Volume 19 Number 1 February 1995 ISSN 0350-5596

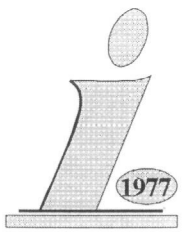
Informatica

An International Journal of Computing and Informatics

Special Issue: Parallel and Distributed Real-Time Systems

Guest Editors: Marcin Paprzycki
Janusz Zalewski

Profile: Haneef Fatmi



The Slovene Society Informatika, Ljubljana, Slovenia

Informatica

An International Journal of Computing and Informatics

Basic info about Informatica and back issues may be FTP'd from ftp.arnes.si in magazines/informatica ID: anonymous PASSWORD: <your mail address>
FTP archive may be also accessed with WWW (worldwide web) clients with URL: ftp://ftp.arnes.si/magazines/informatica

Subscription Information: Informatica (ISSN 0350-5596) is published four times a year in Spring, Summer, Autumn, and Winter (4 issues per year) by the Slovene Society Informatika, Vožarski pot 12, 61000 Ljubljana, Slovenia.

The subscription rate for 1994 (Volume 18) is
– DEM 50 (US$ 34) for institutions,
– DEM 25 (US$ 17) for individuals, and
– DEM 10 (US$ 4) for students
plus the mail charge DEM 10 (US$ 4).

Claims for missing issues will be honored free of charge within six months after the publication date of the issue.

LaTeX Tech. Support: Borut Žnidar, DALCOM d.o.o., Stegne 27, 61000 Ljubljana, Slovenia.
Lectorship: Fergus F. Smith, AMIDAS d.o.o., Cankarjevo nabrežje 11, Ljubljana, Slovenia.
Printed by Biro M, d.o.o., Žibertova 1, 61000 Ljubljana, Slovenia.

Orders for subscription may be placed by telephone or fax using any major credit card. Please call Mr. R. Murn, Department for Computer Science, Jožef Stefan Institute: Tel (+386) 61 1259 199, Fax (+386) 61 219 385, or use the bank account number 900–27620–5159/4 Ljubljanska banka d.d. Slovenia (LB 50101-678-51841 for domestic subscribers only).

According to the opinion of the Ministry for Informing (number 23/216-92 of March 27, 1992), the scientific journal Informatica is a product of informative matter (point 13 of the tariff number 3), for which the tax of traffic amounts to 5%.

Informatica is published in cooperation with the following societies (and contact persons):
Robotics Society of Slovenia (Jadran Lenarčič)
Slovene Society for Pattern Recognition (Franjo Pernuš)
Slovenian Artificial Intelligence Society (Matjaž Gams)
Slovenian Society of Mathematicians, Physicists and Astronomers (Bojan Mohar)
Automatic Control Society of Slovenia (Borut Zupančič)
Slovenian Association of Technical and Natural Sciences (Janez Peklenik)

The issuing of the Informatica journal is financially supported by the Ministry for Science and Technology, Slovenska 50, 61000 Ljubljana, Slovenia.

PROFILES

Profiles is a peculiar gallery of personalities in the field(s) covered by *Informatica.* To remember the readers, in this column we have had the profiles of the following researchers: Terry Winograd—AI and cognitive science philosopher and researcher (1/93); Jiří Šlechta—physicist and cyberneticist (2/93); Hubert L. Dreyfus—American philosopher of Being and critic of the traditional AI approach (3/94); Gheorghe Tecuci—machine learning and knowledge acquisition researcher (4/93); Robert Trappl—AI scientist and medical cyberneticist (1/94); Branko Souček—computer engineer and author of the six generation computing (2/94); and Hiroaki Kitano—inventor of computer simultaneous translation and AI researcher (3/94).

Professor Haneef Fatmi is a well-distinguished researcher, teacher, and organizer especially in the field where the most attractive development of scientific thought and technological development is expected, that is, in the field of cybernetic systems and machines. It is to stress that he is the chairman of a symposium held within the large cybernetics conference in the coming August, in Namur, Belgium.

Haneef Fatmi

Haneef Fatmi is currently Honorary President of the Cybernetics Society of the United Kingdom. He is also a member of the Governing Body of the International Association for Cybernetics, Namur, Belgium and the Governing Body of the journal "Cybernetica". In addition he is a Senior Member of the American Institute of Electronic and Electrical Engineers. For his outstanding services for the advancement of Cybernetics and information theory he was presented a Deed of Appreciation by the Cybernetics Society in 1992, founding in perpetuity a "Fatmi Lecture in Cybernetics".

Dr. Fatmi was educated at the Inns of Court School, Lincoln's Inn, London, where he obtained his Barrister's Degree, and Imperial College London where he was awarded a PhD degree for his work on plasma electrodynamics and information theory in 1961. From 1961 he worked on various problems of plasma electrodynamics and information theory in collaboration with Professor Dennis Gabor, FRS, Nobel laureate at Imperial College London, and the Atomic Energy Research Establishment Harwell, U.K. In 1968 he was appointed Director of the Cybernetics Research Group at the University of London and served in that capacity until recently. He has successfully directed over 30 PhD and 100 Master's projects in cybernetics.

In 1970 Haneef Fatmi published a world famous Definition of Intelligence in *Nature,* London which was a subject of comments by the leading researchers all over the world. It was republished by the Institute of Physics as one of the 2000 leading quotations of all times. The subject matter of the Definition was used by Professor H.B. Barlow to investigate a new approach to the measurement of intelligence and to relate guesswork, language and intelligence under the same common ground.

During the last 25 years Haneef Fatmi and his collaborators published over 150 technical papers on various aspects of cybernetics, informatics, artificial intelligence, robotics, knowledge-based systems, communications, control and information theory, all over the world.

His main research interests include the development of a novel approach to human psychology based on the understanding of psychophysiological mechanism of perception and awareness; development of a novel theory of cybernetics and intelligence machines; definition of intelligence in humans and the machines; intelligent neural networks and systems; distributed computing, control and communications networks; compression of data by pattern recognition and ruled-based algorithms; forecasting of financial data. Some of his papers has been published in *Informatica.*

Publications

Limited space does not allow to give a complete list of Dr. Fatmi's publications. Below is a short list of selected papers.

Gabor, D. and Fatmi, H.A., "A thermionic generator," Nature, London, 1961, 868.

Gabor, D. and Fatmi, H.A., "The theory of gas discharges with extraneous ion supply," Advanced Energy Conversion, IEE, London, 1964, 307.

Fatmi, H.A. and Young, R.W., "A definition of

intelligence," Nature, London, 1970, 97.

Fatmi, H.A., "The concept of a creative society," Electronics and power, IEE London, 1974.

Fatmi, H.A. and Resconi, G., "A new computing principle," Nuovo Cimento, Bologna, Italy, 1988, 239–242.

Fatmi, H.A. et al, "Theory of cybernetics and intelligent machine," in Proc. General Systems, USA, 1990, 123–164.

Lee C.C. and Fatmi, H.A., "Run-time support for parallel functional programming," J. of Systems and Software, Sept. 1991.

Fatmi, H.A. et al, "The nature of the stochastic method in cybernetics," in Proc. Int. Congress on Cybernetics, Belgium, 1972, 328–333.

Fatmi, H.A. et al, "Principles of discrimination in pattern recognition," Biokybernetik, Germany, 5, 1974, 281–287.

Fatmi, H.A. et al, "The concept of associative memories," in Proc. Int. Congress on Cybernetics, 1980, 303–310.

Fatmi, H.A. et al, "Parallel processors for cybernetic systems," Cybernetics and Systems, USA, 11, 1982, 179–192.

Fatmi, H.A. and Todd, S.J., "A cybernetic approach to intelligence based space system," in Proc. IEEE-SMC, New York, 1176–1181.

Ciupa, M. and Fatmi, H.A., "An expert system for data networks and services offering strategic planning support," in Proc. Network 90, UK, 241–250.

Oliver, A. and Fatmi, H.A., "Nonlinear adaptive filtering by the Gabor-Kolmogorov method," IEE Control, UK, 1991, 105–110.

Russel, I.E. and Fatmi, H.A., "A novel approach to interface design for a neural network expert system," IEEE IJCNN, USA, 1991, 384–388.

Sherif, H.T. and Fatmi, H.A., "3-dimentional moving object recognition by the neuro-optic field," IEEE IJCNN, China, 1992, 637–640.

Gilani, S. and Fatmi, H.A., "Organizational professional contextual issues," in Proc. Int. Congress on Cybernetics, 1992, 199.

Khan, H.U., Ahmad, J., Mahmood, A. and Fatmi, H.A., "Text compression as rule based pattern reorganization," IEE Electronics Letters, 29(20), 1993, 1752–1753.

Khan H.U. and Fatmi, H.A., "Text compression using rule base encoder," IEE Electronics Letters, 30(3), 1994, 199–200.

Mahmood, A., Khan, H.U. and Fatmi, H.A., "Adaptive file allocation in distributed information systems," Informatica, 18(1), 1994, 37–46.

Mahmood, A., Khan, H.U. and Fatmi, H.A., "Data reorganization in distributed information systems," Informatica, 18(3), 1994, 325–336.

Mahmood, A., Khan, H.U. and Fatmi, H.A., "Adaptive file allocation in distributed computer systems," IEE Distributed systems and engineering journal, December 1994.

Khan, H.U., Mahmood, A. and Fatmi, H.A., "A novel approach to text compression," Informatica, 18(4), 1994, 485–490.

Ahmad, J. and Fatmi, H.A., "Recognition of objects data in computer integrated manufacturing," in Proc. IEE Control'94, 1994, 805–808.

Ahmad, J. and Fatmi, H.A., "Signal recovery by feed forward neural networks," in Proc. ITA'94, UK, 1994, 79–83.

Ahmad, J. and Fatmi, H.A., "A novel method of speech recognition using feed forward neural network," in Proc. IEEE-SMC, USA, 1994, 21–25.

Russel, I. and Fatmi, H.A., "A novel definition of expert knowledge in expert systems," in Proc. IEEE-SMC, USA, 1994, 2208–2211.

Khan, H.U. and Fatmi, H.A., "Application of pattern recognition in text compression," in Proc. IEEE-SMC, USA, 1994, 1657–1659.

Ahmad, J. and Fatmi, H.A., "Quadric neural network for the prediction of financial time series data," IEE world congress on computational intelligence, USA, 1994, 3667–3670.

Edited by *A.P. Železnikar*

PARALLEL AND DISTRIBUTED REAL-TIME SYSTEMS: INTRODUCTION TO THE SPECIAL ISSUE

Marcin Paprzycki
Science and Mathematics
University of Texas–Permian Basin
Odessa, TX 79762-0001, USA
paprzycki_m@gusher.pb.utexas.edu
and
Janusz Zalewski
Dept. of Computer Science
Embry-Riddle Aeronautical University
Daytona Beach, FL 32114-3900, USA
zalewski@db.erau.edu

The purpose of this Introduction is to present the rationale behind selecting the structure of this Special Issue. It follows the general scheme of real-time systems' development: from requirements specification, to design, to implementation. A bibliography of books on parallel and distributed real-time systems for the last ten years is also included.

Real-Time Systems have two major characteristics: they always interact with the environment other than the human operator, and usually deal with timing constraints, mostly in a form of deadlines on the reaction to external stimuli. Because responsiveness and timeliness are so important in their behavior, real-time systems are almost exclusively concurrent, that is, consist of multiple program units, usually called tasks or processes, running simultaneously to perform required functions. Concurrent execution of tasks on a single processor may be in many respects inadequate for achieving the required level of performance or required level of reliability – the two primary system requirements. Therefore the tasks are often moved to different interconnected processors, making a real-time system parallel or distributed. Although it is sometimes hard to distinguish between parallel and distributed systems, especially in real-time computing, the principal distinction between the two is that of the communication speed versus processing speed. If the communication time between processing units is negligible with respect to the processing speed, then the system is called parallel; otherwise it is distributed.

Our approach to real-time systems, in general, is based on the system development view: from application requirements, to specification and design, to implementation issues considered on three different levels, that is, programming languages, operating systems, and hardware architectures. From the system development point of view, it does not make any difference whether a system is to be implemented on a single processor or on a parallel or distributed architecture; the development process must proceed in the same way. Therefore the sequence of articles in this special issue resembles the system development process and is structured in that way.

The first article, by McKay and Atkinson, discusses one of the most demanding applications for a real-time system: a part of the NASA's Mission project. Our interest in this paper is not that much in the solutions, which are described on a relatively general level, but in system requirements, which include reliability, safety and security. The major characteristic of an application described in this article is that as systems get more and more complicated, a unified approach to account for critical system properties, such as those listed above, combined with real-time properties is needed. Such systems are usually called high-assurance systems or high-integrity systems.

Because of extremely critical nature of high-assurance systems, whose failure may involve loss of lives, loss of precious property or significant environmental damage, unconventional development methods are needed to ensure their correctness. One very promising, although not fully te-

sted yet, approach to ensure correctness on the high level of development is the use of formal methods. These are the methods that employ proof techniques to ensure that the system is correct. In the article by van Katwijk and Toetenel, one such formal method, named MOSCA, is presented. MOSCA, based on an extension of VDM (Vienna Development Method), is a specification language providing facilities to specify real-time requirements for parallel and distributed applications.

Moving from the specification to the design level, system developers need to be equipped with modern methodology, usually consisting of the rigorous notation, techniques for development, and support tools. One such approach, object-oriented technology received significant attention in the last decade, but not necessarily for real-time systems development. In the article by Lin, Kung and Hsia, an object-oriented approach is presented to designing real-time systems whose critical system properties constitute a dominant part of the requirements and play a significant role in the development.

To convert the system/software design into a running application, the implementor usually faces a problem of dealing with programming language constructs to support design concepts. Of the many constructs that explicitly support real-time, parallel, and distributed programming, one still needs further development: exception handling. Colnarič, Verber and Halang deal with eception handling in their paper. Again, this problem is especially important because of the necessity to meet critical requirements in exceptional situations and the major contribution of this article is in providing exceptions on the language level.

If the implementation is to run according to the specification, the language constructs ought to be adequately mapped onto and supported by the operating system kernel. A real-time kernel, and especially a parallel or ditributed real-time kernel, needs to provide specific functionality which is very different from traditional understanding of an operating system. Corresponding problems are so critical that this special issue includes four articles related to this subject.

The first paper in this group, by Yu and Welch, presents an off-line scheduling approach based on the analysis of tasks' behavior for concurrency enhancement. The second paper, by Davari and Dhall, discusses two heuristic on-line algorithms to solve the allocation problem, that is, the assignment of tasks to processors so they can successfully meet deadlines. The next paper, by Erciyeş, Özkasap and Aktaş, describes a dynamic load balancing mechanism for massively parallel processing systems, and finally, a paper by Wòjcik and Wòjcik presents a universal method of achieving fault tolerance in a distributed system via checkpointing.

As the implementors well know, finding the perfect solutions to the most difficult software problems may be not enough, if the underlying hardware architecture is not functioning properly. From the multitude of problems which can be listed on the architecture level of a parallel or distributed real-time system, only one is tackled here, that of hardware guarantees on communication deadlines. Tchouaffe and Zalewski, in their article, deal with the problem of predictability of Ethernet – one of the most widely used local area networks.

Although we attempted to provide readers with a comprehensive coverage of problems and their solutions in parallel and distributed real-time systems, certainly no such coverage can be exhaustive. Those readers who are especially interested in this topic and want to pursue further studies may want to look into three other collections of papers [12, 18, 27] or access some of the books on this subject which have been published throughout the last ten years and are listed below.

Acknowledgements

The following reviewers are gratefully thanked for their time and effort to make this special issue a reality:

- Azer Bestavros, Boston University, USA
- Travis Craig, University of Washington, USA
- Hesham El-Rewini, University of Nebraska–Omaha, USA
- Rod Howell, Kansas State University, USA
- Eric Johnson, New Mexico State University, USA

- Gilad Koren, Bar-Ilan University, Israel
- Phil Laplante, Fairleigh-Dickinson University, USA
- Bud Lawson, Lawson Publishing & Consulting, Sweden
- Joseph Y-T. Leung, University of Nebraska–Lincoln, USA
- Doug Locke, Loral Federal Systems, USA
- Daniel Mossé, University of Pittsburgh, USA
- Jerzy Nogieć, Fermilab, USA
- Warren Persons, Lawrence Livermore National Laboratory, USA
- Gary Preckshot, Lawrence Livermore National Laboratory, USA
- Michael Quinn, Oregon State University, USA
- Felix Redmill, Redmill Consultancy, UK
- Bo Sanden, George Mason University, USA
- Przemysław Stpiczyński, University of Maria Curie-Skłodowska, Poland
- Ken Tindell, University of York, UK

References

[1] Atkinson C., T. Moreton, A. Natali (Eds.), Ada for Distributed Systems, Cambridge University Press, London, 1988

[2] Bishop J. (Ed.), Distributed Ada: Developments and Experiences, Cambridge University Press, London, 1990

[3] Conte G., D. Del Corso (Eds.), Multi-Microprocessor Systems for Real-Time Applications, Reidel Publishing, Dordrecht, 1985

[4] de la Puente J.A., M.G. Rodd (Eds.), Distributed Computer Control Systems 1994, Pergamon Press, Oxford, 1994

[5] Fleming P.J. (Ed.), Parallel Processing in Control: The Transputer and Other Architectures, Peter Peregrinus, Stevenage, UK, 1988

[6] Fortier P.J., Design and Analysis of Distributed Real-Time Systems, McGraw-Hill, New York, 1985

[7] Garcia Nocetti F., P.J. Fleming, Parallel Processing in Digital Control, Springer-Verlag, London, 1992

[8] Irwin G.W., P.J. Fleming (Eds.), Transputers in Real-Time Control, John Wiley and Sons, New York, 1992

[9] Knuth E., M.G. Rodd (Eds.), Distributed Databases in Real-Time Control, Pergamon Press, Oxford, 1989

[10] Kopetz H., M.G. Rodd (Eds.), Distributed Computer Control Systems 1991, Pergamon Press, Oxford, 1992

[11] Lawson H.W., Parallel Processing in Real-Time Applications, Prentice Hall, Englewood Cliffs, NJ, 1992

[12] Lawson H.W. (Ed.), Special Issue on Parallel Processing in Embedded Real-Time Systems, Microprocessing and Microprogramming, Vol. 40, No. 2-3, April 1994

[13] Liebowitz B.H., J.H. Carson, Multiple Processor Systems for Real-Time Applications, Prentice Hall, Englewood Cliffs, NJ, 1985

[14] Lukas M.D., Distributed Control Systems: Their Evaluation and Design, Van Nostrand Reinhold, New York, 1986

[15] Motus L., S. Narita (Eds.), Distributed Computer Control Systems 1989, Pergamon Press, Oxford, 1990

[16] Nielsen K., Ada in Distributed Real-Time Systems, McGraw-Hill, New York, 1990

[17] Popovic D., V.P. Bhatkar, Distributed Computer Control for Industrial Automation, Marcel Dekker, New York, 1990

[18] Reeves D.S., K.G. Shin (Eds.), Special Issue on Parallel and Distributed Real-Time Computing, IEEE Parallel and Distributed Technology, Vol. 2, No. 4, Winter 1994

[19] Rodd M.G., Th. Lalive d'Epinay (Eds.), Distributed Computer Control Systems 1988, Pergamon Press, Oxford, 1989

[20] Rodd M.G., K.D. Mueller (Eds.), Distributed Computer Control Systems 1987, Pergamon Press, Oxford, 1988

[21] Rogers E., Y. Li (Eds.), Parallel Processing in a Control Systems Environment, Prentice Hall, Englewood Cliffs, NJ, 1993

[22] Schuetz W., The Testability of Distributed Real-Time Systems, Kluwer Academic Publishers, Boston, MA, 1994

[23] Stein R.M., Real-Time Multicomputer Software Systems, Ellis Horwood, New York, 1992

[24] Suski G.J. (Ed.), Distributed Computer Control Systems 1985, Pergamon Press, Oxford, 1986

[25] Theus J., The Futurebus+ Handbook, VITA, Scottsdale, AZ, 1993

[26] Thoeni U., Programming Real-Time Multicomputers for Signal Processing, Prentice Hall, New York, 1994

[27] Welch L., D.K. Hammer (Program Chairs), Proc. 2nd IEEE Workshop on Parallel and Distributed Real-Time Systems, Cancun, Mexico, 28-29 April 1994, IEEE Computer Society Press, Los Alamitos, CA, 1994

[28] Zalewski J. (Ed.), Advanced Multimicroprocessor Bus Architectures, IEEE Computer Society Press, Los Alamitos, CA, 1995

Supporting the Evolution of Distributed, Non-stop, Mission and Safety Critical Systems

Charles W. McKay and Colin Atkinson
University of Houston - Clear Lake
2700 Bay Area Boulevard, Houston, TX. 77058.
Phone: +713 283 3830, Fax: +713 283 3869
E-mail: mckay@cl.uh.edu

Keywords: distribution, environments, non-stop, real-time, safety-critical

Edited by: Marcin Paprzycki and Janusz Zalewski
Received: February 12, 1994 **Revised:** October 20, 1994 **Accepted:** January 9, 1995

In coming years embedded systems which are distributed, non-stop and "mission and safety critical" (MASC) are likely to assume increasing importance. The construction, operation and maintenance of this class of system presents a unique blend of problems which many traditional tools and techniques, targeted to just one problem area, cannot currently address. This paper provides an overview of a promising, model-based framework for supporting such systems that has been developed as part of NASA's MISSION project. Based on well-established research advances in computing, the MISSION approach provides a domain-specific, life-cycle support framework encompassing three separate environments: host, integration and target. Although the individual elements of the framework are not all new, their synergistic packaging within the MISSION project is believed to be unique. This paper focuses upon the systems-level support for applications executing in the target environment.

1 Introduction

An embedded system is a computer system which is constructed to monitor and/or control a set of devices and processes constituting some larger engineering system. The term "embedded" is used to reflect the fact that such computing systems are physically encapsulated by the engineering system they monitor/control. An important characteristic of embedded systems is that they are typically real-time - not only must they produce the correct result, but they must do so within a specified period of time. Because of their monitoring and controlling role, the reliable execution of an embedded system is often critical to the success of the overall mission and to the safety of life, health, property or the environment. In such circumstances the embedded system is termed a mission and safety critical (MASC) system.

As the reliability and efficiency of networking technology has increased, and the cost of microprocessors has plummeted, there has been an increasing trend towards the implementation of embedded systems as distributed systems made up of autonomous, cooperative processors interconnected by communication channels. Not only does such an implementation enable processing power to be located physically close to the individual devices in the system, but it also opens up the possibility of extending, or modifying, parts of a system while other parts are still running. In other words, it opens up the possibility of building non-stop systems which can be dynamically upgraded and reconfigured.

In coming years there is likely to be an increasing need for embedded systems which exhibit all the properties identified above, namely the properties of being mission and safety critical, real-time, distributed and non-stop. Such systems are essential in extremely hostile and/or inaccessible environments, such as space or the depths of the ocean, and are therefore crucial to pending NASA projects (e.g., space station, lunar outpost, human missions to Mars). Such systems are also

likely to be used in large process control applications such as factory automation, power plant control, etc.

In recent years numerous projects have addressed one or more of the issues mentioned above. To meet the real-time requirements of embedded systems, for example, advanced scheduling techniques have been developed (e.g., rate monotonic scheduling [37] and best effort decision making [20]). The requirements of distribution, on the other hand, are addressed by new and more powerful networking hardware and communications protocols such as the Open Systems Interconnection Model [33]. Reliability and safety are addressed by advanced software features such as distributed nested transactions [24], while the needs of non-stop operation and dynamic upgradeability [44, 42] are addressed by modular approaches to operating system organization.

Because of the complex way in which the above characteristics are interrelated in embedded systems, however, it is not always possible to use these tools and techniques together in a system which exhibits several, if not all, of these properties. Often a technique which is very successful at solving one particular problem cannot be used with another technique developed to solve another problem because of the way they overlap and interact. The different techniques, and in particular the combination of technologies, have the potential to introduce new problems or exacerbate others. This difficulty is compounded by the fact that systems of this kind are inherently complex and typically very large. In fact, some of the largest software systems to date fit into this category.

For this reason, rather than tackling individual aspects of the problem of supporting the evolution of non-stop, distributed, real-time, MASC systems, the MISSION[1] project has focused on defining the overall development strategy and infrastructure into which such solutions will fit. Specifically, this work has two main thrusts. The first part is to lay the foundation for a new generation of integrated systems software for the target environment in which MASC computing applications are deployed and operated. The second part is to define an accompanying infrastructure which is capable of supporting the construction, verification, reuse and maintenance of the kind of software artifacts required in the target environment. The MISSION approach is believed to be unique in the integration of these advancements across the three environments.

This paper provides an overview of the MISSION approach for supporting distributed, non-stop MASC systems with a particular focus upon the systems software support for applications executing in the target environment. Before describing the approach itself, however, we first describe the main issues that arise in the construction and maintenance of this type of system. In addition to providing a definition and description of each issue, we identify some of the applicable terminology and technologies. The following section then describes the MISSION strategy for dealing with these issues, first introducing the general context in which MASC software is developed, operated and maintained, and then describing the target architecture. We conclude by describing each of the subsystems making up this architecture.

2 Principal Issues

Important issues and requirements for MASC computing systems operating in hostile environments have been discussed in publications such as [1, 14, 36, 38]. This section discusses only five of the principal issues: life cycle approaches; distribution; safety; reliability, security and integrity; and fault tolerance.

Clearly, the requirements for the project as a whole are driven by the target environment. The life cycle requirements for the integration environment, which serves as the site from which the target is monitored, controlled and updated, are principally driven by the need to provide safe and affordable support for the target environment over its complete lifetime. The requirements for both the target and integration environments are, in turn, the principal drivers of the life cycle requirements of the host environment, which is the place where the initial application development and testing takes place. Since the entire set of life cycle requirements for this class of MASC computing applications and systems will probably never be known in advance, an iterative approach to life cycle support is essential.

[1] MISsion and Safety crItical SuppOrt ENvironment

2.1 Life Cycle Approaches

As might be expected, one of the major deficiencies in the current state of the practice for this domain is the lack of predictably-dependable, integrated approaches [11, 23, 29]. Such approaches should be traceable, controllable, and applied iteratively from the system's initial inception through to its retirement. MISSION's goal of defining and verifying such approaches is mirrored in other projects such as Spring [11] and the PDCS project [29].

An important goal of MISSION is to demonstrate that an object-oriented discipline can be used to control the complexity of this MASC target environment. Related issues include the application of the object-oriented discipline to the design of the generic architecture for the target environment systems software. Of particular importance is the evolution of a MASC kernel for this systems software [26, 39, 29]. The kernel is intended to provide a small but powerful set of mechanisms designed especially to support tractable, rigorous reasoning about MASC functions and systems. Support for such reasoning is critical for the infrastructure in the integration and host environments. In addition, safe and affordable approaches should consider the integrated issues of the software, (both applications level and systems level), the hardware, communication links and human-machine subsystems as well as interactions with the environment in which the system is deployed and operated. Techniques currently addressing these system level issues are not well integrated. The Alpha project [26] shares the goal of using the object paradigm to develop systems software that supports tractable, rigorous reasoning about MASC properties.

2.2 Distribution

Providing support for distributed operations is both a problem and an opportunity. Distribution should facilitate new and more powerful forms of fault tolerance along with opportunities to improve performance for real- time command and control systems [30, 41]. Related issues include when and how to assign software components to physical processing sites [5] and what support can and should be provided for migrating components among processing sites [45]. This support must be integrated with the ability to dynamically evolve and reconfigure both the applications and the systems software in the non-stop, distributed target environment (DTE). Unfortunately, no known system currently integrates a full set of acceptable solutions to these requirements with the needed attention to safety.

The need to capture a broad spectrum of information for system objects is even more crucial when real-time decisions are to be made [40]. In a distributed system the universal system state changes faster than can be communicated throughout the system [15]. Furthermore it may never be possible to "snap-shot" a view of the entire system state at any point in time. Decisions therefore must often be made in environments of incomplete and sometimes inaccurate data [20]. The goal of safely supporting dynamic evolution and reconfiguration of non-stop, distributed systems is shared by the Real Time Mach project [43].

2.3 Safety

The following working definition of safety is used in this project "safety is the probability that a system, including all hardware, software, communication links, human-machine subsystems, and interactions with the environment, will provide appropriate protection against the effects of faults, errors, and failures which could endanger life, health, property, or the environment." Safety depends upon related issues such as integrity, reliability, security and others to be discussed in the following subsections. Safety cannot be guaranteed, especially not for the class of MASC computing applications under discussion in this paper. Many important risks, nevertheless, can be managed to improve the probability of sustaining safety across the life cycle [28, 7]. MISSION supports the traditional goal for aerospace applications that no single point of failure can endanger a mission and no two points of failure can endanger safety.

Safety is the most important aspect of any distributed MASC computing system. The system must guard itself against any event or action, intentional or accidental, that compromises its safety [6]. Safety requirements should be considered at each point of the system's life cycle [19, 34].

The ultimate aim of the work reported in this

paper is to define a small but powerful set of constructs that can be used to compose MASC computing applications and systems. These constructs are being defined to support safety properties. Systems composed of such constructs should facilitate tractable, rigorous reasoning about safety. The MISSION project is fairly unique in its emphasis on evolving and verifying approaches to composing safe, non-stop, real time, distributed systems.

2.4 Reliability, Security and Integrity

The safe and affordable support of lives, health, property, environment, and mission in the target environment depend upon system level reliability, security and integrity. System reliability refers to the ability of the system to function under stated conditions for a stated period of time [25], and should be maximized for MASC applications and systems. This requires more than certification of correct software components and highly reliable hardware components. It also requires systems level design for fault tolerance and survivability [16, 31].

System security refers to the protection of the system from accidental or malicious access, use, modification, destruction, or disclosure [9]. Distributed systems which support a diverse group of users are particularly vulnerable to problems which result from improper access to information and other resources. At the minimum, protection is necessary for inadvertent access due to program or operation error. At the other extreme, deliberate disruption must be prevented. The MISSION project seeks to provide security to at least the multilevel security class B3 of the DoD standard for security [9]. Such security should be supported within the target environment and in all its interactions with the integration environment.

System integrity refers to the ability of the system to perform its intended function irrespective of changes in its operational environment [32, 8, 31]. The MISSION approach for ensuring integrity in the target environment builds upon research in executable assertions[35]; monitors [18]; checkpointing and recovery schemes [21]; and distributed, nested transactions [24]. The approach also introduces the concept of the integration environment. These aspects of the approach are discussed in more detail in the following section.

2.5 Fault Tolerance and Recovery

In a perfect world, functionally correct software, hardware, communication links, and human machine subsystems would operate safely and reliably in their intended environment. Unfortunately, in the domain addressed by MISSION, faults, errors, and failures will occur which could be disastrous if not detected and handled properly. MASC systems are needed which can tolerate such problems or, when the problems cannot be tolerated, enact survivability policies.

A failure means that a functional unit can no longer satisfy its requirements at run-time, and may be caused by a defect in the software design or implementation. A fault occurs at run-time and may leave errors in some part of the system, and may sometimes lead to failures. Detection may refer to the detection of either a fault, an error or failure [27]. Recovery refers to the process of restoring normal operation after the occurrence of a fault or failure [21].

Classes of faults, errors, failures, and their combinations should be identified and prioritized according to their probability of occurrence during execution, and the consequences of not properly dealing with them [7, 12]. A safe system is not only able to monitor its status and detect an occurrence of such classes as soon as possible, but can also analyze and control the propagation of the effects and recover safely.

The fundamental issue behind MASC software support is handling the consequences of faults. Two approaches are commonly identified: fault tolerance and fault avoidance. Fault avoidance depends on ultra-reliable hardware, early detection of low-level faults with redundant processing, and the ability to use this redundancy to mask faults in the system from its environment. Specifically, the faults are masked from the system state vectors. Avoidance techniques are valuable but not sufficient [13, 41].

Large, complex systems with intricate dynamic interactions severely limit the ability of fault avoidance to assure safe and correct performance. Even if systems with millions of lines of defect-free code could be built (and they currently can not), they would not execute without faults, errors and failures throughout a long, non-stop, operational

lifetime. Some combination of hardware failure, communication links failure, operator errors, latent software defects or acts of providence will cause problems at runtime. Many of these can be tolerated if the software is built to do so. Others cannot be tolerated but survivability can be maximized if the software is so designed [26].

Fault tolerance is a complementary approach to fault avoidance. Fault tolerance is based upon the assumption that any computation might become defective and result in an erroneous system state vector. Either forward or backward recovery schemes may be used to restore the system to a safe and correct state. Since the possibility for the introduction of such problems exists at all levels of the software hierarchy, it should be considered and addressed at all levels. In so doing, the ability to manage or at least mitigate the effects of faults, errors and failures throughout large and complex systems may be made possible [4, 12, 13, 16, 17, 41]. The MISSION goal to leverage combinations of fault avoidance and fault tolerance in support of MASC requirements is similar to a goal of the MARS project [17].

3 The MISSION Approach

The previous section has described some of the principal issues involved in the construction and maintenance of distributed, non-stop MASC systems. In this section we provide an overview of the MISSION approach for tackling the issues, and integrating the various separate technologies that have been developed to date. In particular, we describe how the MISSION approach addresses the need for precise (semantic) modeling, three computing environments, and a generic architecture for the systems software that executes MASC applications.

3.1 Semantic Modeling

As depicted in Figure 1, the key requirements originate within the distributed target environment (DTE), flow traceably and cumulatively across the integration environment to the host environment and back. System level modeling is fundamental to improved understanding and progress toward safe solutions. The need for such modeling extends beyond the final executing system, and encompasses also the interrelated processes that produce the improved solutions. Such modeling of products and processes has implications for all three environments in the MISSION approach.

A key requirement for an integrated solution is the capability to model system level components and interrelationships among software, hardware, communication links, human-machine subsystems, and their operational environment. The representation of such system level components and their interrelationships should facilitate automated support for tractable, rigorous reasoning about their MASC properties.

To respond to these needs, the MISSION team has adapted an object-oriented modeling approach developed by Embley, Kurtz and Woodfield [10] and augmented the approach with additional semantics in entity- attribute/relationship-attribute (EA/RA) form. The approach by Embley et. al. is based upon a formal definition and depicts object-oriented models in three views. Object-relationship models provide the structural view of the part of the system being modeled. The behavior of each object class that appears in the object-relationship models is depicted in an object-behavior model. Interactions among object classes are depicted in object-interaction models. Although the combination of the three modeling views does support a large degree of tractable, rigorous reasoning about the systems being modeled, the semantics defined in the approach do not provide sufficient granularity to capture all details of interest in the MISSION project. Examples include redundant objects, bindings between software and hardware, workload profiles, reconfiguration of systems resources, etc. An entity- attribute/relationship-attribute (EA/RA) form of representation which has been systematically extended to include object classes, relationship sets, states, transitions, interactions and attributes is a feasible choice for representing these system level components, interrelationships, and their MASC properties. The IRDS standard [3] for this form of semantic representation has been legally extended by the MISSION team to meet these needs. However, a discipline is required to systematically address the inherent complexity within the problem space. The same discipline should also control the associated com-

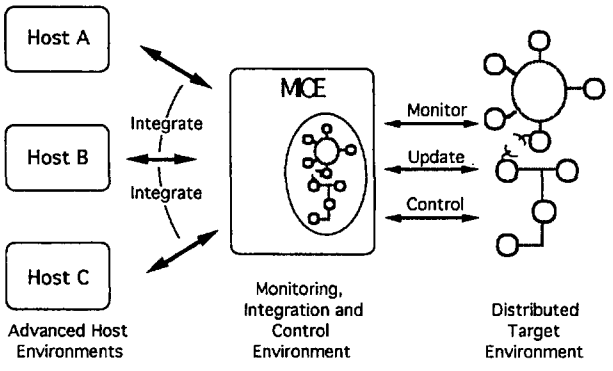


Figure 1: Three Environments

plexity of the processes of evolving and sustaining safe and affordable solutions.

As a scenario to illustrate the modeling discipline and processes advocated by MISSION, consider a proposal to replace and to add types and instances of vehicles in NASA's Space Transportation System. The MISSION process would begin with domain analysis to determine the number of product lines needed (types of vehicles in this example) and the variations needed among instances of each type. Along with attributes such as costs, benefits, risks, opportunities, etc., this "business model" would be captured in object-oriented form and conveyed to the client. Based upon priorities, constraints, and other business and political factors shaping decisions and commitments, the business model would be mapped to a scoping model to identify which product lines and their variations will be evolved, when, and in what order. The object-oriented scoping model would then be mapped to a "concept of operations" model for each product line and its variations. System requirements modeling for the domain would then proceed by revising the concept models to represent common requirements and constraints as-well-as differences among the product lines and their variations. Later, this "domain model" would be mapped to a partitioning and allocation of requirements and constraints among models of: software, hardware and communications, and human interfaces. This stage would be followed by the creation, evaluation and selection of generic architectures appropriate for the domain. The domain engineering process would continue and would eventually be followed by application engineering to create specific instances of the product lines.

Some important points to be noted about this scenario are as follows. First, all products of the process are represented in an extended object- oriented form (i.e., extended via EA/RA notations) whether the products are business models, models of system requirements and constraints, or models of software, hardware and communications, human interfaces, and interactions with the environment. Second, a complete set of semantic information typically requires three views of the object models. Third, tools exist to facilitate such modeling and reasoning about the models. Fourth, the domain engineering processes and the application engineering processes that evolve these products are also represented as object models.

Precise semantic modeling using an object-oriented discipline provides the foundation for constructing system level fault tolerance and avoidance. Systems built from such models can also be designed and verified to enforce policies for survivability when faults and failures occur that cannot be tolerated or avoided. For example, to support fault tolerance, classes of faults, errors, and failures can be identified and modeled for the software, hardware, communication links, human-machine subsystems and operational environment that comprise the intended MASC computing system. Assertions can be formulated to provide context sensitive detection and responses for certain classes of faults, errors, or failures - namely, those classes that are not only likely to occur but which will also produce unacceptable behavior and effects if they not properly handled. One or more monitors to enforce these assertion checks and responses can then be generated to accompany the functional software to the target environment.

Of the research projects that focus on domains overlapping with that of MISSION, MISSION is somewhat unique in its emphasis on process and methodologies that leverage object modeling as a unifying paradigm at the systems level. Alpha shares the commitment to software objects and Spring shares the commitment to tools and methods for the host and the target environment.

3.2 Three Environments

Developers of software for embedded systems have traditionally been concerned with two environments: the host environment (the computers on

which all software requirements analysis, design, implementation, and testing is performed) and the target environment (the embedded computers on which the software is intended to execute). However, these two types of environments are insufficient for MASC systems which are developed by several different organizations, and which are required to execute non-stop. Typically there will be many "host" environments, each used to develop a part of the final system. For example, different host environments could be responsible for different (sub)applications to be added to the existing system. To enable the products from the various "hosts" to be combined, and to provide an interface to the software executing on the target environment, MISSION envisions a third environment - the monitoring, integration and control environment (MICE). The provision of a coherent framework for modeling the structure and behavior of MASC systems impacts all three environments throughout the full life-cycle of the system.

The Monitoring, Integration and Control Environment (MICE), is intended to mitigate the risk in evolving and sustaining remotely distributed, non- stop, MASC computing applications and systems. The MICE serves as an interface between the various hosts and the target environment and is the environment where software from the hosts is integrated. The MICE additionally serves to safely upgrade software components in the target environment, monitor the performance of the target environment, and possibly assist the target environment in performing major reconfigurations in response to faults. To properly perform these tasks the MICE must have up-to-date models of the structure, functionality, behavior and constraints of the elements of the executing target environment. The MICE must also present an appropriate command interface, and provide powerful diagnostic support.

The MISSION project is believed to be unique in its attention to the integration environment within a research context, although environments of this type have historically been an important part of NASA applications (e.g., the Mission Control Center for shuttle operations).

3.3 Generic Architecture

A generic solution architecture is proposed for the domain of MASC computing applications and systems addressed by the MISSION research. As shown in Figure 2, the target environment is a distributed system composed of interacting, multiprocessor clusters. Local area networks (LANs) may be configured from these clusters, and wide area networks (WANs) may be configured from these local area networks. The applications software on each cluster is supported by systems software providing intra- and inter-cluster communication and reliable execution in the presence of component failures. To limit the damage caused by faults, and to increase the feasibility of developing and sustaining such a system, the software on the processor clusters is separated into the following "firewalled" partitions[2] -

1. MASC Kernel
2. Distributed Application System (DAS)
3. Distributed Monitoring system (DMS)
4. Distributed Policy Systems (DPS)
5. Distributed Information System (DIS)
6. Distributed Communication System (DCS)

If, for example, a new space vehicle were required, the number and type of applications and the profile of the intended workload can be used to determine how many clusters (and with what resources), and what LAN and WAN resources will be needed.

Much of the research and development of distributed systems has evolved from an assumption of single processor nodes interconnected by LANs and WANs. Even multiple processor nodes have frequently been configured as "N redundant" processors to avoid certain types of faults. In effect, such processors process a single instruction and data stream with a "voting mechanism" to assure majority rule (e.g., the primary flight control system of NASA's space shuttles).

As a partial result of the "single processor node" mind set, attempts to evolve distributed

[2]By firewalled, we mean that certain steps have been taken to ensure that a fault, failure or error in one partition does not adversely affect other partitions.

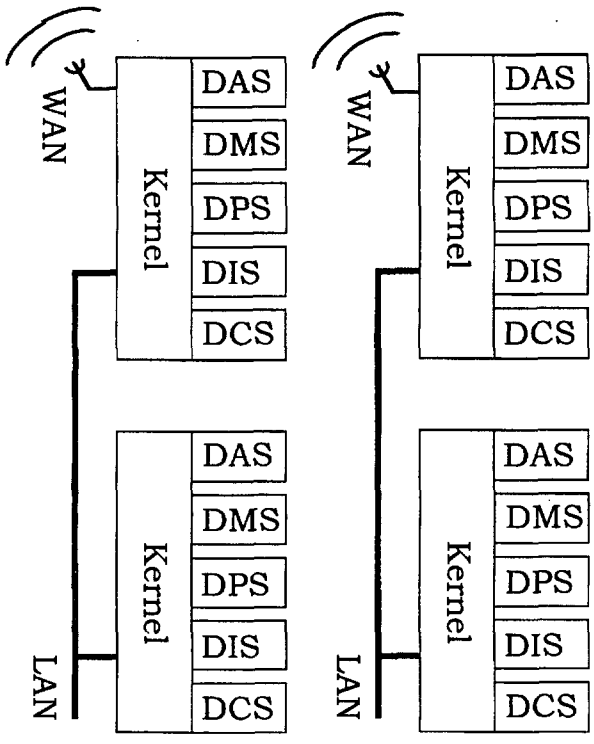


Figure 2: Generic Architecture

systems with tightly constrained, real-time control functions have not been widely successful. Such systems typically experience severe performance problems in meeting their functional requirements. Attempts to integrate a software based approach to supporting systems level fault tolerance tend to exacerbate the overhead problem responsible for the poor performance.

Much of the performance overhead in a single processor node is associated with the time required for context switches. Unfortunately, the elimination of context switches can result in the loss of opportunities to help prevent faults that occur in the execution of one instruction stream from corrupting the subsequent execution of other instruction streams. A key concept of the MISSION approach is to "flatten" the traditional software architecture to take advantage of multiprocessing clusters as illustrated by the cluster architecture in Figure 2. If, for example, such a cluster was located at a geographical site with requirements for four local, hard constrained, real- time control functions, then as many as four or more processors could be assigned to the parallel processing of these control functions. Even if interaction existed among the four functions, parallel processing may offer benefits over a single processor. In the MISSION architecture, the units of functional code are intended to execute in parallel with coroutines on other processors that check for faults, errors, or failures. As long as no flaws are detected, only a minimal performance overhead is added to the execution of the functional code of the applications. Still another performance benefit may be derived by also allowing parallel execution of services and resources that are shared among the applications. For example, persistent information and communications may be organized in such as way as to maximize parallel processing among these subsystems and the applications as indicated in Figure 2.

The MISSION goal to exploit parallel processing capabilities among LANs and WANs of multiprocessing clusters is also a goal of other projects such as Alpha, Spring and Real-Time Mach. The approach to "flattening" the architecture to achieve the intended throughput improvements is particularly evident in Alpha and MISSION.

3.3.1 The Clusters

MISSION clusters have the following properties. Clusters:

- do not share physical memory,
- have access to a hierarchy of memory subsystems including stable storage controlled by transaction mechanisms,
- may be connected to any number of LANs and WANs,
- may have predetermined types of hardware resources, including processors, added to a cluster without changing systems software,
- may fail completely or partially,
- may be repaired and returned to full service, typically without stopping processing,
- may be added/removed at any time,
- may have changes to applications and systems software made without stopping processing, and
- may control access to both physical and virtual systems resources.

3.3.2 The Communications Links

MISSION communications links must be able to tolerate faults, errors and failures which include messages which have lost parts, garbled parts, out-of- order parts, duplicated parts, or parts which are arbitrarily delayed.

3.3.3 MISSION Computing Systems

MISSION computing systems are expected to tolerate, to a specified level, combinations of faults, errors and failures to include: communications failures, abortion of application and system program components, crashes of one or more clusters participating in an application, and lock cycles.

3.4 The Kernel

The MASC kernel is a critical part of the MISSION approach to improving runtime support for the execution and evolution of MASC functions and components in the distributed target environment (DTE). It is similar to the"microkernels" of other projects such as Alpha, Mars and Spring, and provides the foundation on which the firewalled subsystems are built. These mechanisms directly affect the ability of the infrastructure in the integration and host environments to support the DTE. This is because the integrated approaches to semantic modeling are based upon the generic architecture of the DTE systems software.

The kernel is responsible for encapsulating hardware and providing mechanisms to support the policies, operations, and interactions of the other five firewalled partitions. Any communication entering or leaving a partition is a result of invoking the kernel for a message passing service. No direct communication among partitions is allowed apart from with the kernel. The five firewalled partitions, shown in Figure 2, (DAS, DIS, DCS, DPS and DMS) are also referred to as the five firewalled subsystems. Thus, for example a DAS component that wishes to request a resource from the local DIS or from a remote DAS component must invoke a message passing service from the kernel. This modularity allows rigorous reasoning about the kernel independent of the sources or destinations of messages. Since the properties of the structure, functionality, behavior and constraints of the kernel can be assured, the same approach to rigorous reasoning can be independently extended to each of the five firewalled subsystems.

As in [2], MISSION treats the kernel's message-passing relationships with the other subsystems as explicit, first class semantic entities. Protocols are used to describe allowable interactions, their constraints, and their responses to constraint violations, much as in the Mars project. In contrast with the Mars approach, however, MISSION does not assume a clock that is universally available to all clusters in real time.

3.4.1 Twelve Features of the Generic Architecture

The MISSION system architecture embodies twelve features which are either not found at all in today's systems software or are not found as an integrated set. There are at least two important reasons why this set of features is used. First, they facilitate the provision of runtime support needed for the domain of MASC computing applications and systems addressed by the project. Second, they facilitate precise modeling and the associated discipline of rigorous reasoning about the system. These twelve features are identified below:

F1. Model-based reasoning

F2. Firewalled partitions of applications and subsystems

F3. Tailorable interfaces based on classes, objects and messages

F4. Life cycle unique identification of classes, objects and messages at runtime

F5. Extensible and modifiable sets of classes, objects and messages at runtime

F6. Separation of policies and mechanisms

F7. Multiple and adjustable levels of security and integrity

F8. Synchronous and asynchronous communications mechanisms

F9. Adaptable policies for scheduling, redundancy management and the management of other runtime services and resources

F10. Stable storage for checkpointing and recovery

F11. Distributed, nested transactions

F12. "System" level fault tolerance and survivability through systems software.

We elaborate upon these features below.

F1. Model-Based Reasoning

MISSION engineering processes and products emphasize semantically rich, object-oriented models to support tractable, rigorous reasoning about MASC properties. These models can be partially leveraged in the target environment since the kernel contains a finite set of mechanisms designed especially to support the interpretation, maintenance and modification of runtime models. For example, runtime policies are maintained in the DPS as models. In addition, current configuration details are also maintained as on-line models. When an overload condition arises at a cluster, interpretation of the overload policy in terms of the current configuration will determine the response (e.g., load sharing with another cluster or local load shedding).

Although model based reasoning is certainly not new, MISSION is believed to be one of the first projects to investigate its application to nonstop, distributed, MASC systems. Initial studies have focused upon its use in configuration management For example, a resource might initiate one particular recovery response under one set of conditions, and a different recover response under different conditions. Since most elements of the workload and system configuration are well-defined in the DTE models, context sensitive contingency determinations can often be made in parallel with workload processing and be available for rapid response in the presence of one or more anomalies of a predetermined type.

F2. Firewalled Partitions

Firewalled partitions are used in MISSION to maximize the opportunities for identification, isolation, and selection of recovery capabilities. In the host environment, objects are created and assigned to one-and-only-one of the five firewalled subsystems or to the kernel. As the semantic models of the DTE applications and system are evolved, these objects are further allocated to specific clusters. This partitioning and allocation information is exported to the DTE for use by the kernel and the five subsystems. This means that if, for example, an application object executing in a cluster's DAS requests information from an object in the local DIS, the message is passed from the first subsystem to the second by invoking the kernel. Similarly, if an object in the cluster's DAS requests information from a DAS object in a remote cluster, the kernel recognizes that a local object is requesting information from a remote object and invokes the appropriate operation. The message is passed to the local DCS where a communication object will prepare to effect the remote communication.

The result of this organization is to isolate each partition of objects by explicit message passing through the kernel services. For example, suppose a DAS object passes a message to a DIS object which accepts the message and then fails. The opportunities for tolerating the failure are enhanced since the DAS object was preserved in a healthy state when the message was sent. In much the same way, different applications within the DAS, different information systems within the DIS, etc. are also protected from corruption within their own subsystems.

F3. Tailorable and Extensible Interfaces

Dynamic extensibility and other forms of dynamic reconfiguration are facilitated by this feature. Each segment of the generic architecture for the DTE systems software interacts with other segments of the local cluster and with peers in remote clusters through carefully defined interfaces. These interfaces are specified in CIFOs (Catalogues of Interface Features and Options). The interfaces are tailorable in that the given set of applications and system requirements for a given cluster determine which features and options will be selected as CIFO subsets for each cluster. The interfaces are extensible in that precisely modeled rules exist for extending these CIFOs as needed over time. As an example of such rules, no device driver can be replaced until certain preconditions are satisfied such as: "Complete all input/output

operations in progress when the replacement command arrives until a 'recoverable' state is reached. Then effect the replacement."

F4. Life Cycle Unique Identification

This feature also supports tractable and rigorous reasoning about the MISSION models. In the DTE, classes templates, executable images of objects and messages are uniquely identifiable. For example, suppose an object is a part of an application that requires about five minutes to complete and that is intended to run every hour on the hour. The executable image retains its unique identification but, in addition, each hourly activation receives a different thread-of-control identifier. Each thread assignment is provided a unique identification so that the effects of each activation are traceable. Similarly, an iterative object structure may complete and send the same message structure many times during the life span of each object. In the MISSION approach, the effects of each message are intended to be traceable through the unique identifiers of each message, source, and destination(s). The element of the MISSION approach has been strongly influenced by the work of Moss [24].

F5. Extensible and Modifiable RunTime Sets

This feature complements all the preceding features, but is particularly germane to: "F3. Tailorable and Extensible Interfaces". The ability to tailor and extend CIFOs in the host and integration environment is important, but a corresponding capability is needed for objects inside any cluster partition of an operational, non-stop DTE. More specifically, the interfaces to each segment of a cluster architecture should allow existing class definitions internal to the segment to be modified or new ones to be added. Once the modified or new class definitions are installed, the interfaces should encapsulate the ability to create new objects and messages of the new and modified classes. In addition, the interfaces should support the retirement and replacement of old classes, objects, and messages as needed. This mechanism is analogous to the polymorphism/dynamic binding mechanisms of object-oriented languages

F6. Separation of Policies and Mechanisms

This feature not only facilitates tractable, rigorous reasoning, but it also facilitates the domain and application engineering processes through separation of concerns. The MISSION approach partitions and allocates policies to various members of the firewalled subsystems. The shared mechanisms used to effect these policies are in the kernel. For example, the DPS is intended to contain polices for the management of shared services and resources within and among clusters. These policies are encapsulated within DPS modeling objects. The effects are somewhat analogous to earlier techniques of operating systems enforcing "table driven" policies. The interpretation and enforcement of the policies encapsulated by the firewalled subsystems is dependent upon the utilization of the kernel mechanisms. This feature is also supported in Alpha.

F7. Multilevel Security and Integrity

All threads-of-control are created, assigned, sustained and retired via the MASC kernel. A requirement for each active object (i.e., one with its own thread-of-control) is to maintain a registration of its unique identity and its current capabilities. This is particularly important when the active object is about to request a service of another object. A unique identity is required for the destination object and its services and resources. In addition, two other points should be noted. First, the match of a sender's capabilities to a receiver's list of required access rights should be enforced for each access. Second, these rights may sometimes have to be temporarily sacrificed in the cause of higher level policy issues related to a system's fault tolerance and survivability.

F8. Synchronous and Asynchronous Communications Mechanisms

The domain of interest to MISSION researchers includes applications requiring telemetry data to be broadcast as it becomes available and without regard for the status of intended receivers at the time of the broadcast. The domain also includes applications such as multidimensional collision avoidance and proximity operations that require hard constrained, real time synchronization

and control. The literature on communication mechanisms to support distributed and concurrent processing requirements reveals two distinct solutions with certain advantages claimed for each [22].

The first type of mechanism supports the use of asynchronous transmissions and receptions without blocking the sending process or the receiving process(es). Instead, transmission is a case of "send when ready and then proceed". Reception is a case of "receive when ready, if message is available, and then proceed". Variations of this type of mechanism have also been studied.

The second type of mechanism is used for two distinct cases of synchronous communication. The first case involves an active object which calls for a service from a passive object (a passive object borrows its thread-of-control). This case is analogous to a local thread-of-control in a "main" procedure calling a remote subroutine. That is, the thread and its request are passed to the environment of the called subroutine. After "borrowing" the thread-of- control to execute, the passive object returns both the results and the thread- of-control to the calling environment.

The second case of synchronous communications involves a need for synchronization and exchange of information among two-or-more cooperating, active objects. This case addresses, among other things, the issues of the Ada rendezvous among two cooperating threads-of-control. This support for multiple forms of communications is very different than the approaches taken in many other related projects such as Mars which only use datagrams.

F9. Adaptable Runtime Services and Resources

The provision of shared system services and resources to an evolving collection of applications is intended to be based upon well-defined policies, configurations and circumstances. Some resources and services will be replicated to maximize availability and fault tolerance. Such redundancy will need to be managed at a variety of levels. At one extreme, the redundant copies could be managed as "hot standbys" which are ready to be substituted for the primary copy at any time. At another extreme, the redundant copy can be substituted for the primary copy only after processing is performed to prepare the "cold standby" to take over. Depending upon criticality, workload, and the status of system resources, the type and amount of redundancy is intended to vary according to adaptable policies.

Another important aspect of adaptable policies is scheduling. Some real-time applications map naturally to a collection of periodic processes. Others are interrupt driven and are aperiodic. Still others have sporadic service requirements that may be of varying frequency and duration. An important aspect of the approach, therefore, is the use of adaptable scheduling policies to maximize support for MASC functions and components under conditions that vary from normal to various types of emergencies. A similar feature is also found in Real Time Mach.

F10. Stable Storage

Fault tolerance among clusters of distributed MASC systems benefits from the next feature, distributed nested transactions. However, implementation approaches to such transactions require stable storage. Stable storage has two characteristics that facilitate check pointing and recovery. First, it survives temporary losses of power. Second, it is always updated in an atomic operation.

F11. Distributed, Nested Transactions

Fault tolerance among interactive, distributed processing clusters is facilitated by support for distributed, nested transactions [24, 26]. This is particularly true when a fault, failure or error can not be detected in a single state vector, but depends instead upon detection of incorrect sequences of processing. Transactions bracket a named collection of operations between "Begin transaction X" and "End transaction X". The effects of the transaction are to make the set of enclosed operations appear to be a single atomic action. That is, either all of the operations complete successfully or the system can detect and recover from the effects of partial completion. Distributed transactions support hierarchies of parallel and distributed operations. Nesting allows higher level transactions to be composed of sets of enclosed transactions. Transactions of this kind can be used to provide fault tolerance and survivability

in the DTE, and also facilitate reasoning in the host and integration environments. Other related projects employing this mechanism include Alpha and Mars.

F12. System Level Fault Tolerance and Survivability

The MISSION approach leverages systems software to support true systems level fault tolerance and survivability. Since object classes and relationship sets are used to model software, hardware and communications, human interfaces, and interactions with the environment, systems software monitors can be used to monitor and control systems level resources as appropriate.

An important component of the MISSION approach is the concept of coroutines which associate monitors in the DPS with functional objects elsewhere in the system. The job of the monitors is to detect faults, errors, and failures as soon as possible and to then provide support for effecting isolation, analysis, and recovery. Such detection is based upon assertions that are associated with MASC properties. These assertions may be about values of state or about sequences of state transformations. The Mars project also employs kernel-level mechanisms to support system-level fault tolerance.

3.5 Firewalled partitions

As mentioned above, and illustrated in Figure 2, the generic architecture employs five firewalled partitions that interact by means of the message passing services provided by the kernel. In this subsection we outline further the role of each subsytem.

Distributed Applications System

The DAS is the firewalled subsystem containing MASC applications that are to be executed on the MASC computing system. The focus of the research in the DTE is on the generic architecture of the systems software rather than the DAS. The DAS developers are intended to leverage the features and options of this generic architecture to improve runtime support of MASC functions and components.

Only two aspects of the DAS are within the scope of this research project. The first is the set of interfaces to the local cluster and to DAS peers in remote clusters. The second is the set of abstractions made available to applications programming teams to improve safety and affordability. However, another important point should also be understood about a DAS partition of a cluster. Any component within a DAS application is firewalled from the other partitions and from other applications within the DAS. That is, different applications and partitions have no direct means of communication, but must invoke a message service of the kernel. This additional firewalling of applications is also supported within the other partitions and is used to facilitate tractable, rigorous reasoning about the individual parts of a partition.

Distributed Information System

The DIS is responsible for managing shared and persistent information services and resources. Whenever information is shared by more than one application, access to the information is provided via a virtual interface set by requesting services from the DIS. For example, a DAS component could request a unit of shared information from the DIS by invoking a message service from the kernel. Also, some applications do not execute continuously and have requirements for persistent information. For example, a program that takes five minutes to complete may be scheduled to execute once every eight hours. At each execution, the program updates some information in the DIS that must persist between executions. In addition, the DIS manages shared and persistent information on behalf of the systems software. Examples include: performance and workload by cluster, LAN, WAN, and system; health and status of ..., etc. As with the other firewalled partitions, portions of multiple DISs may reside on the same cluster. Each DIS represented on the cluster is firewalled from the other DISs also on the cluster.

The class of MASC computing applications and systems addressed by MISSION will typically be long lived. Many type definitions that will be needed in the future cannot be known when the system is initially developed and deployed. Since non-stop operation requirements prohibit brin-

ging the system down to recompile existing code in the context of the new type definitions, an alternative is needed to upgrade the system. The approach under study is based upon controlled inheritance. A set of commands in the Distributed Command Interpreter is intended to allow the MICE to first extend/add the definitions and then create instances of the types. The reader should note that the problem of dynamic type extensibility is not limited to just the DIS.

Distributed Communications System

The DCS corresponds to the upper three layers and a portion of the fourth layer of the seven layer ISO model for Open Systems Interconnection [33]. (The lower layers are encapsulated as device drivers within the kernel.) The DCS is responsible for managing communications services and resources among clusters, LANs and WANs. Within a cluster, whenever an applications component or a systems software component needs to communicate with a peer at another cluster, the DCS is responsible for effecting this communication. A virtual interface set shared with its DCS peers at other clusters is used to resolve issues of routing, congestion control, relocation, and other services. Such resolution is transparent to the applications components or to any systems software components located outside the DCS partition.

Distributed Policy System

The DPS is responsible for the evolution and enforcement of policies regarding the sharable services and resources of the integrated systems software. The DPS contains a library of policies which are used in conjunction with the mechanisms of the kernel to manage such issues as: contention between local cluster priorities and universal system priorities, multiparameter scheduling, emergency load shedding, dynamic reconfiguration and others. An important premise is that support can be predictably and dependably provided for different policies needed by different applications if a known set of sufficient resources are available and if a known set of universal and local policies permit. This is somewhat similar to the approaches taken in Alpha and Spring.

Distributed Monitoring System

One of the most important and unique features of the MISSION "smokestack" model is the distributed monitoring system. This contains the objects responsible for monitoring the correct execution of the application objects. In fact, monitor objects are also introduced to monitor the correct execution of system level objects.

For any MASC component or any set of communicating MASC components in any of the other firewalled partitions, the engineers in the host environment are responsible for identifying those classes of faults that must be tolerated or that must invoke survival policies. Context sensitive assertion checks can then be generated to detect such faults at run time, and handlers can be prepared to respond to such detections. These assertions and handlers can then be combined into monitors. Together with policies in the DPS, they are responsible for system level fault tolerance and survivability.

When a work module (i.e. an application or system module) is installed in the DAS, or other appropriate partition, the corresponding monitors are installed in the DMS. The work module and associated monitors are scheduled to run concurrently on separate processors, although the work module is modified to write key information about state values and state changes to designated bulletin boards as it executes. The monitor is programmed to read this information for its assertion checks, and as long as no violations are detected, the work module is allowed to continue. However, if a violation is detected, the corresponding policy is consulted and the appropriate handler is invoked. If the fault is entirely local to a single work object, then the associated monitor may be able to insure proper tolerance by itself. However, faults that will cause temporal, spatial, or value errors in other objects or faults among cooperating objects are addressed by monitors that coordinate the activities of the monitors of the affected objects (i.e., monitors that monitor and coordinate other monitors).

Another primary function of the DMS is to be the "window" to the target environment for the MICE. Under normal operation, the DMS will monitor the health and status of the clusters, LANs and WANs and report this information (via the DCS) to the MICE. Other normal facilities

that it will control or monitor include: the introduction or removal of a new object to a cluster; the movement of an object from one cluster to another; linking, loading and starting new programs downloaded from the MICE; suspending or aborting threads of control; etc. Although other projects such as Mars have explored the use of monitors for fault detection at the cluster level, MISSION is somewhat unique in its use of monitors to detect faults and coordinate recovery among multiple applications spanning multiple clusters.

4 The Testbed

The MISSION testbed uses Sun workstations for the host and integration environments. A version of a Verdix Ada (1983) compiler that supports post- partitioning and distribution of code has been used to generate code for the target environment. Although other processor types have been successfully used in this target environment (e.g., object-oriented processors by Ericsson), the major clusters consist of multiprocessing clusters of Motorola 68030s.

5 Conclusion

Distributed, non-stop MASC systems are some of the largest and most complex computer systems to have been tackled to date. As described in section 2, they require many independent technologies, developed separately for smaller systems, to be brought together and integrated into a single unified whole. This integration, and the definition of the environment to support it, presents a major technological challenge. This paper has outlined some of the major issues which arise in the construction and maintenance of this category of embedded system, and has provided an outline of the MISSION approach to achieving this goal. This strategy has two principal components: the definition of a generic architecture for the target systems software, and the design of a supporting infrastructure and processes.

A key component of the proposed infrastructure is the monitoring, control and integration environment (MICE) which bridges the gap between the traditional host and target environment used today for embedded systems. The MICE serves as the location at which new software components and (sub)applications from the various contractors can be tested, assembled and eventually downloaded to become part of the executing MASC embedded system. To perform this function the MICE employs a set of precise semantic models which describe the current structure, functionality and behavior of the executing system. Such semantic modeling pervades all three environments, over the full life-of the system, and forms the cornerstone of the MISSION software process used to develop and sustain distributed, nonstop, MASC systems. Each process is domain-specific and leverages the object paradigms for modeling all aspects of the systems across the life cycle.

The paper also outlined the nature of the generic architecture for the multi- processor clusters, interconnected by LANs and WANs, which make up the distributed target environment. This architecture is based on the principal of segregating functionally cohesive components into separate, firewalled, partitions which can only interact indirectly via the special MASC kernel. Preliminary prototypes of these subsystems have demonstrated the feasibility of the architecture and the overall approach, but further work is needed to elaborate upon the detailed make up of the separate subsystems, and to evaluate the concepts in a pilot project.

Acknowledgments

The MISSION research has been partially supported by NASA. The authors wish to thank: the NASA sponsors and monitors; our fellow researchers from the faculty, staff and students at the University; our fellow researchers from industry; our support staff in RICIS; and the volunteers on our Industrial Advisory Committee.

References

[1] AIA/SEI (Aerospace Industries Association/Software Engineering Institute), *Workshop on Research Advances Required for Real Time Software Systems in the 1990s*, Software Engineering Institute, 1991.

[2] Allen, R., D. Garlan, "Formalizing Architectural Connection", *Proceedings of the 16th In-*

ternational Conference on Software Engineering, Sorrento, Italy, 1994.

[3] "American National Standard Information Resource Dictionary System", American National Standards Institute, Group X3H4, New York, 1985.

[4] Arlat, J., K. Kanoun, J. Laprie, "Dependability Modeling and Evaluation of Software Fault Tolerance: Recovery Blocks, N Version Programming, N Self Checking Programming", *First Year Report on Predictably Dependable Computing Systems*, Volume 3 of 3, Esprit Project 3092, 1990.

[5] Atkinson, C., T. Moreton, A. Natali, *Ada for Distributed Systems*, Ada Companion Series, Cambridge University Press, 1988.

[6] Burns, A. and C. McKay, "A Portable Common Execution Environment for Ada", *Ada: The Design Choice - Proceedings of the Ada- Europe International Conference*, Madrid, 1989, Cambridge University Press, 1989.

[7] Charette, R., *Software Engineering Risk Analysis and Management*, McGraw Hill, 1989.

[8] Deswarte, Y., J. Fabre, J. Laprie, D. Powell, "A Saturation Network to Tolerate Faults and Intrusion", *Proceedings of the 5th Symposium on Reliability in Distributed Software and Database Systems*, IEEE Computer Systems Press, 1986.

[9] DOD (Department of Defense, United States of America), "Trusted Computer System Evaluation Criteria", *DOD 5200.28-STD*, 1985.

[10] Embley, D., B. Kurtz, S. Woodfield, *Object-Oriented Systems Analysis: A Model-Driven Approach*, Yourdon Press, 1992.

[11] ESPRIT (European Strategic Program for Research and Development in Information Technology), *First Year Report on Predictably Dependable Computing Systems*, Volumes 1, 2, 3, ESPRIT, 1990.

[12] Ezhilchelvan, P. and S. Shrivastava, "Characterization of Faults in Systems", *Proceedings of the 5th Symposium on Reliability in Distributed Software and Database Systems*, IEEE, 1986.

[13] Ezhilchelvan, P. and S. Shrivastava, "A Distributed Systems Architecture Supporting High Availability and Reliability", *Proceedings of the 2nd International Working Conference on Dependable Computing For Critical Applications*, IEEE, February 1991.

[14] GAO (Government Accounting Office), "Space Station: NASA's Software Development Approach Increases Safety and Cost Risks – Report to the Chairman", Committee on Science, Space and Technology, House of Representatives, GAO, 1992.

[15] Jensen, E., Chapter 8, *Distributed Systems: Architecture and Implementation*, (B. Lampson, M. Paul and H. Siegert, editors), Springer-Verlag, 1981.

[16] Knight, J. and J. Urquhart, "On the Implementation and Use of Ada on Fault-Tolerant Distributed Systems", *IEEE Transactions on Software Engineering*, Vol SE-13, No. 5, May 1987.

[17] Kopetz, H., A. Damm, C. Koza, M. Mulazzani, W. Schwabi, C. Senft, R. Zainlinger, "Distributed Fault-Tolerant Real-Time Systems: The MARS Approach," IEEE Micro, February 1989.

[18] LeBlanc, R. and A. Robbins, "Event Driven Monitoring of Distributed Programs", *Proceedings the 5th International Conference on Distributed Computing Systems*, IEEE 1985.

[19] Leveson, N., "Building Safe Software", *Proceedings of COMPASS*. 1986, IEEE, 1986.

[20] Locke, D., "Best Effort Decision Making for Realtime Scheduling", *CMU-CS-86-134*, Carnegie Mellon University, 1986.

[21] Long, J., W. Fuchs, J. Abraham, "Implementing Forward Recovery Using Checkpointing in Distributed Systems", *Proceedings of the 2nd International Working Conference on Dependable Computing For Critical Applications*, IEEE, February 1991.

[22] McKay, C. W. and C. Atkinson, *Volumes I, II and III of the MISSION Concept Document*, RICIS Report, University of Houston-Clear Lake, 1992.

[23] McKay, C., D. Auty, K. Rogers, "A Study of System Interface Sets (SIS) For the Host, Target, and Integration Environments of the Space Station Program (SSP)", SERC(UHCL) Report SE. 10, NCC9-16, 1987.

[24] Moss, J., *Nested Transactions: An Approach to Reliable Distributed Computing*, MIT/LCS/TR 260, Massachusetts Institute of Technology, April 1981.

[25] Musa, J., A. Iannino, K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw Hill, 1987.

[26] Northcutt, J., *Mechanisms for Reliable Distributed Real- Time Operating Systems: The Alpha Kernel*, Academic Press, Boston, 1987.

[27] Parker, D., G. Popek, A. Rudison, A. Stoughton, B. Walker, E. Walton, J. Chow, D. Edwards, S. Kiser, C. Kline, "Detection of Mutual Inconsistency in Distributed Systems", *IEEE Transactions on Software Engineering* Vol. SE-9, No. 3, IEEE, May 1983.

[28] Pyle, I., *Developing Safety Systems: A Guide Using Ada*, Prentice Hall, 1991.

[29] Ramamritham, K., J. Stankovic, *Overview of the Spring Project*, University of Massachusetts Amherst, COINS Technical Report 89-03, January 1989.

[30] Randall, C., P. Rogers, C. McKay, "Distributed Ada: Extending the Runtime Environment for the Space Station Program", *Sixth National Conference on Ada Technology*, March, 1988.

[31] Randell, B. and J. Dobson, "Reliability and Security Issues in Distributed Computing Systems", *Proceedings of the 5th Symposium on Reliability in Distribute Software and Database Systems*, IEEE, 1986.

[32] Redmill, E. (Editor), *Dependability of Critical Computer Systems 2*, Elsevier Applied Science, 1989.

[33] "Reference Model of Open Systems Interconnection". *ISO/TC97/SC16/N227*, International Standards Organization, 1979.

[34] Rogers, K., M. Bishop, C. McKay, "An Overview of the Clear Lake Life Cycle Model (CLLCM)", *Proceedings of the 9th Annual Conference on Ada Technology*, ACM, March, 1991.

[35] Sankar, S., "Automatic Runtime Consistency Checking and Debugging of Formally Specified Programs", *STAN-CS-89-1282*, Stanford University, 1989.

[36] SATWG (Space Avionics Technology Working Group), Space Avionics Requirements Study, (Integrated by General Dynamics), 1990.

[37] Sha, L. and J. Goodenough, "Realtime Scheduling Theory and Ada", *IEEE Computer*, April 1990.

[38] Shankar, K., C. McKay, "Why NASA, Code R, Should Sponsor Advanced Research in Software Engineering: A White Paper", *Proceedings of the Computing in Aerospace 9 Conference, American Institute of Aeronautics and Astronautics*, San Diego, October 1992.

[39] Stankovic, J. and K. Ramamritham, "The Design of the Spring Kernel", *Proceedings of the Real Time Systems Symposium*, IEEE, December 1987.

[40] Stankovic, J., K. Ramamritham (editors), Tutorial: Hard Real Time Systems, *IEEE Computer Society Press*, 1988.

[41] Strigini, L., "Software Fault Tolerance", *First Year Report on Predictably Dependable Computing Systems*, Volume 2 of 3, Esprit Project 3092, 1990.

[42] Tindell, K., "Dynamic Code Replacement and Ada", *Ada Letters*, Vol. X, No. 7, 1990.

[43] Tokuda, H., T. Nakajima, P. Rao, "Real-Time Mach: Towards a Predictable Real-Time System", *Proceedings of the Usenix Machine Workshop*, October 1990.

[44] Vincente, B., A. Alonso, J. Amador, "Dynamic Software Replacement Model and It's Ada Implementation", *Proceedings of TriAda'91*, ACM, 1991.

[45] Zicari, R., "Operating System Support For Software Migration in a Distributed System", *Proceedings of the 5th Symposium on Reliability in Distributed Software and Database Systems*, IEEE, 1986.

Loose Specification of Real Time Systems

Jan van Katwijk and Hans Toetenel
Delft University of Technology
Faculty of Technical Mathematics and Informatics
P.O. Box 356, 2600 AJ Delft The Netherlands

Keywords: MOSCA language, multiple threads, semantic basis

Edited by: Marcin Paprzycki and Janusz Zalewski
Received: February 15, 1994 **Revised:** October 14, 1994 **Accepted:** January 31, 1995

MOSCA is an experimental language that equips the Vienna Development Method specification language VDM-SL to be applicable in the area of developing distributed, parallel and real-time systems. As is generally known, plain VDM is not adequate for these application areas since it lacks facilities to specify multiple threads of control and does not allow the use of time within specifications. MOSCA is designed to overcome these restrictions. This paper presents an overview of some of the process specification capabilities of the notation. It highlights the semantic basis and treats in particular the interpretation of loosevaluespecification and looseprocessspecification.

1 Introduction and overview

In the last two decades many techniques and methodologies were developed with as common aim the systematic and orderly development of computer software. With the emerging techniques came the recognition that the development of software was not a simple huge homogeneous activity, but should be split up in different phases. Analysis results in a description of a solution (a model) which after thorough design leads to the desired software product.

Research over the last 20 years has developed formal techniques for the modeling and design of sequential systems. These techniques are accepted within the research community and are adopted in many industrial applications. Sequential systems can be characterized as computation oriented systems having strictly limited and well-guarded interaction with their environment, as opposed to computer systems with an ongoing interaction with their environment.

Unfortunately systems of the latter class cannot be sensibly viewed within the sequential framework. Most of these systems are highly concurrent, distributed and often have real-time properties. The development of these systems is far more complex than the development of data driven systems. Many authors like e.g. BROOKS [6], PARNAS [18], and LEVESON [13] have stressed that standard techniques currently used in the development of data driven systems do not match the demands put on development techniques for real-time and distributed systems. Even worse, attempts to build such systems using the same approaches developed and used for data processing and information processing systems must lead to grave failures.

LEVESON argues in [13] that modeling and analysis form the main challenges in building real-time (control) systems. STANKOVIC [22] states that the main challenge in the development of advanced computing systems lies in modelling and verification of timing constraints. Inclusion of a time metric increases the semantic power of concurrency models to a great extend, and thus complicate the verification. One of the more difficult problems is verification of these systems, which requires the satisfaction of timing constraints.

Figure 1 schematically depicts the relationships between notations and tools that partake in the notational framework for system modelling and analysis. The primary part is the *system specification language* which enables the creation of a *model* of the system, based on the overall requirements. Often a simple *conceptual model* is used to

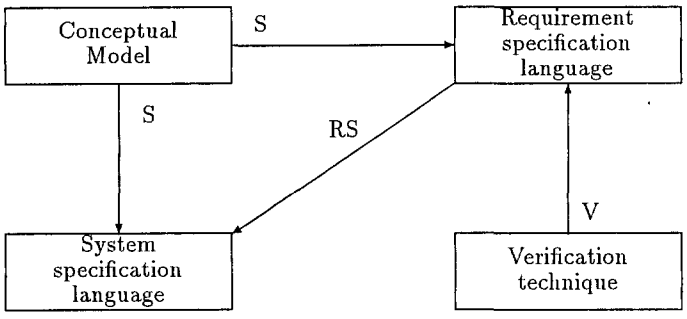


Figure 1: The notational framework

give a semantics (**S**) to both the system specification language and the *requirement specification language* (RSL). A RSL offers means to specify requirements or properties (**RS**) of the system model. To decide whether the system model models the requirements and has the desired properties a *verification technique* must be applied (**V**), such as model-checking or axiomatic reasoning.

In this paper the system specification language MOSCA is introduced as a vehicle for modelling reactive and real-time systems. One of its distinguished features is the possibility for *loose specification*. Loose specification is the possibility of writing constructs that may have several different meanings. The specification of a system is necessarily loose specified if the final system may wait until execution time to determine the outcome of the constructs. Loose specified constructs may also arise in specifications even when the system finally developed is deterministic. The phenomenon of looseness arises in this context because a specification (of a model) is often stated at a greater level of abstraction than that of the final source code of the deterministic system.

The article is further organized as follows. In section 2 an overview is presented of the syntax of the specification language MOSCA. Section 3 presents the conceptual model on which the semantics of MOSCA is defined. It highlights in particular the semantics of time dependent constructions and the incorporation of looseness into the semantics. Section 4 presents a larger example of the use of MOSCA. Finally section 5 summarizes the results so far and compares the MOSCA notation to related work.

2 The MOSCA notation

MOSCA[1] [23], [24], [25] is an experimental notation based on the Vienna Development Method specification language VDM-SL [2] [7], [11]. The aim of its development was to increase the applicability of VDM in the area of distributed, parallel and real-time systems. MOSCA incorporates notational aspects of CCS, the Calculus of Communicating Systems [15]. The model-oriented specification language of VDM acts as the process algebra's value manipulation language. The combination is further extended with capabilities to describe *time dependent behaviour*. MOSCA offers a timing facility based on the *Timed* CCS notation of WANG [27].

MOSCA is an experimental specification language. Although both VDM and CCS have a very firm position as accepted notations, a combination of the two is rather experimental. Recently some results are established by HENNESSY [10] on combining a process algebra like CCS with a value passing mechanism capable of manipulating natural numbers. Semantically the problems of combining VDM-SL with TCCS are substantial. Without claiming to give a solution to all problems a fairly simple semantic model for the combination was constructed. [3]

A MOSCA specification describes four aspects of complete systems of communicating processes: their data-containment, their functional

[1] MOSCA is an acronym for *M*odel-*O*riented *S*pecification of *C*ommunicating *A*gents.

[2] The specification language for VDM for which an ISO standard is currently being developed (ISO SC22/WG19/N-20).

[3] For a full rationale on the design of the MOSCA language the reader is referred to [23].

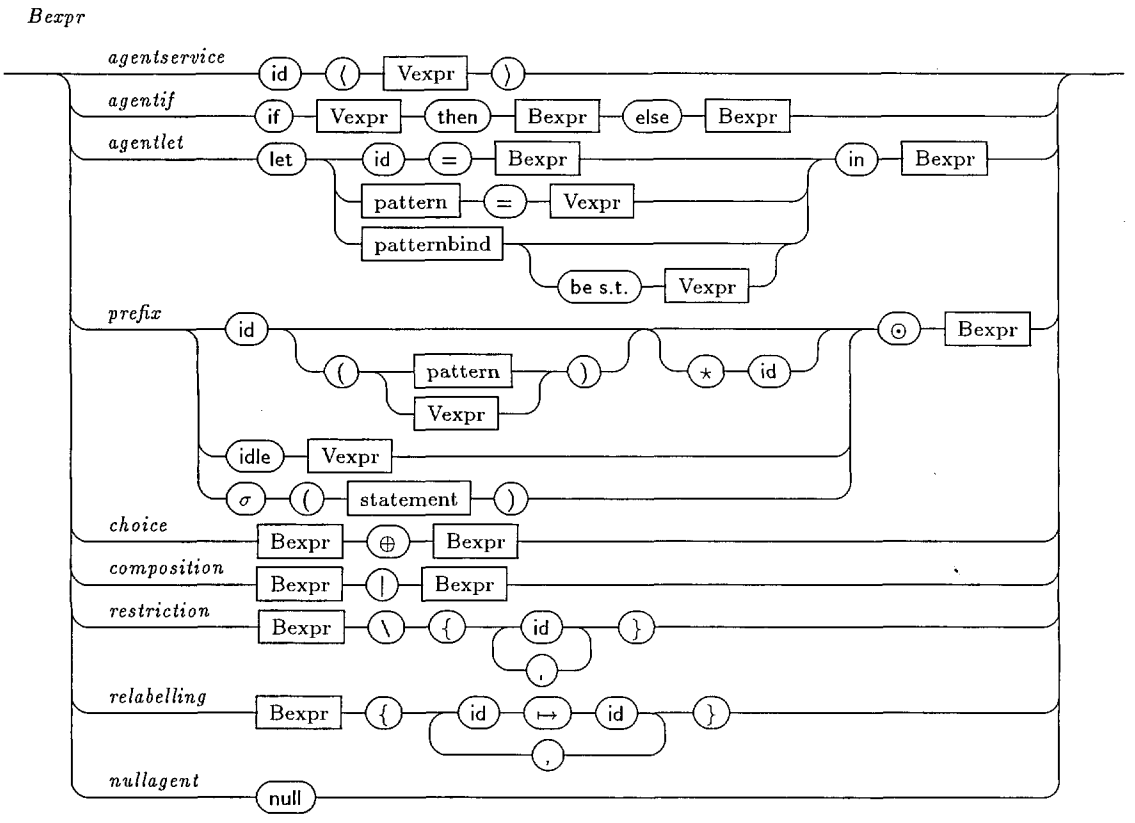


Figure 2: Syntax of behaviour expressions

behaviour, their process-structure and their time characteristics. Associated with these aspects are different constructions : data type definitions, functions and operations on the data types, agent definitions, and timed actions. The basic element in the MOSCA model of a system is a *process*, called *agent*. The constructions to specify the behaviour of agents are shown in Figure 2. In the following sections these constructions are introduced by example to describe the various aspects of agent behaviour.

The first option in the syntax diagram is dedicated to agent invocation. The syntax class *Vexpr* denotes all value expressions as defined by the VDM-SL part of MOSCA. It offers basic types like natural numbers ($\mathbb{N}$), integers ($\mathbb{Z}$), reals ($\mathbb{R}$), booleans ($\mathbb{B}$), product and union types, record types optional types, function and operation types. Further it offers complex data structuring facilities based on set types, sequence types and map types. It features subtyping through type invariants.

The *agentif* expression enables conditional behaviour specification. The *agentlet* enables local agent definitions and local value bindings, either fixed or loose. E.g. in

$$\mathsf{let}\ x = 3\ \mathsf{in}\ x$$

the value 3 is bound to the identifier x in a deterministic way. In

$$\mathsf{let}\ x \in \{1,2,3\}\ \mathsf{in}\ x$$

it is not deterministically decided what the value of the expression will be. It may be either 1, or 2 or 3, depending on the context of the expression whether the choice is made in specification time or in execution time. The last form of the *agentlet* offers a means to constrain the set of bindings. The expression

$$\mathsf{let}\ x{:}\,\mathbb{N}\ \mathsf{be\ s.t.}\ x < 10\ \mathsf{in}\ x$$

defines a value expression for which the value ranges from 0 to 9.

The next option is dedicated to the various forms of *prefix* expressions. The first form is regular CCS extended with an explicit synchronization prefix. The second form describes the idle

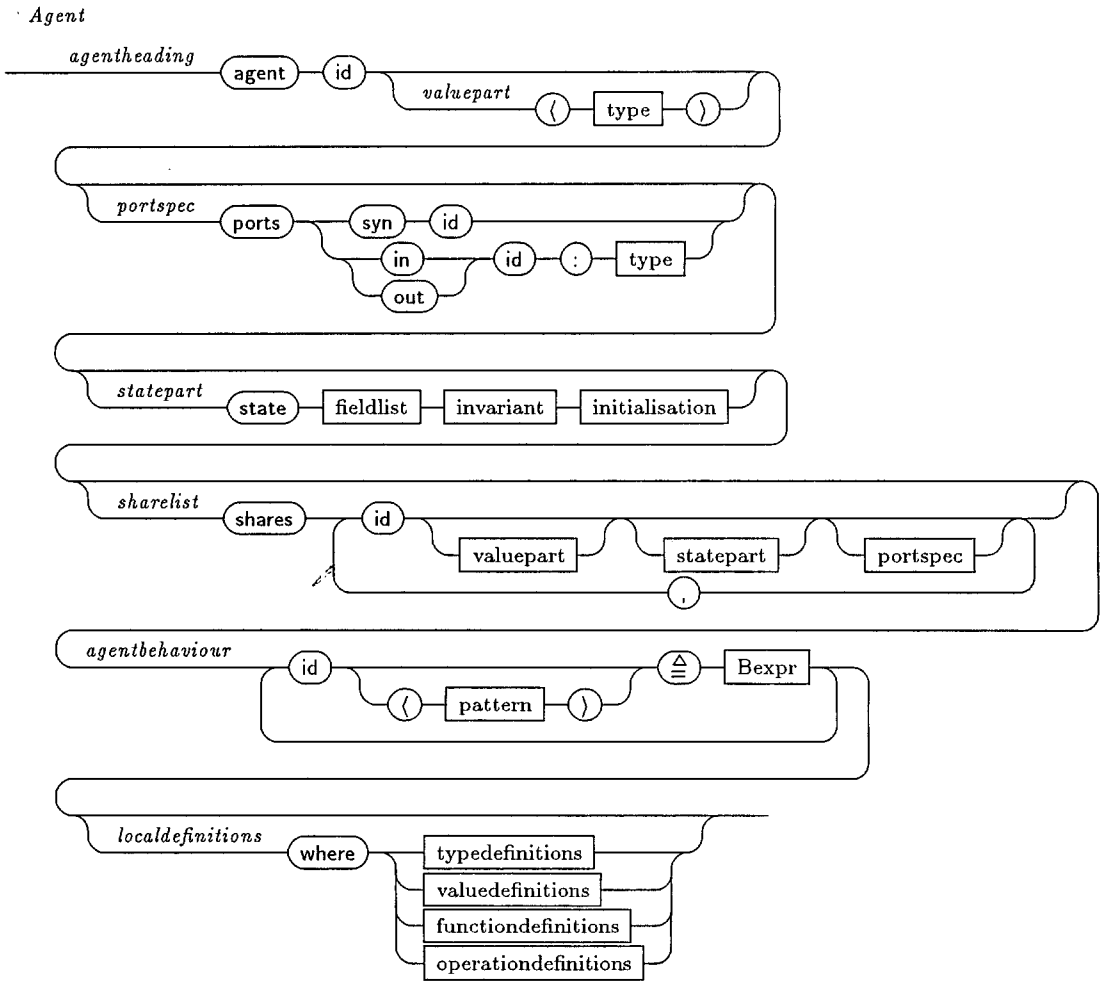


Figure 3: Syntax of agent definitions

prefix, which involves time manipulation. The third form handles the infusion of value state manipulation. It specifies a VDM-SL statement that is allowed to manipulate the state of the agent in which the construction appears.

The next four options describe standard CCS operators. The *choice* construction enables nondeterministic selection of specific behaviour. *composition*, and *restriction* involve agent communication. Through *relabelling* port labels can be renamed. They originate in CCS and have a meaning equivalent to their CCS counterparts.

The syntax of agent definitions is defined by the *Agent* syntax rule, presented in Figure 3. Agent definitions can be given in various styles, ranging from simple agents without any means to handle values — through agents with value parts — to agents with *local* state and associated operations that act on the state of the agent.

Specification 2.1

agent $Clock$
ports in $reset$: $\mathbb{T}$
out $\overline{time}$: $\mathbb{T}$

shares $RClock\ \langle \mathbb{T} \rangle$

$Clock \triangleq$
$\quad reset(inittime) \odot RClock\ \langle inittime \rangle$

$RClock\ \langle t \rangle \triangleq$
$\quad \mathsf{idle}(tick) \odot RClock\ \langle tick + t \rangle$
$\quad \oplus\ reset(inittime) \odot RClock\ \langle inittime \rangle$
$\quad \oplus\ \overline{time}(t), \star d \odot$
$\quad \mathsf{idle}(tick - d) \odot RClock\ \langle tick + t \rangle$

where

values
$tick : \mathbb{T} = 0.001$
end

In specification 2.1 the first line states the *agentheading*. The value part is absent. The next two lines define two communication ports, the input port *reset* and the output port *time* (an output portname is marked by an over bar, e.g. like $\overline{time}$. Both ports handle data elements from the time domain $\mathbb{T}$. The *Clock* has two different behaviour definitions, *Clock* and *RClock*. The second behaviour definition uses a value part to hold the current reading of the clock. The shares clause defines additional behaviour definitions that *share* the interface of the surrounding agent definition in the following sense: additional ports and state extend the interface of the surrounding agent, an additional value part replaces the value part of the surrounding agent definition. The *Clock* behaviour definition

$$Clock \triangleq reset(inittime) \odot RClock\ \langle inittime \rangle$$

is defined through the *agentbehaviour* syntax construction. The behaviour expression

$$reset(inittime) \odot RClock\ \langle inittime \rangle$$

is a prefix construction. The left operand is an input action where *reset* is the name of the port through which the input action will operate and *inittime* is the pattern on which the input value will be matched to form a new pattern bind. The right operand is an example of an *agentservice* construction. It resembles the function call of the VDM-SL part of MOSCA. An agent service construction consists of the agent name it invokes and optionally an actual value to match the valuepart. The behaviour expression

$$Rclock\ \langle inittime \rangle$$

is an agentservice expression that sets the value part of the *RClock* behaviour definition to the value bound to *inittime* and behaves just like the behaviour expression associated with the behaviour definition *RClock*.

An slightly different specification of the *Clock* agent is presented in specification 2.2. It illustrates the state facilities of MOSCA.

Specification 2.2

agent *SClock*
ports in $reset : \mathbb{T}$
out $\overline{time} : \mathbb{T}$

state $reading : \mathbb{T}$

init $reading \triangleq reading = 0$

$$\begin{aligned} SClock \triangleq\ & \mathsf{idle}(tick) \odot \\ & \sigma(reading := \overleftarrow{reading} + tick) \\ & \odot SClock \\ \oplus\ & reset(inittime) \odot \\ & \sigma(reading := \overleftarrow{reading} + tick) \\ & \odot SClock \\ \oplus\ & \overline{time}(t), \star d \odot \mathsf{idle}(tick - d) \\ & \odot \sigma(reading := \overleftarrow{reading} + tick) \\ & \odot SClock \end{aligned}$$

where

values
$tick : \mathbb{T} = 0.001$
end

The reading of the clock is now stored in a state component. State components act as global variables. In contrast with *Clock* will *SClock* always start with a zero reading. *SClock* can also be initialized with a start reading. The form

$$\sigma(reading := \overleftarrow{reading} + tick) \odot \ldots$$

is a special form of an prefix construction. It is not associated with an external action, but with a completely internal action, i.e. a manipulation of one or more state variables. In state manipulations old values are 'hooked', i.e. they appear with an hooked overbar, like $\overleftarrow{reading}$.

The model of time in MOSCA is centered around the following elements.

- There is neither a central clock, nor any other ticking device that registers the current time.
- Passing of time is measured related to actions: from the moment an action becomes enabled, i.e. offered to the environment, to the moment the action is actually taken. The

passed time is recorded in a variable associated with the involved action. E.g. in

$$\overline{time}(t), \star d \odot \mathsf{idle}\ (tick - d) \odot \ldots$$

the passing of time between the moment the $\overline{time}$ action becomes enabled and the actual taking of the action is bound to the variable d.

- Time progression is modelled by taking idle actions. Time flows continuously. There are no time stops.

Time identifiers of timed prefix constructions may occur as free variable within the behaviour expression following the $\odot$ operator. After taking the action of the prefix construction the actual value of the variable t is substituted in each free occurrence of the identifier within the behaviour expression following the $\odot$ operator. Thus in

$$act, \star t \odot P$$

each free occurrence of t in P is bound by $\star t$. Upon taking the action *act* the actual value of the time variable t is substituted in all free occurrences of t in *Bexpr*.

Time progression is due to the taking of *idle* actions. E.g.

$$\mathsf{idle}\ tick \odot RClock\ \langle inittime \rangle$$

specifies a prefix expression, in which taking the action causes the time to progress with the value *tick*. Time progression is strongly connected to the semantics of the prefix construction, choice and composition operator.

The domain $\mathbb{T}$ for the values of the idle actions is chosen to be $\mathbb{R}_0^+$, the set of real values greater or equal to zero. The set of reals creates a model that fits more closely to reality than e.g. $\mathbb{N}$, the natural numbers. For a fully synchronous system one may take a discrete time domain (like with SCCS in [14]), since all subagents refer to the same global intuition of time (e.g. a global clock), and events happen at certain moments in time. In the asynchronous case, any two agents may perform actions at times that are not equal, but arbitrary close to each other. Hence a dense time domain is preferred.

The parallel composition is synchronous with respect to time actions. E.g. in

$$\mathsf{idle}\ 6 \odot P \mid \mathsf{idle}\ 8 \odot Q$$

after 2 time units have passed a situation described by

$$\mathsf{idle}\ 4 \odot P \mid \mathsf{idle}\ 6 \odot Q$$

is reached. However, time progression is asynchronous for the normal actions (except for communications between its different components). In

$$P \mid Q$$

P and Q can perform actions asynchronously until synchronization between two subcomponents is due.

In the example above the reading of the clock, modelled by the action $\overline{time}$ may take some time, due to idling elsewhere. Thus in

$$\overline{time}(t), \star d \odot \mathsf{idle}(tick - d) \odot RClock\ \langle tick + t \rangle$$

the passing of time between the enabling of the $\overline{time}$ action and the taking of the action is bound to d. The structure of the specification of *RClock* ensures that d will never be greater than *tick* so after idling $tick - d$ time units exactly one whole tick is passed and the reading of the clock is updated accordingly.

3 The semantics of MOSCA

MOSCA's structured operational semantics is based on the common notion of labeled transition systems ([19]) and exhibits three particular properties, i.e.

- the state domain of the labeled transition system holds a notion called environment, capturing bindings from identifiers to both semantic value denotations and syntactic agent definitions;
- the label domain holds values from (i) the set of visual ports (as usual), (ii) values of the semantic value domain of VDM-SL and (iii) values of the semantic domain *TIME*, corresponding with the syntactic domain $\mathbb{T}$.
- the state transition relation reflects looseness. To this end the transition rules are defined over states constructed of *sets* of items. Each item corresponds to a possible different transition.

The values in the semantic value domain of VDM-SL are defined through a denotational semantic approach. E.g. the syntactic domain of natural numbers, $\mathbb{N}$ is represented by a semantic domain of natural number denotations, *NATURALS*, which is constructed as a chain complete partial ordering. The semantics of VDM-SL are defined through recursive functions over the semantic domains. Currently the semantics is as part of the draft VDM-SL ISO standard, under review. It is rather elaborate. It contains over 200 semantic functions together taking 400 pages of formatted output [11].

The state space is infinite. Although the infinity aspect has difficult theoretical properties the labeled transition system offers a base for the operational description of the behaviour of agents.

3.1 The state space

The labeled transition system for MOSCA is a triple

$$\langle (Bexpr \times \rho)\text{-set}, Act, \longrightarrow \rangle.$$

The first part defines the state space, the second part the label space and the third part the transition relation. *Bexpr* ranges over behaviour expressions, ρ ranges over environments, *Act* ranges over the set of defined port labels and $\longrightarrow$ is a relation

$$(Bexpr \times \rho)\text{-set} \times Act \times (Bexpr \times \rho)\text{-set}.$$

The state space is built from states, consisting of *sets* of tuples (be, env), where $be \in Bexpr$ and *env* is an environment. A single element (P, ρ) of a state is further referenced by the notion *state item*.

The environment captures the set of definitions in which behaviour expressions operate. These definitions include all data definitions like types, values and functions, agent definitions, the set of local bindings resulting from value let constructions, pattern matching etc. The environment records

- all possible value bindings, i.e. bindings between identifiers and *semantic* denotations of value constructions such as types, functions, values, operations and states, and
- all possible agent bindings, i.e. bindings between identifiers and *syntactic* representations of agent definitions.

3.2 The label set *Act*

The label set *Act* for MOSCA is defined by the union of the external action names, the timed action labels and the internal action. The external action names correspond to the port identifiers. The timed action labels τ are the semantic counterparts of the idle actions. Each idle action specifies a delay d that is attached to the timed action label τ like $\tau(d)$. The class of time actions *TimeActs* contains the idle action specifiers $\tau(d)$. Each value of d specifies a denotation of a time value. E.g. the transition

$$\{(P, \rho)\} \xrightarrow{\tau(d)} \{(Q, \rho')\}$$

denotes that agent P will become Q after $d \in TIME$ units of time. The behaviour expression idle $0.5 \odot P$, operating in an environment ρ could make the transition

$$\{(\text{idle } 0.5 \odot P, \rho)\} \xrightarrow{\tau(0.5)} \{(P, \rho')\}.$$

The last action label is ι, labeling an internal action. Notice the difference with CCS where there is an action τ that models an internal action on the syntactic level. In MOSCA an internal action with very similar semantics can be constructed by taking e.g. a state manipulation with a dummy statement, or an idle action with delay 0.

3.3 The transition relation

The $\longrightarrow$ relation is defined implicitly as the least relation defined by a set of inference rules. The general form of the inference rules are more complex than in the structural operational semantics of CCS. This is caused by the state extension with environments and the interplay of the loose value semantics with the agent behaviour expressions.

A transition step is denoted with the notation $S \xrightarrow{a} S'$ with $a \neq \iota$ or $S \Longrightarrow S'$ where $\Longrightarrow$ is an abbreviation for $\xrightarrow{\iota}$. S is called the *basis* of the transition step, a the action label and S' the *result* of the transition. When S is a singleton set the notation $\{s\}$ is used.

The semantic rules fall into different classes: the internal action rules, external action rules and timed action rules. In general these rules appear as

$$\mathbf{L}\ \frac{S \xrightarrow{a} S'}{T \xrightarrow{b} T'}$$

where **L** is an identifying tag. The intuition undelying the inference rule is something like " if S can proceed to S' by taking an a action, T can proceed to T' by taking a b action ". The part above the line is the hypothesis for the inference, the part under the line is the conclusion. The hypothesis may be absent, in which case the action step in the conclusion can be taken always.

The internal action rules cover all cases where evaluation or manipulation of values is involved. Examples are the rules for *agentservice*, *agentlet*, *agentif* and *prefix* expressions. Their basic form is like

$$\mathbf{SE}\ \frac{}{\{s\} \Longrightarrow S}.$$

These rules describe the effect of taking an internal action in the context of singleton states and result in general in state expansion (**SE**). The effect of taking internal actions in the context of states built from multiple state items is described by the next rule.

$$\mathbf{EP}\ \frac{\{s\} \Longrightarrow S}{S' \cup \{s\} \Longrightarrow S' \cup S}.$$

It signifies state expansion propagation (**EP**). Next there are rules that describe the effect of taking internal actions in the context of the standard CCS operations. E.g. the rule for state expansion in the context of a choice expression is as follows.

$$\boxed{\mathbf{CE}\ \frac{\{(A_i, \rho)\} \Longrightarrow S}{\{(A_1 \oplus A_2, \rho)\} \Longrightarrow S}\ i = 1,2}$$

It states that whenever an operand of a choice expression leads through state expansion to some expanded state S, the whole choice expression leads to this state S. That is, taking an internal action will force the choice to be evaluated. The choice is thus not only directed by external action steps but is also directed by internal actions.

Opposed to state expansion there are state reduction rules. There are two different causes for state reduction: (i) external actions and (ii) idling actions. Action state reduction is defined as

$$\mathbf{ASR}\ \frac{\{s\} \xrightarrow{a} \{s'\}}{S \cup \{s\} \xrightarrow{a} \{s'\}}$$

The rule states that whenever a single state item can take an external action, the whole state is reduced to the result of the action step. Idling state reduction is defined as

$$\mathbf{ISR}\ \frac{\{s\} \xrightarrow{\tau(d)} \{s'\}}{S \cup \{s\} \xrightarrow{\tau(d)} \{s'\}}$$

which states that whenever a single state item takes an idling action, the whole state is reduced to the effect of the idling step. The following examples may elucidate the effect of state expansion and reduction in different contexts.

Example 3.1 The evaluation of value constructions is captured in an internal action step. E.g. the rule for the agent value let expression is

$$\boxed{\mathbf{AVL}\ \frac{}{\{(avl, \rho)\} \Longrightarrow EvalAVL(avl, \rho)}}$$

where *avl* represents an agent value let expression and *EvalAVL* is a semantic evaluation function that results in a set of state items that share the body of the let expression as continuation behaviour but hold different values in the environment part that resulted from loose value evaluations. E.g. suppose the expression

$$\mathsf{let}\ x \in \{1,2,3\}\ \mathsf{in}\ P\ \langle x \rangle$$

operates in an environment ρ in which P is defined. The rule **AVL** states that the state

$$ll = \{(\mathsf{let}\ x \in \{1,2,3\}\ \mathsf{in}\ P\ \langle x \rangle\ , \rho)\}$$

is transformed through an internal action ι into the result of $EvalAVL(ll)$, which in this particular case would be the following state

$$\{(P\ \langle x \rangle\ , \rho \cup \{x \mapsto 1\}),\\ (P\ \langle x \rangle\ , \rho \cup \{x \mapsto 2\}),\\ (P\ \langle x \rangle\ , \rho \cup \{x \mapsto 3\})\}.$$

The cardinality of this state is equal to the cardinality of the set in the patternbind of the let construction. Each state item is the starting point of the same behaviour expression operating in a different environment. □

Example 3.2 demonstrates the effect of the action state reduction rule **ASR**.

Example 3.2 The evaluation of agent service construction is also captured in an internal action

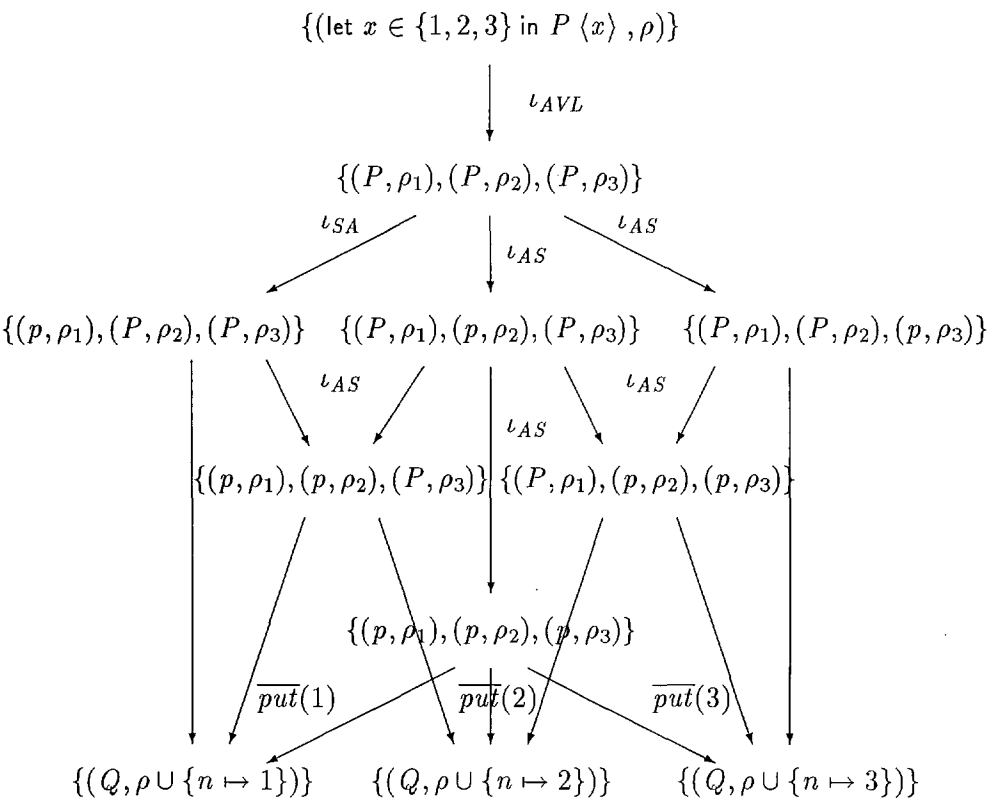


Figure 4: Transition tree for loose let construction

step. The rule, completely analogue to the **AVL** rule, is

$$\textbf{AS}\ \frac{}{\{(as,\rho)\} \Longrightarrow EvalAS(as,\rho)}$$

where *as* represents an agent service expression and *EvalAS* a semantic function, that takes the syntactical form of a specific agent service and an environment as arguments and computes the set of state items resulting from the agent service application.

Suppose the agent P in example 3.1 is defined through the single agent behaviour definition

$$P\ \langle n\rangle \ \triangleq\ \overline{put}(n) \odot Q.$$

The rule for agent service can now be applied on each of the state items of state *ll*. The effect of the rule applied on state item $\{(P\ \langle x\rangle\ , \rho \cup \{x \mapsto 1\})\}$ is as follows.

$$\{(P\ \langle x\rangle\ , \rho \cup \{x \mapsto 1\})\} \Longrightarrow$$
$$\{(\overline{put}(n) \odot Q, \rho \cup \{n \mapsto 1\})\}$$

According to rule **EP** the state *ll* now will have a transition resulting in

$$\{(\overline{put}(n) \odot Q, \rho \cup \{n \mapsto 1\}),$$
$$(P\ \langle x\rangle\ , \rho \cup \{x \mapsto 2\}),$$
$$(P\ \langle x\rangle\ , \rho \cup \{x \mapsto 3\})\},$$

but also an internal transition leading to

$$\{(P\ \langle x\rangle\ , \rho \cup \{x \mapsto 1\}),$$
$$(\overline{put}(n) \odot Q, \rho \cup \{n \mapsto 2\}),$$
$$(P\ \langle x\rangle\ , \rho \cup \{x \mapsto 3\})\},$$

and equally an internal transition leading to

$$\{(P\ \langle x\rangle\ , \rho \cup \{x \mapsto 1\}),$$
$$(P\ \langle x\rangle\ , \rho \cup \{x \mapsto 2\}),$$
$$(\overline{put}(n) \odot Q, \rho \cup \{n \mapsto 3\})\}.$$

In Figure 4 these state transitions are depicted as a derivation tree. The internal actions ι are indexed with the action rule label. The state item (P, ρ_i) is a shorthand for $(P\ \langle x\rangle\ , \rho \cup \{x \mapsto i\})$. State item (p, ρ_1) symbolizes the state item $(\overline{put}(n) \odot Q, \rho \cup \{n \mapsto i\})$.
A rule for external output actions is

$$\textbf{EOA}\ \frac{}{\{(a(expr) \odot X, \rho)\} \xrightarrow{a} \{(X,\rho)\}}$$

This rule can be applied to state items (p, ρ_i) which have all external actions $\overline{put}(i)$. This action step will reduce the states according to rule **ASR**. □

3.4 Semantics of Time

The basic properties of the semantics of time are summarized below. They originate from the work on TCCS in e.g. [16] and [27] and are extended to fit into the semantic setting of MOSCA.

- **Maximal Progress** Whenever an agent can proceed by taking an internal action it will not wait.

$$\{s\} \Longrightarrow \{s'\} \Rightarrow$$
$$\nexists d \in TIME \cdot \{s\} \xrightarrow{\tau(d)} \{s''\}$$

  Whenever an agent can proceed by taking an external action it will not wait.

$$\exists a \in ExtAct \cdot \{s\} \xrightarrow{a} \{s'\} \Rightarrow$$
$$\nexists d \in TIME \cdot \{s\} \xrightarrow{\tau(d)} \{s''\}$$

- **Time Determinacy** To ensure the progress of time each agent, even the null agent and divergent agent can take idle actions. When time goes, if an agent idles, it can never reach different states:

$$\{s\} \xrightarrow{\tau(d)} \{s'\} \wedge \{s\} \xrightarrow{\tau(d)} \{s''\} \Rightarrow$$
$$\{s\} \equiv \{s''\}$$

  where $\equiv$ means state equality. Time determinacy results from the properties of idling in combination with and choice and composition. The divergent agent $\perp$ must be able to take idle actions as well. As time progression in compositions is synchronous, $P \mid \perp$ would not allow the time to proceed if $\perp$ would only take internal actions and no idle actions as well.

- **Time Continuity** During idling no time moment may be passed without notice.

$$\{s\} \xrightarrow{\tau(d+e)} \{s''\} \Leftrightarrow$$
$$\{s\} \xrightarrow{\tau(d)} \{s'\} \wedge \{s'\} \xrightarrow{\tau(e)} \{s''\}$$

- **Time persistency** By idling an agent will not loose the ability to perform an action that it is able to perform originally.

$$(\{s\} \xrightarrow{\tau(d)} \{s'\} \wedge \{s\} \xrightarrow{x} \{s''\}) \Rightarrow$$
$$\{s'\} \xrightarrow{x} \{s''\}$$

These basic properties of time in MOSCA reflect the incorporation of idling in the semantic rules of prefix, choice, composition, the null agent and divergent agents. The null agent is inactive but can be engaged in idling, to enable time progression in compositions that involve a null agent. The divergent agent is either idling or busy with internal actions.

4 Case study: a railroad controller

In order to be able to evaluate notations for use in the development of real-time software systems, we are performing a comparative review of some selected system specification notations. The study emphasizes the use of the notations in the domain of real-time (control) applications. Our review will be based on a simple railroad controller model. This case contains data modeling aspects, functional aspects as well as temporal aspects. A (toy) railroad with computer interface, is available in our laboratory, used for lab assignments. An Ada encoded controller, loosely based on the specification is running on a PC for demonstrations. Typical elements to consider are usability with regard to the specification in relation to the requirements, and secondly, usability with respect to further program development. In this section we briefly address some issues of the MOSCA specification of the controller. For a complete MOSCA specification the reader is referred to [3].

This section is further organized as follows. Section 4.1 shortly covers the problem description. Section 4.2 presents an analysis of the problem. Section 4.3 present fragments of a MOSCA specification.

4.1 Informal problem description

A (toy) railroad system is built up from connected rail elements (straight elements, bowed elements, crossings and switches). Connected rail elements

are grouped into *physical blocks*. These blocks are connected to hardwired *function decoders*. The average length of rail elements is 23 cm, the speed of a train can be varied between 0 and 30 cm per second.

Through a computer (serial) interface, commands can be given:

- status enquiry commands (i.e. blocks being occupied or not).
- switch setting/resetting commands.
- train commands, setting a particular speed, setting lights, a horn and a direction.

The problem is to specify (and subsequently design and implement) a software controller through which the behaviour of the system can be controlled. The behaviour of the train should conform to *requirements*, expressed as missions, one for each train. Missions specify routes which are expressed in terms of *tracks* and the train behaviour along the route. A track is a path over a sequence of connected rail elements. The sequence of rail elements itself, is called a *block*.

Obviously, the system should obey elementary safety precautions:

- trains should not collide, modeled by preventing two trains to ride on the same block;
- when a train is on a block, no switch in this block should be changed. Setting a switch while a train passes, might cause considerable physical damage.

The complete reference specification of the railroad controller is contained in [2].

4.2 Analysis of the problem

For each train, the controller has been given a mission that specifies actions to be taken by that train.

A mission is built up from a sequence of *commands*. A command consists of an *action specification*, a *position specification* and a *specification of temporal constraints*.

Command = INIT | START | STOP | PASS

Mission = *Command**

INIT causes a train to be initialized. The initial position is assumed to be in the middle of the specified track, the latter specified by the block identification (a number) and identifications of entry and exit points on this block (numbers as well).

START causes a train to start from the (assumed) position at the middle of the specified track (at *Start time*), and to reach the end of the track no later than *Exit time*.

STOP causes the train to stop, at *Stop Time*, near the middle of track *Track*.

PASS causes the train to run over a given track *Track*, with a speed sufficient to reach the end of this track no later than *Exit Time*.

A typical sequence of commands for a particular train that is to drive from the middle of a starting track (1, 1, 2) to the middle of track (4, 1, 2) and vice versa, may look like:

```
INIT   (1, 1, 2)
START  (1, 2, 1), 1000, 1010
PASS   (2, 1, 2), 1020
PASS   (3, 1, 2), 1030
STOP   (4, 1, 2), 1035
START  (4, 2, 1), 1035, 1040
PASS   (3, 2, 1), 1050
PASS   (2, 2, 1), 1060
STOP   (1, 2, 1), 1070
```

According to the above mentioned informal requirements, the system consists of (i) a railroad with a number of trains, (ii) some logic private to the railroad, and (iii) a controlling computer system. In line with observations in e.g. [21], we prefer a closed description, i.e. one that provides an integral description of both the behaviour of the controller and of the controlled system. Only then, the model can be used for (safety) analysis.

A first step in the development is to identify the different elements in the requirements model. Different brands of trains provide different features, abstraction from physical details is therefore necessary. The resulting structure of the requirements specification is schematically depicted in figure 5. In this model, we identify:

- The controller (further indicated with SCM), observes the positions of the trains and issues (speed setting and switch setting) commands

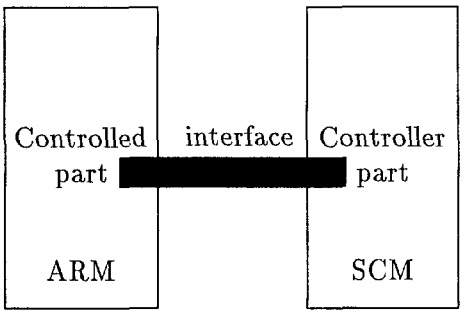


Figure 5: Structure of the controller

to correct observed deviations of the required position. The SCM is responsible for maintaining safety on the rail road.

- The controlled part (further indicated with ARM) provides information over train positions to the controller. It reacts upon the SCM by setting switches and by handling speed setting and direction commands for the trains.
- An *interface.* The interface between the ARM and SCM is chosen to be *data oriented.* It contains:
  - for each train the specification of the *required speed*, maintained by the SCM, to be interpreted by the ARM.
  - for each train an indication of the *observed position*, as determined by the ARM.
  - a specification of the *required* settings of switches.

Within the requirements model, trains, blocks and switches are typical entities with state. The state of a block indicates whether or not it is occupied, the state of a switch indicates its setting. A train is in one of eight states, the states and transitions are given in figure 6. Relevant for the discussion in section 4.3 are the following states.

- A train in state PASS remains running until it enters the next track on its route. If the command to be performed on this next track is a *PASS* command, an attempt will be made to allocate the next track on the route. If allocation fails, the train will slow

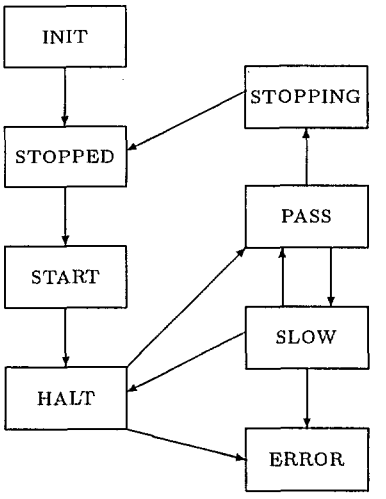


Figure 6: Train state diagram

down (and enter state SLOW), if allocation succeeds state PASS will be reentered. If the command on this track is *STOP* command, state STOPPING will be entered.

- In state STOPPING, the train will move forward until reaching the middle of the track. There it will take state STOPPED and halt. In this state, the only valid command is a *START* command.
- In state SLOW, the train will continue to attempt to allocate the required resource. If the train reaches the middle of the track *before* allocation succeeds, it enters state HALT and the train halts. If the train does obtain the required resource, state PASS will be (re)entered, and the train will move with the speed required to meet its temporal constraint.

The SCM 'observes' the position of each of the trains and sets (i) the switches for tracks that are entered next, and (ii) the speed of the trains. In this process, the SCM reacts upon the occurrence of events. The events are chosen such that the change in state for each individual train becomes clear.

The temporal constraints on the system only follow indirect. They are determined by the speed of the trains and the speed with which the system needs to react upon changes.

Typical elements from which specific temporal constraints are to be be derived, are:

- Switches should be set *before* a train enters the block containing these switches. if the settings for switches for a next track are given, before the train reaches the middle of the current track, more than 0.5 seconds is available for the ARM to set the switches.

- accuracy of position reading depends on the frequency with which this reading is updated.

The speed of the train dictates the setting of deadlines. Since the maximum speed of the train is app. 30 cm/second, and most of the rail elements are app. 23 cm, most of the deadlines can be set to app 100 msec, leaving room for a deviation of app. 3 cm.

Specification 4.1

CONTROLLER
state *ReqSpeed* : *SpeedTable*
ReqSwitches : *SwitchTable*
Position : *PositionTable*
Conf : *RailRoad*

shares *TIME*
ports syn *DurationEvent*
syn *InitEvent*
out $\overline{Time}$: $\mathbb{N}$
ARM
ports syn *DurationEvent*
in *SpeedEvent* : *SpeedTable*
in *SwitchEvent* : *SwitchTable*
out $\overline{TrackEvent}$: *Train*
out $\overline{PositionEvent}$: *Train*
state *CurrSpeed* : *SpeedTable*
CurrSwitches : *SwitchTable*
SCM
ports in *Time* : $\mathbb{N}$
syn *InitEvent*
in *TrackEvent* : *Train*
in *PositionEvent* : *Train*
out $\overline{SpeedEvent}$
out $\overline{SwitchEvent}$
state *AllocatedTracks* : *TrackTable*
TheTrains : *TrainTable*

$CONTROLLER \triangleq TIME \mid ARM \mid SCM$

4.3 A model for the system

The MOSCA model is systematically derived from the reference specification, which is based on an event/action paradigm [12]. In [26] we describe a simple method to transform a specification based on the event/action paradigm into a process oriented specification.

The complete MOSCA model of the railroad controller consists of (i) an elaborate model of the physical railroad, (ii) a description of the mission data structure, (iii) a distributed model of the state of the system, (iv) specifications for the timing device, the ARM and the SCM. The overall structure of the model is presented in figure 7.

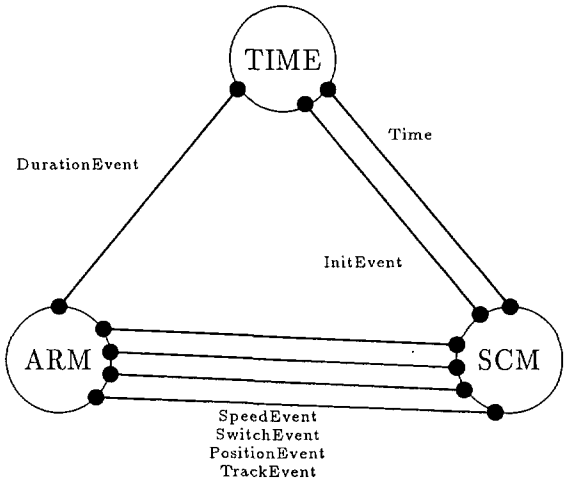


Figure 7: Structure of MOSCA model

The MOSCA specification is given in specification 4.1. From the globally addressable variables (the state) of the *CONTROLLER* agent are *ReqSpeed* and *ReqSwitches* read by the *ARM* agent, set by *SCM*. The component *Position* is set by *ARM* and read by *SCM*. The model of the railroad *Conf* is (for convenience) assumed to be properly initialized. The local variables of *ARM* model the actual devices, i.e. the trains and the switches. The local variables of *SCM* model the controller's view of the trains and the actual allocation of tracks per train. The *TIME* agent offers the time service to the *SCM* agent and two interval timers, the duration interval timer and the init event timer.

The timing device consists of two parallel components which are the clock and the interval timer. The clock ticks measure 1 msec. The du-

ration interval measures 100 msec. The specification of the timing device (4.2) starts by defining its interface to the other two processes. It consists of three ports offering a time service $\overline{Time}$ and signals that mark the ending of the timed interval. The behaviour of the *TIME* agent is defined as two processes running in parallel, the *Clock* and the *IntervalTimer*. The clock merely ticks and offers the time service. The interval timer times an interval of 100 msec. The event that marks the end of the interval is signaled by two parallel running signal servers that stop after passing the signal on to the *ARM* and the *SCM* processes.

Specification 4.2

```
agent TIME
ports syn DurationEvent
      syn InitEvent
      out Time              : ℕ
shares Clock ⟨ℕ⟩
       ports syn Tick
       IntervalTimer ⟨ℕ⟩
       ports syn Tick

TIME ≙
    (idle tick_delay ⊙ Clock ⟨1⟩ |
    IntervalTimer ⟨0⟩ ) \ {Tick}

Clock ⟨r⟩ ≙
        Tick, ⋆t ⊙
        idle (tick_delay − t) ⊙
        Clock ⟨r + 1⟩
      ⊕ Time(r), ⋆t1 ⊙
        Tick, ⋆t2 ⊙
        idle (tick_delay − (t1 + t2) ⊙
        Clock ⟨r + 1⟩

IntervalTimer ⟨count⟩ ≙
    if count = duration
    then (DurationEvent ⊙ null |
          InitEvent ⊙ null |
          Tick ⊙ IntervalTimer ⟨1⟩ )
    else Tick ⊙ IntervalTimer ⟨count + 1⟩

where

values
    tick_delay : 𝕋 = 0.001
      duration : ℕ = 100
end
```

The *TIME* agent models an ideal clock, an infallible device without any skew. The specification is unbiased with respect to its realization which can be either in hardware, software or a mixed implementation partly in hardware and partly in software. A clock with an expected skew can easily be modeled by adding a skew factor to the idle action following the *Tick* action,

```
Tick, ⋆t ⊙
idle (tick_delay − t + skew()) ⊙ ...
```

such that the function *skew* delivers a skew value.

Specification 4.3

```
agent ARM
ports syn DurationEvent
      in SpeedEvent       : SpeedTable
      in SwitchEvent      : SwitchTable
state CurrSpeed     : SpeedTable
      CurrSwitches  : SwitchTable
      extConf       : RailRoad
      extPosition   : PositionTable

shares NewPosHandler
       ports out PositionEvent
             out TrackEvent
       SpeedSettingHandler ⟨SpeedTable⟩
       SwitchSettingHandler ⟨SwitchTable⟩

ARM ≙
       DurationEvent ⊙
       NewPosHandler | ARM
     ⊕ SpeedEvent(train_speed) ⊙
       SpeedSettingHandler ⟨train_speed⟩
     ⊕ SwitchEvent(switch) ⊙
       SwitchSettingHandler ⟨switch⟩
```

Specification 4.3 presents the *ARM*. It is modeled as an event handler offering for each event a specialized handler. The *DurationEvent* signals the computation of new positions of the trains.

Specification 4.4

```
agent NewPosHandler
ports out PositionEvent : Train
      out TrackEvent    : Train
state ct : 𝔹

shares NewPosHandling ⟨Train-set⟩
```

```
NewPosHandler ≜
    let ts = dom CurrSpeed in
    NewPosHandling ⟨ts⟩

NewPosHandling ⟨ts⟩ ≜
    if ts = { }
    then null
    else let t ∈ ts in
        σ(ct:= ComputeNewPos(t)) ⊙
        if ct
        then TrackEvent(t) ⊙
             NewPosHandling ⟨ts \ {t}⟩
        else PositionEvent(t) ⊙
             NewPosHandling ⟨ts \ {t}⟩
```

Specification 4.4 presents the event handler that is activated on the duration event from the interval timer. Each event is given a associated handler. For each train in the system (registered in the domain of the *CurrSpeed* state element) it computes a new position through the VDM-SL operation *ComputeNewPos*.

NewPosHandling features a loose let construction. By selection of arbitrary trains from the set of trains in its value part it computes for each member of the set a new position. The changed position is subsequently signaled to the *SCM* by either a *PositionEvent* when the train has just moved within its current track, or by a *TrackEvent* when the train has moved on to a new track. The boolean state value *ct* holds the value true if a track change occurred during the change of position. The position handler, being part of *SCM* is given as a final example of a MOSCA agent. It handles the mode changes from SLOW to HALT and from STOPPING to STOPPED which were indicated in figure 6. In mode PASS First the current reading of the clock is bound to the name *now*. The computation proceeds by checking whether the train can reach its target within its time limit. If so, the train proceeds running, else the train enters mode ERROR. If the position of the train is near the middle of the track it either slows down to halt, or it stops completely.

5 Summary

The work on MOSCA is inspired by many other approaches on combining value manipulation, behaviour and time. In this context at least the following pioneering research should be mentioned: BJØRNER and JONES for their work on VDM [4], HENNESSY for his work on denotational semantics for process algebras [9]), MILNER for his work on CCS and scientists working on time extensions of process algebraic notations [1]), [27]. The LOTOS [5] and RAISE [8] specification languages come most closely to MOSCA in its current form. The main differences are the nature of the semantic treatment of the embedded value manipulation language. Whereas LOTOS and RAISE offer a value manipulation mechanism based on the algebraic approach, MOSCA offers a *model oriented* approach, taken from VDM-SL, which is based on combinations of simple mathematical models like sets, sequences, maps, trees, etc. and which is given a denotational semantics.

The specification language LOTOS (*Language Of Temporal Ordering Specification*) was designed to enable formal description of OSI architectural concepts such as services, protocols service access points, etcetera. Contrary to the name suggest, LOTOS is not related to temporal logic. LOTOS is based on CCS, with influences of CSP. CCS is used as semantical basis for the process part of the language. The requirements for abstractness favored the choice of an abstract datatype definition technique as value description device. Here ACT-ONE was chosen as a starting point. LOTOS is an executable notation. Recently experiments have been carried out to incorporate a notion of time in LOTOS [20].

RAISE (*Rigorous Approach to Industrial Software Engineering*) is a collection consisting of a specification language called RAISE Specification Language (RSL), a development method, and a set of supporting tools. The RAISE Method is, just like VDM, based on the notion of stepwise refinement. Its specification language RSL contains notions to express data-abstraction through model-oriented *and* property-oriented facilities. Control abstraction for parallel activities is based on the process concept of CSP extended with the facility to specify processes implicitly through *trace* and *failure* assertions. RSL does not handle the notion of time.

MOSCA shares with RAISE the notion of data-refinement. MOSCA lacks the facility to specify processes implicitly. MOSCA offers a prim-

Specification 4.5

```
agent PositionHandler ⟨Train⟩
ports out StoppedEvent  : Train
      out OnErrorEvent  : Train
state ext TheTrains     : TrainTable
      ext Position      : PositionTable
      NewSpeed          : SpeedTable

PositionHandler ⟨t⟩ ≙
    if TheTrains(t).Mode = PASS
    then Time(now) ⊙ let CD = TheTrains(t).CurrentTarget.Distance in
                     let CO = Position(t).CurrentOffset in
                     let DistanceToPass = CD − CO in
                     if now > TheTrains(t).CurrentTarget.Time
                       ∧ DistanceToPass > MAXDIFFERENCE
                     then σ(TheTrains(t).Mode:= ERROR) ⊙
                          OnErrorEvent(t) ⊙
                          null
                     else null
    else if NearMiddleOfTrack(Position(t))
        then if TheTrains(t).Mode = SLOW
             then Time(now) ⊙
                  σ(FromSlowToHalt(t, now)) ⊙
                  (SpeedEvent(NewSpeed) ⊙
                  null | CDTimer ⟨MAXDELAY⟩ )
             else if TheTrains(t).Mode = STOPPING
                  then σ(FromStoppingToStopped(t)) ⊙
                       SpeedEvent(NewSpeed) ⊙
                       StoppedEvent(t) ⊙ null
                  else null
        else null
```

itive structuring mechanism based on packaging concepts found in Ada and Modula-2 to control the complexity of large specifications. RSL offers, similar to LOTOS, the notion of abstract datatype specification as main structuring mechanism. MOSCA enables state-based development, like RAISE, and unlike LOTOS. This work differs from the other approaches mainly in the definition of the semantics, by combining a denotational semantics with an operational semantics. The combination was chosen on practical reasons. The VDM-SL language has been given a full denotational semantics, whereas the TCCS notations has been given a SOS semantics. It is an open question whether a single denotational or SOS semantics can be given that covers both the VDM-SL and TCCS notations.

The MOSCA project is in its beginning phase. A system specification language together with a conceptual model have been defined. The definition of the syntax and semantics forms a firm basis to address the other topics of interest concerning the notation such as a requirement specification language (a logic) and a verification technique (proof system). A further interesting topic is refinement of value-manipulation expressions and equivalences between behaviour expressions and their interrelationship.

This article has put the emphasis on the presentation of some aspects of handling looseness

and timing facilities of MOSCA. Currently the main focus in the project is on two related topics: tool development and analysis techniques. A rapid prototyper has been developed that generates ADA code from MOSCA specifications [17]. Next to the prototyper a state space analysis tool is being developed. State space analysis is a technique that can be successfully applied to safety and liveness problems. A serious problem remains the efficient generation of finite derived state spaces out of infinite full state spaces which is a current topic of research.

References

[1] J.C.M. Baeten and J.A. Bergstra. Real Time Process Algebra. *Formal Aspects of Computing*, 3:142–188, 1991.

[2] T. Biegstraaten, K. Brink, J. van Katwijk, W.J. Toetenel. A simple railroad controller: a case study in real-time specification using AE/VDM. Delft University of Technology, Faculty of Technical mathematics and Informatics, report 94-86

[3] T. Biegstraaten, K. Brink, R. Lutje-Spelberg, J. van Katwijk, W.J. Toetenel. A simple railroad controller: a case study in real-time specification using MOSCA. Delft University of Technology, Faculty of Technical mathematics and Informatics, report 94-87

[4] D. Bjørner and C.B. Jones. *Formal Specification & Software Development.* PHI. Prentice Hall, 1982.

[5] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. In P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors, *The Formal description Technique LOTOS*, pages 23–77. North Holland, 1989.

[6] F.P. Brooks Jr. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer*, 20(4):10–19, April 1987.

[7] J. Dawes. *The VDM-SL Reference Guide.* Pitman, 1991.

[8] A. Haxthausen and K. Havelund. RSL Reference Manual. Technical Report RAISE/CRI/DOC/2/V1, CRI, April 1990.

[9] M. Hennessy. A Proof System for Communicating Processes with Value-Passing. *Formal Aspects of Computing, Springer Verlag*, 3(4):346–366, 1991.

[10] M. Hennessy and Ingólfsdóttir. A Theory of Communicating Processes Value-passing. In M.S. Paterson, editor, *Automata, Languages and Programming (ICALP)*, volume 443 of *LNCS*, pages 209–220. Springer Verlag, 1990.

[11] Inormation Technology, Programming Languages – VDM-SL *First Commitee Draft Standard: CD 13817-1, November 1993.* Document ISO/IEC/JTC1/SC22/WG19 N-20

[12] F. Jahanian and A.K-L. Mok. Safety Analysis of Timing Properties in Real-Time Systems. *IEEE TSE*, se-12(9):890–904, September 1986.

[13] N.G. Leveson. The Challence of Building Process-Control Software. *IEEE Software*, pages 55–62, November 1990.

[14] R. Milner. Calculi for Synchrony and Asynchrony. *TCS*, 25:267–310, 1983.

[15] R. Milner. *Communication and Concurrency.* PHI. Prentice Hall, 1989.

[16] F. Moller and C. Tofts. A Temporal Calculus of Communicating Systems. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR'90: Theories of Concurrency: Unification and Extension*, volume 458 of *LNCS*, pages 401–415, Amsterdam, The Netherlands., August 1990. Springer Verlag.

[17] A. Ottens and W.J. Toetenel. Simulation of Mosca Specifications in Ada. In J. van Katwijk, editor, *Proceedings of the Ada-Europe'92 conference*, LNCS. Springer Verlag, 1992.

[18] D.L. Parnas. Software Aspects of Strategic Defence Systems. *Communications of the ACM*, 28(12):1326–1335, 1985.

[19] G. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[20] J. Quemada, A. Azcorra, and D. Frutos. TIC: A Timed Calculus for LOTOS. In S.T. Vuong, editor, *Formal Description Techniques II.* Elsevier Science Publishers B.V. (North Holland), 1990.

[21] K .Sima'an. Design principles for Real-Time Process Control Systems. Delft University of Technology, Faculty of Technical mathematics and Informatics, report 94-42

[22] J.A. Stankovic. Misconceptions About Real-Time Computing: A Serious Problem for next generation Systems. *IEEE Computer*, 21(10):10–19, October 1988.

[23] W.J. Toetenel. *Model Oriented Specification of Communicating Agents.* PhD thesis, Faculty of Technical Mathematics and Informatics, Delft University of Technology, 1992.

[24] W.J. Toetenel. VDM + CCS + TIME = MOSCA. In *Proceedings of the 18th workshop of IFIP/IFAC WRTP'92.* Brugge, 1992.

[25] W.J. Toetenel. Loose Real-time Communicating Agents. In *Proceedings first workshop on Semantics of Specification Languages (SOSL), Utrecht*, London, 1993. Springer Verlag.

[26] W.J. Toetenel and J. van Katwijk. Stepwise development of model-oriented real-time specifications from action/event models. In J. Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *LNCS.* Springer Verlag, 1992. Proceedings of RTFT'92, Nijmegen.

[27] Y. Wang. Real-Time Behaviour of Asynchronous Agents. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR'90 Theories of Concurrency: Unification and Extension*, volume 458 of *LNCS*, pages 502–520. Springer Verlag, 1990.

An Object-Oriented Approach for Modeling and Analysis of Safety-Critical Real-time Systems

Jyhjong Lin, David Chenho Kung and Pei Hsia
Computer Science Engineering
The University of Texas at Arlington
P.O. BOX 19015, Arlington, TX 76019-0015
E-mail:jlin|kung|hsia@cse.uta.edu

Keywords: object-oriented conceptual modeling, distributed and parallel real-time system, control flow, safety, failure, analysis

Edited by: Marcin Paprzycki and Janusz Zalewski
Received: February 15, 1994 **Revised:** October 21, 1994 **Accepted:** December 22, 1994

This paper presents an object-oriented approach that deals with modeling and analysis of concurrent real-time systems whose behavior must satisfy certain safety considerations. The approach describes the structural aspect and the time dependent behavioral aspect of objects in one model, and allows formal analysis of the model properties. The description of object behavior also contains a representation of control flow that specifies desirable execution sequences of object activities. For designing for fault tolerance and safety, it also supports modeling of failures to object behavior and their resultant faults. In particular, including the types of faults is useful for modeling and analysis of failures in more general situations.

1 Introduction

Conceptual modeling is an important step in developing a computer based application which collects adequate user requirements about the application domain (i.e., the structural and behavioral aspects of the application). It has been recognized that failure to identify the real application domain knowledge may result in late delivery, poor quality, and high maintenance costs. Traditionally, conceptual modeling is done by using function-oriented approaches (Cameron 1986, Jackson 1983) or data-oriented (Hull & King 1987, Peckham & Maryanski 1988).

In function-oriented approaches, software development begins from the analysis of system functionality to obtain a data flow diagram of the system. Entity-relationship diagrams are also used to identify the relationship between the data entities. The contents and logical structures of the data flows and data entities are often specified in a data dictionary. This paradigm has several drawbacks: (1) Consistency between the data flow and the data dictionary is hard to maintain; (2) Since the functions/data are not mappings to the objects in the real world, a change in the application could result in the modifications to many functions/data; (3) The separate treatment of data and functions makes software reuse difficult.

In contrast, data-oriented approaches start from the modeling of data structures. The specification of system functionality is then done according to the data structures. Usually, these two aspects are specified in two separate models: a structural model and a process model. The drawbacks of this paradigm are: (1) Consistency between the structural model and the process model is difficult to maintain; (2) Interaction between functions is not explicitly modeled. So, its effect is difficult to comprehend; and (3) Since data and functions are treated separately, software reuse is still difficult.

The development of object-oriented modeling approaches is motivated by the problems in function-oriented and data-oriented approaches. It starts with the modeling of objects which represent as close as possible real-world entities.

The data and operations are encapsulated in an object. This allows software to be easily adapted to changes in the application. Other significant features and benefits of object-oriented models are: (1) Inheritance greatly simplifies the design and development of applications; (2) Encapsulation supports information hiding, increases software reuse, and reduces maintenance costs.

As modern real-time systems become larger and more complex, the features and benefits of object-oriented techniques have also stimulated using them to produce real-time software that is easy to understand, maintain, and reuse. This paper presents an object-oriented approach for modeling and analysis of concurrent real-time systems whose behavior must satisfy certain safety considerations. For the complex features of these systems such as concurrency, nondeterminism, safety, and fault tolerance, the approach should have the following features: (1) The modeling constructs are intuitive and easy to comprehend (e.g., the use of graphical representations); (2) The structural and behavioral aspects of objects are described in one *cognitive model* (Balzer & Goldman 1979). This enhances maintaining consistency between these two aspects when changes are made to one of them; (3) Timing characteristics are encapsulated in an object. This facilitates object reuse for various real-time applications. For example, different objects have the same functionality with the same interface but with different timing characteristics; (4) The model is able to represent concurrency and synchronization to manage concurrent access to objects; (5) The behavioral modeling contains a representation of desirable execution sequences of object activities; (6) Failures to object behavior and their resultant faults must also be able to be modeled for the requirements of designing for fault tolerance and safety; and (7) The model supports sufficient formality so that formal analysis can be conducted to verify its properties.

The proposed model is an extension and refinement of an object-based executable conceptual model (Kung 1989). The extensions include object structure, encapsulation, timing characteristics, modeling of control flow, and specification of failures and faults. It describes the structural aspect and the time dependent behavioral aspect of objects in one model, and allows formal analysis of the model properties. Structural modeling describes object types, attributes, and relationships between object types, while behavioral modeling describes the operations of objects and their effects on other objects. The description of object behavior also contains a representation of control flow that specifies desirable execution sequences of object activities. In particular, it also supports modeling of failures to object behavior and their resultant faults for the requirements of designing for fault tolerance and safety. Including the types of faults is useful for modeling and analysis of failures in more general situations. An advantage of the presented approach is that it encapsulates object states for ease to comprehend and to use. System behavior from individual objects can be readily obtained and analyzed. The modeling of system behavior is based on a timed transition net. This net is chosen here for the following reasons: (1) It is able to deal with concurrency, synchronization, and nondeterminism in a reasonable manner; (2) It is very convenient for analyzing and verifying desirable properties.

This paper is organized as follows. Section 2 presents the modeling constructs of our approach. For illustration, a simple example of the alternating bit protocol is presented and modeled. The modeling of control flow, failures, and faults, is described in Section 3. Section 4 discusses how system behavior is obtained and analyzed. Finally, Section 5 has the related work and conclusions.

2 Object Type

In our approach, objects are divided into *persistent* ones, *control* ones, and *fault* ones. A persistent object models an entity or thing in the application domain with behavior satisfying application imposed constraints (i.e., static constraints and time/temporal constraints). Control objects are used to give control flow between the operations of persistent objects, while fault objects are used to model failures of the behavior of persistent objects and faults which result from the failures. We shall present in the next section how control and fault objects are used for the requirements of designing for fault tolerance and safety. In this section, we introduce persistent object types that model the persistent objects.

A persistent object type T is a 7-tuple given

by: $T = (N_T, A_T, S_T, G_T, \Theta_T, \Phi_T, \Gamma_T)$
(except for N_T, the other components are optional), where

1. N_T is its name to be denoted by a string of characters.

2. A_T is a set of attributes. Each attribute has an associated type, which can be a set, a tuple, a list, or a persistent object type.

3. S_T is sets of subtypes, disjoint subtypes (ds), generalization subtypes (gs), and/or partition subtypes (ps). Objects of two subtypes may be overlapping; objects of disjoint subtypes are disjoint; the union of the objects of all the generalization subtypes must equal to the objects of the supertype at all times; partition subtypes are both disjoint and generalization subtypes. For a formal definition of these subtypes, the reader is referred to (Kung 1990). Attributes, constraints, and operations of a supertype are inherited by all of its subtypes.

4. G_T is a set of *component* object types with associated ranges. The ranges indicate the number of objects of the component types that an object of the type may have.

5. Θ_T is a set of static constraints. These constraints are specified as closed tuple calculus like expressions and must be satisfied by every object of the type at any time.

6. Φ_T is a set of time/temporal constraints. They are specified in a real-time logic and must be satisfied by every permissible evolution of every object of the type.

7. Γ_T is a set of operations. Each operation $\gamma \in \Gamma_T$ is a 5-tuple $\gamma = (N_\gamma, P_\gamma, I_\gamma, O_\gamma, R_\gamma)$
(except for N_γ, the other components are optional), where N_γ is its name, P_γ is a list of parameters with associated types, I_γ is a set of input object types, O_γ is a set of output object types, and R_γ is a list of execution rules.

   The parameters in P_γ specify additional inputs that must be supplied externally when γ is executed. Objects of the types in I_γ are either consumed or referenced by γ, while objects of the types in O_γ are produced by γ. An object type can be both in I_γ and in O_γ, meaning that its objects are updated by γ. Each execution rule Ri $\in R_\gamma$ is a 4-tuple Ri $= (L, [l, h], pre, post)$
   where L is a label 'Ri' denoting (it is) the ith rule, $[l, h]$ is a time interval, *pre* is a precondition, and *post* is a postcondition. The time interval $[l, h]$ is closed in its two ends l and h, and l must be less than or equal to h. The precondition *pre* is specified in a QUEL-like language that refers to (references/consumes) the objects of the types in I_γ. The postcondition *post* that produces new objects of the types in O_γ is a conjunction of assignments of the form $x.A = expr$ meaning that the attribute A of object x is set to the value of the expression *expr*.

   γ is executable if and only if a tuple of input objects (i.e., one object per input type in I_γ) makes the precondition *pre* of some execution rule Ri true (for some i, γ.Ri.*pre* is true). Note that the specification of the execution rules must be that only one rule's precondition can be satisfied by such a tuple of input objects. That is, if for some i, γ.Ri.*pre* is true, then for all j, j $\neq$ i, γ.Rj.*pre* must be false. Once γ becomes executable on rule Ri, the time interval $[l, h]$ of Ri specifies a period during which an execution on Ri must be performed. The execution produces a tuple of output objects that satisfy the postcondition *post* of Ri. (If γ.Ri.*pre* is true, then γ.Ri.*post* must become true during l to h units of time.)

In illustration, we show in Figure 1 the specification of a simple alternating bit protocol between two entities (Bartlett et al. 1969). An *entity* object type is specified to model the two entities. The clause "Component Object Type: *sender*[1], *receiver*[1]" specifies an aggregation relationship between *entity* and *sender/receiver*: an *entity* object is composed of a *sender* object and a *receiver* object. The static referential constraint

$$(\exists s \in sender)(\exists r \in receiver)(e\# = s.e\# \wedge e\# = r.e\#)$$

for the $e\#$ attribute of the *entity* object type is specified to illustrate this. In the *sender* object

Persistent Object Type *entity*;
Attribute *e#*: Int $(\exists s \in sender)(\exists r \in receiver)(e\# = s.e\# \wedge e\# = r.e\#)$;
Component Object Type *sender*[1], *receiver*[1];
...
End Persistent Object Type;

Persistent Object Type *sender*;
Attribute *e#*: Int, *outmsg*: String[80], *dest*: Int, *seq#*: Int, *waitack*: boolean;
Operation send_msg(Message: String[80], Dest: Int);
Persistent Object Updated $s \in sender$, Produced $m \in message\_packet$;
Execution Rule:
R1: $[0, \infty]$ pre $s.e\# \neq$ Dest $\wedge s.waitack =$ false;
post $m(from = s.e\#, dest =$ Dest, $seq\# = s.seq\#, msg =$ Message$)\wedge$
$s(outmsg =$ Message, $dest =$ Dest, $waitack =$true);
End Operation;
Operation receive_ack();
Persistent Object Updated $s \in sender$, Consumed $a \in ack\_packet$;
Execution Rule:
R1: [0,1] pre $s.e\# = a.dest \wedge s.seq\# = a.seq\# \wedge s.waitack =$true;
post $s(seq\# = (s.seq\# + 1) \bmod 2, waitack =$false);
End Operation;
End Persistent Object Type;

Persistent Object Type *receiver*;
Attribute *e#*: Int, *inmsg*: String[80], *from*: Int, *seq#*: Int, *acksent*: boolean;
Operation receive_msg();
Persistent Object Updated $r \in receiver$, Consumed $m \in message\_packet$;
Execution Rule:
R1: [0,1] pre $r.e\# = m.dest \wedge r.seq\# \neq m.seq\#$;
post $r(inmsg = m.msg, from = m.from, seq\# = m.seq\#, acksent =$false);
R2: [0,1] pre $r.e\# = m.dest \wedge r.seq\# = m.seq\#$;
post $r(acksent =$false);
End Operation;
Operation send_ack();
Persistent Object Updated $r \in receiver$, Produced $a \in ack\_packet$;
Execution Rule:
R1: [0,1] pre $r.acksent =$ false;
post $a(from = r.e\#, dest = r.from, seq\# = r.seq\#) \wedge r(acksent =$true);
End Operation;
End Persistent Object Type;

Figure 1: Object type specification

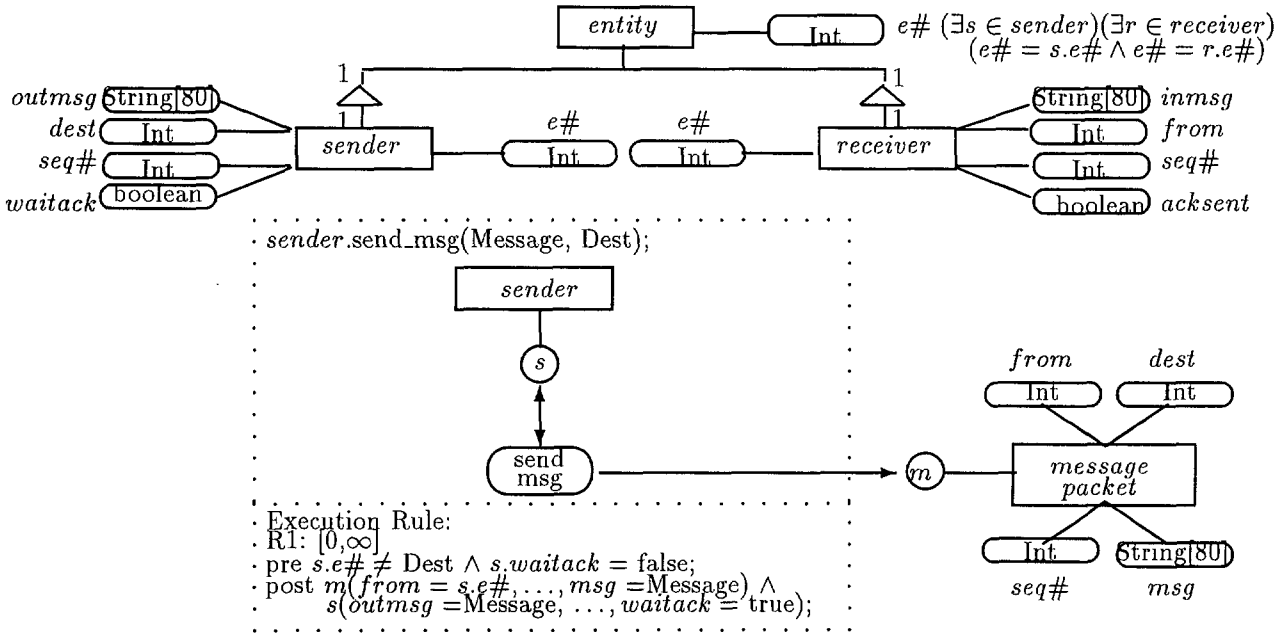


Figure 2: Graphical representation of object type

type, the clause "Persistent Object Updated $s \in$ *sender*, Produced $m \in$ *message_packet*" specifies the input/output of its send_msg() operation. The operation has an execution rule R1 that assures if a message is going to be sent out and the sender currently is not waiting for some *ack* from the other entity, then a packet carrying the message is produced and transmitted to the other entity (although its time interval $[0, \infty]$ indicates that no time constraint is given for sending the packet). After sending the packet, the sender waits for a corresponding *ack* (set *waitack* = true). Rule R1 of the receive_ack() operation assures that the sender can transmit another packet (numbered a modulo-2 sequence number by setting $s.seq\# = (s.seq\#+1) \bmod 2$) after it receives the *ack*.

The (modulo-2) sequencing function is introduced to let the receiver, upon reception of a packet, be able to decide whether this packet carries a new message or a duplicate of the immediately preceding one. The modulo-2 is chosen because it is sufficient to distinguish messages which are transmitted one at a time. The receive_msg() operation of the *receiver* object type is to receive (numbered) messages from the other entity. Rule R1 assures that it receives a new message, while rule R2 makes sure that the message received is a duplicate, and hence, simply discards it. However, an *ack* will be sent out after receiving a message no matter whether this message is new or duplicate. This is modeled by rule R1 of the send_ack() operation.

Here, it should be noted that since a packet from the sender of an entity must be received by the receiver of the other entity (by recognizing $s.e\# \neq$ Dest), two senders are able to send out their message to each other at the same time (if there exists a transmission medium between each pair of sender and receiver). However, since our specification considers only the transmission between two entities, it cannot correctly model the case of three or more entities. For example, the receiver of an entity is unable to distinguish which packet comes from which sender of other entities.

Figure 2 is the graphical specification of part of the *sender* object type that is textually specified in Figure 1. The upper part depicts its structural aspect and the relationship with other object types, while the lower part specifies its send_msg() operation together with the interaction with other objects. The objects updated and produced by the operation are referred to by the small circles which are the interface between these objects and the operation.

This graphical representation is an important feature of our approach because it gives a high level abstraction of the modeled system which makes it easier to communicate with users and understand the conceptual model. The operations of an object type are specified one at a time, displayed together with the structural aspect of the object type and its interaction with other objects. Hence, the effects of the modifications on objects and attributes by an operation become explicit and visible. We feel this is a good way to model encapsulation because an analyst can concentrate on one operation at a time while specifying the interaction between objects. The complexity of a conceptual model is thus largely reduced so as to alleviate the difficulty of modeling a non-trivial application. When the graphical specifications of the operations are put together, they show the complete behavior of the objects of the object type as in the textual specification. Figure 3 shows the most commonly used graphical symbols of our approach.

3 Designing for Safety and Fault Tolerance

The modeling of control flow, failures, and faults, is essential in conceptual modeling for safety-critical real time systems. In our approach, it is achieved by introducing *control* objects to model control flows between the activities of persistent objects, *failure* operations to denote the failure events, and *fault* objects to specify the faulty conditions caused by the failures. (Note that the modeling of failures and faults was first proposed in (Leveson & Stolzy 1987) upon which our approach is based.)

A control object type C contains a name N_C and a set A_C of attributes (see their counterpart in Section 2). Control objects are in general introduced between two persistent object operations, to be consumed/produced by these two operations, to model the desirable control flows between them (i.e., the chronological ordering of their executions). Figure 4 shows how some com-

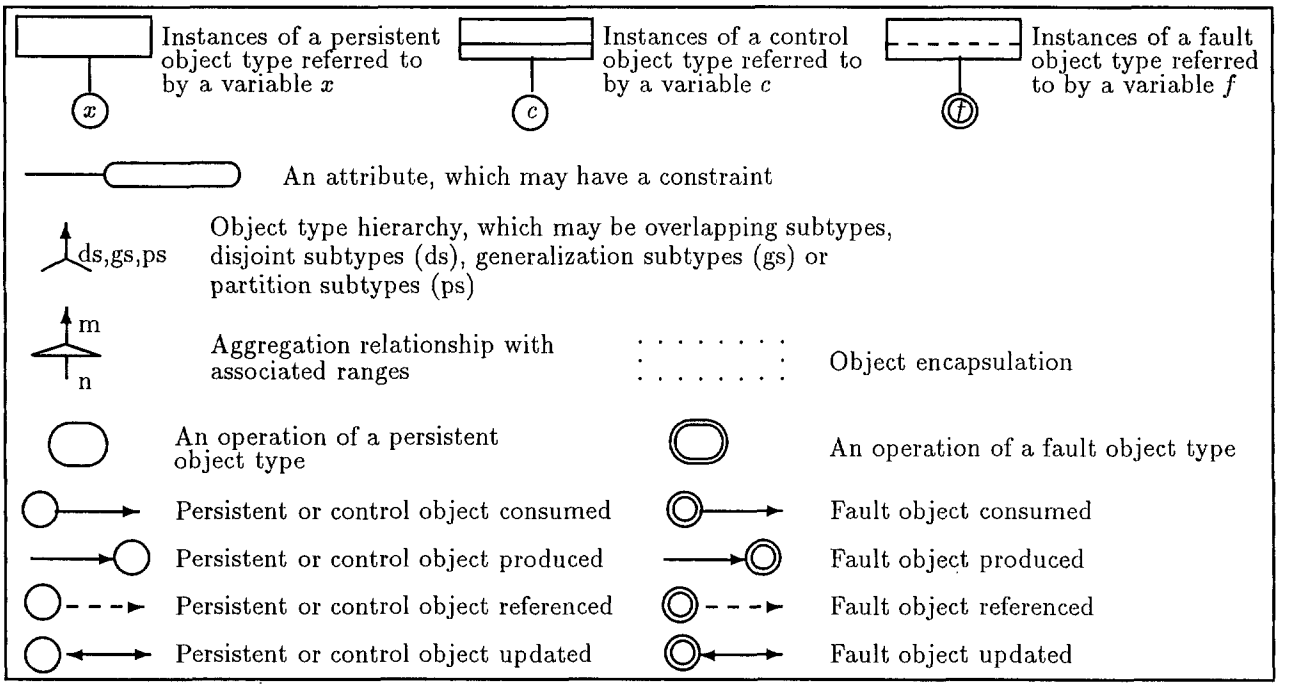


Figure 3: Graphical symbols for object-oriented models

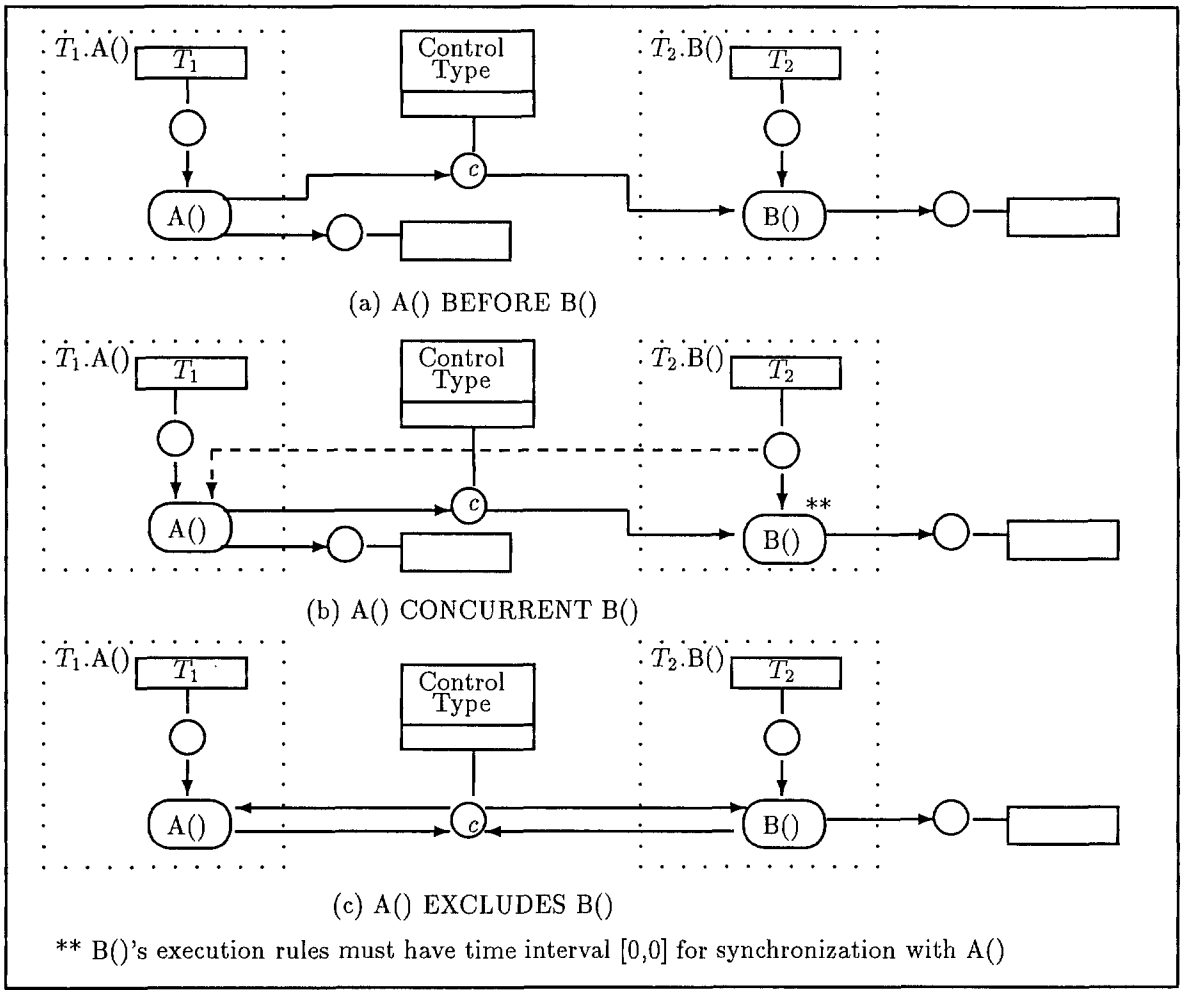


Figure 4: Specification of some common chronological orderings

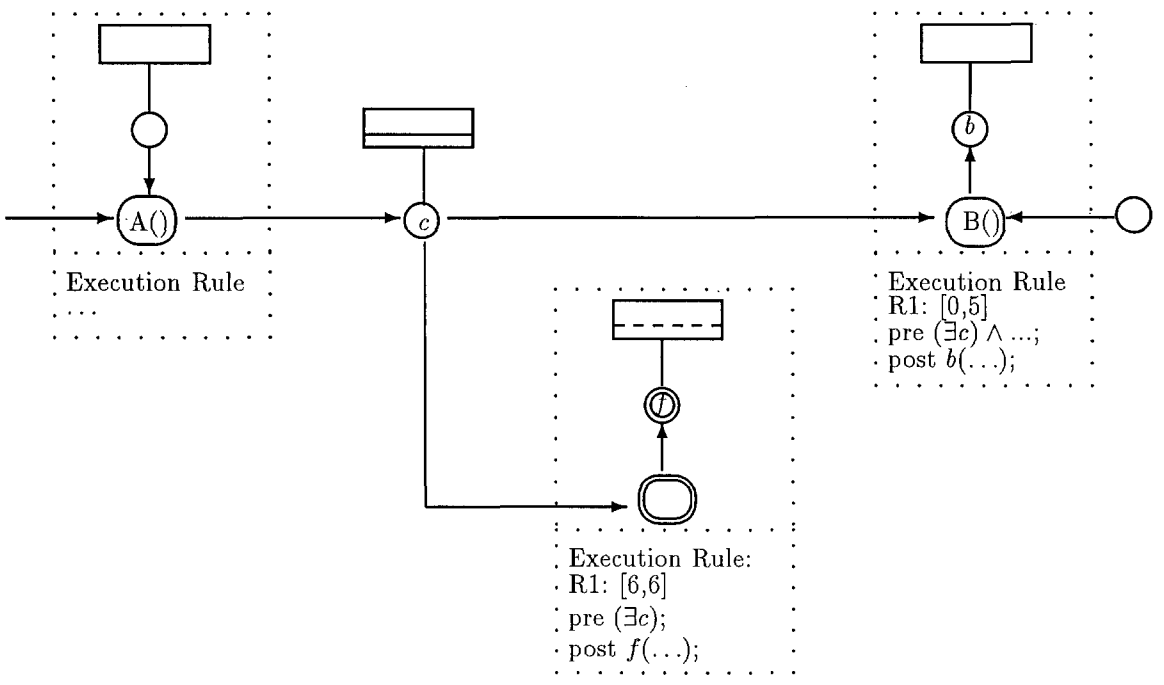


Figure 5: maximum time constraint between operations A() and B()

mon chronological orderings between two persistent object operations can be modeled by using control objects. We use two example operations A() and B() in the figure; each one consumes an input persistent object and produces an output persistent object. A control object c is consumed/produced by them to model their possible relationships.

For safety requirements, failures of the behavior of persistent objects must be detected during analysis of the model. The modeling of failures and faults is done by specifying fault object types. A fault object type F contains a name N_F, a set A_F of attributes, and a set Γ_F of operations (see their counterpart in Section 2). These operations consume (or produce spurious) persistent/control objects to denote the occurrences of failures. If it is needed to determine the states after these failures have occurred (i.e., to analyze their consequences on the system to differentiate their costs), fault objects of the type will be produced to specify the resultant faulty conditions by these failures.

With the fault object types, one can model and analyze failures in more general situations. In this paper, we illustrate only their usefulness on control failures such as failing to ensure minimum (or maximum) time constraints between two operations and the occurrence of an undesirable event. Figure 5 shows, for example, a maximum time constraint between two persistent object operations: *executing operation* A() *implies that sometime within 5 time units operation* B() *must be executed.* The time interval [6,6] of rule R1 of the failure operation is specified such that it can be executed only if operation B() is not executed within 5 time units (a failure by exceeding the maximum time constraint). A fault object is produced to specify the faulty condition. This object (faulty condition) will remain in the system until a terminating activity for it (i.e., the execution of a persistent object operation that consumes it).

In our illustrative example of the alternating bit protocol, a possible failure is the loss of a message or ack packet being transmitted. As shown in Figure 6, this is modeled by specifying the lose_msg() and lose_ack() operations of the fault object type F_1 that consume the message packet and the ack packet respectively. Objects of the control object type C are used to model the order of executing the send_msg() and receive_ack() operations: *after sending a message, the sender waits for a responsive acknowledgment.* Because of the possible loss of messages or acknowledgments, it is necessary to have a maximum time constraint between them: *the sender waits for the*

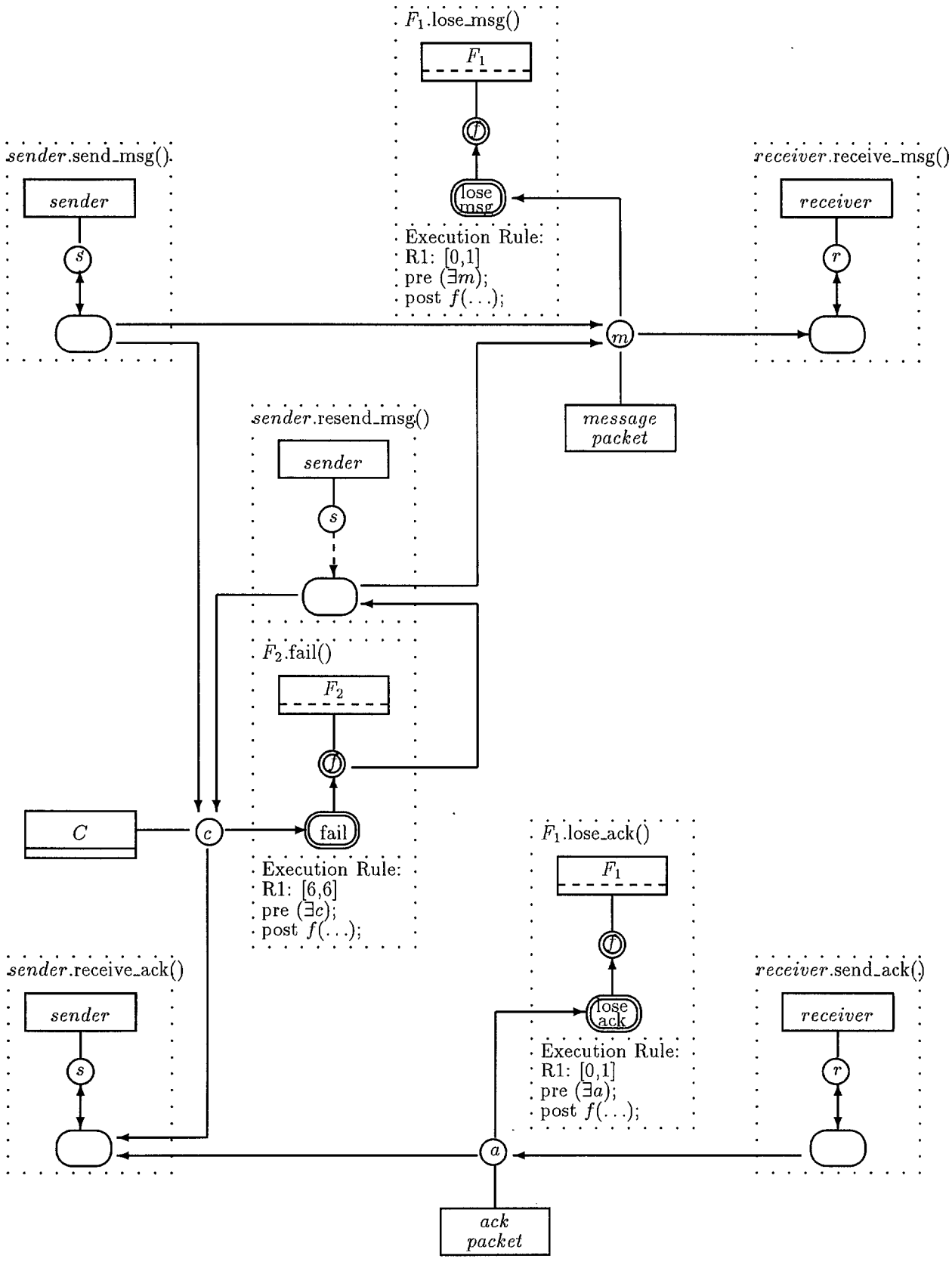


Figure 6: Specification of recoverable alternating bit protocol

Persistent Object Type *entity*;
Attribute $e\#$: Int $(\exists s \in sender)(\exists r \in receiver)(e\# = s.e\# \wedge e\# = r.e\#)$, *state*: String[8];
Component Object Type *sender*[1], *receiver*[1];
...
End Persistent Object Type;

Persistent Object Type *sender*;
Attribute $e\#$: Int, *outmsg*: String[80], *dest*: Int, $seq\#$: Int, *waitack*: boolean;
Operation send_msg(Message: String[80], Dest: Int);
Persistent Object Updated $s \in sender$, Produced $m \in message\_packet$,
Control Object Produced $c \in C$;
Execution Rule:
R1: [0,∞] pre $s.e\# \neq$ Dest $\wedge s.waitack =$ false;
post $m(from = s.e\#, dest =$ Dest, $seq\# = s.seq\#, msg =$ Message$) \wedge$
$s(outmsg =$ Message, $dest =$ Dest, $waitack =$ true$) \wedge$
$c(from = s.e\#, dest =$ Dest, $seq\# = s.seq\#)$;
End Operation;
Operation resend_msg();
Persistent Object Referenced $s \in sender$, Produced $m \in message\_packet$,
Control Object Produced $c \in C$, Fault Object Consumed $f \in F_2$;
Execution Rule:
R1: [0,1] pre $s.e\# = f.from \wedge s.dest = f.dest \wedge s.seq\# = f.seq\# \wedge s.waitack =$ true;
post $m(from = s.e\#, dest = s.dest, seq\# = s.seq\#, msg = s.outmsg) \wedge$
$c(from = s.e\#, dest = s.dest, seq\# = s.seq\#)$;
End Operation;
Operation receive_ack();
Persistent Object Updated $s \in sender$, Consumed $a \in ack\_packet$,
Control Object Consumed $c \in C$;
Execution Rule:
R1: [0,1] pre $s.e\# = a.dest \wedge s.seq\# = a.seq\# \wedge s.waitack =$ true $\wedge$
$c.from = a.from \wedge c.dest = a.dest \wedge c.seq\# = a.seq\#$;
post $s(seq\# = (s.seq\# + 1) \bmod 2, waitack =$ false);
End Operation;
End Persistent Object Type;

Persistent Object Type *receiver*;
Attribute $e\#$: Int, *inmsg*: String[80], *from*: Int, $seq\#$: Int, *acksent*: boolean;
Operation receive_msg();
Persistent Object Updated $r \in receiver$, Consumed $m \in message\_packet$;
Execution Rule:
R1: [0,1] pre $r.e\# = m.dest \wedge r.seq\# \neq m.seq\# \wedge r.acksent =$ true;
post $r(inmsg = m.msg, from = m.from, seq\# = m.seq\#, acksent =$ false);
R2: [0,1] pre $r.e\# = m.dest \wedge r.seq\# = m.seq\# \wedge r.acksent =$ true;
post $r(acksent =$ false);
End Operation;
Operation send_ack();
Persistent Object Updated $r \in receiver$, Produced $a \in ack\_packet$;
Execution Rule:
R1: [0,1] pre $r.acksent =$ false;
post $a(from = r.e\#, dest = r.from, seq\# = r.seq\#) \wedge r(acksent =$ true);
End Operation;
End Persistent Object Type;

Figure 7: Textual specification of recoverable alternating bit protocol (*to be continued*)

Fault Object Type F_1;
  Attribute $f\#$: Int, $from$: Int, $dest$: Int, $seq\#$: Int;
  Operation lose_msg();
    Persistent Object Consumed $m \in message\_packet$, Fault Object Produced $f \in F_1$;
    Execution Rule:
    R1: [0,1] pre $(\exists m)$;
        post $f(from = m.from, dest = m.dest, seq\# = m.seq\#)$;
  End Operation;
  Operation lose_ack();
    Persistent Object Consumed $a \in ack\_packet$, Fault Object Produced $f \in F_1$,
    Execution Rule:
    R1: [0,1] pre $(\exists a)$;
        post $f(from = a.from, dest = a.dest, seq\# = a.seq\#)$;
  End Operation;
End Fault Object Type;

Fault Object Type F_2;
  Attribute $f\#$: Int, $from$: Int, $dest$: Int, $seq\#$: Int;
  Operation fail();
    Control Object Consumed $c \in C$, Fault Object Produced $f \in F_2$,
    Execution Rule:
    R1: [6,6] pre $(\exists c)$;
        post $f(from = c.from, dest = c.dest, seq\# = c.seq\#)$;
  End Operation;
End Fault Object Type;

Figure 7: Textual specification of recoverable alternating bit protocol (*continued*)

acknowledgment at most 5 time units. Failure to meet this constraint will result in resending the message. The fail() operation of the fault object type F_2 denotes the failure. It consumes the control object if the sender cannot receive the acknowledgment timely (also, cannot consume the control object timely). An object of type F_2 is produced to specify the faulty condition, which is then used to invoke the resend_msg() operation to retransmit the message. This fault object is consumed by the resend_msg() operation to terminate the faulty condition. Figure 7 shows textual specification of the recoverable alternating bit protocol.

4 System Behavior Analysis

Analyzing the system behavior of our model is achieved by first modeling overall behavior among persistent objects in a timed transition net. Then, the reachability graph for the net is constructed to analyze its properties.

4.1 Timed Transition Net

Based upon the approach in (Chao & Kung 1991, Kung 1989), the net consists of a set of transitions and places. Transitions model the events that occur instantaneously in a system, and places are the input/output of transitions. Each transition is associated with a time interval $[l, h]$, where l and h represent the minimum and maximum time delay that must elapse before the transition is executed. Times l and h are relative to the moment at which the transition becomes executable.

In addition, functional capabilities are incorporated in the net. That is, typed tokens are assigned to places, and each token in a place represents an object of some (object) type. A transition is executable if and only if each of its input places contains a token and its precondition is true. Tokens in input places are either consumed or referenced by the transition, while tokens in output places are produced by the transition. For modeling failures and faults in the net, transitions are divided into *normal* ones and *failure* ones (Leveson & Stolzy 1987). The faulty conditions caused by failure transitions are denoted by the *fault* output places of these failure transiti-

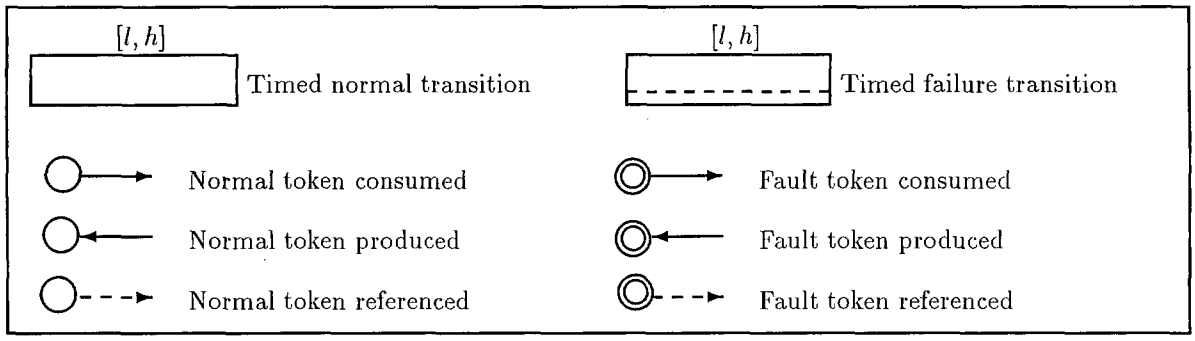


Figure 8: Graphical symbols of timed transition nets

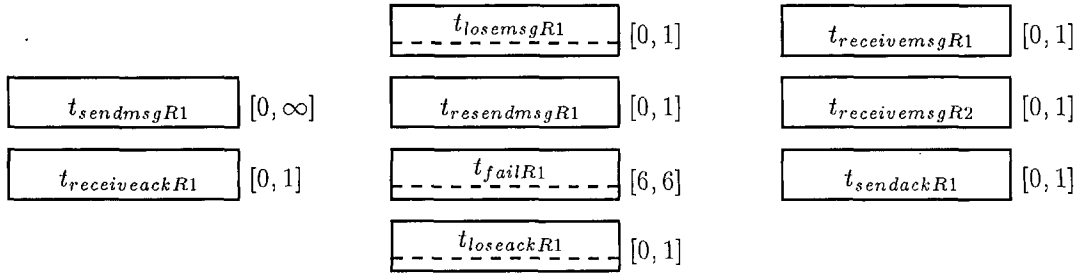


Figure 9.

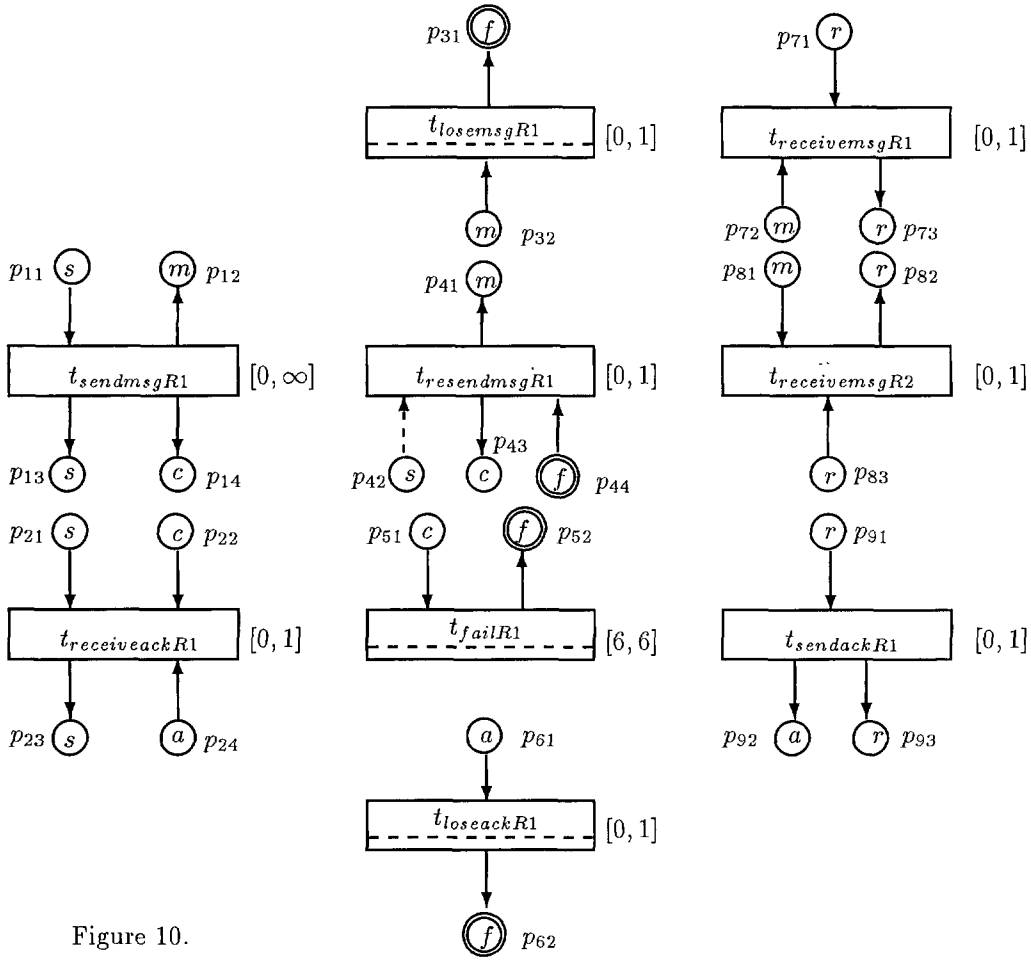


Figure 10.

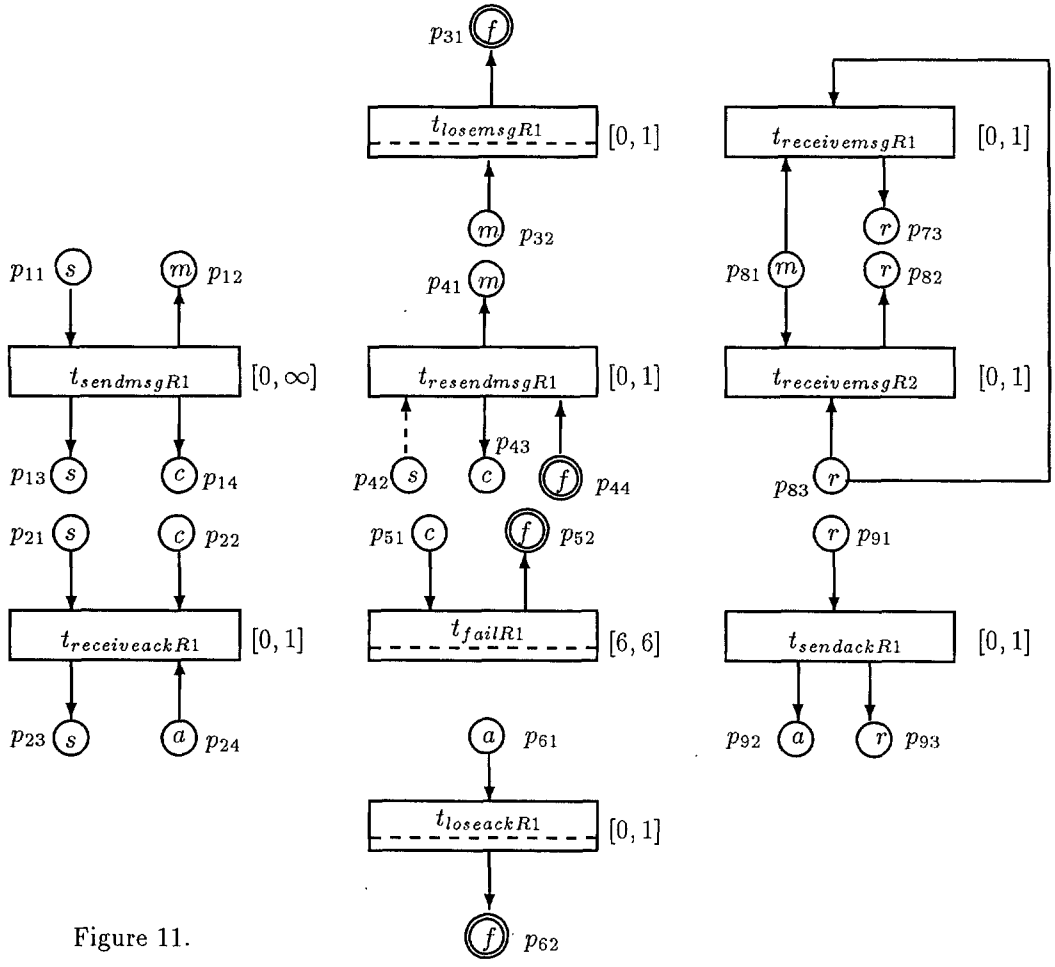


Figure 11.

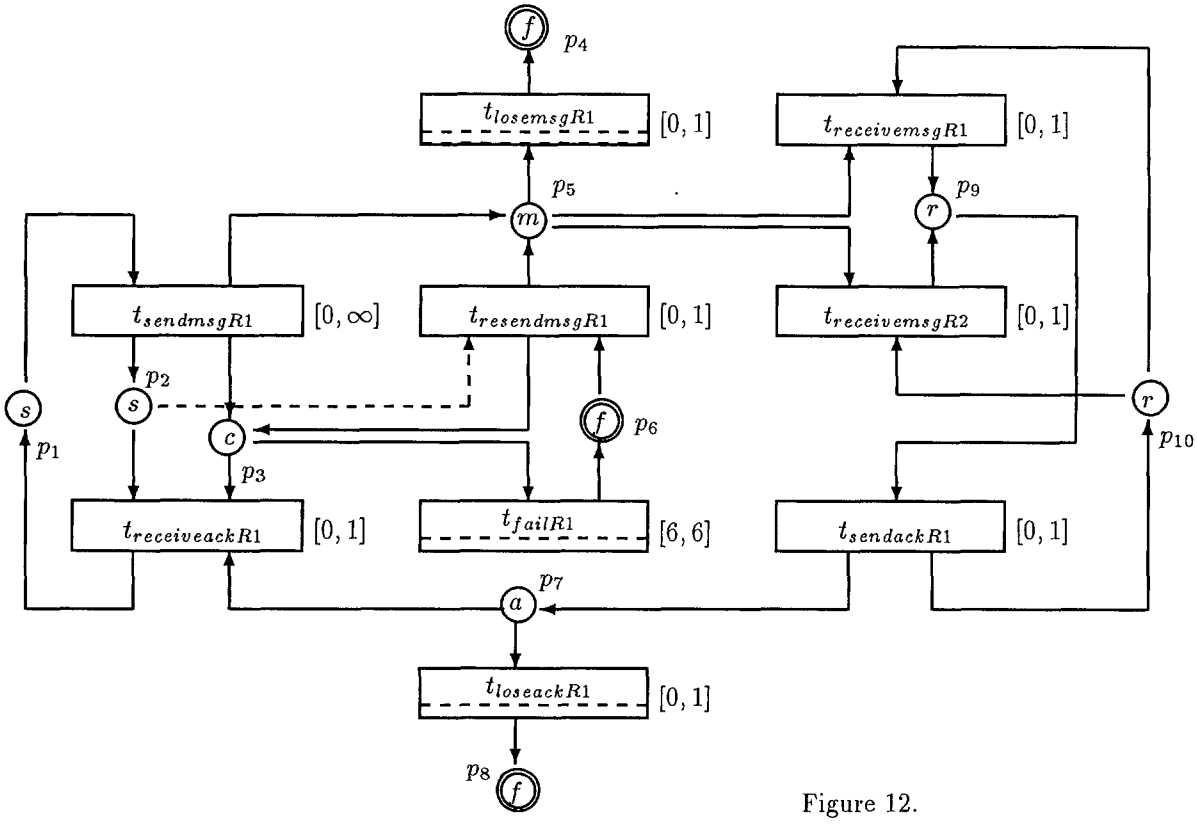


Figure 12.

ons. Figure 8 shows the graphical symbols of the net.

4.2 Modeling of System Behavior

The following steps are used to model overall behavior of our model.

Step 1: For each operation O_i of an object type, create a transition t_{ij} for each execution rule R_j which has the same time interval and pre/post conditions as in the rule. For example, as shown in Figure 9, the send_msg() operation in Figure 7 corresponds to the $t_{sendmsgR1}$ transition. The receive_ack() operation corresponds to transition $t_{receiveackR1}$. All these transitions have the same time interval and pre/post conditions as their corresponding execution rules have.

Step 2: For each transition t_{ij} derived in the last step, create its input/output places, depending upon input/output object types of O_i:

- For each consumed (resp. referenced) object type whose object is referred to in the precondition of t_{ij}, create a consumed (resp. referenced) input place whose containing objects are of the type.

- For each produced object type whose object is produced by some assignment statements of the postcondition of t_{ij}, create a produced output place whose containing objects are of the type.

- For each updated object type whose object is referred to in the precondition of t_{ij}, create a consumed input place and a produced output place where their containing objects are of the type.

Figure 10 shows the input/output places created for each transition.

Step 3: Merge redundant input places. Two places p_1 and p_2 are redundant if (1) their containing objects are of the same type and (2) each one is an input place of a transition where the two transitions were derived in Step 1 from two execution rules of one operation. For our example in Figure 10, the two consumed input places p_{72} and p_{81} are redundant and hence should be merged. Figure 11 results from Figure 10 after merging redundant places.

Step 4: Connect related transitions by merging common places. Two places p_1 and p_2 are common if (1) their containing objects are of the same type, (2) p_1 is an output place of a transition t_1, and p_2 is an input place of a transition t_2, and (3) the objects in p_1 produced by t_1 are useful for satisfying the precondition of t_2, meaning that they can used through p_2 for the firings of t_2. For example, in Figure 11 the message packets in place p_{12} produced by transition $t_{sendmsgR1}$ are useful for satisfying the precondition of transition $t_{receivemsgR1}$ or $t_{receivemsgR2}$ (through place p_{81}), hence p_{12} and p_{81} are common and then merged. Figure 12 results from Figure 11 after merging common places.

Applying these steps, the system behavior of our model can be described in a timed transition net (as shown in Figure 12).

4.3 Construction of Reachability Graph

With the timed transition net, its reachability graph then can be constructed for analyzing desirable properties such as liveness, fairness, and timing properties. Since our timed transition nets use the same timing constructs as in time Petri nets (TPN's) (Merlin 1974, Merlin & Faber 1976), the technique in (Berthomieu & Diaz 1991) for building the reachability graph of a time Petri net can be applied directly for our nets. That is, each node (or *state class* defined in (Berthomieu & Diaz 1991) of the resulting reachability graph is a pair $n = (M, D)$ where M is the marking that contains the current status of objects and D is a set of inequalities that specify the bounds of firing times of the transitions made executable by M (assume all parameters, such as Message and Dest in Figure 1, that must be supplied externally for executing these transitions are provided adequately).

Given any node $n = (M(n), D(n))$, a succeeding node $n' = (M(n'), D(n'))$ can be defined when a transition $l \leq t \leq h$ in $D(n)$ is executed. The edge between them is denoted as $(n, t[l, h], n')$. In node n', $M(n')$ is the marking that contains the new object status resulting from the effects of t on $M(n)$. Transitions in $D(n')$ can be divided into two sets: (1) The transitions that are newly made executable by $M(n')$; and (2) The transitions that were already executable

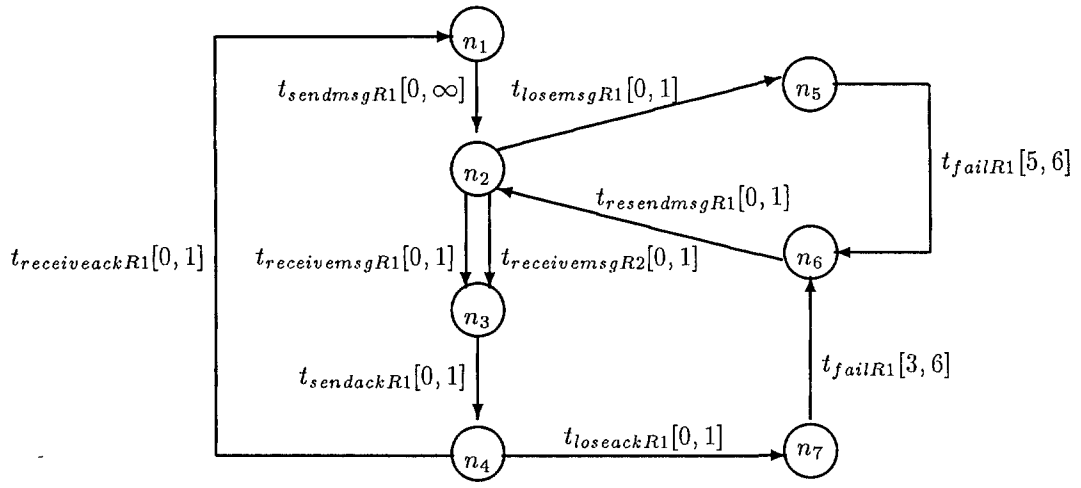


Figure 13: Reachability graph for the timed transition net

in n (already in $D(n)$) are still executable in n'. In the first set, the firing bounds of each transition are equal to the bounds of the time interval associated with the transition. But, in the second set, the firing bounds of each transition t_k are given by: $\max(0,l_k-h) \leq t_k \leq \max(0,h_k-l)$, where l_k and h_k are the firing bounds of t_k in $D(n)$. For more details about state classes and reachability, the reader is referred to (Berthomieu & Menasche 1982, 1983, Berthomieu & Diaz 1991).

Figure 13 shows the resulting reachability graph from the net in Figure 12. In the graph, nodes are labeled with the present state class and arcs represent transitions between state classes with associated bounds of firing times. Since the fault objects in places p_4 and p_8 are produced only to denote losses of messages and acknowledgments, which are not used for retransmitting messages, we do not include them in the markings of state classes for simplifying the graph. The nodes of the graph are as follows.

Node n_1: $M : p_1, p_{10}$, $D : 0 \leq t_{sendmsgR1} \leq \infty$
Node n_2: $M : p_2, p_3, p_5, p_{10}$
$D : 0 \leq t_{losemsgR1} \leq 1$, $0 \leq t_{receivemsgR1} \leq 1$,
$0 \leq t_{receivemsgR2} \leq 1$, $6 \leq t_{failR1} \leq 6$
Node n_3: $M : p_2, p_3, p_9$
$D : 0 \leq t_{sendackR1} \leq 1$, $5 \leq t_{failR1} \leq 6$
Node n_4: $M : p_2, p_3, p_7, p_{10}$
$D : 0 \leq t_{loseackR1} \leq 1$, $0 \leq t_{receiveackR1} \leq 1$,
$4 \leq t_{failR1} \leq 6$
Node n_5: $M : p_2, p_3, p_{10}$, $D : 5 \leq t_{failR1} \leq 6$
Node n_6: $M : p_2, p_6, p_{10}$, $D : 0 \leq t_{resendmsgR1} \leq 1$
Node n_7: $M : p_2, p_3, p_{10}$, $D : 3 \leq t_{failR1} \leq 6$

The graph then can be used as a usual state transition system for verifying properties that characterize the correct behaviors of our model. For instance, we can verify real-time logic assertions (desired time/temporal constraints) by traversing its nodes (state classes). In addition, we can also decide the safety of our model by checking whether in the *control* place p_3 its marking (number of control objects) has only a value 0 or 1 during the execution of the system. Containing more than one object in p_3 is considered as a design error (only one message packet can be transmitted at a time), and hence, must be uncovered during the behavior analysis. Likewise, we might consider the maximum permissible number of fault objects in the place p_4 or p_8 as the maximum allowable number of times of losing messages or acknowledgments. Excessive number of message or acknowledgment losses could mean the extremely unreliable transmission medium, and then, result in the invocation of some necessary actions (i.e., replace the medium with a new one).

5 Related work and Conclusions

Object-oriented conceptual modeling starts with the description of objects which represent as close as possible real-world entities. An object-oriented method must support the encapsulation of the structural and behavioral aspects in an object. In particular, it must provide constructs for modeling of *object types, relationships, object behavior,*

and *interactions between objects.*

Shlaer and Mellor (Shlaer & Mellor 1988) proposed a model that describes an application in an information model, a state model, and a DFD process model. Since these three models are not integrated and the state model and DFD notations are less formal, their work is prone to inconsistencies and ambiguities. In addition, because there is no formal object interaction mechanism, overall system behavior can not be deduced from the behavior of individual objects. Thus, it is impossible to check that the state model is consistent with the process model. These drawbacks also present in other similar approaches such as Booch's (Booch 1991) and Rumbaugh et al's (Rumbaugh et al. 1991).

de Champeaux (de Champeaux 1991, de Champeaux & Olthoff 1989) presented a similar model which augments the state model by attaching to transitions: (1) a *trigger* that indicates whether a triggering event is required; and (2) a *casual list* that describes the events that are generated as a consequence of the transition and act as triggers for subsequent transitions. The process model uses these triggers and messages to describe the casual interactions between objects. As a result, consistency between the state model and the process model can be checked. However, his approach has drawbacks: (1) The structural aspect and the behavioral aspect are not integrated; and (2) The object interaction mechanism is less formal (based on *broadcast* communication), and hence, formal analysis of system behavior is difficult.

Kim and Moon (Kim & Moon 1992) propose a diagrammatical representation, called *Object-Relationship Diagrams*, which provides a uniform model of the structure and the behavior of objects. Object-Relationship diagrams consist of *structure diagrams* and *behavior diagrams.* Structure diagrams describe the structure of objects and their relationships. Behavior diagrams describe the behavior of objects by identifying states, events, and interactions between objects. Since object interaction is described by the same informal way as that in de Champeaux's approach, formal analysis is still difficult.

Hayes and Coleman (Hayes & Coleman 1991) propose a coherent analysis model to capture both the structure and the behavior of objects. Three models are used in this approach. The object structure model describes the static aspect of objects that provides the information to be used in behavioral (dynamic and function) models. In the dynamic model, objectchart (Coleman 1992), which is extended statechart (Harel 1988), is used to describe object behavior. A formal object interaction mechanism is used to describe interactions *one at a time*, so system behavior can be deduced and analyzed from individual object behaviors. In the function model, pre/post conditions are used to describe system level behavior. The consistency between the dynamic model and the function model can be checked.

Our approach presents a different way of modeling both the structural aspect and the behavioral aspect of objects in a uniform representation. Unlike the methods surveyed above that describe object behavior by identifying and specifying all the possible states an object can be in its life cycle, our model supports encapsulation of object states by allowing an analyst to focus on modeling of one operation for one object at a time. The complete behavior (states) of an object can be obtained by putting together and mapping all its operations into a timed transition net. This largely reduces the complexity of our model, and hence, alleviates the difficulty of modeling a nontrivial application. In addition, our modeling constructs can have a graphical representation and a textual representation. The graphical representation makes it easier to communicate with users and makes the conceptual model easier to understand, while the textual representation can be used for other purposes (i.e., compile for syntax and consistency checking). For the requirements of designing for safety and fault tolerance, it also supports modeling of failures to object behavior and their resultant faults. Since system behavior can be easily deduced and modeled in a timed transition net, many existing techniques (Berthomieu & Diaz 1991, Leveson & Stolzy 1987) can be exploited for analyzing desirable properties.

Acknowledgement

We are grateful to Chaoying Chen for her help in preparing the art work of the manuscript.

References

[1] Balzer R. and Goldman N. (1979) Principles

of good software specification and their implications for specification languages. *Proc. of Specifications for Reliable Software,* Cambridge, MA, p. 58-67.

[2] Bartlett K., Scantlebury R. and Wilkinson P. (1969) A note on reliable full-duplex transmission over half-duplex link. *Communications of the ACM,* Vol. 12, No. 5.

[3] Berthomieu B. and Menasche M. (1982) A state enumeration approach for analyzing Time Petri Nets. *Proc. of 3rd European Workshop Applications and Theory of Petri Nets,* Varenna, Italy.

[4] Berthomieu B. and Menasche M. (1983) An enumerative approach for analyzing Time Petri Nets. *Proc. of IFIP Congress 1983,* Paris.

[5] Berthomieu B. and Diaz M. (1991) Modeling and verification of time dependent systems using Time Petri Nets. *IEEE Trans. on Software Engineering,* Vol. 17, No. 3, p. 259-273.

[6] Booch G. (1991) Object-Oriented Design with Applications. Benjamin/Cummings.

[7] Cameron J. (1986) An overview of JSD. *IEEE Trans. on Software Engineering,* Vol. 12, p. 222-240.

[8] Chao J. and Kung C. (1991) Rapid prototyping of conceptual database design on a relational database management system. *Proc. of 10th Int'l Conference on Entity-Relationship Approach,* San Mateo, p. 93-109.

[9] Coleman D., et al. (1992) Introducing Objectcharts or how to use Statecharts in object-oriented design. *IEEE Trans. on Software Engineering,* Vol. 18, no. 1, p. 9-18.

[10] de Champeaux D. (1991) Object-oriented analysis and Top-Down software development. *Proc. of ECOOP,* p. 361-376.

[11] de Champeaux D. and Olthoff W. (1989) Towards an object-oriented analysis method. *Proc. of 7th Annual Pacific Northwest Software Quality Conference,* Portland, OR, p. 323-338.

[12] Harel D. (1988) On visual formalisms. *Communications of the ACM,* Vol. 31, No. 5, p. 514-531.

[13] Hayes F. and Coleman D. (1991) Coherent models for object-oriented analysis. *Proc. of OOPSLA Conference,* p. 171-183.

[14] Hull R. and King R. (1987) Semantic data modeling: survey, applications, and research issues. *ACM Computing Surveys,* Vol. 19, No. 3, p. 201-260.

[15] Jackson M. (1983) System Development. Prentice-Hall, Englewood Cliffs, New Jersey.

[16] Kim Y. and Moon S. (1992) Object-relationship diagrams for object-oriented modeling with concurrency feature. *Microprocessing and Microprogramming,* Vol. 33, North-Holland, p. 207-221.

[17] Kung C. (1989) Conceptual modeling in the context of software development. *IEEE Trans. on Software Engineering,* Vol. 15, No. 10, p. 1176-1187.

[18] Kung C. (1990) Object subclass hierarchy in SQL: a simple approach. *Communications of the ACM,* Vol. 33, No. 7, p. 117-125.

[19] Leveson N. and Stolzy J. (1987) Safety analysis using Petri Nets. *IEEE Trans. on Software Engineering,* Vol. SE-13, No. 3, p. 386-397.

[20] Merlin P. (1974) A study of the recoverability of computer system," Thesis, Dept. of Computer Science, Univ. of California, Irvine.

[21] Merlin P. and Faber D. (1976) Recoverability of communication protocols. *IEEE Trans. on Software Engineering,* Vol. COM-24, No. 9, p. 1036-1043.

[22] Peckham J. and Maryanski F. (1988) Semantic data models. *ACM Computing Surveys,* Vol. 20, No. 3, p. 153-190.

[23] Rumbaugh J., et al. (1991) Object-oriented modeling and design. Prentice-Hall, Englewood Cliffs, New Jersey.

[24] Shlaer S. and Mellor S. (1988) Object-Oriented Systems Analysis. Yourdon Press.

Supporting High Integrity and Behavioural Predictability of Hard Real-Time Systems

M. Colnarič and D. Verber
University of Maribor, Faculty of Technical Sciences
Smetanova 17, Maribor, Slovenia
colnaric@uni-mb.si
AND
W. A. Halang
FernUniversität Hagen, Faculty of Electrical Engineering
D-58084 Hagen, Germany
wolfgang.halang@fernuni-hagen.de

Keywords: hard real-time systems, high-integrity requirements (safety-related systems), exception handling, real-time programming languages, process run-time estimation

Edited by: Marcin Paprzycki and Janusz Zalewski
Received: February 19, 1994 **Revised:** October 30, 1994 **Accepted:** December 19, 1994

The main objective of this paper is to present a method for handling non-preventable and non-avoidable catastrophic exceptions in embedded hard real-time environments in a well-structured and predictable way, and as painlessly as possible.
First, apt hardware and software platforms which are pre-requisite for predictable system behaviour are briefly presented. Then, some existing techniques are shown and their suitability for implementation in embedded hard real-time environments is discussed. Further, a classification of exceptions and our own approach for handling them is presented and elaborated. Finally, a method for the estimation of the resulting temporal behaviour is described.

1 Introduction

In this paper, embedded hard real-time systems are dealt with. In general, they are employed to control different processes; the integrity of these applications relies on their temporally and functionally correct operation. Depending on the application, these systems can be extremely safety critical; their malfunction may cause major damage, material loss, or even endangerment of human lives. Thus, for such systems high integrity and safety is required, and mechanisms must be devised to cope with partial or complete failures.

While in the systems, which are usually used in process control, testing of conformance with functional specifications is well established, temporal circumstances are seldom consistently verified. It is almost never proven at design time that such a system will meet its temporal requirements in every situation that it may encounter.

In his reference paper [20], Stankovic is unmasking several misconceptions in the domain of hard real-time systems. Seemingly the most characteristic one is that real-time computing is often considered fast computing. It is obvious that computer speed itself cannot guarantee that specified timing requirements will be met.

Instead, a different ultimate objective was set: predictability of temporal behaviour. Being able to assure that a process will be serviced within a predefined time frame is of utmost importance. In multiprogramming environments this condition can be expressed as schedulability: the ability to find a schedule such that each task will meet its deadline [22].

For schedulability analysis, execution times of tasks must be known in advance. These, however, can only be determined if a system functions predictably. To assure overall predictability, all system layers must behave predictably

in the temporal sense, from the processor to the system architecture, language, operating system, and exception handling (layer-by-layer predictability, [21]).

In recent years, the domain of real-time systems substantially gained research interest. Certain sub-domains have been examined very thoroughly, such as scheduling and analysis of program execution times. It is typical that most of the research done was dedicated to higher level topics and presumes that the underlying features behave fully predictably.

Exception handling is one of the most severe problems to be solved when a system is to behave predictably. By an exception any unexpected intrusion into the normal program flow which cannot be considered during schedulability analysis phase is meant. It is usually related to residual specification and implementation errors and to failures. Anticipated timing events and events from the environment, which trigger associated processes, do not belong to this category. They should be implemented in a way, which does not cause any non-deterministic delays in the execution of the running tasks. That can be achieved by migrating event recognition and operating system services out of main task processors [11], and was also implemented in the Spring project [19]. Results of our previous studies were presented in [4] and are used in the design of an experimental platform as described in the next section.

When an exception occurs in a program, the latter is inevitably delayed causing a serious problem with respect to the a priori determined execution time. Therefore, exceptions should be prevented by all means, whenever and wherever it is possible [1]. If it is not possible to prevent them to happen, they should be handled in a consistent and safe way in conformity with the hard real-time systems design guidelines, i.e. timeliness, simultaneity, predictability, and dependability [13]. The need for consistent solutions to the exception problem is exacerbated by the fact that exceptions are often the results of some critical systems states, which is when computer control is needed most.

In this paper we show constructively how behavioural predictability can be achieved by presenting an experimental system and considering different aspects of its design. Although the main emphasis of the paper is on consistent exception handling, it is necessary to present some principles used to provide the necessary pre-conditions for deterministic system behaviour; only then it is reasonable to consider the upper layers of a system design. In Section 2 we start off with describing the basic layers of an asymmetrical parallel hardware architecture and the operating system concepts which prevent process control tasks to be disturbed (and thus delayed) by events occurring in the environment. Further, in Section 3, a real-time programming tool supporting the architecture is described by which process control programs with deterministic temporal behaviour can be designed and their run-times determined.

Exception handling was integrated into a high-level programming language, which is the subject of Section 4. First, we classify exceptions and show that a number of them can be either prevented or avoided. Further, we summarise some known solutions to handle the remaining exceptions, which were all combined in the implemented approach. Finally, an analysis of the impact the approach has on overall process timing predictability is given.

2 Concept of an Experimental Hardware Platform

In multi-tasking systems, dynamic scheduling algorithms to generate appropriate schedules must be implemented. The ones which fulfill the requirement that all tasks must meet their deadlines are referred to as feasible. In the literature, several such algorithms have been reported (an overview is given in [13]). For our purpose, the earliest-deadline-first scheduling algorithm is chosen. It has been shown that it is feasible for scheduling tasks on single processor systems; with the throw-forward extension it is also feasible on homogeneous multiprocessor systems. However, this extension leads to more pre-emptions and is more complex and, thus, less practical.

For process control applications, where process interfaces are usually physically hard-wired to sensors and actuators establishing the contact to the environment, it is natural to implement either single processor systems or dedicated multiprocessors acting and being programmed as separate units. Thus, the earliest-deadline-first sche-

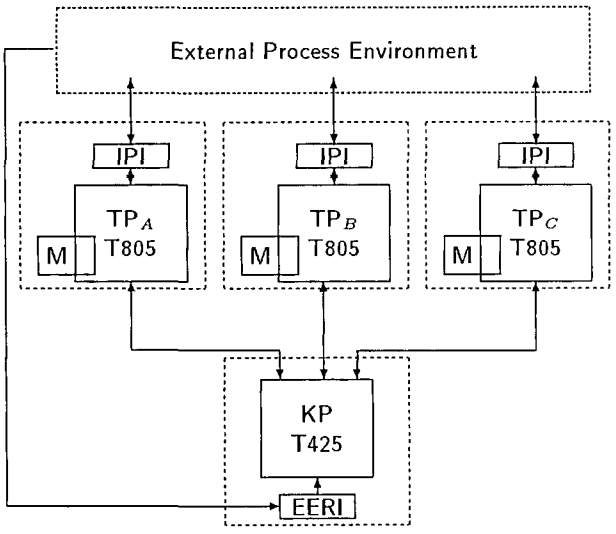


Figure 1: Scheme of an experimental hardware platform

duling policy can be employed without causing any restrictions, resulting in a number of advantages discussed by Halang and Stoyenko [13].

In the classical computer architecture the operating system is running on the same processor(s) as the application software. In response to any occurring event, the context is switched, system services are performed, and scheduling is done. Although it is very likely that the same process will be resumed, a lot of performance is wasted by superfluous overhead. This suggests to employ a second, parallel processor, to carry out the operating system services. Such an asymmetrical architecture turns out to be advantageous, since, by dedicating a special-purpose, multi-layer processor to the real-time operating system kernel, the user task processor(s) are relieved from any administrative overhead.

This concept was in detail elaborated in [11] and further refined in [3, 4]. Our experimental hardware platform is to a high extent complying with these principles, and is currently under construction. In Figure 1 it is shown that it consists of task processors (TPs) with inteligent process interfaces (IPI) and a kernel processor (KP) with an external event recognition interface (EERI), which are fully separated from each other.

The external process is controlled by tasks running in task processors without being interrupted by the operating system functions. Any event from the environment is fed to the kernel processor and scheduling is performed based on the modified earliest-deadline-first policy: the intention is to find a schedule such that all waiting tasks including the newly arrived one meet their deadline while the running task remains in execution. Thus, a running task is only pre-empted if it is necessary to assign the highest priority to an incoming task in order to allow that all tasks meet their deadlines.

The task processors are implemented with INMOS T805 transputers. In the task processors' external memory the code of each task assigned to be run is loaded. Also, a part of the control blocks of these tasks is residing there, holding the context of eventually pre-empted tasks. The fast on-chip RAM of the transputers is holding task internal variables except for the large data structures which are held in the external memory.

The IPI process interface is based on a Motorola MC68000 microprocessor which adds the necessary intelligence to peripheral devices. It is accessible by a bi-directional link via an INMOS converter and is acting as a slave to the task processor(s). Services of the intelligent process interface are available by calling pre-defined peripheral device drivers and providing parameters and data.

Synchronisation of tasks running in different task processors is carried out with the help of semaphores residing in the kernel processor and being accessible through systems calls.

The kernel processor is responsible for all operating system services. It consists of an INMOS T425 transputer performing the operating system kernel services, and a Motorola MC68000 based external event recognition interface. The latter's task is administering the real-time clock in the form of Julian time, receiving signals from the process environment, providing them with time stamps, and periodically triggering events by sending messages to the transputer containing information about all events that happened recently, and serving as a synchronisation means.

The time between two synchronisation messages from the EERI is further sub-divided in slots in the kernel processor. In these slots the information from the external event recognition module is processed, time events are administered, and OS service calls from the task processors are serviced, each triggering scheduling of an associated application task.

It is to be mentioned that our nomenclature

is not strictly conforming with [11], although the functions implemented in our architecture comply with the layers proposed there. The external event administration part of the hardware layer functions is implemented in the EERI, the others — primary and secondary reaction level — in the kernel processor. The hierarchy is retained by executiong these functions in strictly defined slots. We are considering migrating the scheduling-related secondary reaction level services into a separate transputer to enhance performance.

Through this concept, preventing nondeterministic interruptions from the environment, careful avoidance of the sources of unpredictable processor and system behaviour, loose coupling of task processors, and synchronous operation of the kernel processor, the predictability of the temporal system behaviour will provide the necessary basis for the higher system design levels.

3 Concept of a Real-Time Programming Tool

To program applications on the above hardware platform a tool is being constructed, in which the proposed exception handling mechanism is built in. Its ultimate objective is to produce temporally predictable and optimal program code for embedded hard real-time applications, and estimations of their execution times.

In the tool two parts are closely integrated: a compiler for an adapted standard real-time programming language, and a program execution time analyser. The latter is providing the necessary information for a schedulability analyser which is currently beyond the scope of our research.

In the design of the tool, the following guidelines were followed:

1. *Target system independence.* The compiler should produce executable application code for a variety of target systems. This is achievable by implementing target system specific macros which transpose each element of intermediate code into a corresponding piece of executable code. In the specification file for each target system its own set of macros is defined.

2. *Generation of efficient code.* Although being system independent, the compiler is expected to generate fast and compact code. This can be achieved by the simplicity of the programming language, and the possibility of global syntax tree optimisation (register scheme, local and global variables' locations and other implementation specific information are given in the system specification file). Also, in translation macros full information about the operands (constant, register, local or global variables) is contained.

3. *Realistic estimation of task execution times.* A drawback of many methods for task execution time estimation is that they yield such pessimistic results that their relevance is seriously diminished [18]. To cope with that, the tool supports two different methods to determine the execution time of a task: compile-time program analysis and direct measurement of worst-case (partial) task execution time.

3.1 MiniPEARL

To program an application, the programming language miniPEARL is introduced. It is a simplified version of PEARL [9], a standard language for programming real-time applications, which, however, may produce temporally unpredictable code for several reasons. To eliminate these problems, PEARL's syntax is modified. Further, to support efficient mapping onto typical target architectures certain features are reduced. Finally, it is enhanced by some constructs specific to real-time systems, proposed by Halang and Stoyenko [13, 12]. MiniPEARL is described in more detail in [23].

The main differences between PEARL and miniPEARL are:

1. *There are no GOTOs.* The use of GOTO statements can result in unstructured and hardly manageable code. Instead of these, EXIT and LOOP statements are introduced for preliminary exit from an innermost structure, and for immediate initiation of the next iteration of a loop, respectively.

2. *Each loop block is strictly bounded.* In the REPEAT statement, lower and upper counts of a loop are obligatory and defined with compile-time constant expressions to limit the number of iterations.

3. *Pointers and recursion are not allowed.* Dynamic data structures and recursion can result in severe memory management problems. They

may cause temporally non-deterministic actions that cannot be considered in timing analysis.

4.*Signals are not directly supported.* In our architecture model interrupts and signals are managed by the kernel processor. Events can be induced by the synchronisation mechanism.

5.*Each statement execution is temporally bounded.* Commands whose execution time is non-deterministic must be either forbidden, or taken special care of in a real-time systems. Each of such commands which are unavoidable must be temporally guarded, and time-out alternatives must be defined explicitly. Commands that must be guarded are the ones that are dealing with process inputs and outputs (if handshaking is implemented), and synchronisation mechanisms.

6.*Explicitly asserted execution time.* Frequently, because of the nature of a program, estimation may yield very pessimistic execution times. To resolve this problem, additional information about program execution must be given by the programmer. This can be done by adding new constructs (pragmas) into program code as proposed in [18]. But such constructs require complex analysis and are not feasible for all situations. To overcome this problem, the execution time of code segments known through competent measurement, detailed analysis of the program behaviour, experience, reuse, etc. may be explicitly asserted by the system developer who also takes the responsibility. In such case, execution time analysis is overriden. However, to guarantee that the actual execution time will not be longer than declared, blocks must be guarded by time-out controls, and time-out action must be present.

8.*DATIONs are not used.* Mass storage and asynchronous input/output devices as used in PEARL are not suitable for hard real-time systems. For this reason and because of the relative complexity of these structures, DATIONs are excluded from the structure of the language. Input/output devices (registers) are accessed at the lower programming level.

9.*Improved task activation scheme.* In miniPEARL task activation, deactivation, etc. can be done through signals from the environment, time-related conditions, or specific states of synchronisers, as proposed in [13]. A time-related and one non-time-related condition may be combined.

10.*Scheduler support.* The scheduling algorithm performed in the kernel processor relies on the residual execution time of a task. This time is computed as maximum execution time of the task minus cumulative running time. However, the actual execution time is expected to be shorter than the estimated one. To achieve better performance, the actual residual task execution time can be explicitly asserted at ceratin points to update the estimated one.

3.2 Estimation of Task Execution Times

To allow for schedulability analysis, precise execution times of application tasks must be known in advance. In our tool, two methods for the estimation of program run-times are supported:

1. *Analysis of executable code.* In this method, an automatic analyser is used to estimate execution times (compare also [18, 17]). Source code is transformed into an intermediate form (modified syntax tree) prior to executable code generation. Each element of this form is associated with a macro block that is used for two purposes. The first is to generate the code and the second is to obtain its execution time. Because the execution time of the same block can be data-dependent, as much information as possible about operands should be passed to it. The operand can be a register, a constant, a local variable or a global variable. When the macro is expanded, the sum of times needed for accessing these operands is added to the basic execution time of the macro.

2. *Direct measurement of executable code.* This method can be used when more precise execution time than estimated is desired. To achieve that, object code is executed on the target system and the execution time is recorded. Direct implementation of this method has some disadvantages:

- The complete target system must be implemented. That is inappropriate in early phases of development when the target-system is not completely implemented, yet.
- Through recording, only average execution times can be obtained. For usable analysis, however, worst-case execution times are needed. A test scenario to obtain that situation is usually difficult to determine.
- The input/output devices must be active and interact with the environment. Thus, the embed-

ding environment or a simulation of it is needed.

By our approach, these disadvantages are eliminated. Only a task processor or its equivalent must be implemented. The longest path through a task is determined by the compiler and a pilot code is generated running only through that path. From a set of alternative constructs (IF and CASE statements, for example), the longest one is statically routed. All time-guarded commands and input/output variable accesses are replaced by appropriate delays. This pilot code is then executed on the hardware platform or, because of the substitution of every system-specific function, a delay is inserted.

4 Handling of Exceptions in Hard Real-Time Systems

Our previous work in the domain of dealing with exceptions was published in [5]. With the goal to avoid non-deterministic delays in the execution of application tasks it was shown that a great number of exceptions can be either prevented from happening, or they can be handled within the context of task requirements:

– *Preventable exceptions:* Some exceptions can be prevented by restricting the use of potentially dangerous features. Compliance with these restrictions must be checked by the compiler. For example, no dynamic features like recursion, references, virtual addressing, or dynamic file names and other parameters etc. are allowed.

Other features are, e.g., strong type checking (see [8]), or extensions of the input and output data types by two "irregular" values representing "signed infinity" to accommodate overflows and underflows and "undefined", as proposed in the IEEE 32-bit floating point standard [2] implemented also in the INMOS transputers' Floating Point Unit (FPU) (compare also [16]). Thus, computed irregular values do not raise exceptions, but are propagated to the subsequent or higher-level blocks, which must be able to handle them.

– *Non-preventable, anticipated exceptions:* If the potential danger of irregularity can be recognised during design time, it has to be taken care of in the specifications. For example, peripheral devices shall be intelligent, fault-tolerant and self-checking in order to be able to recognise their own malfunctions, and to react in a predefined way if a value which is sent to them is irregular. Further, a number of exceptions resulting from irregular data can be avoided by prophylactic run-time checks before entering critical operations. Many tasking errors are also avoidable by previously using monadic operations to check the system state.

Falling into this category, an obvious and frequently used way of avoiding critical failures in hard real-time systems design is redundancy (an example for consistent implementation of redundancy is the MARS system [15]). Redundant system components must be implemented according to thorough analysis of fault hypotheses.

If there is no way to predict an error, an exceptional situation caused must be handled in order to survive it. These are situations when "the impossible happens" [1], in which programs do not follow their specifications due to hardware failures, residual software errors, or wrong specifications. For example, failure of a part of memory can result in the change of constant values; an error in file management or on a disk is usually unexpected. In safety-critical control systems non-anticipated exceptions may have catastrophic consequences. There it is especially important to implement a mechanism for their safe and consistent handling.

In his early paper, Goodenough [10] presented the idea of assigning default or programmed exception handlers to every potentially dangerous operation. According to the severity of an exception raised the running process was either terminated, or suspended and resumed later. A similar mechanism although considerably more elaborate and adapted for use in hard real-time systems was implemented in Real-Time Euclid [14]. There, exception handlers were (optionally) located within block constructs and were executed in the case of an exception. If there were no exceptions the handlers had no effect except for their impact on a block's execution time estimated by a schedulability analyser, thus making it more difficult to be scheduled. Exceptions may be raised by kill, terminate or except statements, to terminate a process entirely or only its frame, or to execute the handler without termination of the process, respectively.

A reference study in the domain of non-

preventable exceptions was done by Cristian [6, 7]. Certain principles from this work were further detailed in [1] and were also adopted in our exception handling mechanism.

According to Cristian, exceptional situations can be handled (a) by programmed exception handling and (b) by default exception handling based on automatic backward recovery using recovery blocks. Since in embedded hard real-time systems programmed exception handling should be included in the system requirements and can, thus, be treated as normal actions, in the following the alternative technique will be dealt with briefly.

The principle of *backward recovery* is to return to the previous consistent system state after an inconsistency is detected by consistency tests called post-conditions. It can be done in two ways, (a) by the operating system recording the current context before the program is "run" and restoring it after its unsuccessful termination, or (b) by recovery blocks inside the context of a task whose syntax is as follows:

RB ≡ **ensure** *post* **by** P_0 **else by** P_1 **else by** ...
else *failure*

where P_0, P_1, etc. are alternatives which are tried consecutively until either consistency is ensured by meeting the *post*-condition, or the *failure* is executed. Each alternative should be independently capable to ensure consistent results.

In the *forward error recovery technique* it is tried to obtain a consistent state from partly inconsistent data. Which data are usable can be determined by consistency tests, error presumptions, or with the help of independent external sources.

To handle catastrophes we propose a combination of pre-conditions, post-conditions and modified recovery blocks implementing both backward and forward recovery. Its syntax is shown in Figure 2.

A block (plain block structure, task, procedure, loop, or other block structure) consists of alternative sequences of statements. Each alternative can have its own pre- and/or post-conditions, represented by Boolean expressions. When the program flow enters a surrounding block, the state variables, that are modifiable by alternatives which might fail, are stacked (see below).

```
block ::= block_begin block_tail

block_begin ::= BEGIN
              | PROCEDURE parameters & attributes;
              | TASK parameters & attributes;
              | parameters REPEAT

block_tail ::=[declaration_sequence]
              [alternative_sequence] END;

declaration_sequence ::= block-specific declarations
    [PRESERVE global_var_list]
alternative_sequence ::=
   {[ALTERNATIVE [PRE bool--exp;] [POST
bool--exp;]]
   [statement_sequence]}
```

Figure 2: Syntax of an exception handling mechanism

Then, the first alternative statement sequence, whose pre-condition (if it exists) is fulfilled, is executed. At the end, its post-condition is checked, and if this is also fulfilled, execution of the block is successfully terminated. If the post-condition is not fulfilled, the next alternative is checked for its pre-condition and eventually executed. If necessary, values of the state variables recorded at the beginning of the block are first restored.

If an alternative fails, any effect on the system state should be discarded; thus, it is necessary that the original value of any variable is restored, which was modified and lies outside of the scope of the failed alternative. For that purpose, the state of any such variable must be stacked at the time of entering the block. Whether and which variables must be stacked can be determined by the compiler. It is only necessary to restore non-local variables that appear on the left hand side of an assignment in alternatives which have post-conditions, since only they may fail after modifying the state. It is a task of the compiler to scan the block for such variables and take care of their stacking. Further, after a non-successful evaluation of a post-condition, only the variables that were modified in this alternative are automatically restored.

Stacking all global variables that can be modified within a block may require a relatively large amount of time. There are situations where the value of a global variable is not needed any more after an unsuccessful termination of an alterna-

tive. In such situations the application programmer may wish to declare which modifiable global variables should be restored after the unsuccessful alternative. This can be requested by the optional PRESERVE declaration in the declaration_sequence. If this declaration is present, the automatic search for modifiable global variables is prevented; hence, the explicitly given list must contain the complete set. The compiler then scans for global variables that are both in the list and appear on a left hand side in the alternative program, and restores their original values after an unsuccessful try.

A good technique which prevents the above problem is to work with private copies of global state variables inside the alternatives that may cause backward recovery, and to export their values after a successful post-condition check. However, this is more time-consuming, especially when there are more such alternatives in a block, which require (counter-productive) transfer of global into local variables and back.

Since embedded hard real-time systems, which are the main subject of this paper, are, as a rule, used in process control a severe problem arises if there are any actions triggered like commencing a peripheral process which causes an irreversible change of initial state inside an alternative that failed. In this case, backward recovery is generally not possible. As a consequence, it is our suggestion that no physical control outputs should be generated inside the alternatives which may cause backward recovery in case of failure, i.e., inside those which have post-conditions. In this case only forward recovery is possible, bringing the system to a certain predefined, safe, and stable state.

Both forward and backward recovery methods can be implemented using the proposed syntax. In the following these approaches will be shown:

- **Backward recovery:** bearing in mind the dangers of backward recovery in process control systems, it may be (carefully) implemented. Backward recovery can be recognised by the post-conditions an alternative must meet. Its functioning is obvious: if an alternative fails to meet its post-condition, the next alternative fulfilling its pre-condition is used to do the task of the block. Thus, it is necessary to restore the system state variables possibly modified in previous unsuccessful alternatives.

- **Forward recovery:** this technique may be somewhat less obvious. Consider the case where an alternative is checking the success of its operation, according to its design specifications. According to the results of the check, different actions may be taken to resolve different situations. To control the program flow, this alternative then, according to the outcome of these checks, sets some states with which the pre-conditions of alternatives in the subsequent block are set. There may be an alternative with empty statement_sequence whose pre-condition is met if a previous alternative was successful; by this example, classical exception handling can be implemented.

The alternatives should contain independently designed and coded programs to comply with specifications and to eliminate possible implementation problems or residual software errors. They can contain alternative design solutions or redundant resources, when problems are expected. A further possibility is to assert less restrictive pre- and/or post-conditions and to degrade performance gracefully. By the means presented in [23] it is also possible to bound the execution times of alternatives. If one of them fails to complete inside a predefined period, a less demanding alternative is taken.

If there is no alternative, whose pre- and post-conditions are fulfilled, the block execution was unsuccessful. If the block was nested inside an alternative on the next higher level, this alternative fails as well and the control is given to the next one, thus providing a chance to resolve the problem in a different way. On the highest level, the last alternative must not have any pre- or post-conditions. It must solve the problem by applying some conventional actions like employing fault-tolerance measures or performing smooth power-down. Since the system is then in an extreme and unrecoverable catastrophic condition, different control and timing policies are put in action, requesting safe termination of the process and possibly post-mortem diagnostics.

Using this exception handling mechanism the worst-case program execution times required for

schedulability analysis can be estimated at compile time. In the following paragraphs three different cases will be considered.

(a) Exclusive backward recovery (all alternatives have post-conditions): in the worst-case execution time estimation all times must be considered, i.e., time for stacking all global variables' contents, for evaluating all pre- and post-conditions, alternative program execution times, and times to restore used variables.

$$t_{wc} = t_{st} + \sum_{i=1}^{n} t_{pre_i} + t_{body_i} + t_{post_i} + t_{rest_i}$$

where

i	number of alternatives in the block
t_{wc}	worst-case block execution time
t_{st}	time to store global variables
t_{pre_i}	i-th alternative pre-condition evaluation time
t_{body_i}	i-th alternative program execution time
t_{post_i}	i-th alternative post-condition evaluation time
t_{rest_i}	time to restore global variables in i-th alternative

(b) No backward recovery (no alternatives have post- conditions): in this case it must be scanned for the maximum time composed of an alternative body execution time plus the sum of non-successful pre-condition evaluation times of all preceeding alternatives; there is no stacking or restoring of variables.

$$t_{wc} = max_{k=1,n}\left(t_{body_k} + \sum_{i=1}^{k} t_{pre_i}\right)$$

(c) Mixed alternatives with and without post-conditions: in this case, estimation of the worst-case execution time is slightly more complicated. During operation, alternatives are tried one after another according to their sequence in the block. Thus, execution times are evaluated as follows:

- the execution time of the sequence of alternatives with post-conditions is calculated as in case (a) and is added to the execution time of the body of the subsequent alternative without post-condition if it exists, or forms a virtual alternative without pre- or post-condition if it is at the end of the block.
- afterwards, one proceeds as in case (b).

Actually, the last method (c) is generally valid and also applicable in both previous cases.

Especially the backward recovery method inevitably yields pessimistic execution time estimations. However, this is not due to this specific solution. In safety-critical hard real-time systems it is necessary to consider worst-case execution times, which must also include exceptional conditions. Depending on the performance reserve of a system, more or less alternatives may be provided, performing more or less degraded functions. In extremely time-critical systems just a single alternative in the highest level block may be implemented only performing a safe and smooth power-down.

To cope with the problem of the pessimism of run-time estimation of execution of alternatives some further solutions are possible. Each subsequent alternative of a set of backward recovery alternatives may be bounded to half of the execution time of the previous one; thus, the block will terminate in at most twice the execution time of the primary alternative. Also, from a failure of an alternative it is possible to deduce which subsequent alternatives in subsequent blocks are reasonable and which are not, and to set their pre-conditions accordingly. However, this requires a sophisticated run-time analyser.

5 Conclusion

In order to assure a predictable behaviour of real-time systems, it is necessary to determine a priori bounds for the task execution times. In this paper a consistent design of a computing system for embedded applications operating in the domain of hard real-time is described. While the experimental hardware platform and program development tool are only outlined, the exception handling mechanism, which represents the most severe obstacle to overall predictability, is dealt with in more detail. Catastrophic exceptions are coped with in a well-structured environment by providing sequences of gradually more and more evasive software reactions.

Embedded hard real-time systems for process control often operate in safety critical enviro-

nments. Uncontrolled malfunctions can have drastic consequences with regard to repair costs, production loss, or even endangerment of human health or lives. By our approach, overall system safety is greatly enhanced. Possible system failures are already being considered during the design phase, and alternative solutions are devised and prepared. Having to use them, performance may be reduced, but safety is retained, since they will either solve the problem, or bring the system into some controlled and safe state. These alternative measures either employ software approaches or redundant hardware means, or are gradually less complex and, thus, less sensitive to disturbances and failures. Therefore, they rely on very simple fault-tolerance measures, employing minimum resources. They may even employ electrical or mechanical means, such as safe passive state of inactive relays or automatic activation of mechanical brakes when the system loses control, etc.

Applications designed this way fulfill the requirements of hard real-time systems, viz., timeliness, simultaneity, predictability, and dependability. Although the worst-case analysis necessarily introduces pessimism in run-time estimation, the proposed methodology is practically usable for the development of safety critical embedded hard real-time applications if the alternative solutions to the critical parts of control tasks are designed reasonably.

References

[1] Andrew P. Black. Exception handling: The case against. Technical Report TR 82-01-02, Department Of Computer Science, University of Washington, May 1983. (originally submitted as a PhD thesis, University of Oxford, January 1982).

[2] W.J. Cody, J.T. Coonen, D.M. Gay, K. Hanson, D. Hough, W. Kahan, R. Karpinski, J. Palmer, F.N. Bis, and D. Stevenson. A proposed radix- and word-length-independent standard for floating-point arithmetic. *IEEE Micro*, 4(4):86–100, August 1984.

[3] Matjaž Colnarič. *Predictability of Temporal Behaviour of Hard Real-Time Systems.* PhD thesis, University of Maribor, June 1992.

[4] Matjaž Colnarič and Wolfgang A. Halang. Architectural support for predictability in hard real-time systems. *Control Engineering Practice*, 1(1):51–59, February 1993. ISSN 0967–0661.

[5] Matjaž Colnarič and Wolfgang A. Halang. Exception handling and predictability in hard real-time systems. In *Proceedings of the 12th International Conference on Computer Safety, Reliability and Security SAFECOMP '93*, pages 371–378, Poznań – Kiekrz, Poland, October 1993. Springer–Verlag, London, 1993.

[6] Flaviu Cristian. Exception handling and software fault tolerance. *IEEE Transactions on Computers*, 31(6):531–540, June 1982.

[7] Flaviu Cristian. Correct and robust programs. *IEEE Transactions on Software Engineering*, 10(2):163–174, March 1984.

[8] Ian F. Currie. NewSpeak: a reliable programming language. In *High-integrity Software*, pages 122–158. Pitman Publishing, London, 1988.

[9] *DIN 66 253: Programmiersprache PEARL, Teil 1 Basic PEARL.* Berlin, 1981.

[10] John. B. Goodenough. Exception handling: Issues and a proposed notation. *Communication of the ACM*, 18(12):683–696, 1975.

[11] Wolfgang A. Halang. Definition of an auxiliary processor dedicated to real–time operating system kernels. Technical Report UILU-ENG-88-2228 CSG-87, University of Illinois at Urbana Champaign, 1988.

[12] Wolfgang A. Halang and Alexander D. Stoyenko. Comparative evaluation of high–level real–time programming languages. *Real-Time Systems*, 2(4):365–382, 1990.

[13] Wolfgang. A. Halang and Alexander D. Stoyenko. *Constructing Predictable Real Time Systems.* Kluwer Academic Publishers, Boston–Dordrecht–London, 1991.

[14] Eugene Kligerman and Alexander Stoyenko. Real-time Euclid: A language for reliable real-time systems. *IEEE Transactions on*

Software Engineering, 12(9):941–949, September 1986.

[15] Hermann Kopetz, A. Damm, Ch. Koza, M. Mulazzani, W. Schwabl, Ch. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The MARS approach. *IEEE Micro*, 9(1):25–40, February 1989.

[16] Barbara H. Liskov and Alan Snyder. Exception handling in CLU. *IEEE Transactions on Software Engineering*, 5(6):546–558, November 1979.

[17] Chang Yun Park. Predicting program execution times by analyzing static and dynamic program paths. *Real-Time Systems*, 5(1):31–62, March 1993.

[18] Peter Puschner and Christian Koza. Calculating the maximum execution time of real-time programs. *Real-Time Systems*, 1(2):159–176, 1989.

[19] Krithi Ramamritham and John A. Stankovic. Overview of the SPRING project. *Real-Time Systems Newsletter*, 5(1):79–87, Winter 1989.

[20] John A. Stankovic. Misconceptions about real–time computing. *IEEE Computer*, 21(10):10–19, October 1988.

[21] John A. Stankovic and Krithi Ramamritham. Editorial: What is predictability for real–time systems. *Real-Time Systems*, 2(4):246–254, November 1990.

[22] Alexander Stoyenko. *A Real-Time Language With A Schedulability Analyzer*. PhD thesis, University of Toronto, December 1987.

[23] Domen Verber and Matjaž Colnarič. A tool for estimation of real-time process execution times. In *Proceedings of Software Engineering for Real-Time Applications Workshop*, pages 166–171, Cirencester, September 1993. IEE.

A Novel Approach to Off-line Scheduling in Real-Time Systems

Guohui Yu and Lonnie R. Welch
Department of Computer and Information Science
New Jersey Institute of Technology
University Heights
Newark, NJ 07102
Email: gray@earth.njit.edu, welch@vienna.njit.edu

Keywords: real-time, multiprocessor, off-line scheduling, ADTs, replication, concurrency

Edited by: Marcin Paprzycki and Janusz Zalewski
Received: February 25, 1994 **Revised:** October 31, 1994 **Accepted:** December 26, 1994

This paper presents a new off-line scheduling approach for object-based real-time systems using a periodic model of execution. The approach differs from previous off-line scheduling work in that it constructs a feasible schedule by exploiting concurrency. Concurrency is achieved by replication of abstract data type (ADT) module instances to reduce contention caused by multiple accesses to shared ADTs. Concurrency is also achieved by asynchronous remote procedure calls (ARPCs), which allow caller and callee to execute concurrently. By enhancing concurrency, the execution times of processes are reduced, and the chance for finding a feasible schedule is significantly increased.

1 Introduction

In hard real-time systems, the most important goal is to guarantee that all timing constraints are satisfied. Since scheduling problems are NP-hard in multiprocessor systems [28], no algorithm exists to solve the problems efficiently. On-line scheduling approaches are not typically sufficient to guarantee timeliness, due to the limited amount of time for scheduling and the overhead of optimal scheduling. However, on-line scheduling techniques [16, 19, 22, 42, 23] are necessary in applications that have unpredictable environments. For fully predictable, or almost fully predictable, environments, off-line scheduling techniques are used to guarantee timeliness. Contention for shared resources (processors, devices, and communications) is avoided by constructing schedules before runtime for each shared resource. If there exist a few unpredictable factors, several schedules are constructed according to the factors. At run-time, one of the schedules is chosen. Off-line scheduling is being used successfully in many application areas including factory automation, telecommunication, aerospace, and robotics [29, 7, 32].

Traditional off-line scheduling approaches try all possible permutations of the *scheduling objects* (processes, tasks, segments of processes, etc.) to seek a feasible solution. The timing behavior of scheduling objects is unchanged during scheduling, thus all effort is devoted to optimizing the search path for finding feasible schedules [21, 37, 30, 14, 17, 24, 2]. This paper presents a new approach for constructing off-line schedules for applications composed of abstract data type (ADT) modules using a periodic model of execution. The execution time of scheduling objects is reduced by enhancing concurrency. Given an initial schedule and a list of processes missing deadlines, the scheduler identifies a list of candidate opportunities to enhance concurrency within processes missing deadlines. Candidates are evaluated by analyzing effects on the entire schedule, the utilization and availability of demanded resources, and the amount of concurrency produced. The best candidate (possibly a combination of several candidates) is chosen to improve the assignment and schedule. The chance of finding a feasible schedule is significantly increased by concurrency enhancement.

1.1 ADT Modules

This paper considers concurrent real-time systems built from reusable components which are constructed from ADT modules. ADTs, which are supported by languages such as Ada, Clu and Modula-2, provide a way to limit the complexity of the interface between data structures and associated methods, and programs that use the data structures and methods. ADTs also provide a mechanism for information hiding and encapsulation, which are desired properties for reusable software components [34, 13, 25, 33, 39]. In distributed systems terminology, ADT instances are servers that provide services, and the programs built on top of ADTs are clients that request the services.

Applications constructed with ADTs tend to have many method calls, which may cause inefficiencies at execution time. Asynchronous remote procedure call (ARPC) [15, 36] is used to allow the caller to continue execution until it requires a *busy parameter*. With the ARPC model [36, 31], there are three factors that may prevent two method calls from running concurrently. They are (1) control dependence, which describes the required sequence of execution; (2) data dependence, which describes the exclusive use of a data object; and (3) code (ADT instance) dependence, which dictates exclusive access to the code of a module instance. Control dependence can be resolved by techniques such as concurrent speculative execution of multiple branches of if-statements and case-statements, or concurrent execution of loop iterations. Data dependence can often be resolved by data replication techniques and variable renaming. Instance dependence can be resolved by instance replication (cloning) techniques. This paper focuses on how cloning is used for constructing feasible schedules.

While cloning increases concurrency, it may increase CPU and network contention, and may also increase synchronization costs. These overheads vary for different kinds of ADTs. For cloning stateless ADTs, there is no need to maintain consistency of data. For cloning ADTs with state, additional effort on maintaining data consistency is needed. Therefore, the balance between the increased concurrency and the synchronization overhead is an important issue. In this paper, only stateless ADTs are considered for cloning. In applications constructed by abstract data types, typically, an ADT instance is used to manage more than one data object. It is the client's responsibility for maintaining the consistency of data. Therefore, most ADT module instances are stateless. That is one of the design principles for developing ADT modules [13]. We make the ADT module as general as possible (generic ADT modules). Applications built with ADT module instances typically have many stateless ADT instances [13, 34]. Note that cloning of ADT instances allows granularity of concurrency at the method level, not at the statement level. Therefore, communication costs are typically much smaller than the execution times of methods, especially since only pointers to data structures need to be passed in most remote procedure calls [36]. Since instance cloning introduces overheads to the systems, we do not want to place a clone on every processor. Several clients may share an ADT instance without any contention if they call the instance at different times. To determine the minimum number of clones of an ADT instance needed to resolve all possible contention in a given application, dependence relations among statements, method calls, and ADT instances must be analyzed [41].

1.2 Previous Work

The cloning considered here is the replication of aggregate code components (not data or hardware components, which are considered in data management systems [12] and hardware design). Previous work on cloning has mainly concentrated on compiler optimization and fault tolerance. Keith Cooper [5] uses cloning techniques for compiler optimization. Clones of procedures are used to inherit an environment that allows for better code optimization. Procedure or task cloning for fault tolerance is discussed in [21, 3, 4], where clones of procedures or tasks are used to obtain high availability. Cloning of ADTs for concurrency was first addressed in [35]. The contention for an ADT is revealed by partitioning the statements of an ADT module into *units*, where an unit is a sequence of statements that must be executed in order, due to data dependence. The approximate upper bound on the number of clones of a module instance that can be used concurrently is determined by a polynomial-time algorithm. This

work is different from [35] in the following aspects:

1. the program dependence graph (PDG) [8, 6, 1, 11] is extended to represent instance dependence;
2. the new PDG is used to perform dependence and cloning analysis at three levels (intra-method, inter-method, and inter-instance);
3. a more accurate upper bound than [35] on the number of clones of each ADT module instance that can be used concurrently is determined;
4. the new PDG is used to trace the incremental changes of contention of ADT module instances during cloning;
5. cloning of ADT module instances is used for schedule construction.

Previous work on assignment and scheduling using cloning of ADT module instances for enhancing concurrency has not been observed.

Much of the previous off-line scheduling algorithms deal with only one or two processor scheduling problems [9, 18, 38]. Multiprocessor scheduling problems are addressed in work [17, 24, 2], but the scheduling problem is simplified by various assumptions, such as same release times, same deadlines, no precedence relations among processes, identical computation times, and others. General models are used in [30, 21, 37], where arbitrary release times, arbitrary deadlines, arbitrary computation times, and precedence relations among processes are considered in multiprocessor systems. This work uses a more general model [31, 30], which uses processes as assignment and scheduling units. Concurrency exists only at the process level. In this work, ADT module instances are used as assignment and scheduling units. Concurrency is exploited at the method level and is enhanced in processes missing deadlines. The framework of our scheduling approach was first introduced in [40], the work is elaborated and details of the approach are presented in this paper.

In the following sections, the programming and execution paradigms are introduced. This is followed by discussion of our scheduling approach and concurrency enhancement. Finally, we discuss the contributions of this work, and identify future work.

2 Programming Paradigm

The programming paradigm used in this work supports object-based programming. The constructs of the programming paradigm include *module classes* (such as Ada packages [20] and C++ classes), *activity classes* (such as Ada tasks and DEDOS activities [10]), and *applications*. In this section, a brief introduction to this language model is given. An example of an application program which is a portion of a factory simulation application is used to illustrate the programming model. Complete details of the model can be found in [31].

2.1 Module Classes

An abstract data type (ADT) is defined as a module class template, the basic reusable software component. A typical ADT exports a type that can be used to declare variables, and a set of operations used to manipulate the variables. In other words, variables declared to have the type exported by an ADT instance can be accessed only through the operations exported by the instance. ADTs can be defined as generic templates (i.e., they can be parameterized by types and operations) to increase their reusability. Templates must be instantiated with actual parameters before being used. To make the programs analysible, no aliased variables or goto statements are allowed, unbounded loops and unbounded recursion are forbidden (to enable timing analysis), the types of variables are determined statically, and instantiation of classes must be done statically. Those restrictions are acceptable for real-time systems since nondeterministic behavior cannot guarantee timeliness of real-time applications [26]. Figure 1 (a) shows a module class *Queue* which is declared as a generic ADT module. The *Queue* takes a parameter T which is a type used to define the elements of the queue. When different types are provided, different queues can be instantiated from this ADT module. The ADT module *Queue* provides a type *QueueType* which can be used to declare queue variables such as *InQueue* and *OutQueue* in ADT module *Machine* (shown in Figure 1 (b)). Two methods *insert* and *remove* are provided by the *Queue* which is used to manipulate variables of *QueueType*.

```
module class Queue(type T);
  provided type QueueType;
  instance ···
  method insert(var Q:QueueType;var x:T);
    begin
      ··· access Q and x for 5 time units
    end method insert;
  method remove(var Q:QueueType; var x:T);
    begin
      ··· access Q and x for 5 time units
    end method remove;
end module class Queue;
            (a)

module class Machine;
  provided type MachineType;
  instance part is Part(···);
  instance product is Product(···);
  instance Q is Queue(part.PartType);
  var inQueue, outQueue : Q.QueueType;
  var part : part.PartType;
  var prod : product.ProductType;
  method perform_next(var M:MachineType);
    begin
      ··· instructions for 2 time units
      Q.remove(inQueue,part);
      ··· access M, part, and prod for 3 time units
      Q.insert(outQueue,prod);
      ··· instructions for 2 time units
    end method perform_next;
end module class Machine;
            (b)

activity class Task;
  instance M is Machine;
  var M1, M2 : M.MachineType;
  begin
    M.perform_next(M1);
    ···
end activity class Task;
            (c)

application factory;
  process Task1 is Task
    with period=30, deadline=12,release-time=0;
  process Task2 is Task
    with period=60, deadline=21,release-time=0;
end application factory;
            (d)
```

Figure 1: Factory Simulation

2.2 Activity Classes

An *activity class* can be used to define a process template, the basic thread of execution. Each activity class is a sequence of statements and procedure calls to methods provided by instances of module classes. An activity class is used for instantiating processes. A process can be created as a normal process or an periodic process. The timing constraints of a periodic process (such as frame, deadline, and release time) are parameters given when processes are instantiated. Thus, using the same activity class with different parameters, different processes can be instantiated. To increase reusability, an activity class can also have types and/or operations as parameters. An instance created in one activity class can be used by operations in another activity class by using *export* and *import* mechanisms. In Figure 1 (c), activity class *Task* is defined. An ADT instance M is created from module class *Machine*, and is used to declare two machines $M1$ and $M2$. Figure 1 (c) shows only one statement of the *Task* which is a call to method *perform_next*() provided by the ADT instance M.

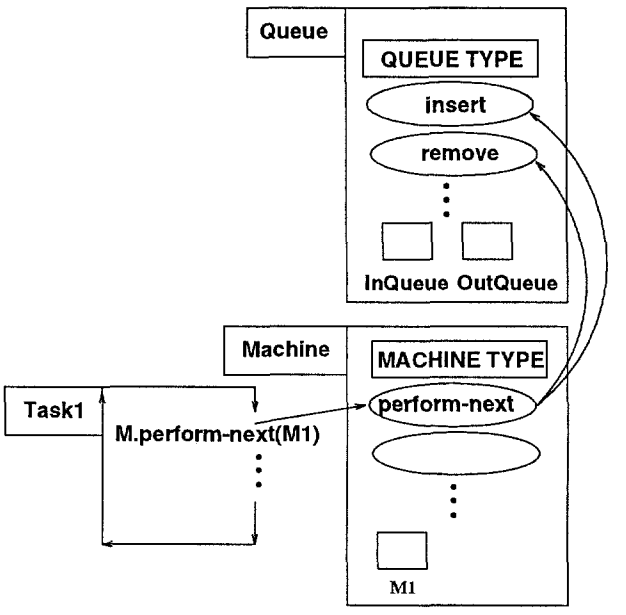


Figure 2: Instance relations created in factory application

2.3 Applications

An application is defined by instantiating global ADT instances (i.e., module class instances) and processes (i.e., activity class instances), and by describing timing relations among processes. In an application definition, activity classes are instantiated as processes with timing constraints and other actual parameters. Timing constraints of processes can be expressed by either *absolute timing constraints (ATCs)* or *relative timing constraints (RTCs)*. An ATC is a time or time period which is imposed on a process to relate the timing behavior of the process with the system time. Frames (periods), deadlines, and release-times of processes are described with ATCs. An RTC is a time or time period which is imposed on a process to relate the timing behavior of the process with another process. Precedence relations and exclusive access relations between processes can be described with RTCs. Figure 1 (d) shows a portion of a factory simulation which is defined by instantiating two processes, Task1 and Task2. Figure 2 shows a few call relations (edges) among three instances in the application.

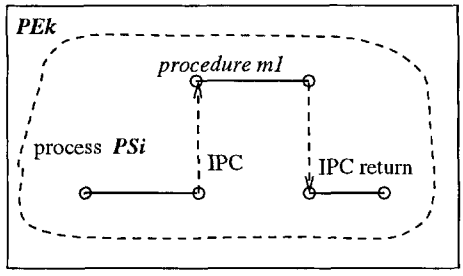


Figure 3: Internal Procedure Calls

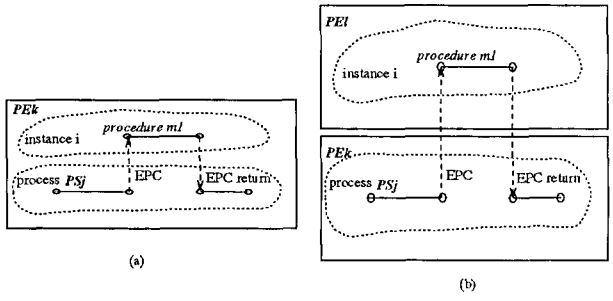


Figure 4: External Procedure Calls

3 Execution Paradigm

The execution platform used in this paper is a distributed memory MIMD system which is described in [36, 31]. Each processing element (PE) consists of a CPU, a communication co-processor and local memory. PEs are connected by either buses and/or high-speed, bidirectional point-to-point links. We assume that there is a physical route of buses and links between any pair of PEs.

Communication is handled by the communication co-processor in either a synchronous or in an asynchronous manner. Device resources are either *physical devices* or *logical devices*. Physical devices are hardware devices managed by software packages such as disks, sensors, and monitors. Logical devices are ADT module instances.

As shown in Figure 3, *execution graphs* are used to describe the executions of processes. In an execution graph, solid lines show executions of process segments, and dashed directed lines indicate calls and returns. The solid boxes indicate PEs, and the dashed ovals enclose operations of instances. EST stands for earliest start time, FT for finish time, and D for deadline of a process.

An inter-instance call is an *internal procedure call* (IPC) if the calling operation is provided by an instance created within the activity class (as in Figure 3). For example, call

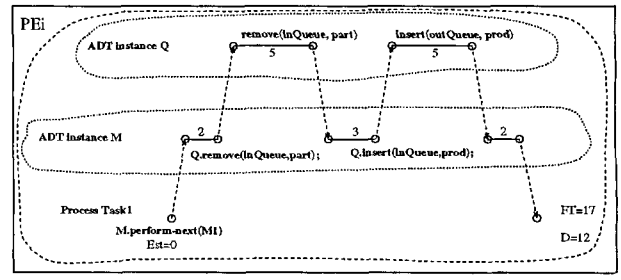


Figure 5: Assignment of Task1 and ADT instances in the factory application

$M.perform\_next(M1)$ in activity class $Task$ is an IPC since method $perform\_next()$ is provided by M which is instantiated within the activity class. An inter-instance call is an *external procedure call* (EPC) if the calling operation is provided by an instance created outside the activity class (as shown in Figure 4). Based on the physical location, IPCs and EPCs are implemented with *local procedure calls (LPCs)* and *remote procedure calls (RPCs)*. An LPC occurs when the caller and the callee are on the same PE. An RPC occurs when the caller and callee are on two different PEs (as shown in Figure 4 (b)). An IPC is implemented by using an LPC which is simply a local context switch. An EPC is implemented by an LPC if the called operation is on the same PE (as shown in Figure 4 (a)), or by an RPC if the called operation is on a different PE (as in Figure 4 (b)). Processes are distributed among the PEs. A process and all of its ADT module instances are initially assigned to the same PE (as shown in Figure 5 for Task1 in the factory simulation). Concurrency can be gained among processes if they are running on different PEs. RPCs can be *synchronous remote procedure calls (SRPCs)* or *asynchronous remote procedure calls (ARPCs)*. With SRPCs, the caller is blocked to wait for the call to return. With ARPCs, concurrency is obtained by allowing the caller to continue execution until it requires a busy parameter. When the required parameter is returned, the caller resumes execution. Therefore, not only calls in different processes, but also calls in the same process, run concurrently.

To control the complexity of scheduling, a hybrid scheduling model is used. Full preemption indicates that the execution of a process can be interrupted by other processes (with higher priorities) at any time. Nonpreemption describes a

scheduling approach wherein a process must run to completion once it starts. Full preemption gives the scheduler more flexibility to find a feasible schedule, but it increases the steps of scheduling because more cases are considered, and also may cause too much overhead (context switch) at run time. On the other hand, nonpreemption simplifies the scheduling job, but reduces the flexibility (chances to find a feasible schedule). To balance the flexibility and the complexity, we use a *semi-preemption* model [31] that restricts preemption to points called *preemption points*. When statements are used as scheduling units, there is maximal flexibility for scheduler and maximal overhead. Statements are grouped into segments called *beads* [31]. A bead is a group of statements that have to be executed together without preemption, i.e., a bead is a nonpreemptable scheduling and execution unit. There are the following types of preemption points:

1. an EPC or an EPC return;
2. the beginning and ending of a blocking device access; and
3. the beginning and ending of the sending phase of an inter-process communication.

Semi-preemption is a balance between the flexibility and the complexity of traditional scheduling approaches. More details about this execution model can be found in [31].

4 The Off-line Scheduling Approach

The job of scheduling is to decide the start times of scheduling objects so that their timing constraints are satisfied. As mentioned previously, the general scheduling problem on multiprocessors is NP-hard. Optimal solutions [38, 2, 24, 17] are not practical for large applications. Therefore, a heuristic approach is used in this work to construct a schedule before run-time [40, 32].

The framework of our off-line scheduling approach (shown in Figure 6) consists of the following elements:

1. *Application designer* designs applications using reusable ADT module components and process templates.

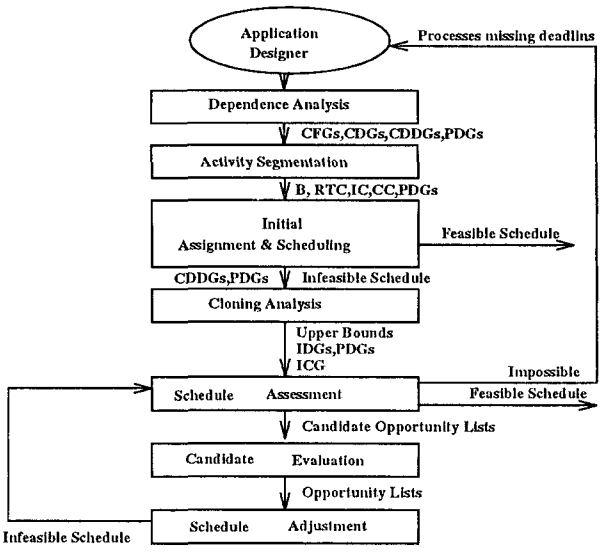


Figure 6: The scheduling approach

2. *Dependence analysis* identifies the dependence relations among statements, beads, blocks, and module instances. A control flow graph (CFG), a control dependence graph (CDG), a control and data flow graph (CDDG), and a program dependence graph (PDG) are constructed for each module instance in the application.

3. *Activity segmentation* groups statements into beads based on the dependence relations. It may also include the clustering of beads into blocks, to reduce scheduling complexity and the number of context switches.

4. *Initial assignment and scheduling* generates an assignment and schedule for each PE. In this work, the technique described in [30] is used to perform the initial assignment and scheduling. If the timing constraints of an application are quite loose, even the initial scheduling can generate a feasible schedule. If the initial schedule is not feasible, concurrency is enhanced and applied to the processes (or beads) missing deadlines.

5. *Cloning analysis* produces the PDGs. Instance dependence graphs (IDGs) are generated to expose the instance dependence relations in each method of a module instance. An upper bound on the number of clones of each ADT module instance that can run concurrently is determined.

6. *Schedule assessment* evaluates the infeasible schedule. If all resources have 100% utilization and no extra resources are available, the only thing we can do is to ask the application designer to relax the timing constraints. The evaluation process includes identification of (1) the beads of the processes missing deadlines and (2) a list of candidate opportunities to enhance concurrency. The opportunities that we may consider to be a candidate include:
   - cloning module instances to resolve contention;
   - using ARPCs to allow caller and callee to run concurrently;
   - reordering statements to enable ARPCs;
   - load distribution;
   - classical parallelizing compiler techniques.
7. During *candidate evaluation* and *schedule adjustment*, the candidate opportunities identified are evaluated to balance the amount of concurrency they would produce against the overhead they would cause. Since the evaluation process is NP-hard, we use dependence relations to narrow the search for effects of applying opportunities. Among the candidate opportunities, one of them is chosen to be applied to the infeasible schedule to shorten the response time of a process missing its deadline.

5 Concurrency Enhancement

The goal of scheduling is to resolve contention for shared resources. Generally, there are two kinds of shared resources: hardware resources (CPUs, I/O devices, and communication media) and software resources (ADT module instances). One way to resolve contention for shared resources is by replication of resources, as a multiprocessor system is used to resolve contention for the single CPU in an uniprocessor system. On the extreme, if the number of resources allows every client to get a resource at any time, there is no contention and there is minimal need for scheduling. However,

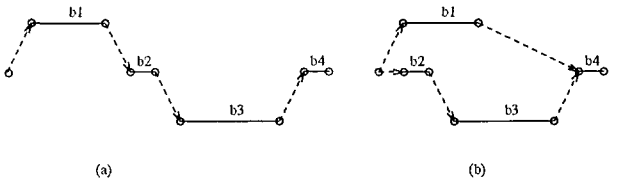


Figure 7: (a) SRPCs and (b) ARPCs

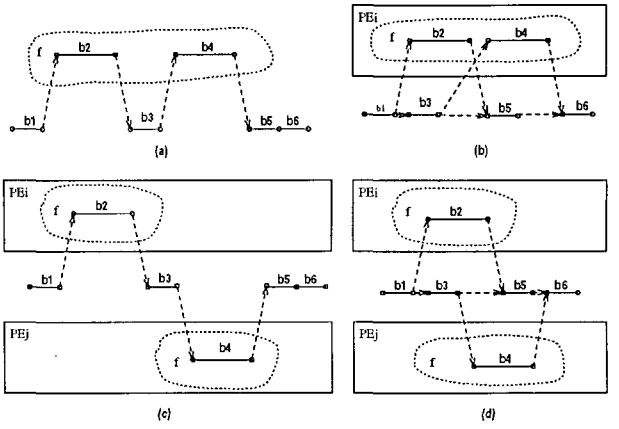


Figure 8: (a) Using neither ARPCs nor cloning. (b) With ARPCs only. (c) With cloning only. (d) With ARPCs and cloning.

the quantity of hardware resources is typically fixed in computing systems. Therefore, scheduling of hardware resources is necessary. Similarly, replication of software resources is also a way to resolve contention. In programs built by layering ADT module instances, an instance is often used to manage several data objects, and there will be contention for getting access to the instance if multiple data objects need to be accessed concurrently by multiple clients. Cloning an ADT instance allows each clone to manage only one data object, or a subset of the data objects.

In the following sections, several ways to enhance concurrency are presented. The example in Figure 1 is used to show how each kind of concurrency enhancement works in conjunction with the scheduling approach.

5.1 Enhancing Concurrency via ARPCs

One way to introduce concurrency is to use ARPCs instead of SRPCs. With SRPCs, the caller is blocked after making a remote procedure call. Most of the scheduling approaches switch the calling processor to another process, but the

calling process is blocked until the call returns. To reduce the execution time of the calling process, ARPCs can be used to let the caller continue execution if no memory conflict is caused. For the example in Figure 7 (a), we can see that the execution of $b2$ and $b4$ are blocked until the return of the two synchronized calls to $b1$ and $b3$, respectively. If we change the calls from SRPCs into ARPCs and assume the parameters used by $b1$ and $b3$ are not used until $b4$, the two calls can run concurrently as shown in Figure 7 (b). In Figure 8 (a), all calls are SRPCs and no concurrency exists. If instance f is allocated on another PE, and parameters used by bead $b2$ are not used until bead $b5$, and the parameters used by bead $b4$ are not used until bead $b6$, with the use of ARPCs, beads $b3$ and $b5$ can run concurrently with beads $b2$ and $b4$, respectively, and concurrency is achieved (as shown in Figure 8 (b)).

ARPCs make the caller and called method run concurrently if they do not access the same memory location at the same time. By looking at the dependence relations among method calls, the opportunities for applying ARPCs can be identified [41]. In our scheduling approach, ARPCs are also used to reduce the execution times of processes missing deadlines.

5.2 Enhancing Concurrency via Instance Cloning

Cloning of resources can reduce contention for resources. If the number of clones of software resources in every processor is sufficient, then no contention exists. The question is, "How many clones of a resource is enough?" Another important question is "How many clones are needed to enhance concurrency to enable a schedule to meet deadlines?". To determine the lower bound on the number of clones needed, program dependence relations are analyzed. In [41], techniques are presented for determining the lower bound on the number of clones of each ADT module instance needed to resolve all possible contention of the ADT module instances. The technique employs dependence analysis techniques at the statement, method, and instance levels of granularities. The program dependence graph (PDG), which was previously used to describe data and control dependence relations among statements, is extended to include instance dependence relations in object-based systems. Several theorems are proved with respect to the instance dependence properties of the new PDG graph in [41]. In Figure 8 (c), ADT instance f is cloned and is placed on a different PE. Now two clones of instance f can serve two calls at the same time.

5.3 Enhancing Concurrency via a Combination of Cloning and ARPCs

If two calls do not access the same memory location at any time, but they call the same method or different methods provided by the same ADT module instance, the two cannot run concurrently since they have contention to access same ADT module instance. Cloning can be used to resolve the contention, and the SRPC can be converted into an ARPC. In the example in Figure 8 (d), ARPCs are combined with instance cloning so that maximum concurrency is achieved.

For the application in Figure 1, there exists no feasible schedule (no matter how process and module instances are assigned and scheduled) if concurrency enhancement is not applied. Assume an initial schedule is constructed as shown in Figure 5. Since all instances are in the same PE, all calls are LPCs. Therefore, only one thread of execution exists, i.e., the schedule of PE_i contains only one bead. Given such a schedule, our approach is to try to improve the schedule by enhancing concurrency. By examining the execution graph of the process $Task1$, we see that $Queue$ is shared by two calls. Only one call is granted access to the $Queue$ at any time; the other call is put into waiting. We also see that the two calls $Q.remove(inQueue, part)$ and $Q.insert(outQueue, prod)$ do not have common parameters, and do not call the same method, but they call the methods provided by the same ADT module instance Q. The two calls cannot run concurrently due to the contention for the instance Q. Cloning of Q can resolve the contention and turn the SRPC to an ARPC. In Figure 9, ADT instance $Queue$ is cloned and placed on a different PE, and the two calls ($Queue.insert()$ and $Queue.remove()$) made by instance M can run concurrently. The one thread of execution in Figure 5 is broken into 4 beads. Beads $b2$ and $b3$ run concurrently. The execution time of $Task1$ is reduced from 17 to 13. Therefore, concurrency

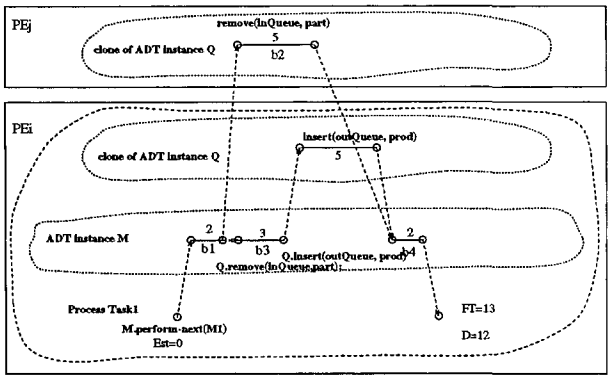


Figure 9: Two clones of Queue serve two calls concurrently, so that an almost feasible schedule is constructed by cloning.

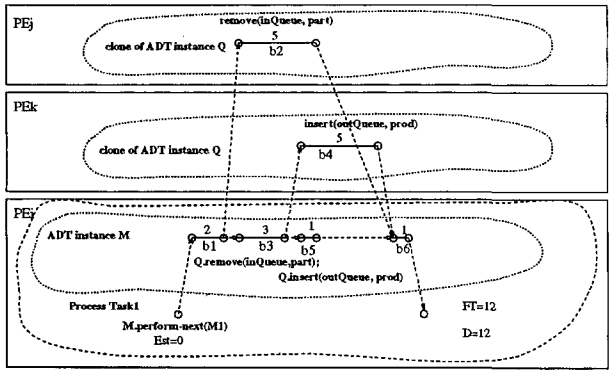


Figure 10: Enhancing Concurrency by Load Distribution

is enhanced and an almost feasible schedule is found by using cloning and ARPCs, as shown in Figure 9.

5.4 Load Distribution to Allow Cloning and ARPCs

A good schedule has high utilization of resources. If one process is unable to continue execution, the resource is given to another process. As we mentioned before, if the utilization of a PE reaches 100 percent, no ARPCs and cloning can be applied. To enable ARPCs and cloning, load distribution is necessary. In the example shown in Figure 1, note that the deadline of process $Task1$ is 12 time units, but its finishing time is 13 time units. The schedule in Figure 9 needs to be further improved. Although an ARPC opportunity exists (the first part of $b4$ will not access variables $OutQueue$ and $prod$ which are the parameters used by the call $insert()$), PE_i is scheduled to execute method call $insert(OutQueue, prod)$. If the ADT instance Q on PE_i is placed on PE_k as shown in Figure 10, $bead$ $b4$ can be an ARPC and can run concurrently with $bead$ $b5$. Thus, the finishing time of the task is reduced to 12 time units, and process $Task1$ can meet its deadline due to the load distribution.

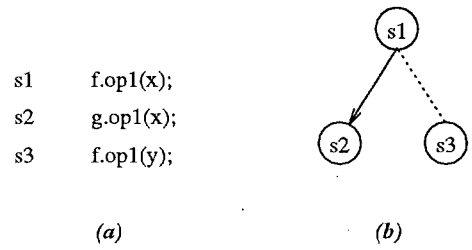


Figure 11: A program segment (a) and its PDG (b) where solid arrows represent data dependent relations, and dashed arcs denote instance dependent relations

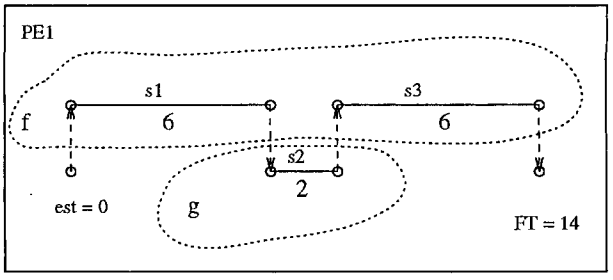


Figure 12: Statements s2 blocks the execution

5.5 Statement Reordering to Expose Concurrency

Although the upper bound of clones of an instance tells the maximum number of clones that can run concurrently, the statement order might prevent part of the concurrency. Let us see a simple example in Figure 11 (a). From its extended PDG [41] in Figure 11 (b), we can see that the maximum number of clones of instance f is 2, and the maximum number of clones of instance g is 1. However, the statement order s_1, s_2, s_3 prevents statements s_1 and s_3 from being executed in parallel, since statement s_2 would be blocked due to the busy parameter x until statement s_1 completes. The schedule is shown in Figure 12.

If we reorder the statements as (s_1, s_3, s_2) or (s_3, s_1, s_2), statements s_1 and s_3 can be executed concurrently (allow two clones of instance f to be used in parallel). For the schedule shown

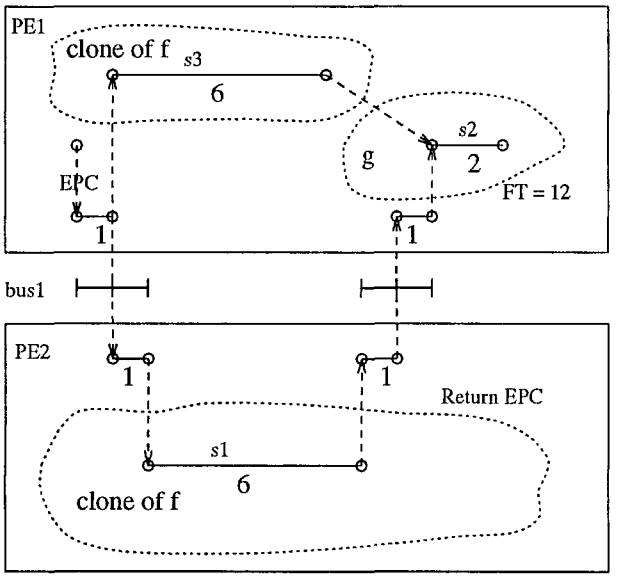


Figure 13: Statements s1 and s3 run concurrently by reordering statements s2 and s3

in Figure 12, if we swap the two statements $s2$ and $s3$ and place a clone of instance f on $PE2$, the execution time of this program segment can be reduced from 14 time units to 12 time units, as shown in Figure 13.

6 Conclusion and Future Work

ADT modules increase reusability, but potential inefficiencies may occur at execution time. Concurrent execution with ARPCs and instance cloning can greatly improve the performance of programs constructed with ADTs. This is especially useful for real-time systems, where the timing constraints are a concern. The scheduling approach presented in this paper differs from previous off-line scheduling techniques by employing concurrency enhancement to reduce the execution time of processes missing deadlines.

A platform [27] (including ADA+RESOLVE compiler, linker, parallel virtual machine, real-time kernel, timing analysis tools, global scheduler (allocation algorithms) and graphical interface) has been developed for executing programs constructed with ADTs on SUN workstations and Ncube. Dependence analysis and cloning analysis techniques are being developed currently. Ongoing research includes implementing and experimentally evaluating the scheduling algorithm, developing concurrency metrics to measure amount of concurrency, and applying concurrency enhancement techniques to reengineer existing systems. Future work includes method cloning to reduce the number of clones of instances needed to resolve contention of shared ADT module instances. When the number of clones of an ADT module instance does not reach the upper bound, assignment of variables to clones so that maximum concurrency can be achieved is a research issue.

7 Acknowledgments

Thanks is due to A. K. Ganesh, D. Hammer, T. J. Marlowe, J. McHugh, P. Ng, B. Ravindran, A. D. Stoyenko, J. P. C. Verhoosel, and W. Zhao for discussions that helped to refine this work. We are very grateful to *NJIT (SBR 421290)* and the *US NSWC (N60921-94-M-G096)* who have in part supported this work.

References

[1] R. A. Ballance, A. B. Maccabe, and K. J. Ottenstein. The program dependence web: A representation supporting control-, data- and demand-driven interpretation of imperative languages. In *Proceedings of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation*, pages 257–271. ACM, June 1990.

[2] P. Bratley, M. Florian, and P. Robillard. Scheduling with earliest start and due date constraints on multiple machines. *Nav. Res. Log. Quart.*, 22:165–173, 1975.

[3] V. Cherkassky. Redundant task-allocation in multicomputer systems. *IEEE Trans. on Soft. Eng.*, 41(3):336–342, September 1992.

[4] E. C. Cooper. Circus: A replicated procedure call facility. In *Proc. 4th Symp. Reliability in Distributed Software and Databases*, pages 11–24, 1984.

[5] K. D. Cooper, M. W. Hall, and K. Kennedy. A methodology for procedure cloning. In *The International Conference on Computer Languages*. IEEE, April 1992.

[6] R. Cytron, J. Ferrante, B. K. Rosen, and M. N. Wegman. Efficiently computing static

single assignment form and the control dependence graph. *ACM Trans. on Programming Languages and Systems*, 13(4):451–490, October 1991.

[7] K. Driscoll and K. Hoyme. The airplane information management system: An integrated real-time flight-deck control system. In *Real-Time Systems Symposium*, 1992.

[8] J. Ferrante, K. J. Ottenstein, and J. D. Warren. The program dependence graph and its use in optimization. *ACM Trans. on Programming Languages and Systems*, 9(3):319–349, July 1987.

[9] M. R. Garey, D. S. Johnson, B. B. Simons, and R. E. Tarjan. Scheduling unit-time tasks with arbitaray release times and deadlines. *SIAM Journal of Computing*, 10:256–269, May 1981.

[10] D.K. Hammer and O.S. van Roosmalen. An object-oriented model for the construction of dependable distributed systems. In *International Workshop on Object-Orientation in Operating Systems (I-WOOOS 92)*, Paris, France, September 1992.

[11] M. J. Harrold, B. A. Malloy, and G. Rothermel. Efficient construction of program dependence graphs. Technical Report 92-128, Clemson University, December 1992.

[12] Maurice Herlihy. Concurrency versus availability: Atomicity mechanisms for replicated data. *ACM Trans. on Computer Systems*, 5(3):249–274, August 1987.

[13] Joseph E. Hollingsworth. *Software Component Design-for-Reuse: A Language-Independent Discipline Applied to Ada.* PhD thesis, The Ohio State University, 1992.

[14] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The MARS approach. *IEEE MICRO*, 9(1):25–40, February 1989.

[15] B. Liskov and L. Shrira. Promises: Linguistic support for efficient asynchronous procedure calls in distributed systems. In *Proceedings of the SIGPLAN '88 Conference on Programming Language Design and Implementation*, pages 260–267. ACM, June 1988.

[16] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, 20(1):46–61, Jan. 1973.

[17] C. Martel. Preemptive scheduling with release times, deadlines, and due times. *Journal of ACM*, 29(3):812–829, July 1982.

[18] G. McMahon and M. Florian. On scheduling with ready times and due dates to minimize maximum lateness. *Operation Research*, 23:475–482, 1975.

[19] A. Mok and M. Dertouzos. Multiprocessor scheduling in a hard real-time environment. In *Proceedings of the 7th Texas Conference on Computing Systems*, pages 5.1–5.12, November 1978.

[20] Department of Defense. *Reference Manual for the Ada Programming Language.* Ada Joint Program Office, Washington, D.C., Government Printing Office, ansi/mil-std-1815a-1983 edition, 1983.

[21] K. Ramamritham. Allocation and scheduling of complex periodic tasks. In *International Conf. on Distributed Computing Systems.* IEEE, May-June 1990.

[22] K. Ramamritham and J. A. Stankovic. Dynamic task scheduling in distributed hard real-time systems. In *Proc. of the 4th IEEE International Conf. on Distributed Computing Systems*, pages 96–107, May 1984.

[23] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. Technical Report Technical Report CMU-CS-87-181, Carnegie-Mellon University, November 1987.

[24] B. Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM Journal of Computing*, 12:294–299, May 1983.

[25] M. Sitaraman, L. R. Welch, and D. E. Harms. Influences of a component-based industry on the expression of specifications of

reusable software. *The International Journal of Software Engineering and Knowledge Engineeeing*, pages 207–229, June 1993.

[26] J.A. Stankovic. Misconceptions about real-time computing. *IEEE Computers*, 21(10):10–19, October 1988.

[27] A.D. Stoyenko, L.R. Welch, P.L. Laplante, T.J. Marlowe, C. Amaro, B. Cheng, A. K. Ganesh, M. Harelick, X. Jin, M. Younis, and G. Yu. A platform for complex real-time applications. In *The Complex Systems Engineering Synthesis and Assessment Technology Workshop*. Naval Surface Warfare Center, July 1993.

[28] J. D. Ullman. Np-complete scheduling problems. *Journal of Comput. System Science*, 10:384–393, 1975.

[29] P.D.V. van der Stok, F. van den Berk, R. Deckers, Y. van de Vijver, J.I.M. Botman, and C.J. Timmermans. Object-oriented design for accelerator control. In *IEEE Trans. on Nuclear Science*, No.1 pages 200-208, Feb. 1994.

[30] J. P. C. Verhoosel and et al. A static scheduling algorithm for distributed hard real-time systems. *Journal of Real-Time Systems*, pages 227–246, 1991.

[31] J. P. C. Verhoosel, L. R. Welch, D. K. Hammer, and A. D. Stoyenko. A model for assignment and pre-runtime scheduling of object-based, distributed real-time systems. *Journal of Real-Time Systems*, 1994 (to appear).

[32] J.P.C. Verhoosel, G. Yu, L.R. Welch, and D.K. Hammer. Pre-run-time scheduling for object-based, concurrent, real-time applications. In *Proceedings of the 2nd IEEE Workshop on Real-Time Applications*, pages 8–11, July 1994.

[33] B. W. Weide, W. F. Ogden, and S. H. Zweben. Reusable software components. In M.C. Yovits, editor, *Advances in Computers*, volume 33, pages 1–65. Academic Press, 1991.

[34] Bruce W. Weide, Stephen H. Edwards, Douglas E. Harms, and David A. Lamb. Design and specification of iterators using the swapping paradigm. *IEEE Transactions on Software Engineering*, 20(8):631–643, August 1994.

[35] L. R. Welch. Cloning ADT modules to increase parallelism: Rationale and techniques. In *Fifth IEEE Symposium on Parallel and Distributed Processing*, pages 430–437. IEEE, December 1993.

[36] L. R. Welch. A parallel virtual machine for programs composed of abstract data types. *IEEE Trans. on Computers*, 43(11), November 1994.

[37] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions on Software Engineering*, 19(2):139–154, February 1993.

[38] J. Xu and D.L. Parnas. Pre-run-time scheduling of processes with execlusion relations on nested or overlapping critical sections. In *Proceedings of 11th Annual IEEE International Phoenix Conference on Computers and Communications (IPCCC-92)*, pages 774–782. IEEE, April 1992.

[39] G. Yu, L. R. Welch, W. Rossak, and A. D. Stoyenko. Automatic retrieval of formally specified real-time software components. In *Fifth Annual Workshop on Software Reuse*, October 1992.

[40] Guohui Yu. Use of concurrency enhancement in off-line schedule construction. In *The Second Workshop on Parallel and Distributed Real-Time Systems*, pages 32–37. IEEE, April 28-29, Cancun, Mexico 1994.

[41] Guohui Yu and Lonnie R. Welch. Program dependence analysis for concurrency exploitation in programs composed of abstract data type modules. In *Sixth IEEE Symposium on Parallel and Distributed Processing*. IEEE, October 1994.

[42] W. Zhao, K. Ramamritham, and J. A. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE Transactions on Computers*, C-36(8):949–960, August 1987.

On-line Algorithms for Allocating Periodic-time-critical Tasks on Multiprocessor Systems

Sadegh Davari
Department of Computing and Mathematics
University of Houston-Clear Lake
Houston, TX 77058
AND
Sudarshan K. Dhall
School of Computer Science
University of Oklahoma
Norman, Ok 73019

Keywords: scheduling, multiprocessors, time critical, on-line, heuristic algorithms, real-time

Edited by: Marcin Paprzycki and Janusz Zalewski
Received: February 9, 1994 **Revised:** October 27, 1994 **Accepted:** January 19, 1995

The problem of allocating a set of Periodic-Time-Critical (PTC) tasks to processors in a multiprocessor system is considered. A PTC task is a real-time task for which requests are made periodically, at some fixed interval of time, by means of some external signals. Associated with each request there is a computation time and a deadline for the completion of the computation. The Rate-Monotonic algorithm, which is an optimal static priority driven algorithm for scheduling PTC tasks on single processor systems (Liu & Layland 1973, Serlin 1972, Sha & Goodenough 1990, Locke 1992), does not perform as well for multiprocessor systems (Dhall 1977, Dhall & Liu 1978, Davari 1985, Davari & Dhall 1986a, 1986b, 1986c, Oh & Son 1994). The allocation problem is to distribute a set of PTC tasks among various processors so that the tasks assigned to each processor can be feasibly scheduled on that processor using the Rate-Monotonic algorithm. Moreover, the aim is to use as few processors as possible. This problem is NP-hard (Davari 1985, Leung & Whitehead 1982). In this paper we present two heuristic on-line algorithms and analyze their complexity and worst case performance.

1 Introduction

Process control computers are now being used to control and monitor a wide variety of time-critical processes. In many of these applications, the computer is required to execute a certain number of time-critical tasks in response to periodic external signals, and to guarantee that each task is completely executed within a specified interval of time following the occurrence of the signal that caused the initiation of the task. Each such task is referred to in this paper as a Periodic-Time-Critical (PTC) task.

1.1 Problem Formulation

In this paper we study the problem of distributing a set of PTC tasks among various processors so that when the allocated tasks are scheduled on the processor according to a given scheduling algorithm, every request of each one of the tasks is executed before the corresponding deadline. Moreover, the aim is to use as few processors as possible.

We assume that the tasks to be allocated satisfy the following characteristics:

[0]Section 2 is based on On-line Algorithms for Real-Time Tasks Allocation by Davari & Dhall, Twentieth Conference on Information Sciences and Systems, 1986, p.178-182, and Section 3 is based on An On-Line Algorithm for Real-Time Tasks Allocation by Davari and Dhall which appeared in the Proceedings of the Real-Time Systems Symposium, Dec 1986, New Orleans, LA, p. 194-200, ©1986 IEEE.

1 The requests of each task are periodic, with constant intervals between requests.

2 Deadlines consist of runability constraints only, i.e. each request must be completed before the next request of the same task occurs.

3 The tasks are independent in that the requests of a task do not depend on the initiation or the completion of the requests of the other tasks.

4 Computation time for the requests of a task is constant for the task. Computation time refers to the time a processor takes to execute the request without interruption.

It follows that a task is completely defined by two numbers, the computation time of each request and the request period. We will denote a task T by the ordered pair (c, t), where c is the *computation time* and t is the *request period.* The ratio $\frac{1}{t}$ is called the *request rate*, and the ratio $\frac{c}{t}$, denoted by u is called the *utilization factor* of the task. Note that the utilization factor of a task is the proportion of the processor time taken by the task. Therefore, the utilization factor of the processor over a set of tasks is the sum of the utilization factors of all the tasks in the set.

A scheduling algorithm provides a set of rules that determines the task to be executed at a particular point in time. We say that a set of tasks can be *feasibly scheduled* by an algorithm, if the schedule produced by the algorithm meets the deadlines of all the requests of each task in the set. In this paper we consider only *preemptive priority driven scheduling algorithms.* In these algorithms, a currently running task of lower priority will be taken off the processor whenever there is a request for a higher priority task, even though the lower priority task has not yet completed. The interrupted task is resumed later from the point of interruption. We will assume the cost of interruption to be negligible. The priority-driven scheduling algorithms can be classified into two categories: *static priority algorithms*, and *dynamic priority algorithms.* In *static priority algorithms*, the priorities of tasks are fixed in advance once and for all; whereas in the *dynamic priority algorithms*, the priorities of tasks may change from time to time, depending upon certain conditions. The implementation of a dynamic priority algorithm requires a lot more overhead than the implementation of a static priority algorithm. In this paper, we will only consider static priority algorithms.

We call a scheduling algorithm *on-line* if it schedules tasks as they arrive. In other words, the tasks are available one at a time, and the algorithm schedules each task as and when it becomes available without taking into consideration the tasks to follow. In contrast, we call a scheduling algorithm off-line, if it has to have complete information about all the tasks before the scheduling process can begin.

1.2 Single Processor Systems

The problem of scheduling a set of PTC tasks on a single processor system has been considered by a number of researchers (Liu & Layland 1973, Serlin 1972, Sha & Goodenough 1990, Labetoulle 1974, Dhall 1977). The *Rate-Monotonic Scheduling (RMS)* algorithm, introduced independently by Liu and Layland (Liu & Layland 1973) and Serlin (Serlin 1972], is the best static priority algorithm available for this problem (Locke 1992). This algorithm schedules tasks in decreasing order of their request rate. For ease of reference, we present the main result about this algorithm in the form of the following theorem:

Theorem 1.1 (Liu & Layland 1973) A set of m PTC tasks can be feasibly scheduled on a single processor by the rate-monotonic scheduling algorithm, if the utilization factor of the set is less than or equal to $m(2^{\frac{1}{m}} - 1)$, and this bound is tight in the sense that for each m, there exists a set of m tasks with utilization factor $m(2^{\frac{1}{m}} - 1)$ which fully utilizes the processor.

The value $m(2^{\frac{1}{m}} - 1)$ approaches $\ln 2$ (≈ 0.69) as m approaches infinity. That is, in the worst case, the processor may be only 69 per cent utilized. Note that Theorem 1.1 provides only a sufficient condition. It implies that sets of m tasks with utilization factor greater than $m(2^{\frac{1}{m}} - 1)$ may or may not be feasibly scheduled on one processor by the rate-monotonic algorithm.

Recently, the original RMS algorithm, which dealt with PTC type tasks only, has been extended to include aperiodic tasks with time constraints (Sha & Goodenough 1990, Sprunt et. al. 1989) and task synchronization (Sha & Goodenough 1990, Sha et. al. 1990, Davari & Sha

1992). Recently, RMS has also been considered for distributed systems (Agrawal *et. al.* 1994). NASA has adapted RMS as a baseline technology for the Space Station Freedom [Davari & Zhao 1991, Davari et. al. 1993). The European Space Agency has also specified RMS as the baseline theory for its Hard Real-Time Operating System project (ESA 1990).

1.3 Multiple Processor Systems

It has been shown that (Dhall 1977, Davari 1985) a simple extension of the scheduling algorithms which perform well for scheduling PTC tasks on single processor systems does not provide satisfactory results for multiprocessor systems. Recently, Leung (Leung 1989) proposed an algorithm, called the Slack-Time Algorithm for scheduling real-time tasks on a single and multiprocessor system. However, the problem of deciding if a task system is schedulable by Slack-Time algorithm, the Deadline algorithm, the Rate-Monotonic algorithm, or any other fixed priority scheduling algorithm was shown to be NP-hard for each fixed m, the number of processors in the system (Leung & Whitehead 1982, Leung 1989). An alternative approach to designing algorithms for multiprocessor systems would be to first partition the set of tasks into different groups, and then schedule the tasks in each group on one processor using the algorithm for a single processor. This reduces the problem to finding a good partitioning scheme in order to use a minimum number of processors. The scheduling algorithm to be used on each group of tasks will certainly influence the partitioning process. Bannister and Trivedi (Bannister & Trivedi 1983) also considered the problem of distributing PTC tasks on different processors. Their goal was to provide fault-tolerance by replicating each task on $r(> 1)$ processors. Further, the distribution was required to achieve load balance for the processors.

A partitioning algorithm is said to be *optimal* if it produces a partition with a minimum number of subsets for any given set of tasks to which it is applicable. Note that since the tasks in each subset of the partition can be feasibly scheduled on a single processor, an optimal partitioning scheme will require a minimum number of processors for the execution of the entire set of tasks. The problem of partitioning a set of PTC tasks with respect to the rate-monotonic-scheduling algorithm has been shown to be NP-hard (Davari 1985, Leung & Whitehead 1982). This partitioning problem is similar to the bin packing problem (Johnson 1974, Johnson *et. al.* 1974, Lee & Lee 1985, Yao 1980) where it is required to pack a given list of pieces into a minimum number of bins of a fixed size. The similarity stems from the fact that the utilization factor of a task can be treated as the size of a piece, and the processor capacity as the bin size. However, major difference is that when tasks are partitioned using the rate-monotonic scheduling algorithm, it cannot be guaranteed that a task set with a total utilization factor less than one, can be feasibly scheduled on the processor.

1.4 Suboptimal Solution

In view of the inherent difficulty of finding efficient optimal algorithms for our problem, we look for sub-optimal heuristic partitioning algorithms which produce satisfactory results with comparatively less effort, and analyze their performance.

Let N^* and $N(A)$ denote, respectively, the number of subsets in the partition produced by an optimal partitioning algorithm and by a heuristic algorithm A when applied to a given set of tasks. Then, the *Worst-Case Performance Ratio* of algorithm A, denoted by $r(A)$ is defined as:

$$r(A) = \lim_{N^* \to \infty} \frac{N(A)}{N^*}.$$

As with many heuristic combinatorial algorithms, we have chosen to measure the worst-case, rather than the average-case, performance of our algorithms. The quantity $r(A)$ gives the asymptotic least upper bound on the ratio $\frac{N(A)}{N^*}$. For large problems, therefore, the algorithm A may be $r(A)$ times as costly as an exhaustive search, but no worse.

Dhall and Liu considered two heuristic algorithms for this problem in (Dhall 1977). They were called the *Rate-Monotonic Next-Fit (RMNF)* and the *Rate-Monotonic First-Fit (RMFF)* algorithm. Davari and Dhall (Davari & Dhall 1986a) considered another one called the *First-Fit-Decreasing-Utilization-Factor (FFDUF)* algorithm. These are all off-line algorithms because they require that information about all the tasks be available before the scheduling decisions can be made. The

Table 1: Comparison of the Worst Case Performance Ratio of Some Off-line Partitioning Algorithms for a set of n tasks

Algorithm	Time	Space	Worst-Case Performance Ratio
RMNF	$O(nlogn)$	$O(n)$	$2.4 \leq r(RMNF) \leq 2.67$
RMFF	$O(n^2)$	$O(n)$	$2 \leq r(RMFF) \leq 2.24$
FFDUF	$O(n^2)$	$O(n)$	$r(FFDUF) = 2.$

results about these algorithms are given in Table 1.

Dhall & Liu (Dhall & Liu 1977) conjectured that the upper bound of RMFF can be reduced to 2. Their conjecture was based on the following Lemma, which they could not prove at the time:

Lemma: If n tasks cannot be feasibly scheduled on $n-1$ processors according to RMFF, then the total utilization of the n tasks is greater than $\frac{n}{1+2^{1/n}}$.

In this paper, we present only on-line algorithms and analyze their performance. Recently, Oh & Son (Oh & Son 1994) have also presented some on-line algorithms. Two on-line algorithms RM-FF and RM-BF have an upper bound of 2.33 and lower bound of 2.28. Further refinements of these two algorithms called *RRM-FF* and *RRM-BF* are shown to have a worst-case bound of 2. The time and space complexity of these algorithms is $O(n \log n)$ and $O(n)$. As will be shown later, the algorithms presented in this paper have a time complexity of $O(n)$, and a space complexity of $O(1)$. The worst-case performance of the *Next-Fit-M* algorithm is better than the worst-case performance of the RM-FF and RM-BF algorithms.

Before we go to the next section we state and prove one more result (Davari 1985) which will be used in the proof of some of the following theorems.

Theorem 1.2: Let $m > 0$. Then, the function $f(m) = m(2^{\frac{1}{m}} - 1)$ is monotonically decreasing with m, and the function $g(m) = (m-1)(2^{\frac{1}{m}} - 1)$ is monotonically increasing with m. As m approaches infinity both these functions approach $\ln 2$.

Proof: To show that $f(m)$ is monotonically decreasing with m, we show that $f'(m) < 0$:

$$f'(m) = 2^{\frac{1}{m}}(1 - \frac{\ln 2}{m}) - 1$$

Since for $x > 0$, $e^{-x} > 1 - x$, we have

$$2^{-\frac{1}{m}} = e^{-\frac{\ln 2}{m}} > (1 - \frac{\ln 2}{m}).$$

Therefore, $1 > 2^{\frac{1}{m}}(1 - \frac{\ln 2}{m})$. Hence $f'(m) < 0$.

To show that $g(m)$ is monotonically increasing with m, we show that $g'(m) > 0$:

$$g(m) = (m-1)(2^{\frac{1}{m}} - 1) = m(2^{\frac{1}{m}} - 1) - 2^{\frac{1}{m}} + 1.$$

and

$$g'(m) = 2^{\frac{1}{m}}(1 - \frac{\ln 2}{m} + \frac{\ln 2}{m^2}) - 1$$

Since for $x > 0$, $e^{-x} < 1 - x + \frac{x^2}{2}$, we get

$$\begin{aligned} 2^{-\frac{1}{m}} &= e^{-\frac{\ln 2}{m}} \\ &< 1 - \frac{\ln 2}{m} + \frac{(\ln 2)^2}{2m^2} \\ &< 1 - \frac{\ln 2}{m} + \frac{\ln 2}{m^2}. \end{aligned}$$

Therefore,

$$1 < 2^{\frac{1}{m}}(1 - \frac{\ln 2}{m} + \frac{\ln 2}{m^2}).$$

Hence, $g'(m) > 0$.

Also, it is well known that $lim_{m \to \infty} m(2^{\frac{1}{m}} - 1) = \ln 2$, and it is straightforward to see that $lim_{m \to \infty} g(m) = \ln 2$.

The rest of the paper is organized as follows. Section 2 contains description and analysis of the algorithm *Next-Fit-2*, and Section 3 introduces and analyzes the algorithm *Next-Fit-M*. Finally, some concluding remarks are made in Section 4.

2 The Algorithm Next-Fit-2

The basic approach of this paper for partitioning a given set of tasks is to use some threshold value(s). A task whose utilization factor lies within a certain range is placed in the corresponding partition. First, we use only a single threshold value and divide the tasks into two classes. Then, in the next section we extend this idea to have M threshold values to improve the performance of the algorithm. Naturally, with the increase in the

number of threshold values, a little bit of extra work will be needed to classify a task into proper partition. The two algorithms thus show a cost-performance tradeoff.

In this section, we describe an algorithm which partitions tasks into two classes. The division into classes is based on some fence value x. An optimum value for x is also derived.

Let $T_1, T_2, \cdots, T_n$ be a set of tasks with utilization factors $u_1, u_2, \cdots, u_n$, respectively. Divide this set of tasks into *two* different classes as follows. Let any task T_i belong to class-1 if $u_i > (2^{\frac{1}{x}} - 1)$, where x is a positive integer greater than 1; otherwise let it belong to class-2, as shown below:

Class of Task	Range of Utilization Factor
1	$(2^{\frac{1}{x}} - 1, 1]$
2	$(0, 2^{\frac{1}{x}} - 1]$

The choice of the irrational number $2^{\frac{1}{x}} - 1$ is motivated by the result of Theorem 1.1. Hidden in the statement of Theorem 1.1 is the fact that for any integer n, there exists a set of n tasks each with utilization factor $2^{\frac{1}{n}} - 1$ which *fully utilize* the processor. We say that a task system *fully utilizes* a processor, if the tasks can be scheduled on a single processor using the Rate Monotonic Scheduling Algorithm honoring the deadline of each and every request, and any slight increase in the computation time of any one of these tasks renders the task system infeasible according to the Rate-Monotonic-Scheduling Algorithm, that is there will be some requests whose deadlines cannot be honored.

Similarly, divide the set of all processors into *two* different classes. A processor designated to process class-k tasks exclusively is referred to as a class-k processor, $1 \leq k \leq 2$. For convenience, let a processor of class-k, $1 \leq k \leq 2$, be called *filled* if it has been used and it is not intended to assign any more tasks to it. Let a class-k processor ($1 \leq k \leq 2$) be called *active* if it is the processor which will be considered for the assignment of the next class-k task. The algorithm that determines the assignment of tasks to various processors is given in Figure 1.

/* $P_{k,j}$ = the j^{th} processor of class-k */
/* U_k = the total utilization factor of all the tasks assigned to the active processor of class-k */
/* M_k = the number of tasks assigned to the active processor of class-k */
/* The final value of N_k will be the number of class-k processors used by the algorithm */

1. for $k = 1$ to 2 do
   set $N_k = 1$;
   set $U_k = M_k = 0$;
   end-for;
2. set $i = 1$;
3. while $i \leq n$ do
   if $u_i > (2^{\frac{1}{x}} - 1)$
   then set $k = 1$ /* T_i is a class-1 task */
   else set $k = 2$ /* T_i is a class-2 task */
   end-if;
   if $U_k > (M_k + 1)(2^{1/(M_k+1)} - 1) - u_i$
   then set $N_k = N_k + 1$;
   set $U_k = M_k = 0$;
   end-if;
   assign T_i to P_{k,N_k};
   set $U_k = U_k + u_i$;
   set $M_k = M_k + 1$;
   set $i = i + 1$
   end while;
4. if $M_k = 0, 1 \leq k \leq 2$,
   then set $N_k = N_k - 1$;

The final values of $N_k, 1 \leq k \leq 2$ would be the number of class-k processors used by the algorithm.

Figure 1: Algorithm Next-Fit-2.

2.1 Worst-Case Analysis of Next-Fit-2

The upper bound for the worst-case performance ratio of *Next-Fit-2* denoted by $r(NF2)$, is obtained in two parts. In part I we calculate the upper bound for $r(NF2)$ when $x = 2$, and in part II we calculate the bound for $x > 2$. We will assume the following definitions throughout this section.

Let N_1 and N_2 denote, respectively, the number of class-1 processors and the number of class-2 processors needed by *Next-Fit-2* to schedule the given set of tasks.

Let S, S_1, and S_2 denote, respectively, the sum of the utilization factors of all tasks, the sum of the utilization factors of all class-1 tasks, and the sum of the utilization factors of all class-2 tasks in the given set.

Part I: $x = 2$.

Lemma 2.1: For $x = 2$, we have $N_1 < \frac{S_1}{2^{\frac{1}{2}} - 1} + 1$

Proof: Because the utilization factor of any class-1 task in this case is greater than $(2^{\frac{1}{2}} - 1)$ and any filled class-1 processor must have at least one task assigned to it.

Lemma 2.2: For $x = 2$, we have $N_2 < \frac{2S_2}{\ln 2} + 2$.

Proof: By Theorem 1.1, the total utilization factors of the tasks assigned to any two adjacent class-2 processors must be greater than $\ln 2$.

Corollary 2.1: By Theorem 1.1, Lemma 2.1, and Lemma 2.2, for $x = 2$, we have

$$N(NF2) = N_1 + N_2 < \frac{2S}{\ln 2} + 3.$$

Theorem 2.1: For $x = 2$, we have:

$$r(NF2) = \lim_{N^* \to \infty} \frac{N(NF2)}{N^*} \leq \frac{2}{\ln 2}.$$

Proof: Since $N^* \geq S$, the proof follows by Corollary 2.1.

Part II: $x > 2$.

Lemma 2.3: For $x > 2$, we have

$$N_1 < \frac{2S_1}{(x-1)(2^{\frac{1}{x-1}} - 1)} + 4.$$

Proof: Divide all the filled class-1 processors into two groups as follows. Let all filled class-1 processors which have less than $(x-1)$ tasks assigned to them along with their next immediate neighbors belong to group-1 and the rest of the filled class-1 processors belong to group-2. Before we continue further, let us make some additional definitions.

- Let $S_{1,i}, 1 \leq i \leq 2$, denote the sum of the utilization factors of class-1 tasks assigned to all group-i processors.
- Let $N_{1,i}, 1 \leq i \leq 2$, be the number of class-1 processors in group-i. For group-2 processors, we obviously have

$$N_{1,2} < \frac{S_{1,2}}{(x-1)(2^{\frac{1}{x}} - 1)} + 1.$$

We next consider group-1 processors. Relabel group-1 processors in increasing order of their index as $P_1, P_2, ..., P_{N_{1,1}}$. Let U_i be the total utilization factors of the tasks assigned to processor $P_i, 1 \leq i \leq N_{1,1}$. For now, let us assume that $N_{1,1}$ is an even number. If we consider these processors as a group of $\frac{N_{1,1}}{2}$ adjacent pairs, then, by definition, the first processor in each pair has less than $(x-1)$ tasks assigned to it. Let P_i and P_{i+1} be one such pair, where P_i has $m < (x-1)$ tasks assigned to it. Then, we must have $U_i + U_{i+1} > (m+1)(2^{1/(m+1)} - 1)$. For otherwise, the assignment of tasks to P_{i+1} is made illegally. Since by Theorem 1.2, $(m+1)(2^{1/(m+1)} - 1)$ decreases as m increases, by substituting the maximum possible value of m in terms of x, we get $U_i + U_{i+1} > (x-1)(2^{\frac{1}{x-1}} - 1)$. Since this is true for any such pair of group-1 processors, we have

$$N_{1,1} < \frac{2S_{1,1}}{(x-1)(2^{\frac{1}{x-1}} - 1)} + 2.$$

If $N_{1,1}$ is odd, then we have

$$N_{1,1} < \frac{2S_{1,1}}{(x-1)(2^{\frac{1}{x-1}} - 1)} + 3.$$

We proceed to finish the proof of Lemma 2.3.

$$\begin{aligned} N_1 &= N_{1,1} + N_{1,2} \\ &< \frac{2S_{1,1}}{(x-1)(2^{\frac{1}{x-1}} - 1)} \\ &\quad + \frac{2S_{1,2}}{2(x-1)(2^{\frac{1}{x}} - 1)} + 4. \end{aligned}$$

We next show that $(x-1)(2^{\frac{1}{x-1}} - 1) < 2(x-1)(2^{\frac{1}{x}} - 1)$.

By Theorem 1.2, for $x > 1$, $(x-1)(2^{\frac{1}{x-1}} - 1)$ is a decreasing function of x, that is it has its maximum value when x has its minimum possible value. Similarly, $(x-1)(2^{\frac{1}{x}} - 1)$ has its minimum value when x has its minimum possible value. Then, by substituting the minimum value of x, which is 3, in both sides of the above inequality the proof of our claim becomes obvious.

Thus, we have

$$N_1 = \frac{2S_1}{(x-1)(2^{1/(x-1)} - 1)} + 4.$$

Lemma 2.4: For $x > 2$, we have

$$N_2 < \frac{S_2}{\ln 2 - (2^{\frac{1}{x}} - 1)} + 1.$$

Proof: By Theorems 1.1 and 1.2, as long as the utilization factor of a set of tasks is less than or equal to $\ln 2$ the set of tasks can be scheduled

on one processor by *Next-Fit-2*. Since any class-2 task has a utilization factor of at most $(2^{\frac{1}{x}} - 1)$, the total utilization factors of the tasks assigned to each class-2 processor must be greater than $\ln 2 - (2^{\frac{1}{x}} - 1)$. The proof of the Lemma then follows from this fact.

Corollary 2.2: For $x > 2$, we have

$$N(NF2) < \frac{2S}{(x-1)(2^{\frac{1}{x-1}} - 1)} + 5.$$

Proof:

$$\begin{aligned} N(NF2) &= N_1 + N_2 \\ &< \frac{2S_1}{(x-1)(2^{\frac{1}{x-1}} - 1)} + \\ &\quad \frac{2S_2}{2(\ln 2 - (2^{\frac{1}{x}} - 1))} + 5. \end{aligned}$$

We next show that $(x-1)(2^{\frac{1}{x-1}} - 1) < 2(\ln 2 - (2^{\frac{1}{x}} - 1))$.

The right hand side of this inequality has its minimum value when x has its minimum possible value, and by Theorem 1.2, for $x > 1$, the left hand side has its maximum value also when x has its minimum possible value. By substituting the minimum possible value of x, which is 3, in both sides we prove our claim.

As a result, we get

$$N(NF2) < \frac{2S}{(x-1)(2^{\frac{1}{x-1}} - 1)} + 5.$$

Theorem 2.2: For $x > 2$, we have

$$r(NF2) \leq \frac{2}{(x-1)(2^{\frac{1}{x-1}} - 1)}.$$

Proof: Since $N^* \geq S$, by Corollary 2.2, we have

$$r(NF2) = \lim_{N^* \to \infty} \frac{N(NF2)}{N^*} \leq \frac{2}{(x-1)(2^{\frac{1}{x-1}} - 1)}.$$

The numerical values of bounds in Theorems 2.1 and 2.2 for different values of x are shown in Table 2. By looking at the values in this table, we see that the best choice for x is 3.

To establish a lower bound for $r(NF2)$, let us consider the following two examples.

Example 2.1: In this example the set of tasks to be scheduled consists of a combination of three different types of tasks:

Table 2: The upper bounds for worst-case performance ratio of *Next-Fit-2*.

x	Upper Bound
2	2.8853...
3	2.4142...
4	2.5648...
5	2.6426...
6	2.6900...
.	...
.	...
∞	2.8853...

- Type-1: Tasks with utilization factors $(2^{\frac{1}{2}} - 1)$.
- Type-2: Tasks with utilization factors $\ln 2 - (2^{\frac{1}{2}} - 1)$.
- Type-3: Tasks with utilization factors $1/n^2$, where n is a sufficiently large integer.

The appearance of tasks in the list is as follows:

- n repetitions of (one type-1 task, n type-3 tasks, one type-2 task, n type-3 tasks).

An optimal algorithm can schedule this set of tasks on $\frac{3n}{4}$ processors as follows:

- $\frac{n}{2}$ processors each with (one type-1 task, two type-2 tasks, $2n$ type-3 tasks)
- $\frac{n}{4}$ processors each with (two type-1 tasks, $4n$ type-3 tasks)

When $x = 2$, the *Next-Fit-2* algorithm would use $2n$ processors as follows:

- n processors each with (one type-1 task, n type-3 tasks)
- n processors each with (one type-2 task, n type-3 tasks)

Therefore, for $x = 2$, we have $\frac{N(NF2)}{N^*} = \frac{8}{3} = 2.6666...$.

Example 2.2: In this example the set of tasks to be scheduled consists of a combination of two different types of tasks:

- Type-1: Tasks with utilization factor $\frac{1}{2}$.

– Type-2: Tasks with utilization factor $\frac{1}{3}$.

The appearance of tasks in the list is as follows:

– n repetition of (one type-1 task, one type-2 task)

An optimal algorithm can schedule this set of tasks on $\frac{5n}{6}$ processors as follows:

– $\frac{n}{2}$ processors each with two type-1 tasks

– $\frac{n}{3}$ processors each with three type-2 tasks

When $x > 2$, the *Next-Fit-2* algorithm would use $2n$ processors as follows:

– n processors each with one type-1 task

– n processors each with one type-2 task

Therefore, for $x > 2$ we have $\frac{N(NF2)}{N^*} = 2.4$

Example 2.1 establishes a lower bound for $r(NF2)$ when $x = 2$, and the Example 2.2 establishes a lower bound for $r(NF2)$ when $x > 2$.

The best choice of x will depend on the type of tasks to be scheduled. But in general, when the tasks to be scheduled are not always a specific type, then the choice of $x = 3$ is the best. The lower bound and upper bound for $r(NF2)$ for $x = 3$ are: $2.4 \leq r(NF2) < 2.4143$.

2.2 The Complexity of Next-Fit-2

For each task $T_i, 1 \leq i \leq n$, this algorithm first determines its class by a single test, and then by an additional test it determines whether or not it is feasible on the active processor of its class. If the task is feasible on the active processor of its class, the algorithm assigns it to the processor. Otherwise, it picks a new processor to be the active one. Therefore, by a constant amount of computation this algorithm assigns a task to a processor. Hence, the time complexity of *Next-Fit-2* is $O(n)$.

If we consider a filled processor as the output of the algorithm, then the storage requirement of *Next-Fit-2* will depend on the number of active processors, which is *two,* and not the input size. Therefore, the space complexity of *Next-Fit-2* is 0(1).

3 The Algorithm Next-Fit-M

Let $T_1, T_2, \cdots, T_n$ be a set of tasks with utilization factors $u_1, u_2, \cdots, u_n$, respectively. Let M be a positive integer greater than 2. Divide the set of tasks into M different classes as follows. Let a task T_i belong to class-k if $(2^{\frac{1}{(k+1)}} - 1) < u_i \leq (2^{\frac{1}{k}} - 1)$, for $1 \leq k < M$, and let it belong to class-M if $0 < u_i \leq (2^{\frac{1}{M}} - 1)$, as shown below.

Class of task	Range of Utilization Factors
1	$(2^{\frac{1}{2}} - 1, 1]$
2	$(2^{\frac{1}{3}} - 1, 2^{\frac{1}{2}} - 1]$
3	$(2^{1/4} - 1, 2^{\frac{1}{3}} - 1]$
.	...
.	...
M	$(0, 2^{\frac{1}{M}} - 1]$

Similarly, divide the set of all processors into M different classes. A processor designated to process class-k tasks exclusively is referred to as a class-k processor. Note that since the utilization factor of a task in class-k is less than or equal to $(2^{\frac{1}{k}} - 1)$, by Theorem 1.1, at least k class-k tasks can be scheduled by the rate-monotonic algorithm on one class-k processor. The algorithm in Figure 2 assigns exactly k class-k tasks to each processor (except possibly the last processor) used from class-k, for $1 \leq k < M$. The algorithm assigns class-M tasks to class-M processors so that the total utilization factors of all the tasks assigned to each class-M processor does not exceed ln 2. Since *Next-Fit-M* assigns k class-k tasks to each class-k processor and since the utilization factor of any class-k task is greater than $(2^{\frac{1}{k+1}} - 1), 1 \leq k < M$, therefore, the total utilization factors of all the tasks assigned to any filled class-k, $1 \leq k < M$, processor is greater than $k(2^{\frac{1}{k+1}} - 1)$. Also, since any class-M task has a utilization factor of at most $(2^{\frac{1}{M}} - 1)$, the total utilization factor of all the tasks assigned to each filled class-M processor must be greater than $\ln 2 - (2^{\frac{1}{M}} - 1)$. Another important property of *Next-Fit-M* is that the number of class-k processors, $1 \leq k < M$, used by the algorithm is independent of the order of arrival of the tasks. In other words, except for class-M processors, any permutation of the tasks in the original list will result in the same number of processors used by *Next-Fit-M*. These properties are

```
/* Let P_{k,i} refer to the i^{th} processor of class-k */
1. for k = 1 to M do set N_k = 1;
2. set i = 1;
3. while i ≤ n do
     if T_i is a task from class-k, 1 ≤ k < M,
       then assign T_i to P_{k,N_k};
            if P_{k,N_k} has currently
              k tasks assigned to it,
              then set N_k = N_k + 1
            end-if;
       else /* T_i is a task from class-M */
            if the total utilization factor of
              all the tasks assigned to P_{M,N_M}
              is greater than ln 2 − u_i,
              then set N_M = N_M + 1
            end-if;
            assign T_i to P_{M,N_M}
     end-if;
     set i = i + 1
   end-while;
4. if P_{k,N_k}, 1 ≤ k ≤ M, has no task assigned to it
     then set N_k = N_k − 1;
```

The final values of $N_k, 1 \leq k \leq M$, will be the number of class-k processors used by the algorithm.

Figure 2: Algorithm Next-Fit-M

usefully exploited in the analysis of the worst case performance of *Next-Fit-M*. For convenience, let a processor of class-k, $1 \leq k \leq M$, be called *filled* if it has been used and it is not intended to assign any more tasks to it. Let a class-k processor ($1 \leq k \leq M$) be called *active* if it is the processor which will be considered for the assignment of the next class-k task.

3.1 Worst-Case Analysis of Next-Fit-M

For a given set of tasks, the total number of processors used by *Next-Fit-M*, denoted by $N(NFM)$, is $\sum_{k=1}^{M} N_k$, where N_k is the number of class-k processors used. Let $n_k, 1 \leq k < M$, denote the number of class-k tasks in the set of tasks to be scheduled. Then,

$$\begin{aligned} N(NFM) &= \sum_{k=1}^{M} N_k \\ &= n_1 + \lceil \frac{n_2}{2} \rceil + \lceil \frac{n_3}{3} \rceil + \cdots + \\ &\quad \lceil \frac{n_{M-1}}{M-1} \rceil + N_M. \end{aligned} \tag{1}$$

Let U_M denote the sum of the utilization factors of all the class-M tasks in the set of tasks to be scheduled. Since the total utilization factors of the tasks assigned to each filled class-M processor is greater than $\ln 2 - (2^{\frac{1}{M}} - 1)$, we have,

$$N_M < \frac{U_M}{\ln 2 - (2^{\frac{1}{M}} - 1)}.$$

Thus,

$$\begin{aligned} N(NFM) < {} & n_1 + \frac{n_2}{2} + \frac{n_3}{3} + \cdots + \\ & \frac{n_{M-1}}{M-1} + \frac{U_M}{ln2 - (2^{\frac{1}{M}} - 1)} \\ & +(M-1). \end{aligned}$$

The algorithm *Next-Fit-M* assigns one class-1 task to each class-1 processor it uses, and it assigns k class-k tasks to each class-k processor (except possibly the last one) it uses, for $2 \leq k < M$, therefore, asymptotically, each class-k task costs *Next-Fit-M* $\frac{1}{k}$ processor, for $1 \leq k < M$, and each class-M task T_i with utilization factor u_i costs Next-Fit-M, at most $u_i/(ln2 - (2^{\frac{1}{M}} - 1))$, processors. We, therefore, define a cost function f as:

$$f(u_i) = \begin{cases} \frac{1}{k} & \text{if } T_i \text{ is a class-}k \text{ task} \\ & \text{and } 1 \leq k < M \\ \frac{u_i}{\ln 2 - (2^{\frac{1}{M}} - 1)} & \text{if } T_i \text{ is a class-}M \text{ task} \end{cases}$$

Thus, in terms of the cost function f we can rewrite (1) as

$$N(NFM) < \sum_{i=1}^{n} f(u_i) + (M-1). \tag{2}$$

The term $M-1$ in (2) accounts for the last processors used from class-k, $2 \leq k \leq M$, for which we may not have enough class-k tasks to assign to them. Furthermore, if T_i is a class-k task, for $1 \leq k < M$, then we have $(2^{\frac{1}{k+1}} - 1) < u_i \leq (2^{\frac{1}{k}} - 1)$ and $f(u_i) = \frac{1}{k}$. Therefore,

$$\frac{f(u_i)}{u_i} = \frac{1}{ku_i} < \frac{1}{k(2^{\frac{1}{k+1}} - 1)}.$$

By Theorem 1.2, $k(2^{\frac{1}{k+1}} - 1)$ is monotonically increasing with k. Therefore, we have the following inequality:

$$\begin{aligned} \frac{f(u_i)}{u_i} < {} & \frac{1}{k(2^{\frac{1}{k+1}} - 1)}, \\ & \text{if } 2^{\frac{1}{k+1}} \leq u_i \leq 2^{\frac{1}{k}} - 1 \\ & \text{and } 1 \leq k < M. \end{aligned} \tag{3}$$

Now consider a set of n tasks. Assume an optimal scheduling algorithm uses N^* processors to schedule this set of tasks. Further assume that the tasks assigned to the i^{th} processor, $1 \leq i \leq N^*$, are $T_{i1}, T_{i2}, \cdots, T_{it_i}$ with utilization factors $u_{i1}, u_{i2}, ..., u_{it_i}$, respectively.

Since for any processor, the sum of the utilization factors of all the tasks assigned to it by any algorithm cannot exceed 1, we have

$$\sum_{j=1}^{t_i} u_{ij} \leq 1, 1 \leq i \leq N^*.$$

In terms of the cost function f, let $F_{i,M}$ denote the cost of the i^{th} processor. Then, if $T_{i1}, T_{i2}, ..., T_{it_i}$ are the tasks assigned to the i^{th} processor by *Next-Fit-M*, with utilization factors $u_{i1}, u_{i2}, ..., u_{it_i}$, respectively, we get

$$F_{i,M} = \sum_{j=i}^{t_i} f(u_{ij}), \text{ for } 1 \leq i \leq N^*.$$

Now, let $T' = \{T'_1, T'_2, \cdots, T'_t\}$ be a set of tasks with utilization factors $u'_1, u'_2, \cdots, u'_t$ respectively, with the following properties:

(i) $u'_m > 0, 1 \leq m \leq t$

(ii) $\sum_{m=1}^{t} u'_m \leq 1$.

(iii) $F_M = \sum_{m=1}^{t} f(u'_m) \geq F_{i,M}$, for $1 \leq i \leq N^*$.

Then, we can rewrite (2) as:

$$\begin{aligned} N(NFM) &< \sum_{i=1}^{n} f(u_i) + (M-1) \\ &= \sum_{i=1}^{N^*} \sum_{j=1}^{t_i} f(u_{ij}) + (M-1) \\ &\leq N^* F_M + (M-1). \end{aligned}$$

This implies that F_M is the worst-case performance ratio of *Next-Fit-M* because,

$$r(NFM) = \lim_{N^* \to \infty} N(NFM)N^* \leq F_M.$$

Thus, it is sufficient to find the upper bound on the number F_m, where

$$F_m = \sum_{m=1}^{t} f(u'_m), \text{ subject to } \sum_{m=1}^{t} u'_m \leq 1.$$

According to the property of the *Next-Fit-M* Algorithm, any permutation of a given set of tasks will result in the same number of class-k, $1 \leq k < M$, processors used by the algorithm. Therefore, without loss of generality, we may assume that $u'_1 \geq u'_2 \geq u'_3 \geq \cdots \geq u'_t$. Define a sequence of integers K_i, as follows:

$$K_1 = K_2 = 1.$$

For $i > 2$, K_i is the smallest integer S such that the following inequality is satisfied:

$$(2^{\frac{1}{S+1}} - 1) < 1 - \sum_{j=1}^{i-1} (2^{\frac{1}{K_j+1}} - 1). \qquad (4)$$

Some of the values of K_i are shown in Table 3. It is not hard to see that, for $i > 2$,

$$(2^{\frac{1}{K_i}} - 1) \geq 1 - \sum_{j=1}^{i-1} (2^{\frac{1}{K_j+1}} - 1). \qquad (5)$$

Table 3: Values of K_i in Worst-Case Analysis of *Next-Fit-M*.

i	K_i
1	1
2	1
3	4
4	30
5	2635
6	7145847

Theorem 3.1: For $M > 2$ and $K_i < M \leq K_{i+1}$, where K_i's are given by (4), we have

$$\begin{aligned} F_M &= \sum_{m=1}^{t} f(u'_m) \\ &< \sum_{j=1}^{i} \frac{1}{K_j} + \frac{1 - \sum_{j=1}^{i} (2^{\frac{1}{K_j+1}} - 1)}{\ln 2 - (2^{\frac{1}{M}} - 1)}. \end{aligned}$$

Proof: Let

$$S_M = \sum_{j=1}^{i} \frac{1}{K_j} + \frac{1 - \sum_{j=1}^{i} (2^{\frac{1}{K_j+1}} - 1)}{\ln 2 - (2^{\frac{1}{M}} - 1)}.$$

The numerical values of S_M for different values of M are listed in Table 4.

Table 4: Values of S_M in Worst-Case Analysis of *Next-Fit-M*

M	S_M
3	2.3960...
4	2.3404...
5	2.2920...
6	2.2900...
7	2.2888...
8	2.2879...
9	2.2873...
10	2.2868...
11	2.2864...
12	2.2860...
13	2.2858...
30	2.2841...
31	2.2837...
∞	2.2837...

We attempt to show that F_M is upper bounded by S_M. To prove our claim, we consider the following three cases:

Case 1: T_1' is not a class-1 task. Then $u_m' \leq (2^{\frac{1}{2}}-1), 1 \leq m \leq t$. By (3) we have

$$f(u_m') < \frac{u_m'}{2(2^{\frac{1}{3}}-1)}, 1 \leq m \leq t.$$

Therefore,

$$\begin{aligned}\sum_{m=1}^{t} f(u_m') &< \frac{\sum_{m=1}^{t} u_m'}{2(2^{\frac{1}{3}}-1)} \\ &\leq \frac{1}{2(2^{\frac{1}{3}}-1)} < 1.93 < S_M.\end{aligned}$$

Case 2: T_1' is a class-1 task but T_2' is not a class-1 task. Then,

$$\sum_{m=2}^{t} u_m' < 1-(2^{\frac{1}{2}}-1),$$

and

$$u_m' \leq (2^{\frac{1}{2}}-1), 2 \leq m \leq t.$$

By (3) we have

$$f(u_m') < \frac{u_m'}{2(2^{\frac{1}{3}}-1)}, \ 2 \leq m \leq t.$$

Therefore,

$$\begin{aligned}\sum_{m=1}^{t} f(u_m') &< f(u_1') + \frac{\sum_{m=1}^{t} u_m'}{2(2^{\frac{1}{3}}-1)} \\ &< 1 + \frac{1-(2^{\frac{1}{3}}-1)}{2(2^{\frac{1}{3}}-1)} \\ &< 2.1269 < S_M.\end{aligned}$$

Case 3: Both T_1' and T_2' are class-1 tasks. First we consider the subcase where $T_1' \in$ class-K_1, $T_2' \in$ class-K_2, $\cdots$, $T_i' \in$ class-K_i. Then,

$$\sum_{m=i+1}^{t} u_m' < (1-\sum_{j=1}^{i}(2^{\frac{1}{K_j+1}}-1)). \tag{6}$$

Since $M \leq K_{i+1}$, from equation (6), we see that $T_m' \in$ class-M for all $i+1 \leq m \leq t$. Therefore,

$$\begin{aligned}F_M &= \sum_{m=1}^{t} f(u_m') \\ &< \sum_{j=1}^{i} \frac{1}{K_j} + \frac{1-\sum_{j=1}^{i}(2^{\frac{1}{K_j+1}}-1)}{\ln 2-(2^{\frac{1}{M}}-1)} \\ &= S_M.\end{aligned}$$

We next show that F_M will have its maximum value when $T_j \in$ class-K_j, for all $1 \leq j \leq i$.

Suppose $T_j \notin$ class-K_j for $j = i > 2$. Then, we have $u_m' \leq (2^{\frac{1}{K_j+1}}-1)$, for all $i \leq m \leq t$. This will reduce the above value of F_M by $\frac{1}{K_i}$ and increase it by a maximum value of

$$\frac{(2^{\frac{1}{K_i+1}}-1)}{(K_i+1)(2^{\frac{1}{K_i+2}}-1)}.$$

Thus, total change in the above value of F_M will at most be

$$\frac{(2^{\frac{1}{K_i+1}}-1)}{(K_i+1)(2^{\frac{1}{K_i+2}}-1)} - \frac{1}{K_i}. \tag{7}$$

We claim that

$$\frac{(2^{\frac{1}{K_i+1}}-1)}{(K_i+1)(2^{\frac{1}{K_i+2}}-1)} < \frac{1}{K_i}. \tag{8}$$

From Theorem 1.2, we know that

$$K_i(2^{\frac{1}{K_i+1}}-1) < (K_i+1)(2^{\frac{1}{K_i+2}}-1).$$

Dividing both sides of this inequality by $K_i(K_i+1)(2^{\frac{1}{K_i+1}}-1)$, we get

$$\frac{(2^{\frac{1}{K_i+1}}-1)}{(K_i+1)(2^{\frac{1}{K_i+1}}-1)} < \frac{1}{K_i}.$$

Thus, in this case, the value of F_M decreases further.

Now, suppose that $T_j \notin$ class-K_j for $j = i-1$. Then, repeating the argument, further change in the value of F_M will be given by expression (7) with i replaced by $i-1$. In view of inequality (8), this change also may only result in further decreasing the value of F_M. Repeating these argument for $j = i-2, i-3, \cdots, 3$, it follows that F_M has its maximum value when we have $T_j \in$ class-K_j, for all $1 \leq j \leq i$, and that this maximum value is bounded above by S_M.

This completes the proof.

Comparing the values of F_∞ and F_{31} (Table 4), we see that little improvement on performance is achieved beyond M = 31. Therefore, for practical purposes, one may choose any M within the region $3 \leq M \leq 31$.

To examine the tightness of the bound given by Theorem 3.1, we consider the following example.

Example 3.1: Let $M = K_{i+1}$, for some $i \geq 2$. Let N be an integer divisible by M, and ϵ be a sufficiently small quantity.

Consider a set of tasks consisting of:

> N subsets of i tasks with utilization factor u_j of the tasks in each subset given by $u_j = (2^{\frac{1}{K_j+1}} - 1 + \epsilon)$, for $1 \leq j \leq i$, and
>
> N tasks, with utilization factor of each task denoted by u_{i+1} as follows:

$$u_{i+1} = (1 - \sum_{j=1}^{i} u_j - i\epsilon).$$

Since $M = K_{i+1}$, by (4) and (5), we have

$$(2^{1/(M+1)} - 1) < u_{i+1} < (2^{\frac{1}{M}} - 1).$$

Therefore, *Next-Fit-M* assigns exactly M tasks, each with utilization factor u_{i+1}, to each class-M processor it uses, and it assigns K_j tasks each with utilization factor u_j to each class-K_j processor it uses, for $1 \leq j \leq i$. Thus, the total number of processors needed by *Next-Fit-M* would be

$$N(NFM) = \sum_{j=1}^{i} \frac{N}{K_j} + \frac{N}{M}.$$

Since $\sum_{j=1}^{i+1} u_j = 1$, the number of processors, N^*, needed by an optimal algorithm is $\geq$ N.

Therefore,

$$\frac{N(NFM)}{N^*} = \sum_{j=1}^{i} \frac{1}{K_j} + \frac{1}{M} = \sum_{j=1}^{i+1} \frac{1}{K_j}.$$

Let $Q_M = \sum_{j=1}^{i+1} \frac{1}{K_j}$. The numerical values of Q_M for various values of M, along with those of S_M in Theorem 3.1 are shown below.

M	Q_M	S_M
4	2.2500	2.3404
30	2.2833	2.2841
2635	2.2837	2.2837
.	.	.
.	.	.
∞	2.2837	2.2837

Therefore, for small values of M, the bound given by Theorem 3.1 is close to tight, and for large values of M, it is very tight.

When the set of tasks to be scheduled does not contain any task with utilization factor in the range $(2^{\frac{1}{2}} - 1, \frac{1}{2}]$, then it is not too difficult to show that $r(NFM) < 1.911$ (Davari 1985, Davari & Dhall 1986c).

3.2 The Complexity of Next-Fit-M

For each task $T_i, 1 \leq i \leq n$, this algorithm first determines its class, and then assigns the task to the active processor of its class. Since the class of a task can be determined in $O(logM)$ time and there is only one active processor in each class at any time, the time-complexity of this algorithm is $O(nlogM)$. From Table 4, we see that approximate values of S_3, S_{12}, S_{31}, and S_∞ are, respectively, 2.3960, 2.2860, 2.2837, and 2.2837. Thus, as M increases beyond a certain value, say, 31, the gain in the performance of the algorithm is negligible. For all practical purposes, therefore, one can fix the value of M in the range $3 \leq M \leq 31$.

As a result, M can be considered as a constant. Hence, the time-complexity of *Next-Fit-M*, with $M \leq 31$, is $O(n)$.

If we consider a filled processor as the output of the algorithm, then the storage requirement of *Next-Fit-M* will depend on M, the number of active processors, and not on n, the input size. Therefore, the space complexity of *Next-Fit-M* is O(1).

4 Conclusions

In this paper we studied the problem of partitioning a set of periodic-time-critical tasks into different groups, subject to the conditions that: (i) each group of tasks can be feasibly scheduled on a single processor using the rate-monotonic algorithm (which is the best static priority driven algorithm available for scheduling this type of tasks on a single processor system); and (ii) the number of processors required is minimum.

There exists three off-line algorithms for this problem. Since the nature of the arriving tasks in an on-line processing is unpredictable, an on-line scheduling algorithm is expected to be more difficult than an off-line one. In general the performance of an on-line scheduling algorithm is substantially affected by the permutation of tasks in a given set.

In this paper we presented two $O(n)$-time and $O(1)$-space on-line algorithms, called *Next-Fit-2* ($NF2$) and *Next-Fit-M* (NFM). These algorithms are less complex than the existing off-line algorithms. The Worst-case Performance Ratio (WPR) of $NF2$ was shown to be less than 2.4143, and WPR of NFM was shown to be less than 2.2838. These ratios are comparable to those of the existing off-line algorithms. These ratios are also comparable to the on-line algorithms $RM-FF$ and $RM-BF$ (Oh & Son 1994). However, the performance is not as good as that of *RRM-FF* and *RRM-BF* (Oh & Son 1994). But the redeeming feature of the algorithms presented here is that their time complexity is linear and they require a constant amount of space, as opposed to the time complexity of $O(n \log n)$ and space requirement of $O(n)$. It may also be pointed out that when NFM is applied to the special case in which the set of tasks to be scheduled does not contain any task with utilization factor in the range $(2^{\frac{1}{2}}-1, \frac{1}{2}]$, then the WPR of this algorithm is less than 1.911.

All of the algorithms considered for this problem, so far, are preemptive algorithms. Preemptive scheduling does cost overhead. The overhead is assumed to be negligible in the analysis of all algorithms reported in literature. It would be interesting to analyze the performance of these algorithms by including overhead penalty for preemptions. Also, in the area of non-preemptive scheduling for periodic time critical tasks, not much work has been done. It would be interesting to investigate the behavior of non-preemptive algorithms for this problem.

References

[1] Agrawal G., Chen B., Zhao W., & Davari S. (1994) Guaranteeing Synchronous Message Deadlines with the Timed Token Medium Access Control Protocol. *IEEE Transactions on Computers*, 43(3), p. 327-339.

[2] Bannister J. and Trivedi K. (1983) Task Allocation in Fault Tolerant Distributed Systems. *ACTA Informatica*, 20, p. 261-281.

[3] Davari S. (1985) Scheduling Periodic-Time Critical Tasks on Multiprocessor Computing Systems. *Ph.D. Dissertation,* University of Oklahoma, Norman, Oklahoma.

[4] Davari S. and Dhall S. K. (1986a) On a Real-Time Task Allocation Problem. *Proceedings of the Nineteenth Annual Hawaii International Conference on System Sciences.* p. 133-141.

[5] Davari S., and Dhall S.K. (1986b) A Simple On-line Algorithm For Real-Time Tasks Allocation, *Proceedings of the Twentieth Conference on Information Sciences and Systems, Princeton, New Jersey.* p. 178-182.

[6] Davari S. and Dhall S.K. (1986c) An On-line Algorithm For Real-Time Tasks Allocation. *Proceedings of the Real-Time Systems Symposium,* p.194-200.

[7] Davari S., Leibfried T., Natarajan S., Pruett D., Sha L., & Zhao W. (1993) Real-Time Issues in the Design of the Data Management System

for Space Station Freedom. *First IEEE Workshop on Real-Time Applications, New York, NY,* p. 161-165.

[8] Davari S. and Sha L. (1992) Sources of Unbounded Priority Inversions in Real-Time Systems and a Comparative Study of Possible Solutions. *ACM OSR.* 26(2), p.110-120.

[9] Davari S. and Zhao W. (1991) RMS Aids Real-Time Scheduling. *RICIS Review.* 3(1), p. 2,8-9.

[10] Dhall S.K. (1977) Scheduling Periodic Time-Critical Jobs on Single Processor and Multiprocessor Systems. *University of Illinois, Technical Report No. UIUCDCS-R-77-859.*

[11] Dhall S.K. and Liu C.L. (1978) On a Real-Time Scheduling Problem. *Operations Research.* 26(1), p. 127-140.

[12] ESA. (1990) *Statement of Work, Hard Real-Time OS Kernel.* On-Board Data Division, European Space Agency.

[13] Johnson D.S. (1974) Fast Algorithms for Bin-Packing. *J. Compt. System Sci.* 8, p 272-314.

[14] Johnson D.S., Demers A., Ullman J.D., Garey M.R., & Graham R.L. (1974) Worst-Case Performance Bound For Simple One Dimensional Packing Algorithms. *SIAM Journal of Computing,* 3, p. 299-325.

[15] Labetoulle J. (1974) Some Theorems on Real-Time Scheduling, in *Computer Architecture and Networks,* E. Gelenbe, R. Mahl (Eds.), North Holland Publishing Co., p. 285-298.

[16] Lee C.C., and Lee D.T. (1985) A Simple On-line Bin-Packing Algorithm. *Journal of the ACM.* 32(3), p.562-572.

[17] Leung J.Y.T. and Whitehead T. (1982) On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks. *Performance Evaluation.* 2, p. 237-250.

[18] Leung Joseph Y.T. (1989) A New Algorithm For Scheduling Periodic, Real-Time Tasks. *Algorithmica.* 4, p. 209-219.

[19] Locke D.C. (1992) Software Architecture for Hard Real-Time Applications: Cyclic Executives vs. Fixed Priority Executives. *Journal of Real-Time Systems.* 4, p.37-53.

[20] Liu C. L. and Layland J.W. (1973) Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *JACM.* 20, p.46-61.

[21] Oh Y. and Son S.H. (1994) Allocating Fixed-Priority Periodic Tasks on Multiprocessor Systems. *Journal of Real-Time Systems.* 5, p.1-33.

[22] Serlin O. 1972) Scheduling of Time-Critical Processes. *Proceedings of the SJCC.* p. 925-932.

[23] Sha L. and Goodenough J. B. (1990) Real-Time Scheduling Theory and Ada. *IEEE Computer.* 23, p.53-62.

[24] Sha L., Rajkumar R., & Lehoczky J. P. (1990) Priority Inheritance Protocol: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers.* 39(9), p.1175-1185.

[25] Sha L. and Sathaye S. (1993) A systematic Approach to Designing Distributed Real-Time Systems. *IEEE Computer.* 26(9), p. 68-78.

[26] Sprunt B., Sha L., & Lehoczky J.P. (1989) Aperiodic Task Scheduling for Hard Real-Time Systems. *Journal of Real-Time Systems.* 1, p.27-60.

[27] Yao A.C. (1980) New Algorithms For Bin-Packing. *Journal of the Association for Computing Machinery* 27(2), p. 207-227.

A Semi-Distributed Load Balancing Model for Parallel Real-time Systems

Kayhan Erciyeş, Öznur Özkasap and Nilgün Aktaş
Ege University Computer Eng. Dept.
35100 Bornova, İzmir, Turkey
e-mail: erciyes@baum01.ege.edu.tr, ozkasap@baum01.ege.edu.tr, aktas@baum01.ege.edu.tr

Keywords: parallel processing, load balancing, real-time system, deterministic scheduling

Edited by: Marcin Paprzycki and Janusz Zalewski
Received: February 18, 1994 **Revised:** November 11, 1994 **Accepted:** January 14, 1995

We propose static and dynamic load balancing policies for parallel real-time systems. A parallel real-time system in this context is considered as a computational environment consisting of a number of processors where stringent timing requirements of processes should be met. This would encompass massively parallel systems at one end of the spectrum and a group of computers connected by a local network at the other end. The static and dynamic load balancing policies developed are suitable for both types of systems with parameters such as communication costs to be tuned for each environment. For massively parallel processing systems, we introduce the concept of a domain which is a pool of processors and is governed locally for various services such as dynamic load balancing. The dynamic load balancing is implemented by central load balancers per domain which make use of the group communication facility for distributed communication with the other load balancers. This semi-distributed approach eliminates the need for maintaining a central node or replicated data by providing local data and control confined to that domain. The distributed data and control transfer is performed among the servers of the domains. The static scheduler however works off line for tasks with known characteristics such as execution time, communication constraints and deadlines prior to their execution which would be the usual case for hard real-time tasks.

1 Introduction

Recent developments in hardware technologies have made it possible to build systems consisting of clusters of processors usually referred to as Massively Parallel Processing (MPP) systems. MPP systems are increasingly finding many applications in hard real-time systems such as particle physics. A heterogeneous parallel real-time system is envisioned as a MPP system and host computers connected by a real-time network as shown in Fig. 1.

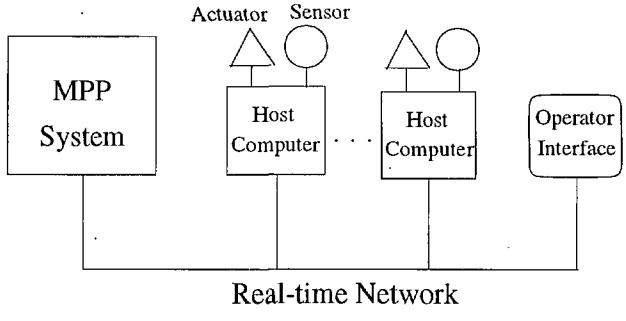


Figure 1: A Parallel Real-Time System

Our study focuses on load balancing in the MPP system of such an environment. We propose a deterministic scheduler, a dynamic load balancing mechanism and operating system modules to support dynamic load balancing. The central theme is to consider the MPP system as a

collection of domains of processors which are managed centrally for various services in a domain and as distributed for interdomain services.

The system comprises minimum functionality required from a distributed memory parallel system to achieve coarse to medium grain parallelism for single or multiple applications. The deterministic scheduler accepts task graphs and the deadlines of tasks in the case of real-time tasks, as its input. It first puts the tasks with heavy communication into sets to be allocated to domains and then calculates various heuristic values for each task in the set and assigns these tasks to the processors in the domain according to these heuristic values. Different than the previous work in this area, namely, static scheduling of real-time tasks (Liu & Layland 1973), (Stankovic & Ramamritham 1988), we have considered communication costs among tasks and defined a heuristic which is a function of communication costs and is a component in the total heuristic value for a task.

The dynamic load balancing mechanism uses a semi-distributed approach. The periodically invoked central load balancer in a domain of processors tries to establish balance among them by first balancing the load within individual domains, then among different domains of the system in the second step. The operating system supports the dynamic load balancing mechanism by providing necessary group communication primitives for multicast communication.

2 Static Scheduling

Task Scheduling is one of the most challenging problems in parallel and distributed computing. Informally, the scheduling problem arises because the concurrent parts of a parallel program must be arranged in time and space so that the overall execution time of the parallel program is minimized. It is known to be NP-complete in its general form as well as several restricted cases. In an attempt to solve the problem in the general case, a number of heuristics have been introduced. The effectiveness of these heuristics depends on a number of factors such as grain size, interconnection topology, communication bandwidth and program structure (El-Rewini 1989).

The maximization of the speedup of a parallel program on a target parallel computer requires the allocation of the tasks among the processors in such a way that the total computational load is distributed as evenly as possible. To minimize the amount of processor idle time, the time required to perform necessary interprocess communication is minimized (Sadayappan & Ercal 1987).

Efe (Efe 1982) developed a heuristic allocation algorithm to balance processor load and to minimize communication cost. His algorithm consists of two phases. First, tasks are clustered with each other to optimize the communication cost and each cluster of tasks is assigned to a processor. Then, tasks are shifted from overloaded to underloaded processors in order to meet load-balance constraints. The algorithm is repeated until a satisfactory degree of load-balancing is achieved whereas we first group closely related tasks for domains and then allocate them individually to the processors of domains.

2.1 Problem Statement

This section describes the components of the static scheduling model we have developed. In general, there are four components in any scheduling system:

1. the target machine
2. the parallel tasks
3. the generated schedule
4. the performance criterion

In our model, the parallel application that contains real-time tasks is characterized by an acyclic directed graph G(V,E) as shown in Fig. 2. Vertex weights $V = \{t_i : i = 1, 2, ..., N\}$ represent computational load, or worst-case execution time of the tasks, and edge weights $E = \{c_{ij} :$ for $\forall t_i, t_j$ where $p_{ij} = 1\}$ represent interprocess communication costs. Precedence relation between tasks is defined as follows :

$p_{ij} = 1$ if there exists a precedence relation
$>$ between t_i, t_j task pairs
$p_{ij} = 0$ otherwise

Since, real-time systems are static and it is assumed that all task characteristics are known a priori (Stankovic & Ramamritham 1988), $\forall t_i$ of

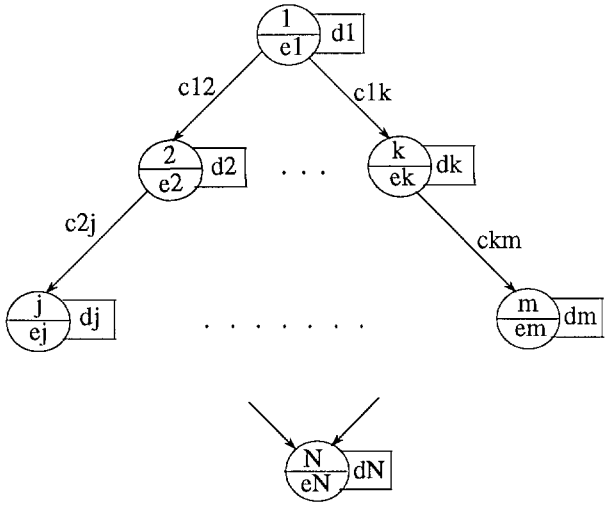


Figure 2: A General Task Graph.

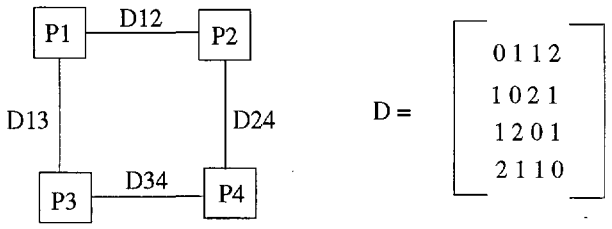


Figure 3: A Target PPA (Parallel Processing Architecture) and its delay matrix.

the parallel application has known timing characteristics such as execution time (e_i for $\forall t_i$) and deadline (d_i for $\forall t_i$)

The topologies of the target Parallel Processing Architecture (PPA) are in the form as shown in Fig. 3. $P = \{P_i : i = 1, 2, ..., M\}$ is a set of homogenous processors with local memory, which communicate via message passing paradigm or channel structures. A delay matrix, D is introduced to represent physical overheads among processors in the parallel processing system.

$$D = \{D_{ij} : \text{number of hops between processors } P_i \text{ and } P_j \text{ for } \forall P_i, P_j \text{ where } i \neq j\}$$

The intraprocessor overhead D_{ii} for $\forall i$ is assumed to be zero. Also, speeds of processors in the system are assumed as equal, that is, they are homogeneous processors.

We considered that N > M where N is the number of tasks in task graph and M is the number of processors in the system.

Definition: A *domain* is a group of fixed-size processors, which includes closely related, therefore heavily communicating tasks of an application. Processors forming a domain are selected as physically closed ones in the parallel processing architecture.

Definition: If tasks t_i and t_j are mapped to processors P_i and P_j respectively, and $p_{ij} = 1$ (that is, there exists a precedence relation between t_i and t_j), then the *cost function* is defined as $F(C) = min(\sum c_{ij} D_{ij}$ for $\forall t_i, t_j)$

2.2 Scheduling Model

The deterministic scheduler of the load balancing model proposed works off-line for scheduling tasks of a parallel application. Inputs of the scheduler are a task graph that shows precedence constraints, communication costs among tasks, execution times and deadlines of tasks, and interprocessor communication delays for the PPA. The off-line scheduler works in two phases:

Domain Allocation : Tasks with heavy communication are put into groups to be allocated to domains.

Task-to-Processor Mapping : Heuristic values for each task in a domain are calculated and the tasks are assigned to the individual processors in their domain.

Domain Allocation Algorithm: The Domain Allocation (DA) Algorithm that is shown in Fig. 4 considers timing constraints of tasks such as execution time, precedence constraints among tasks and interprocess communication. The procedure is to group tasks into domains of the parallel processing system. The objectives considered during this procedure are putting tasks with heavy communication into the same domain in order to minimize communication costs and balancing them by trying to ensure that total execution time of tasks allocated to each domain are approximately the same. The result obtained is an allocation scheme that permits parallel execution of tasks in the target PPA.

Task-to-Processor Mapping Algorithm The Task-to-Processor Mapping (TPM) Algorithm is shown in Fig. 6. Processor allocation for tasks is performed by calculating heuristic values for each task as described in section 2.4 and assigning these tasks to the individual processors in their domain according to these heuristic values.

Once the first step of the scheduling model,

```
Inputs:  Task Graph (TG) with N tasks;
         number of domains (DN);
. TE = Σ_{i=1}^{N} e_i;
. AE = TE/DN;
. Sort c_ij in descending order for ∀ t_i, t_j
  where p_ij = 1;
. Get first element c_ij of the sorted list;
. Set current domain (CD) to first domain;
. Put t_i,t_j into CD;
.DO UNTIL ∀ t_i allocated to a domain
{ if ( Σ e tasks in CD < AE )
    Find max c_ij for t_i and t_j
    that are not allocated to a domain;
    Put t_j into CD
    Delete_comm_cost();
  else
    if(CD < DN) CD++; /*Set current domain
      to next domain*/
    else set CD to domain with min(Σ e);
    if there exists c_ij where
    t_i and t_j are not allocated to a domain
      Put t_i, t_j into CD;
      Delete_comm_cost();
    else if (Σ e of tasks in domain DN==0)
      DN--;
      AE = TE / DN;
    Get next element c_ij;
    if ( t_i or t_j is not allocated to a domain)
      CD = Domain of task t that is allocated;
      Put task into CD;
      Delete_comm_cost();
}
```

Figure 4: Domain Allocation Algorithm

```
SEQ
  Allocate Domains
  PAR i=1 FOR DN
    Do Task-to-Processor Mapping for domain i
```

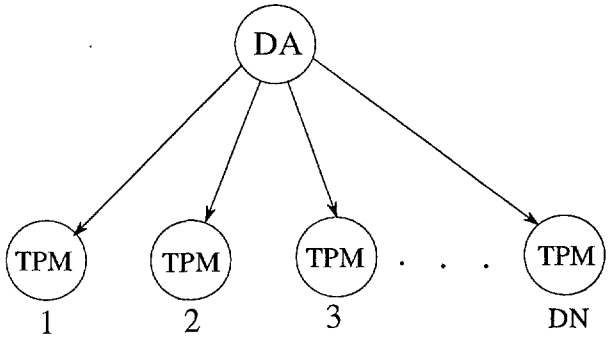


Figure 5: Task to Processor Mapping.

namely, domain allocation is performed, task-to-processor mapping should be completed for each domain of the parallel processing system. This second phase can be performed in parallel for each domain as shown in Fig. 5. This makes the scheduling process faster especially for MPP systems where there are hundreds of processors, and many domains.

2.3 Task Graph Generation

A task graph generator, which generates schedulable task graphs with tasks having timing and precedence constraints is developed. This strategy allows us to evaluate performance of domain allocation and task-to-processor mapping algorithms using various heuristic functions on different parallel applications characterized by generated task graphs.

Task Graph Generation Algorithm: The approach uses the strategy of randomly assignment of tasks to the degrees or positions of the graph. Then predecessor and successor task(s) of each task are determined. Finally, timing constraints of each task and communication costs between related tasks are assigned as shown in Fig. 7.

2.4 Scheduling heuristics

During task-to-processor mapping, we have used the list scheduling method. List Scheduling is a

```
Inputs:  Task Graph (TG) with N' tasks
         allocated for the domain;
         Delay Matrix D;
. n_i = number of immediate predecessors
  of t_i for i = 1, ..., N'
. Task_type=READY for ∀t_i where n_i = 0
. Insert READY tasks into Sched_list
  according to heuristic values at time =0
.WHILE (Sched_list is not empty)
  {Get a task t_i from the Sched_list;
   Switch(Event_type)
    case READY :
      map t_i to an idle processor P_j;
      Event_type = FINISHED;
      Time = Finish_time of t_i on P_j
      Insert_event into Sched_list;
    case FINISHED :
      For each immediate successor t_k of t_i
      n_k = n_k - 1;
      if (n_k == 0) Event_type = READY
        Time=Finish_time of t_i on P_j;
        Insert_event into Sched_list;
  }
```

Figure 6: Task to Processor Mapping Algorithm.

class of scheduling heuristics in which tasks are assigned priorities and placed in a list ordered in decreasing priority. Whenever tasks contend for processors, the selection of tasks to be immediately processed is done on the basis of priority with the higher priority tasks being assigned to processors first. The heuristic functions that determine the priorities of processes can be explained as follows:

EDF (Earliest Deadline First): Priority is given to the real-time tasks with the earliest-deadline.

MLF (Minimum Laxity First): Priority is given to the real-time tasks with minimum laxity where task laxity = task deadline - task execution time.

H3:min(EDF*W1+MLF*W2): The first two heuristic functions are combined by using weight values. We have developed a new heuristic function by using simulation results for EDF and MLF heuristic functions.

3 Dynamic Load Balancing

The random arrival of processes in a parallel processing system can cause some processors to be heavily loaded while other processors are idle or

- *Randomly assign N tasks to d degrees in task graph*
- *Determine predecessor(s) and successor(s) of each task under the constraints such as number of tasks in a degree and number of degrees in graph*
- *Handle timing constraints (e_i for $\forall t_i$) (d_i for $\forall t_i$)*
- *$E = \{c_{ij}$: for $\forall t_i, t_j$ where p_{ij}=1$\}$ of each task generated*

Figure 7: Task Graph Generation Algorithm

lightly loaded. Dynamic load balancing improves the performance by transferring tasks from heavily loaded processors, where service is poor, to lightly processors where the task can take advantage of computing capacity that would otherwise go unused.

Most of the methods used for dynamic load balancing are either fully distributed or centralized methods. Neither fully distributed nor centralized load balancing policies are known to yield good performance for MPP systems. Fully distributed algorithms use a small amount of information about the state of the system. Small systems can yield good performance with limited information, but this may not be true for large systems. Despite the fact that fully distributed algorithms incur less overhead due to message exchange, this overhead linearly increases with the system size. Centralized algorithms do have the potential of yielding optimal performance, but require accumulation of global information which can become a formidable task. The storage requirement for maintaining the state information also becomes prohibitively high with a centralized model of the large system. For a large system consisting of a hundred or thousands nodes, the central scheduler will become a bottleneck and lower the throughput.

Besides, centralized models are highly vulnerable to failures. The failure of any software or hardware component of the central scheduler can stop the operation of the whole system.

In our study, a semi-distributed dynamic load balancing model is developed for a distributed memory computer system. In this case, the processors of an MPP system are divided into domains of fewer processors which are managed centrally for various services and distributed for others. Domains are allocated dynamically during run time in some researches (Kremien et. al. 1993), whereas our system is divided into domains in a

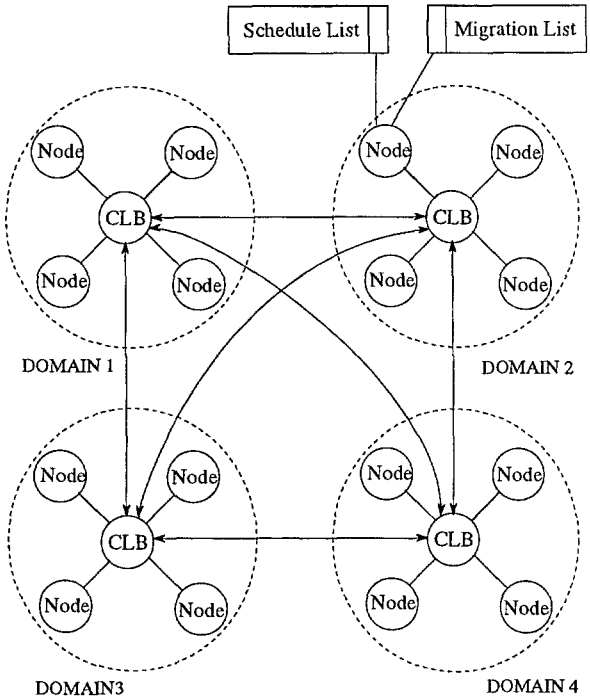


Figure 8: Semi-distributed System Model

static manner before the system starts. A group management module designed for the underlying distributed operating system provides the necessary group communication in multicast mode for the manager processes. A system process in each domain called Central Load Balancer (CLB) first tries to balance the load within its domain. If this is not possible, it communicates with other CLBs to find a destination node for the candidate process for migration as depicted in Fig. 8.

This model proposes a two level load balancing strategy. At the first level, load balancing is carried out within individual domains where the central node of each domain acts as a centralized controller for its own domain. At the second level, the load is balanced among different domains of the system, thus providing a distributed environment among domains. The design of such a strategy involves designing an algorithm for performing optimal task scheduling and load balancing within a domain as well as among domains and developing efficient means for collecting state information at interdomain and intradomain levels.

Central load balancers are responsible for dynamically assigning processes to individual nodes of the domain, transferring the load to other domains if required, and maintaining the load status of the domain and nodes. As a result of load balancing and sharing, a process can be completed earlier due to the utilization of otherwise idle or lightly loaded processors.

3.1 Real-time Approach

In a conventional multitasking operating system, processes are interleaved with higher importance (or priority) processes receiving preference. Little or no account is taken of deadlines. This is clearly inadequate for real-time systems. These systems require scheduling policies that reflect the timeliness constraints of real-time processes .

A realistic hard real time system must guarantee both periodic and non-periodic hard real-time processes on the same processor, utilize spare time by non-critical processes, initialize static allocation of periodic processes and migrate aperiodic processes for response to changing environment conditions or local overload.

Schedulers produce a schedule for a given set of processes. If a process set can be scheduled to meet given pre-conditions, the process set is termed feasible. A typical pre-condition for hard real-time periodic processes is that they should always meet their deadlines. An optimal scheduler is able to produce a feasible schedule for all feasible process sets conforming to a given precondition. For a particular process set, an optimal schedule is the best possible schedule according to some pre-defined criteria. Typically a scheduler is optimal, if it can schedule all process sets.

The system model that has been used allows both periodic and aperiodic processes. Precedence constraints among processes are enforced by using the process's start times and deadlines and no process resource requirements are considered. Context switch have zero cost and multinode, multi-domain systems with dynamic process allocation is allowed.

3.1.1 Deadline Characteristics

Periodic processes are characterized by their period and their required execution time per period. For each periodic process, its period must be at least equal to its deadline. That is, one invocation of a process must be completed before successive invocations. This is termed as the runnability constraint as shown below:

$computation\_time \leq deadline \leq period$

The activation of an aperiodic process is essentially, a random event and is usually triggered by an action external to the system. Aperiodic processes also have timing constraints associated with them; i.e. having started execution, they must complete within a predefined time period. It is not guaranteed that aperiodic processes will certainly meet their deadlines. If it is not possible to schedule an aperiodic process on any processor before its deadline, this process is said to be unschedulable. Aperiodic processes can be invoked at any time.

It has been showed that the algorithms that are optimal for single processor systems are not optimal for increased numbers of processors. In a multiprocessor or distributed system, processes that are considered likely to miss their deadlines have to be migrated to other processors. But it has also been showed that it is better to statically allocate periodic processes rather than let them migrate and, as a consequence, potentially downgrade the system's performance (Audsley & Burns 1990).

3.1.2 Scheduling Policy

Each process is characterized by (A,S,C,D) known at the time of process arrival, where A is the process's arrival time, S is the earliest possible time at which its execution may begin (start time), C is the maximum computation time and D is the deadline by which it must complete its execution. An aperiodic process is described by (A,S,C,D). For each periodic process, an (A,C,P) describes its arrival time, computation time, and period.

The algorithm schedules sets of processes ordered by increasing deadlines. Given such a set, the algorithm selects the first process and schedules it as near to its start time as possible (i.e. at the earliest available time after its start time). The process is scheduled by simply accumulating all unused processor time past the process's start time until sufficient computation time is found. If the resulting schedule permits this process to complete before its deadline then the process is schedulable, else it is unschedulable.

The scheduling information used by this algorithm is recorded in a list. Each element of the list represents a time slot already assigned to a process, and has four fields: starting time, ending time, a pointer to the next list element, a pointer to the previous list element. Given this list, the process schedulability is analyzed by searching the list for available time intervals between two elements. This search starts at an element compatible with the process start time and ends at a time point compatible with the process deadline or when the accumulated length of available time is equal to the process computation time. The process is schedulable if sufficient computation time is found before its deadline during this search, else the algorithm reports the process as unschedulable.

In our system, aperiodic processes do not have hard deadlines, hence they can be migrated to other processors when they can not be scheduled on present processor. On the other hand, periodic processes have hard deadlines and they can not be migrated to other processors. They are statically allocated when the system starts by the deterministic scheduler as explained in Section 2.

In order to schedule an aperiodic real-time process with a soft deadline dynamically, a modified form of Bryant and Finkel's algorithm is employed.

3.1.3 Bryant and Finkel's Algorithm

Bryant and Finkel's algorithm (Bryant & Finkel 1981) is a dynamic and physically distributed algorithm. In our system, Bryant and Finkel's algorithm is used in a semi-distributed fashion, considering the deadlines of the aperiodic processes. To make a decision, processors cooperate by sending negotiation messages. The decisions are sub optimal and heuristic approach is used to find solution.

A newly arriving aperiodic process can be called as schedulable only if its scheduling does not danger previously scheduled processes. First, the new aperiodic process is placed in order into the list, which holds all previously scheduled processes on this processor. Then processes in the list are rescheduled, using the algorithm explained above. If any of the previously scheduled processes is unschedulable now, then newly arriving aperiodic process is determined as unschedulable on this processor. Otherwise it is schedulable. When a process is determined as unschedulable at that node, a timer starts to work. When the timer rea-

ches the value that is equal to deadline minus the execution time of that process, then this process is said to be unschedulable elsewhere.

By using semi-distributed approach, unschedulable newly coming aperiodic processes are tried to be scheduled at intradomain level. Each node in a domain sends its schedule list and unschedulable aperiodic processes list to the CLB periodically. CLB collects this information, then tries to find appropriate empty time slots on different nodes of its domain for unscheduled aperiodic processes. If it can find such a node, then this node is determined as destination node, and process migration takes place. Otherwise the process is said unschedulable within this domain. If such a condition occurs the strategy works in the interdomain level in the following way:

1. The CLB of domain A (CLB_A), sends a query to one of its nearest neighbors CLB of domain B (CLB_B), to form a temporary pair, which enables a controlled, stable environment suitable for process migration. The query has two purposes:
   a. it informs the CLB_B that CLB_A wishes to form a pair
   b. it contains a list of processes and time constraints for each processes.

2. CLB_B after receiving the query can perform one of three options:
   - rejecting CLB_A 's query; this implies that CLB_A must send a query to another neighbor domain
   - form a pair with CLB_A; this implies that CLB_A as well as CLB_B reject all incoming queries until the pair is broken.
   - postpone CLB_A when CLB_B is in a migrating state, that is, sending processes this implies that CLB_A must wait until CLB_B forms a pair with it, or rejects it- CLB_A cannot query anyone else.

3. After establishing a pair, CLB_A sends unschedulable aperiodic real-time processes list to the CLB_B. Then CLB_B broadcasts this information to all of its nodes. Nodes try to schedule these processes on their own schedule table. If this is pos sible, scheduled processor id and its response time are returned to the CLB_B. As the last step, CLB_B compares the response times of the same processes on different nodes and selects the node giving the minimum response time as the destination node.

4. If no processes can be executed on CLB_B, then CLB_B informs CLB_A of this fact and the pair is broken. Otherwise the processes are migrated. This process is repeated for all remaining unscheduled aperiodic real-time processes until no process is left.

4 Operating System Support

In order that the central load balancers can communicate efficiently, the operating system should provide some form of multicast communication. The group management and naming modules described below were added on top of the existing facilities of the NX/2 kernel of the Intel iPSC/2 hypercube simulator.

4.1 Group Management

Processes that are functionally related to finish an overall task are included in a group. These processes communicate frequently in multicast mode where one process which is the member of a group sends a message to all other members of the group as one-to-many communication (Cheriton & Mann 1988).

In our system, groups could be distributed over the processor domains. Each processor domain has a group server that initiates all local group communication primitives, like make_group, kill_group, join_group, leave_group, etc. Each group server is responsible for only the local members of any group. All local members that are located in a domain send their requests to their own group server of the domain. Only group server could be in contact with other members of the group via other group servers if needed.

When a process in a domain wants to create a group, it sends that requests to the group server. Group server creates a group control block and sends that request to the group servers on the other domains. However joining to a group or leaving a group are done locally. No interdomain communication is needed for these frequent services. When a process from a group wants to

send a group message to all other group members over the system, it first sends that request to the local group server which then sends the message to all other members in that domain and pass the message to all other group servers. These group servers on the other domains do the same thing simultaneously. So group communication is handled in parallel by the group servers distributed over the domains of the MPP system.

4.2 Naming

A naming facility in a distributed system, in general should provide a mapping from system names to addresses where the object is residing, and a route to specify how to get there (Goscinski 1991). The naming is implemented by the name servers in each domain which hold a subset of the global naming space for the objects in that domain (Cheriton & Mann 1988). The name servers form a process group and communicate as described above. The name server in a domain contains the address of the objects in its domain and can receive a request from a local processor or another name server for an address of an object. If the request is local, a check is made to find if the object is located at an address in that domain. If this is not the case, a broadcast message is sent to all name servers to receive the address. In this case, only the name server which has the address of the object will respond.

5 Implementation And Results

The proposed load balancing techniques were implemented in an Intel iPSC/2 hypercube simulator running on OSF/1 MACH Unix environment with the following results obtained.

5.1 Static Scheduling Results

The performance of TPM (Task-to-Processor Mapping) and DA (Domain Allocation) algorithms were evaluated by using sample task graphs generated randomly by TGG (Task Graph Generation) algorithm, generally in the form of supervisor-worker model. Results were obtained for different number and topologies of processors, several heuristic functions for real-time tasks, and different number of domains.

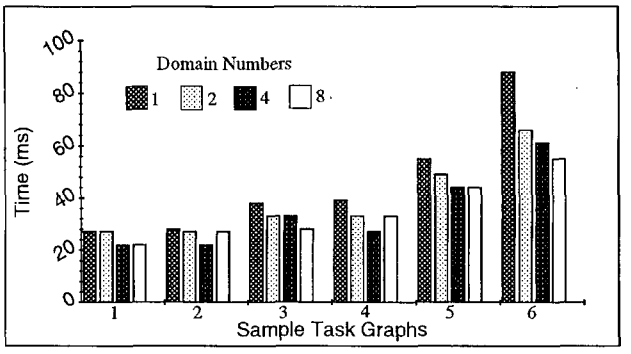


Figure 9: Time Spent for Static Mapping in EDF

Parallel system topologies are selected as 4-processor mesh topology, and 8 and 16-processor hypercube topology. Processor domain numbers are selected as follows:

- DN=1 and 2 for 4-processor case
- DN=1, 2 and 4 for 8-processor case
- DN=1,2,4 and 8 for 16-processor case

DN=1 means there is only one domain in the system. In this case, domain allocation phase is not used.

For different domain numbers, we have obtained time measures for task-to-processor mapping and percentage of real-time tasks whose deadlines are met. The following are some of the results we have obtained for different heuristic functions:

EDF (Earliest-Deadline-First) Heuristic: Fig. 9 shows time spent during static mapping of tasks to processors for DN=1, DN=2, DN=4 and DN=8 where M=16 with hypercube topology. N varies between 24 and 48 interrelated tasks.

Fig. 10 shows the number of real-time tasks whose deadlines are met at the end of our static scheduling scheme versus the number of real-time tasks in sample task graphs for the parallel system with M=16 processors and hypercube topology.

MLF (Minimum-Laxity-First) Heuristic: Fig. 11 shows time spent during static mapping of tasks to processors with varying domain size where M=16 with hypercube topology. N varies between 24 and 48 interrelated tasks.

Fig. 12 shows the number of real-time tasks whose deadlines are met at the end of our static scheduling scheme versus the number of real-time tasks in sample task graphs for the parallel system with M=16 processors and hypercube topology.

$H3 = (EDF * W1 + MLF * W2)Heuristic$:

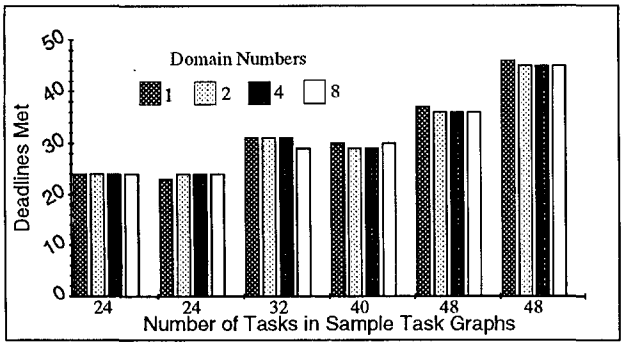


Figure 10: Deadlines met in EDF

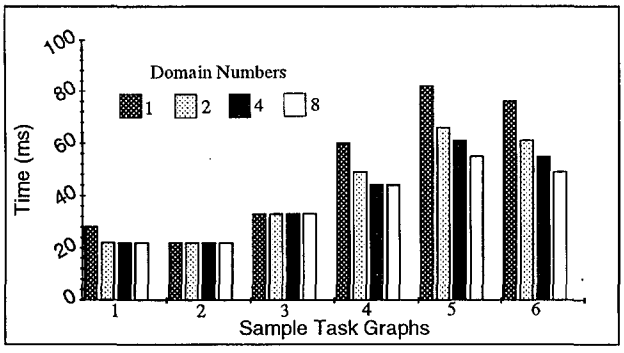


Figure 11: Time Spent for Static Mapping in MLF

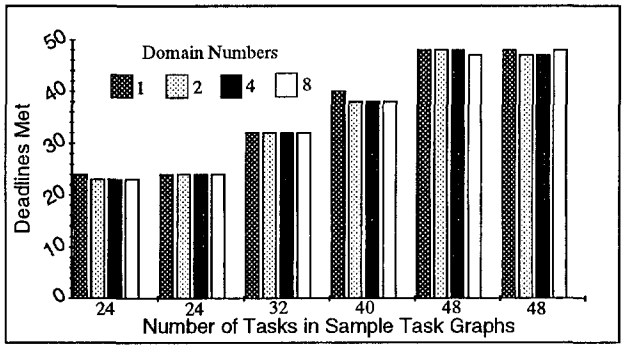


Figure 12: Deadlines met in MLF

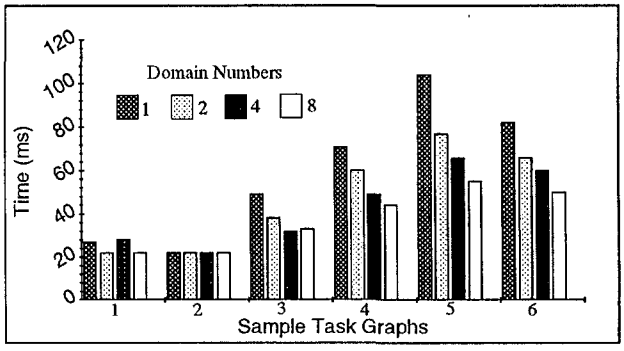


Figure 13: Time Spent for Static Mapping using H3

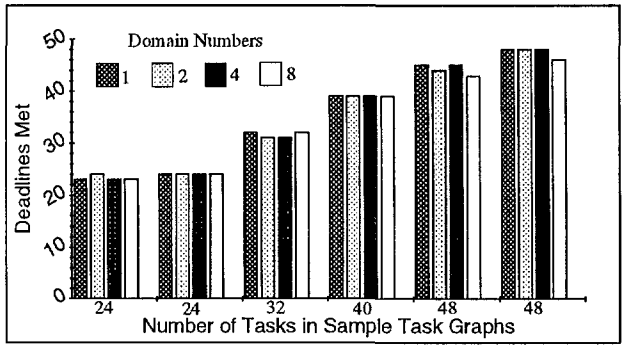


Figure 14: Deadlines met in H3

Fig. 13 shows time spent during static mapping of tasks to processors for DN=1, DN=2, DN=4 and DN=8 where M=16 with hypercube topology. N varies between 24 and 48 interrelated tasks.

Fig. 14 shows the number of real-time tasks whose deadlines are met at the end of the static scheduling scheme versus the number of real-time tasks in sample task graphs for the parallel system with M=16 processors and hypercube topology.

5.2 Dynamic Load Balancing Results

The dynamic load balancing mechanism is implemented for soft real-time processes described in Section 3 by adding group management and naming modules to the existing NX/2 kernel of the hypercube simulator. The semi-distributed central dynamic load balancers are system processes which communicate with other load balancers to perform load transfer. Process migration facility is simulated. Real-time approach is implemented and system's performance is observed for different number of domains and processor numbers. In figures 15-17, the percentages of aperiodic processes meeting their deadlines that have been in

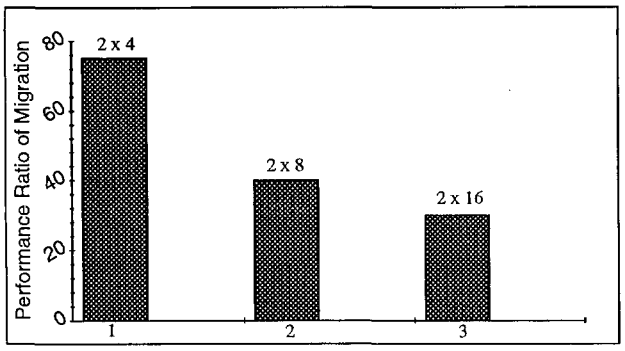


Figure 15: Test of Real-time Approach When Domain Number is Constant and Processor Numbers in Each Domain are Variable.

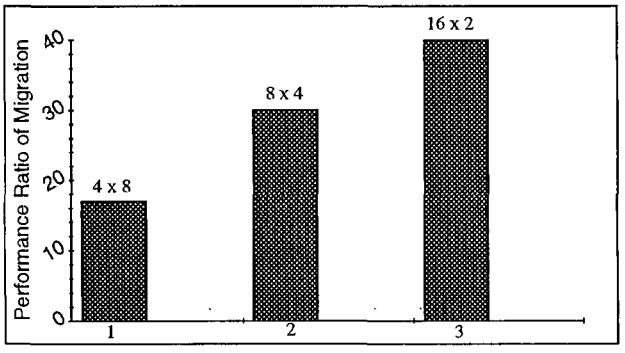


Figure 16: Test of Real-time Approach When Domain Number is Variable But Processor Numbers in Each Domain are Constant

migration list are shown.

Fig. 15 shows the performance of the system when domain number is 2 but number of processors in a domain is varying as 4, 8 and 16. When the domain number is constant but the processor number increases, system goes to a centralized manner and performance of the system decreases.

Fig. 16 shows the system test results when domain number is varying but the total processor number in the system is constant. As the number of domains increases, the system closes to a distributed condition and the system performance increases hence the percentage of the number of aperiodic processes meeting their deadlines rise.

Performance of the system, when the domain number is varying but the processor number in each domain is constant, is shown in Fig. 17. As the domain number in the system increases, the number of aperiodic processes meeting their deadlines increases.

Schedulability ratios for 50 processes on varying number of domains are represented in Fig. 18.

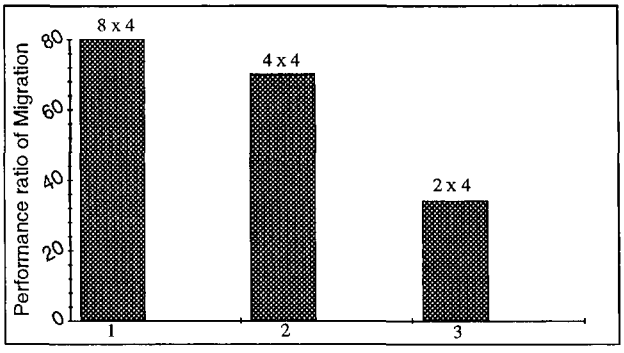


Figure 17: Test of Real-time Approach When Domain Number is Variable But Processor Numbers in Each Domain are Constant

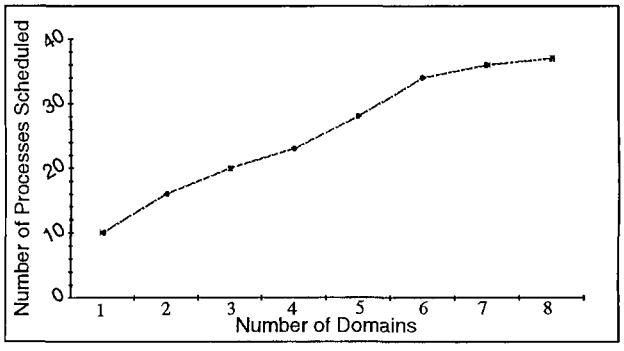


Figure 18: Number of Processes Scheduled Using Real-time Approach For Varying Number of Domains

Each domain has 4 processors in this case. It can be deduced from the figure that as the number of domains in the system increases, schedulability chance of the processes also increases.

6 Conclusions

We have proposed a framework for load balancing and for running applications in a parallel processing system. The main components of the system are the deterministic scheduler, the dynamic load balancing mechanism and operating system support modules for dynamic load balancing. The off-line scheduler is used to allocate periodic, hard real-time processes and the dynamic load balancers transfer aperiodic soft real-time processes from heavily loaded to lightly loaded nodes. The processors of the parallel real-time system are divided into domains which are a collection of processors. Both static and dynamic schedulers try to balance load at intradomain and interdomain levels, but in different directions. The static scheduler starts from global system information and ends in allocation of tasks to processors of individual domains whereas the dynamic load balancers first try to even the load within the domain and if this is not possible, communicate with load balancers of other domains to perform transfers at interdomain level.

The static scheduling model is tested and evaluated for approximately 1500 precedence related tasks. The results can be summarized as follows: It is observed that, when the system is partitioned into domains, time spent during mapping decreases as the number of domains in the system increases. This is caused by the fact that, domain allocation phase minimizes the search space of Task-to-Processor Mapping phase during finding optimal processor for a task. This result is especially important for MPP systems. The percentage of deadlines met for heuristic functions are given in Table 1.

Table 1: Percentage of Real-Time Tasks whose deadlines are met.

	M=4	M=8	M=16
EDF	0.96	0.97	0.94
MLF	0.96	0.98	0.98
H3	0.97	0.98	0.96

Partitioning the system into processor domains is an advantageous approach according to the simulation results we have obtained. When the system is partitioned into domains, time spent for static scheduling decreases. Since the performance of the model we have proposed is approximately the same for domain partitioning approach and the approach without domains, the domain partitioning scheme that has less time complexity will be preferred. Also, the concept of domain will provide the base for centralized use of resources at intradomain level and distributed usage of resources at interdomain level.

For the dynamic load balancing case, the starting point of our research was the fact that neither fully distributed nor centralized load balancing policies yield good performance for MPP systems. We therefore designed a semi-distributed model which makes use of both approaches. The iPSC/2 hypercube simulator unfortunately created problems after 32 processors, so the maximum domain number we could test is 16 each with 2 processors. It is observed that when the system is tested for varying number of domains and processors, the percentage of the scheduled aperiodic real-time processes rise sharply as the number of domains increase which is in accordance with what we expected.

This semi-distributed management of resources by confining local information to domains and acquiring this information if needed by use of the underlying kernel services can be extended to other higher system functions such as file servers, etc. which can be built on this structure using the same principle. For example, a process in a domain, which wants to open a file, would send a message to the file server of that domain. If the file is not found locally, the other file servers can be informed to find the file and transfer data. We think this approach eliminates the need for central or the replicated data, which would lead to bottleneck in the first case and overhead for consistency in the latter.

The results obtained for both static and dynamic load balancing cases are decisive in accepting that partitioning a parallel real-time system into domains and managing them accordingly yields better performance than no-domain case. Secon-

dly, this idea can further be used for other resource management purposes. At a more abstract level, each domain can be considered a node of a distributed real-time system connected by a real-time network. Most of the methodology developed is valid for these loosely coupled systems with the modification of the communication costs accordingly. A future investigation area would be on how to configure the size of the domains to suit an application.

References

[1] Audsley N. & Burns A. (1990) Real-time System Scheduling. Technical Report YCS134. Department of Computer Science, University of York, UK.

[2] Bryant R.M. & Finkel R.A. (1981) A Stable Distributed Scheduling Algorithm. *Proceedings of the 2nd International Conference on Distributed Computing Systems.*

[3] Cheriton, D.R. & Mann T.P. (1988) Decentralizing a Global Naming Facility for Improved Performance and Fault Tolerance. *ACM Transactions on Computer Systems*, 7(2), p. 147-183.

[4] Efe K. (1982) Heuristic Models of Task Assignment Scheduling in Distributed Systems. *IEEE Computer*, June 82, p. 50-56.

[5] El-Rewini H. (1989) Task Partitioning and Scheduling on Arbitrary Parallel Processing Systems. PhD. Thesis, Oregon State University.

[6] Goscinski, A. (1991) Distributed Operating Systems: The Logical Design. Addison-Wesley, p. 300-344.

[7] Kremien O., Kramer J. & Magee J. (1993) Scalable, Adaptive Load Sharing for Distributed Systems. *IEEE Parallel & Distributed Technology*, vol.1, no.3.

[8] Liu J. & Layland J. (1973), Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment. *Journal of the ACM*, 20(1), p.40-61.

[9] Sadayappan P. & Ercal F. (1987) Nearest Neighbor Mapping of Finite Element Graphs onto Processor Meshes. *IEEE Transactions on Computers*, vol.c-36, no.12, p. 1408-1424.

[10] Stankovic J.A. & Ramamritham K. (1988) The Spring Kernel: A New Paradigm for Real-Time Operating Systems. COINS Technical Report 88-97, University of Massachusetts, Amherst.

Optimal Algorithm for Real-Time Fault Tolerant Distributed Processing Using Checkpoints

Zbigniew M. Wojcik and Barbara E. Wojcik
Smart Machines, 13703 Morningbluff Drive, San Antonio, TX 78216, USA
Phone: (210)496-7287, Fax:(210)496-6218

Keywords: fault-tolerant computing, distributed processing, real-time systems

Edited by: Marcin Paprzycki, Janusz Zalewski
Received: February 28, 1994 **Revised:** November 7, 1994 **Accepted:** February 28, 1995

The paper presents optimal algorithm for a real-time deadlock-free fault-tolerant distributed processing. Our fault tolerant computations do not incur any postponements in the underlying distributed processing and need the minimum number of messages. Both the underlying messages and the messages maintaining fault tolerance do not need to arrive in the order in which they have been sent. The method is based on the asynchronous, atomic checkpointing of the sender and receiver of a message. Messages not balanced in the last permanent checkpoints are recorded in the new checkpoints. The fault recovery is based on: a) The repetition of all messages lost according to a record of unbalanced messages in the last permanent checkpoints; and b) Undoing every message re-sent during the fault recovery, or undoing of a computation repeated according to a record of unbalanced messages in the last permanent checkpoints.
Our fault recovery involves only processes which communicated before a failure. Proof of the resilience of the fault recovery algorithm is presented.

1 Introduction

The presented approach[1]is based on checkpoints[2]. Checkpointing assures the overall system consistency after faults during execution of any distributed algorithm. Our checkpoint initiator makes its tentative checkpoint and sends its checkpoint request to its dependant. To the checkpoint request, information is appended about all messages received from the dependant and sent to the dependant by the initiator. Upon receiving the checkpoint request, the dependant treats the initiator's checkpoint as permanent (i.e., if the initiator fails, the dependant will recover the initiator from the initiator's tentative checkpoint). The dependant makes its checkpoint, and then deletes its previous permanent checkpoint treating thus its current checkpoint as permanent. Then sends its checkpoint acknowledgement to the initiator. Upon receiving this checkpoint acknowledgement, the initiator changes its tentative checkpoint to permanent and then removes its previous permanent checkpoint. If the dependant fails before the checkpoint acknowledgement is received, then the initiator recovers the dependant from the previous permanent checkpoints. The reminder of this section discusses advantages of our new approach, and presents technical details associated with it.

1.1. General overview

One approach to resilient fault recovery (so called *backward error recovery)* on a distributed system provides a complete execution of the entire atomic computation (*transaction*) on another site (Randell, [7]). A *commit protocol* enforces the transaction atomicity, i.e., if a transaction is accepted for execution, it must be completed. If the atomic action is too long, it may postpone other tasks making thus the system inconvenient, and it is not clear whether there is still a correct computation or an error. As a result, it becomes more difficult to detect a fault, because the *hand* -

[1]This research has been sponsored in part by the Army Research Office under the contract DAAH04-94-C0019.

[2]The early version of this paper was presented at the 2nd IEEE Symp. on Parallel and Distributed Processing, Dallas, 1990 [14].

shaking procedure cannot be used until the transaction times out. Simply splitting the task into a few parts would make it non-atomic.

This paper presents an approach which allows a node to divide a distributed request into segments without affecting the internal consistency of the transactions and to implement the typical *hand-shaking procedure* for failure detection at any time. In general, goals of the proposed methodology is to enable time-critical real-time fault-tolerant computing by the following means:

- Resilient fault recovery in an asynchronous environment in which the messages are not received in the order in which they have been sent. This is to avoid the necessity of synchronization involving undesired waits for delayed or lost messages;
- Elimination of waits for acknowledgements for checkpoint messages and for checkpoint decisions (e.g., whether to make checkpoint permanent). Our system messages associated with fault-tolerance do not involve synchronization with undesired postponements of underlying computations;
- Deadlock freedom for our send-receive checkpointing. Deadlock freedom and no postponements assure timeliness during our fault-tolerant computing;
- Interruptible tasks (or transactions). Long tasks do not need to be started again from the beginning in case of a fault what assures real-time execution;
- Fault recovery of sequences of short tasks or transactions if it is too expensive to checkpoint every short action; and
- The minimum number of checkpoint messages providing optimality of our algorithm. The optimality criterion is here both the communication time (the number of checkpoint messages) and the waiting time.

A delayed or lost message postpones a checkpoint and a process in CSP [1,3]. One remedy not to postpone indefinitely the processes waiting for lost or delayed messages is to undo tentative checkpoint [3]. Permanent checkpoint may be postponed indefinitely in this way if many messages arrive with some delays. Hence, this approach is abandoned in this paper. Because we also use point-to-point messages, we significantly improve CSP.

1.2. Basic Assumptions

Fault recovery resilience is achieved under the above goal by merging into one atomic entity the sender's and receiver's checkpoints and recording all of the messages in the sender's and receiver's checkpoints. A *send-receive (point-to-point) checkpoint* proposed in this paper is a collection of records and actions involving:

1. Sender q's checkpoint (q is referred to as initiator);
2. Initiator's message $i^k_{q-\{p\}}$ requesting its P receivers (dependants) 1, . . . p, . . . P to perform their checkpoints, where k is the index of subsequent q's message. Each checkpoint request i^k_{q-p} B contains: a) A record of all sent messages $m^{\{k\}}_{q-p}$ which should have been received by p; b) A record of all received messages $m^{\{l\}}_{p-q}$ which have been sent by p, where l is the index of subsequent p's message;
3. All receivers, ('dependants') checkpoints;
4. Dependant's acknowledgements $atc^l_{\{p\}-q}$ to the $i^k_{q-\{p\}}$ sent by all dependants containing: a) A record $pm^{\{l\}}_{p-q}$ of messages $m^{\{l\}}_{p-q}$ which should be received by q; b) A record $pm^{\{k\}}_{q-p}$ of messages $m^{\{k\}}_{q-p}$ received from q.

The information about the initiator's messages is passed to the dependant's checkpoint together with the checkpoint request. The dependant sends a record of its messages attached to its acknowledgement. The atomicity of send-receive checkpoint operations becomes a new specific constraint placed on the structure of distributed operations.

The atomic send-receive checkpoints allow a node to detect a failure occurring at a time of making the checkpoints. If a caller fails during its service and there are other requests to the server, then the failed caller's task or transaction will be aborted or rolled back to its last permanent send-receive checkpoint to unblock the other requests (and to service them by the server).

A failed task or transaction is rolled back together with its server to their last permanent send-receive checkpoint. The lock acquired by a failed task or transaction is relinquished by the fault re-

covery. No other object is able to access the object locked by a failed process before the rolling back (because prevention of task or transaction atomicity is imposed on the execution).

One of the advantages of the proposed scheme is that many requests may be serviced in both the preemptive and the *round-robin* (RR) manner reducing the average waiting time. Using preemption, the driver of the preempted job is forced to be the checkpoint initiator of its tasks. When dealing with RR, a sender (i.e., the transaction initiator) may decide to make its checkpoint when a RR quantum expires. The sender forces the receiver to make its checkpoint too. Atomicity of the operations: send, sender checkpoint, receive, receiver checkpoint, is provided.

The presented method of fault-tolerant computing based on point-to-point message passing (i.e., involving two sites), is directly extendible to broadcasts one-to-many, when many one-to-one messages are considered (e.g., one-to-eight broadcast is treated as eight one-to-one messages). In a similar way, fault tolerance of many-to-many broadcasts can be provided.

1.3. Paper's structure

The rest of the paper is structured as follows: Section 2 discusses other approaches to fault-tolerance. Checkpoint consistency is defined for two-phase-commit protocols and consequences of inconsistency during fault-recovery are discussed. Recovery resilience is defined and a two-phase-commit protocol is shown to be ineffective in case when messages are not received in the order they have been sent. Section 3 develops a remedy for the inconsistency problem of the two-phase-commit fault recovery protocols when the send and receive operations are not synchronized. Send-receive checkpoint consistency of entirely asynchronous messages is defined. Checkpoint protocols and fault recovery protocols for the initiator and dependants are formulated. Section 4 validates the proposed fault recovery protocols. Several interesting and very useful properties of the send-receive checkpoint fault recovery are proved, e.g., deadlock freedom, the minimum number of messages, completeness, effectiveness on a distributed environment with optional lengths of communication lines, and resilience.

2 Some Typical Problems in Fault Recovery

Fault tolerance is an important element of real-time systems to assure service completion before a deadline in case of a fault. When a node in a distributed system fails, a process residing on it must be restarted in a manner which preserves overall real-time system consistency (i.e., the execution must continue as if there was no fault). Shin at al. [9] concentrate on selecting time moments of making the checkpoints to minimize the mean task execution time for real-time applications. System consistency is not considered what may lead to errors in case of a failure, or the system must be resumed from the beginning if a fault happens. Operating system computations to assure consistency may need much more time than making the checkpoints.

Real-time system needs optimization of the consistency computations first. Ramanathan and Shin [6] realize the need to maintain the overall system consistency in real time if a fault happens. Because approaches known from literature introduce undefined waits and postponements, they offer a special hardware to synchronize all clocks on all nodes to make checkpoints at all sites at the same time. This avoids time consuming synchronization of the checkpointing operations by the messages (see the two-phase commit protocol [3] described below). The programmer decides in the parallel program of the time points of making the checkpoints. The programmer should set the checkpoints in the program at points of approximately the same time of execution from the previous checkpoint. Tasks which finish earlier must wait for the longest task.

Our approach to real-time fault-tolerant distributed computing is easier to use, because: (a) It does not need a special hardware; (b) It provides consistency also on a heterogeneous system; (c) It does not spend even a small time for waits; (d) It does not involve any unexpected postponements associated with delayed or lost messages; (e) It is deadlock-free; and (f) It does not need a manual setting of the checkpoints in the program by the programmer (all is done automatically by the operating system).

Generally, to provide system consistency in case of a fault, a strategy of rollbacks to the last per-

manent checkpoint is used to make the recovery [3]. In this section checkpoint consistency and recovery resilience are defined, and then some of the pitfalls associated with checkpointing are discussed. The following assumptions are made for the reminder of the paper:

ASSUMPTION 1. All processes learn of the failure of any processes they communicate with within a finite time. A *time out* (i.e., no response within a prespecified time) signifies a receiver's failure or a communication link failure.

ASSUMPTION 2. Checkpoints are made asynchronously by processes. A message obtained during a checkpoint waits until checkpoint completion.

ASSUMPTION 3. No failure can partition the distributed system.

ASSUMPTION 4. Messages sent during the same distributed computation carry the same unique process ID. All messages of the same ID are numbered in the increasing order.

DEFINITION 1. A checkpoint is *consistent* in a system in which messages arrive in the order they have been sent if [3]: a) It does not contain a record of a received message that is not recorded as sent in a checkpoint of a sender; b) No message is lost after resumption from the last receiver's checkpoint.

LEMMA 1. Checkpoints made independently of send and receive operations can be inconsistent.

PROOF. Suppose, a failure occurs just before or when a checkpoint is made. A message may be registered as received in the receiver's checkpoint (e.g., in the checkpoint $c_1(p)$ in Fig.1.a) but not recorded as sent in the checkpoint of the sender (e.g., in $c_0(q)$ in Fig.1.a), violating DEFINITION 1.a. Also, a message registered as sent (e.g., in $c_1(q)$ in Fig.1.b) but not registered as received may be lost by the failed receiver if a receiver's failure occurs just before or when making a receiver's checkpoint violating DEFINITION 1.b. •

Another danger in a fault recovery is the *domino effect* (Randell [7], Russell [8]) occurring when processes are checkpointed after message receipt. For example, process p (Fig.2) could be resumed from its last checkpoint $c_2(p)$ if not its last

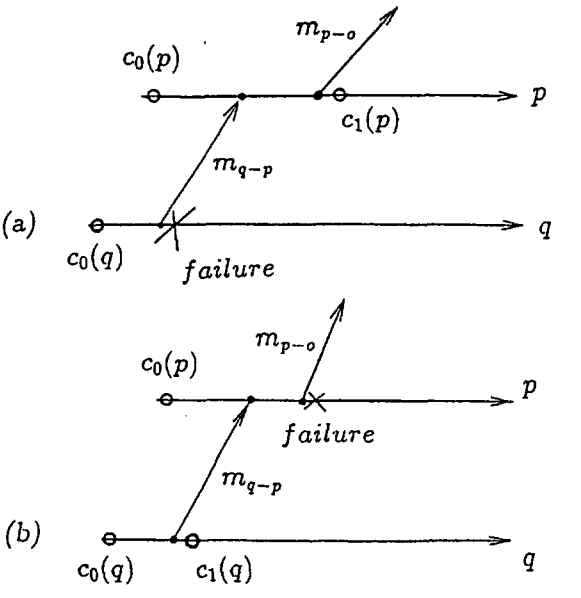


Figure 1: Underlying inconsistencies in fault recovery

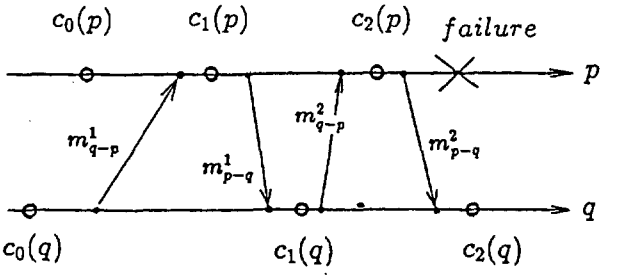


Figure 2: The 'domino effect'

message m^2_{p-q} sent to q. To avoid the service of the message twice by q (after p's recovery), the process q is rolled back to its previous checkpoint $c_1(q)$. But p has the record of the received message from q in its last checkpoint $c_2(p)$. Resuming from $c_1(q)$, this message would be sent again to p, thus would be serviced twice by p. To prevent servicing it twice, process p is rolled back to its previous checkpoint $c_1(q)$. The only stable checkpoint is the original status $c_0(p)$, $c_0(q)$. The *domino effect* involves too much time and storage space to keep all checkpoints.

The *live-lock effect* [3] results from the lack of synchronization and starvation of processes involved in a communication interrupted by a failure. Processes may be rolled back and recovered from the failure an infinite number of times. Consider the scenario presented in Fig.3: q must be recovered from a failure by resuming it from the

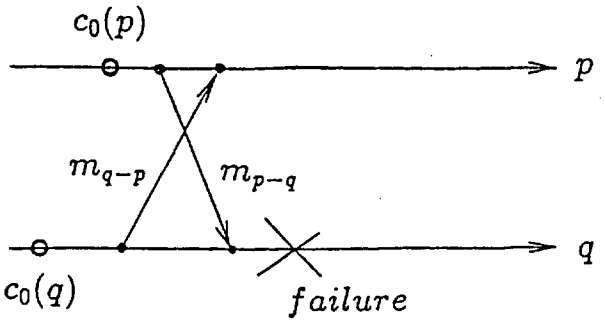


Figure 3: The 'live-lock' effect

checkpoint $c_0(q)$ (defined for both p and q). Receiving the same message m_{q-p} for the second time, process p rolls back itself to its checkpoint $c_0(p)$ (assuming, the fault recovery is not synchronized). Resuming, p sends its message m_{p-q} again, and q receiving it twice rolls back itself to its $c_0(q)$. The two processes perform the recovery (through rollback) infinitely.

DEFINITION 2. A fault recovery algorithm is *resilient* to a process or channel failure if after a rollback operation:
a) there are no lost messages nor is a request serviced multiple times without undoing the excessive computations (i.e., the remaining set of checkpoints is consistent);
b) global atomicity is achieved (e.g., the *commit protocol* for transaction execution is not violated);
c) there are no subsequent rollbacks through a sequence of checkpoints (i.e., the fault recovery is *domino effect* free);
d) synchronization avoids undesired repetition of the recovery process (i.e., there is no *live-lock).*

A straightforward approach to fault tolerance attempts to save only the last checkpoints of processes and the fault recovery takes place by rolling back to the last checkpoints [9]. A fault may happen, however, during checkpointing process. To make the fault recovery resilient to errors during a checkpoint, Koo and Toueg [3] assume an additional tentative checkpoint. The tentative checkpoint is made permanent after the completion of the entire send-checkpoint operation (if no fault occurs). If an error occurs at the time of making the tentative checkpoint, the failed process is rolled back to the last permanent checkpoint. Any operations (including tentative checkpoints) performed after the permanent checkpoints may be undone. Any last checkpoint not completed yet is tentative (i.e., it may be undone without affecting prospective fault recovery). The tentative checkpoint becomes permanent after the receiver sends its acknowledgement and after the initiator makes its own tentative checkpoint. Each permanent checkpoint is consistent. Tentative checkpoints do not need to be consistent.

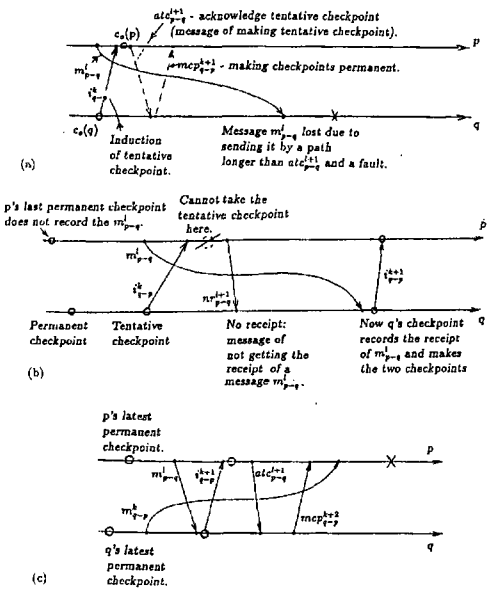


Figure 4: Inconsistencies with fault recovery using the **two-phase commit protocol:** *a) a message m^l_{p-q} lost if sent by a path longer than the acknowledgement of the checkpoint request and as a result of the receiver's failure; b) remedy to the loss of the above message by delaying the permanent checkpoints [3]; c) message m^k_{q-p} lost if protocol [3] is installed (and if sent by a path longer than the checkpoint request i^{k+1}_{q-p} and as a result of a receiver's failure) and recovered using our send-receive checkpoint recovery*

One approach to fault recovery uses a *two-phase-commit protocol.* An initiator q creates its tentative checkpoint and induces tentative checkpoints for other processes (e.g., using a message i^k_{q-p} in Fig.4.a) in the first phase, and then blocks itself and waits for responses. If q receives messages of completion for all of the induced tentative checkpoints, it changes all the tentative checkpoints to permanent in the second phase. Dependents stay blocked waiting for the initiator's decision. However, the permanent checkpoints are not consistent, because a message m^l_{p-q} sent by p before making the permanent checkpoint $c_o(p)$ and received by q after making all checkpoints permanent will be lost if q fails after that.

This problem is approached by the following modification [3]: The dependant p does not make its tentative checkpoint induced by q without an acknowledgement of its outstanding message m^l_{p-q} (Fig.4.b).

However, there is another danger with the modification presented in [3]. Consider the scenario as in Fig.4.c, when the receiver of i^{k+1}_{q-p} fails after making the checkpoints. The i^{k+1}_{q-p} (checkpoint request) contains a record (acknowledgement) of all messages received by q from p (e.g., of m^l_{p-q}). According to [3], if the p's latest permanent checkpoint does not have the record of sending this message to q, then p makes the decision to take its tentative checkpoint. When q learns about taking the tentative checkpoint (by a message atc^{l+1}_{p-q}), then q changes all the tentative checkpoints to permanent (e.g., by a message mcp^{k+2}_{q-p} - make checkpoint permanent) in the second phase. Based on the protocol presented in [3], the processes p and q do not see any inconsistencies. However, the m^k_{q-p} gets lost if it is sent by a longer path and if the p fails!

One of remedies not exploited in this current paper and in [3] is to suspend the second phase of making the tentative checkpoint permanent (by the initiator q) until receiving all the delayed messages (as m^k_{q-p}). This would lead, however, to long delays and even to indefinite postponements if one of messages is lost or when delays happen frequently.

3 Fault Recovery Using Atomic Send-Receive Checkpoints

The send-receive checkpoints introduced in this paper solve the problems discussed above. To prevent inconsistency and any postponements associated with waits for delayed or lost messages, the paper assumes that the request i^{k+1}_{q-p} for a checkpoint carries the record of any messages sent by q to p which are not balanced (i.e., not acknowledged by p). Similarly, acknowledgement to the i^{k+1}_{q-p} contains the record of the p's messages not balanced in the p's record of sent and received messages.

The method presented in this paper does not require the checkpointing of every send. The initiator's checkpoint must be atomic with its last send i^{k+1}_{q-p} (carrying full information of q's unbalanced send messages) and then the receiver of that message makes a tentative checkpoint. To be able to resume correctly in case of a failure, the receiver needs to balance its own record of messages (exchanged with the sender) with the sender's record of the messages (exchanged with the receiver) received from the initiator with the request i^{k+1}_{q-p}. The balancing makes the receive operation atomic with the i^{k+1}_{q-p}.

The atomic send-receive checkpointing enforces transaction atomicity, but it is different from the two-phase-commit protocols. The initiator of the checkpoint expects an acknowledgement with the information as to whether each dependant took its checkpoint, together with the information of the messages which have been received by the dependant from the initiator. Immediately after sending the acknowledgement, each dependant changes its tentative checkpoint to permanent. The initiator does not block itself when waiting for acknowledgement, because records of any delayed messages are exploited for any anticipated fault recovery and the last permanent checkpoints are valid anyway. The receivers of the checkpoint request also do not wait blocked for the initiator's decision on making the checkpoint permanent (as they have to wait in [4]) because they keep track of any delayed messages. Because the dependants do not wait blocked, the message *make-checkpoint-permanent* used in [3] (e.g., mcp^{k+2}_{q-p} in Fig.4.c) is not necessary for the checkpointing in the proposed approach. There is also no requirement that messages be received in the order they have been sent.

DEFINITION 3. A send-receive checkpoint is consistent if:

a) It contains (optionally) a record of a received message that is not recorded as sent in the sender's checkpoint, but the sender's record of all sent and received messages is in the possession of the receiver (e.g., this record is passed with the i^k_{q-p} in Fig.5.b);

b) It has an optional record of sent messages which are not recorded as received in the receiver's checkpoint (compare m^l_{p-q}, Fig.5.a), but the receiver's record is in the possession of the sender (e.g., the record is passed with i^{k+1}_{q-p} in Fig.5.a); and

c) no message is lost after resumption from the last send-receive checkpoint independently of a delay of the message.

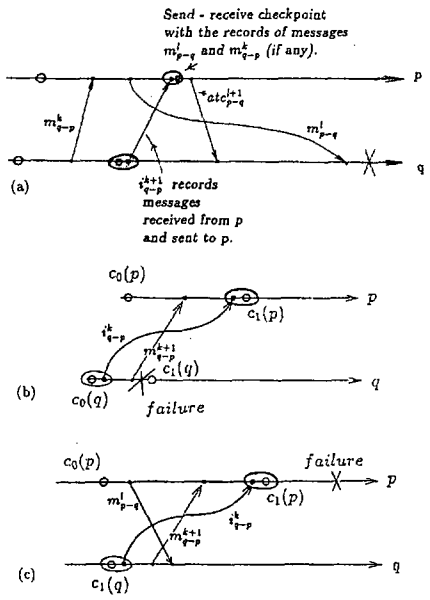


Figure 5: Resilient atomic send-receive checkpointing with the minimum number of messages. The checkpointing does not wait for delayed messages: a) a case when checkpoint request i^{k+1}_{q-p} reaches each dependant p through a path shorter than a regular message m^l_{p-q}; b) a checkpoint request may reach dependant with a longer delay and the initiator fails. A message m^{k+1}_{q-p} is discarded when sent during fault recovery for the second time after the initiator's failure; c) a checkpoint request goes through a path much longer than the messages m^{k+1}_{q-p} and m^l_{p-q} and the dependant has a fault. Message m^l_{p-q} is repeated when it is unbalanced after the dependant's failure, and message m^{k+1}_{q-p} is discarded when it is unbalanced

The **initiator q's checkpoint protocol:**

- Make a q's tentative checkpoint keeping separate records of messages exchanged with every dependant p.
- Send a checkpoint request i^k_{q-p} to each dependant p for $1 \leq p \leq P$. Each i^k_{q-p} contains: a) A record $qm^{\{k\}}_{q-p}$ of all sent messages m^k_{q-p}, $k \in \{1, ...K\}$, which should have been received by p; b) A record $qm^{\{l\}}_{p-q}$ of all received messages m^l_{p-q} which have been sent by p; c) A hand-shaking ARE-YOU-UP message.
- Reset a *Time_Out_k* counter associated with each i^k_{q-p};
- IF *Time_Out_k expired* THEN: a) Undo the q's tentative checkpoint; b) Recover the p from failure.
- IF *any acknowledgement* atc^l_{p-q} *received* THEN make the initiator's tentative checkpoint permanent:

  Balance the q's and p's records $qm^{\{k\}}_{q-p}$ and $pm^{\{k\}}_{q-p}$ of messages m^k_{q-p}. The record $pm^{\{k\}}_{q-p}$ is received in the atc^l_{p-q};

  Balance the q's and p's records $qm^{\{l\}}_{p-q}$ and $pm^{\{l\}}_{p-q}$ of messages m^l_{p-q}. The record $pm^{\{l\}}_{p-q}$ is received in the atc^l_{p-q};

  Keep in the permanent send-receive checkpoint only the record of unbalanced messages.
- Do not block itself when waiting for the atc^l_{p-q}.

The dependant p's checkpoint protocol: IF i^k_{q-p} *received* THEN:

- Make a tentative checkpoint.
- Send the acknowledgement atc^l_{p-q} to q containing: a) A record $pm^{\{l\}}_{p-q}$ of messages m^l_{p-q}, $l \in \{1, ...L\}$, which should have been received by q; b) A record $pm^{\{k\}}_{q-p}$ of messages m^k_{q-p} received from q.
- Change status of the current p's tentative checkpoint to permanent (and do not wait for initiator's decision):

  a) Balance the q's and p's records $qm^{\{k\}}_{q-p}$ and $pm^{\{k\}}_{q-p}$ of messages m^k_{q-p};

  b) Balance the q's and p's records $qm^{\{l\}}_{p-q}$ and $pm^{\{l\}}_{p-q}$ of messages m^l_{p-q}.

DEFINITION 4. A message is *failure-specific* if its number is the minimum of the unbalanced messages within the same ID.

A sequence of messages which should be exchanged when rolling back as a result of a failure starts from a failure-specific message. The necessary minimal rolling-back operation is performed by repetition of this failure-specific message and the entire sequence of messages following it. For instance, a failure specific sequence in Fig.4.c consists of one message m^k_{q-p}.

DEFINITION 5. A *failure-specific sequence* is a message exchange starting with the failure-specific message and ending with the last message before the send-receive checkpoint.

A failure-specific sequence consists of messages which have the same distributed computation ID. The sequence is ordered by increasing numbers. In case of receiver's failure the entire failure-specific sequence must be discarded after the sender resumes (e.g., message m_{q-p}^{k+1} in Fig.5.c is discarded by p when it is re-sent to avoid servicing it twice). See fault recovery protocols given below for particulars.

The dependant's tentative checkpoint lasts only for a time of sending atc_{p-q}^{l}. If a dependant's fault happens when making this checkpoint, the permanent checkpoint is not lost.

Fault recovery protocol carried out by initiator q:

IF *failure of a dependant* p THEN:

- Wait at most a prespecified quantum for the atc_{p-q}^{l}.
- Perform a necessary reconfiguration and roll back to the last permanent send-receive checkpoint (determined by the last atc_{p-q}^{l} received):

  Repeat any failure-specific message m_{q-p}^{k} or m_{p-q}^{l} recorded in $qm_{q-p}^{\{k\}}$ or in $pm_{p-q}^{\{l\}}$ as sent and not present in $pm_{q-p}^{\{k\}}$ or in $qm_{p-q}^{\{l\}}$ (respectively) as received (compare messages m_{p-q}^{l} in Fig.5.c and m_{q-p}^{k} in Fig.4.c).

  Discard any failure-specific sequence (e.g., m_{q-p}^{k+1} in Fig.5.c) recorded in $pm_{q-p}^{\{k\}}$ as received and missing in $qm_{q-p}^{\{k\}}$ as sent. For instance, m_{q-p}^{k+1} in Fig.5.c is missing as sent in last q's checkpoint. After resuming from permanent checkpoints these messages will be repeated, so we prevent from servicing them twice by discarding them.

Note that q keeps separate checkpoint information for each dependant p, so it is possible to roll back only a failed dependant. If the expected atc_{p-q}^{l} is not received within a prespecified quantum, the last tentative q's checkpoint is discarded and the p's failure is assumed to occur during p's tentative checkpoint. The last tentative send-receive checkpoint is not treated as permanent by q before receiving atc_{p-q}^{l}.

Fault recovery protocol carried out by dependant p:

IF *failure of initiator* q THEN:

Perform a necessary reconfiguration and roll back to the last permanent send-receive checkpoint:

1. Repeat any failure-specific message m_{q-p}^{k} or m_{p-q}^{l} recorded as sent in $qm_{q-p}^{\{k\}}$ or in $pm_{p-q}^{\{l\}}$ and not recorded as received in $pm_{q-p}^{\{k\}}$ or in $qm_{p-q}^{\{l\}}$ respectively (compare Fig.5.a).

2. Discard any failure-specific sequence (e.g., m_{q-p}^{k} in Fig.5.b) recorded in $pm_{q-p}^{\{k\}}$ as received and missing in $qm_{q-p}^{\{k\}}$ as sent.

For instance, assume that q fails and its last i_{q-p}^{k} managed to reach p through a path longer than q's message m_{q-p}^{k+1} (Fig.5.b), and that p learns about q's failure. Then p learns also (from q's record $qm_{q-p}^{\{k\}}$ sent with i_{q-p}^{k} and from its own last record $pm_{q-p}^{\{k\}}$ kept in its last checkpoint) that the received m_{q-p}^{k+1} was not recorded as sent by q and that m_{q-p}^{k+1} is failure-specific, hence it discards m_{q-p}^{k+1} during a fault recovery. There could be some more messages with the same process ID_q and higher process numbers, but they are dependant on the failure-specific message and will be discarded after rolling back.

In Fig.5.a process p learns that its message m_{p-q}^{l} is not recorded as received in q's last checkpoint, because the record $qm_{p-q}^{\{l\}}$ received from q with i_{q-p}^{k+1} does not list it as received. The m_{p-q}^{l} occurs to be failure-specific because it has the smallest number l among all messages with ID_p. This dependant's failure-specific message will be repeated after rolling back. There could be more messages with the same ID_p with higher process numbers, but they will be repeated automatically because they belong to the failure-specific sequence with the same ID_p.

4 Validation of the Send-Receive Checkpoint Recovery

THEOREM 1. Any atomic send-receive checkpoint is consistent.

PROOF. Because of balancing all messages with the received checkpoint records, the failed sender will be resumed from its last permanent checkpoint but will be prevented from sending the same message to the receiver again (Fig.5.b) or a repeated message will be removed from the receiver. As a result, the message will be serviced only once. Furthermore, no delayed messages received after making the permanent checkpoints will be lost by a failure of the dependant (Fig.4.c) or by a failure of the initiator (Fig.5.a) because the sender performing the fault recovery will balance the messages and each message received after the permanent checkpoint of a failed receiver will be repeated. Any messages not acknowledged (e.g., m^l_{p-q} in Fig.5.a) by the checkpoint request (e.g., by i^{k+1}_{q-p}) are repeated during the fault recovery. Otherwise the messages would be lost making the send-receive checkpoint inconsistent. •

THEOREM 2. The send-receive checkpointing is deadlock free.

PROOF. The initiator is not blocked waiting for the acknowledgements from its dependants. Also, the dependants do not wait for the initiator's decision regarding making the tentative checkpoints permanent. If any p and q request the checkpoint at the same time from each other, both learn about their messages, will be ready to balance the messages, and will set their tentative checkpoints to permanent. Hence, there are no waits necessary for a deadlock. •

In other words, the send-receive checkpointing does not have the two-phase structure as in [3]. It is not a one phase protocol, because the initiator is not blocked when waiting for dependant's acknowledgement. The acknowledgement is used by the initiator to get rid of the previous permanent checkpoint and to receive records about messages.

In our approach, any dependant is sent (by checkpoint request) information of all other dependants (siblings) taking part in send-receive checkpoints and may make decision of fault recovery of any sibling treating the initiator's checkpoint as permanent and also all sibling's checkpoints as permanent. In the two-phase protocol [3], initiator's decision is necessary regarding making permanent the dependants' checkpoint. If the initiator fails, this decision is difficult to make during fault recovery, because of needing addresses of all the dependants (which should reside in the initiator's tentative checkpoint, but this tentative checkpoint is not safe). Our distributed point-to-point checkpoints are safe.

THEOREM 3 (optimality). The send-receive checkpointing incurs the minimum number of messages, is without postponements and waits, and by having these features is optimal.

PROOF. Only one request and one acknowledgement from each dependant are needed for the induced checkpoint carrying data about the messages which need to be received by appropriate addressees to deduce information about delayed, lost or repeated messages. By this, to maintain consistent distributed computations in case of a fault, we need less messages than any other fault-tolerant distributed computing known from the literature. Using the checkpoint request each dependant is fully informed, and with the aid of the acknowledgements the initiator can perform fault recovery of all its dependants. •

As in [3], no more than one tentative checkpoint is involved in the fault recovery. The number of potential fault handling initiators is limited to the processes sending messages. The checkpoints do not need to be made by all dependants at the same time. The dependants are allowed to complete their current tasks without blocking other dependants who are ready with their checkpoints and wait for initiator's decision (in [3] dependants are blocked waiting for the decision). So, compared to other approaches to fault-tolerance, our method is better because of the minimum number of messages, no waits and no postponements (compare the initiator q's and dependant p's checkpoint protocols).

THEOREM 4. Fault recovery using asynchronous atomic send-receive checkpointing executed after N send operations, $N \geq 1$, is resilient.

PROOF. Any atomic send-receive checkpoint is resilient because it satisfies DEFINITION 2 of fault resilience: a) Is consistent (see *THEOREM 1*). b) If a failure occurs during execution of

the receive fragment of the atomic send-receive checkpoint operation, the send fragment of the atomic send-receive checkpoint operation is tentative and will be undone during the fault recovery. c) Delays in communication are reconciled by sending records of the communication between the processes taking part in the send-receive checkpoint, by balancing the messages and resending any unbalanced messages lost due to a receiver's failure or by undoing unbalanced messages sent twice as a result of sender's failure (see Fig.5.c). d) There is no domino effect, because the fault recovery is synchronized by atomicity of the checkpoints; and e) Also, the live-lock effect is avoided since all processes are rolled back to their last consistent checkpoints. •

THEOREM 5. Send-receive checkpointing and fault recovery in a real asynchronous distributed environment is possible with paths of optional lengths.

PROOF. The checkpointing involving messages sent through long paths or lost messages is possible because of:

a) Receiving the checkpoint request from the initiator with the record of all the messages exchanged with each dependant;

b) Making a record of messages exchanged with the initiator in each dependant's checkpoint and sending that record with the acknowledgement;

c) Balancing the messages which have been sent actually with the received messages (based on their records);

d) Recovering the messages lost by receiver's failure and kept in the permanent send-receive checkpoints;

e) Undoing messages sent twice as a result of sender's fault and recovery. •

DEFINITION 6. For real-time applications with a deadline, an algorithm should satisfy the following requirements: a) Be correct; b) Be optimal (i.e., with the lowest time complexity and the minimum number of messages among all algorithms devised to solved this problem); and c) Be deadlock-free and postponement-free.

LEMMA 2. The initiator can accept messages from other processes before receiving acknowledgement from its dependants (i.e., after sending the checkpoint request), provided that all records of messages are available for prospective recovery of the initiator (if it fails).

PROOF. If the initiator fails after accepting a message and before receiving the acknowledgement, and then is recovered by its dependant, the dependant treats the initiator's checkpoint as permanent. The dependant balances the messages and is able to perform the fault recovery. So, any other messages received by the initiator do not affect the checkpoint atomicity. •

THEOREM 6. Any send-receive checkpoint operation is suitable for fault-tolerant real-time applications (guaranteeing response time) with a deadline.

PROOF. Our send-receive checkpointing satisfies DEFINITION 6 because of the following: a) Is correct (see THEOREMS 1, 4, 5); b) Is optimal (see THEOREM 3); and c) Is deadlock-free (see THEOREM 2) and postponement-free (see THEOREM 2 and LEMMA 2). None of the processes taking part in checkpointing waits indefinitely for delayed or lost messages, nor they are deadlocked by checkpointing procedures. •

5 Conclusion

Because of optimality (no other fault-tolerant algorithm known from literature involves fewer number of messages or is free from waits), deadlock - freedom and no unexpected postponements, the proposed checkpoint protocol can complete execution within a deadline. Our checkpointing is not postponed by any delayed messages, because of immediately exchanging the information (records) of the messages (i.e., which should be received, if sent, and which should be sent, when received). The maximum time required to reach the consistent status of the checkpoints is no longer than the time of making the checkpoint with our method. Other approaches known from literature postpone the underlying computations when exchanging the checkpoint requests and checkpoint acknowledgements to reach the consistency, thus needing much more time than the minimum time to make checkpoints solely.

The atomic send-receive checkpoint has the following advantages: a) The number of messages is minimal (i.e., only one request is needed for the induced checkpoint carrying a data about the

messages which need to be received by the addressee, and one for the acknowledgement of that request). b) It eliminates the danger of starvation (indefinite wait for a lost message); c) Eliminates waiting for checkpoint request acknowledgement and for initiator's decision; d) Is deadlock free and complete (without postponements); e) Is consistent and resilient (i.e., fully correct); f) Is executable at any time moment, even during a long task or atomic transaction; and g) Makes operation in a real asynchronous distributed environment possible with paths of any lengths. Messages do not need to arrive in the order they have been sent. It keeps track over the messages that should be received through different paths (e.g., using packets).

The fault recovery approach using atomic send-receive checkpoints is local and asynchronous, i.e., any process may initiate only its own and the receiver's checkpoint operation (e.g., when preempting, after a *round-robin* quantum expires or on the occasion of a message passing). So, the checkpoint operations are synchronized by some selected process' send and receive operations providing the global atomicity of the distributed computation. Separate processes can be checkpointed independently, rather than all the nodes of a distributed architecture. The checkpoints do not need to be executed every send but after any arbitrary number of sends and receives, and also during a transaction execution.

The fault recovery through the rolling back to the consistent send-receive atomic checkpoint does not affect consistency of real-time tasks. Each task is handled individually according to its consistent checkpoints. Consensus about the consistent state of the entire system is computed dynamically when making the checkpoints in an optimal fashion, without any postponements of underlying computations, by balancing messages which should be received when sent, and also the messages which should be sent if received. This consensus is performed in real time, making the system ready for immediate recovery if a fault happens. If a fault happens, the fault recovery is local, involving only tasks communicating with the faulty node, without postponing other tasks which is ideal for a real-time system. This locality of our fault-tolerant method provides scalability, because independently of the hardware configuration, only the nodes involved in the fault take part in fault recovery if a fault occurs.

The implementation of one-to-many communication by using multiple one-to-one messages is inherent in a number of parallel languages (for example, Occam and Parallel C) and will evolve in this direction, because not shared memory, but private memory enables better scalability of parallel architectures. Nodes with private links and private memories use point-to-point messages. Where global bus data is used (with one-to-many messages), a fault in one of many dependants can be recovered from its checkpoint saved in a neighboring node using private links (and also by means of point-to-point communications).

The presented fault recovery algorithm is also resilient to a fault during the atomic checkpoint operation by making tentative the sender's checkpoint until completing the receiver's checkpoint. The two site case discussed in this paper is easily extendible to a multisite configuration.

Fault recovery of the producer or consumer can be treated in the same way as the fault recovery of the writer, because consumer also modifies (clears) the buffer and is thus also a writer (both consumer and producer impose the exclusive lock on the buffer). The buffer crash should be handled as a fault of the server.

Our fault recovery may be easily combined with a deadlock-free message routing or with a decentralized distributed deadlock protocol [13] and can be embedded on any network or hardware configuration.

The future work needed are the real implementations. Up to now, fault-tolerance of a real-time system is limited by the high costs because of inadequate methods (too time consuming or too expensive, or even incorrect). The limitations for the proposed method are listed as the ASSUMPTIONS 1, 3 and 4, especially that the net must not be partitioned by faults.

A simple real-time system interacting strongly with its environment may no longer need the measurements taken before a fault. A node may simply resume from the very beginning after a fault, and take the new values measured in real time. A more complicated real-time system, especially distributed and possessing higher decision levels, needs to remember, which part of the entire mission have been fulfilled up to the fault, and the roll-

back consistent with the mission accomplished up to a fault is indispensable. Inconsistent resumption may be more disastrous than no resumption.

6 References

[1]. K.M. Chandy, L. Lamport, *Distributed snapshots: Determining global states of distributed system,* ACM Trans. Comput. Syst., v.3, No.1, 1985, pp 63-75.

[2]. W. H. Kohler, *A Survey for techniques for Synchronization and Recovery in Distributed Computer Systems,* Computing Surveys, v.13, no.2, pp 149-183, 1981.

[3]. R. Koo, S. Toueg, *Checkpointing and Rollback - Recovery for Distributed Systems,* IEEE Trans. Soft. Eng., v. SE-13, No.1, pp 23-31, 1987,

[4]. L. Lamport, *Time, Clocks, and the Ordering of Events in a Distributed System,* Communications of the ACM, v.21, no.7, pp 558-565, 1978.

[5]. M. Malek, "Responsive systems: A Marriage between Real-Time and Fault Tolerance", *Fifth International Symposium on Fault-Tolerant Computing Systems*" Nurenberg, 1991.

[6]. P. Ramanathan, G. K. Shin, "Use of Common Base for Checkpointing and Rollback Recovery in a Distributed System", *IEEE Trans. on Software Engineering,* v.19, No.6, 1993, pp 571-583.

[7]. B. Randell, *Reliable computing systems,* in: *Operating systems: an advanced course,* Eds. R. Bayer, R. M. Graham, G. Seegmunller, Springer Verlag, 1979, pp 282-391.

[8]. D. L. Russell, *State Restoration in Systems of Communicating Processes",* IEEE Trans. Soft. Eng., v.SE-6, no.2, pp 183-194, 1980.

[9]. K. G. Shin, T. H. Lin, Y. H. Lee, "Optimal Checkpointing of Real-Time Tasks", *IEEE Trans. on Computers,* v.C-36, No.11, 1987, pp 1328-1341.

[10]. D.P. Siewiorek, R. S. Swarz, *Reliable Computer Systems,* Digital Press, 1992.

[11]. D. Skeen, *A Formal Model of Crash Recovery in a Distributed System,* in: *Concurrency Control and Reliability in Distributed Systems,* Ed. B. K. Bhargava, Van Nostrand Reinhold, 1987, pp.295-317.

[12]. Y. Tamir, C.H. Sequin, *Error recovery in multicomputers using global checkpoints,* Proc. 13th Int. Conf. Parallel Processing, Aug. 1984.

[13]. B.E. Wojcik, Z.M. Wojcik, *Sufficient Condition for a Communication Deadlock and Distributed Deadlock Detection, rm IEEE Trans. on Software Engineering, V.15, No.12, 1989, pp 1587-1595.*

[14]. Z. M. Wojcik, B. E. Wojcik, "Fault-Tolerant Distributed Computing Using Atomic Send-Receive Checkpoints", Proc. 2ns IEEE Symp. on Parallel Distributed Processing, Dallas, Dec. 1990, pp 212-222.

[15]. Z. M. Wojcik, B. E. Wojcik, "Rough Grammar for Efficient and Fault -Tolerant Computing on a Distributed System", IEEE Trans Software Engineering, V. 17, No. 7, July 1991, pp 652-668.

Fully Deterministic Real-Time Protocol for a CSMA/CD Type Local Area Network

Bonaventure Tchouaffe and Janusz Zalewski
Dept. of Computer Science, Embry-Riddle Aeronautical University,
Daytona Beach, FL 32114, USA
Phone: (904)226-7034, Fax: (904)226-6678
E-mail: zalewski@db.erau.edu

Keywords: Ethernet, CSMA/CD, real-time communication

Edited by: Marcin Paprzycki, Janusz Zalewski
Received: February 28, 1994 **Revised:** November 15, 1994 **Accepted:** February 28, 1995

This paper presents a new access protocol, based on a simple CSMA/CD extension to improve Ethernet's performance in real-time applications and guarantee predictability. An analysis is given of both the extended protocol and the standard CSMA/CD. Several simulation experiments are conducted to test the two protocols under a meaningful variety of load levels, measuring the time it takes each protocol to transmit a predefined packet. The results allow making the final analysis to judge each access protocol and determine if and how the problem of predictable Ethernet behavior has been solved.

1 Introduction

Recent developments in distributed computing have brought more stringent requirements on the use of local area networks (LANs). This is because of their spread in real-time applications. The real-time constraints are mostly important on the Medium Access Control (MAC) sublayer of the OSI (Open System Interconnect) Data Link layer, because it is this sublayer that manages the acquisition of the common communication channel through which the actual data packets are physically transmitted. The most common approaches to real-time LAN involve the following communication protocols: MAP (Manufacturing Automated Protocol) [9], FDDI (Fiber Distributed Data Interface) [1], Fieldbus [2], Token Ring [10] and Ethernet [6].

The protocol used by Ethernet is called CSMA/CD (Carrier Sense Multiple Access with Collision Detection). The CSMA/CD uses a shared communication channel managed with a distributed control policy. With this approach there is no central controller managing access to the channel and there is no pre-allocation of time slots or frequency bands.

A station wishing to transmit is said to "contend" for the use of the common communication channel (Ether) until it acquires the channel. Once the channel is acquired, the station uses it exclusively to transmit a packet (not of fixed size). To acquire the channel, the station checks whether the network is busy (uses carrier sense) and defers transmission of its packets until the channel is quiet. When the channel is quiet, the waiting station immediately begins to transmit. During transmission, the transmitting station listens for a collision because other stations may attempt to transmit simultaneously when they all realized the channel was quiet.

In a well functioning system, collision occurs only within a short time interval following the start of transmission. This time interval is called the collision window and is a function of the length of the communication channel. If no collision occurs during this time, a transmitter has acquired the medium and continues the transmission of its packet. If a station detects a collision, the rest of the packet is immediately aborted. To ensure that all parties to the collision have properly detected it, any node that detects a collision invokes a collision enforcement procedure that briefly jams the channel. Each transmitter involved in the collision then schedules its packets

for retransmission at some later time. To minimize repeated collisions, each station involved in the collision tries to retransmit at a different time by scheduling the retransmission to take place after a random delay period (retransmission interval).

In order to achieve channel stability under higher load conditions, the retransmission interval is doubled with each successive collision, thus extending the range of possible retransmission delays. This algorithm has a very short retransmission delay at the beginning and will back-off quickly, preventing the channel from becoming overloaded. After some number of back-offs, the retransmission interval becomes large. To avoid undue delays and slow response, the doubling can be stopped at some point, with additional retransmissions still being drawn from the interval before the transmission is finally aborted. This is referred to as truncated binary exponential back-off. The back-off restarts with a zero retransmission interval for every new packet.

Performance comparison of the CSMA/CD and other types of protocols, such as a token ring, shows that CSMA/CD provides networks with excellent performance at low load levels, but degrades rapidly in high traffic situation. This degradation of performance of the CSMA/CD protocol is due to the fact that the probability of collisions to happen gets higher as the load increases in the network. This probabilistic nature of the CSMA/CD protocol makes it non-deterministic and therefore not suitable for hard real-time applications.

2 Previous Work

Various attempts on improving CSMA/CD for use in real time, were reported in the literature [5, 6, 11], or more recently [3, 12]. The most complete summary of previous studies dealing with Ethernet's (CSMA/CD) use in real-time applications was presented in [6] and is discussed below.

According to this report, Nut and Bayer studied the effect of different backoff algorithms used to resolve collisions by scheduling a different retry interval for each node. They found that although the exponential back-off algorithm used in the standard Ethernet was not very suitable for combined voice and data, it was suitable for voice (periodic sources) provided the network did not get overloaded.

Chlamtac and Eisenger, as reported in [6], studied the behavior of the standard CSMA/CD protocol for combined voice and data using a variety of data traffic patterns. They also concluded that the exponential backoff algorithm is not suitable for mixed voice and data traffic. They also found that an alteration to the backoff policy which reduces the number of collisions helped not only voice but also data traffic throughput. This was their suggestion to the alteration of the back-off policy: increase the first retransmission interval to give better randomization and ensure the separation of voice traffic into non-competing time slots.

Maxemchuk, as reported in [6], simulated a variation of CSMA/CD applicable to any system with periodic and aperiodic (synchronous and asynchronous) sources, with periodic sources having the same transmission requirements. In this case, periodic sources are voice packets and aperiodic sources are data packets. This new protocol uses standard CSMA/CD techniques for data transmission and the following technique to transmit voice. Periodic sources (voice) operate as if a time-division multiplexed (TDM) channel has been assigned to each source. This technique results in a decrease in the network delay (reduced collisions).

Girma and Dunlop, according to [6], reported a hardware implementation of moving TDM slots for Ethernet voice transmission. This technique involves extending the Ethernet frame preamble of 64 bits by another 512 bits. The idea is to resolve collisions during the extended preamble, before the data frame is transmitted.

Hutchison himself [6] used the standard CSMA/CD with no alteration to examine, by means of simulation, the feasibility of carrying data in real-time environment. He first determined the maximum load under which the delay will remain below a given threshold. Next, he investigated the idea of time division multiplexing as described by Maxemchuk, but using data instead of voice. Hutchison found that using the standard Ethernet, nodes can acquire the channel quickly enough to operate in real-time assignment, provided that the load is kept below the threshold value calculated for that particular application.

In a more recent work, Court [4] used a variation of a delay priority scheme, with a parallel arbitration technique to enable a node to obtain priority access to the network. This implementation uses standard network components and requires a low cost hardware adapter. A brief description of his approach follows.

There are two reasons for collisions on the network. The first is when two nodes make a simultaneous check of the network and detect an idle condition; they both talk and a collision occurs. This collision is coincidental and has a low probability of occurring. The second type is when a node detects a busy network and waits for an idle condition at the same time as other nodes may be waiting for the idle condition, and thus they will collide when it occurs. This type of collision has a higher probability of occurring.

To avoid type 1 collisions, the protocol states that access to the network can only be made relative to a busy/idle transition. To avoid type 2 collisions, a delay arbitration protocol is used. Each node is allocated a priority. After a busy/idle transition, a node waits for a time interval that is proportional to its priority; any network activity during this interval will force the node to wait for the next busy/idle transition. If the network is inactive for the node's allotted time, then a narrow time window is opened. A node can then talk; if it does then the window is stretched to encompass the frame, otherwise the window is closed and the node must wait for the next available window. The amount of dead time caused by the delay is proportional to the size of the network. To ensure that a busy/idle transition takes place, all nodes check the idle time. If it exceeds a set time which is longer than any delay time, a node will generate a dummy frame to ensure that the network stays alive. This concludes Court's protocol.

The discussed approaches either change the standard CSMA/CD protocol to fit or tolerate one particular application, or have significant overhead and restrictions or extra rules and are less flexible or performant than the standard CSMA/CD. In general, none of them guarantees a bounded time for message delivery with acceptable restrictions.

3 Problem and Its Solution

The objective of this research is to extend the standard Ethernet CSMA/CD protocol for the MAC to guarantee bounded message delivery time, under the assumption that there is neither a communication error (due to noise) nor a node or channel failure.

The assumed topology includes a series of nodes representing physical units connected to the medium. The medium represents the means by which nodes communicate. Nodes can be inserted or removed at any given moment with no need for synchronization.

3.1 General Description

The main idea behind our solution is to preserve the efficiency of Ethernet in low traffic situations while managing collisions as they occur. This idea is implemented by allowing at most one collision to happen until all nodes involved in the collision transmit their packets. With this technique, a bounded delivery time for a message can be calculated as a function of the maximum number of nodes on the network. A general explanation of this solution called METHOD_X follows.

METHOD_X tries to solve the drawback of the CSMA/CD by preassigning priorities to all nodes in the network. Node interfaces are equipped with two flags (**c_inv_flag** and **c_stat_flag**) and a unique priority number. These interfaces are always active. The protocol of METHOD_X differs from CSMA/CD only from the point where a collision occurs; before any collision, METHOD_X has the same functionality as CSMA/CD. The full protocol is presented in Figures 1-3. When a collision occurs, a jam is invoked by any of the nodes which detects it, and all the nodes in the network set their collision status flag **c_stat_flg** on. All nodes that were involved in the collision also set their collision involved flag **c_inv_flg** on. At this point all nodes involved in the collision (called collided nodes) have both their **c_inv_flg** and **c_stat_flg** on, and all other nodes on the network (called regular nodes) have only the **c_stat_flg** on, so they could not interfere with the collision resolution process.

Each node with its **c_inv_flg** on, fetches a priority number from a counter, in the range $[1..MaxNode]$, where $MaxNode$ is the maximum

number of nodes on the network. This counter is a logical part of the controller which is in turn part of a node. All these counters must be synchronized upon installation, upon upgrading or upon modifying the network to ensure that no two or more nodes can fetch the same priority number from their counters. The node with priority number 1 acquires the channel and therefore informs others of the acquisition by broadcasting a **don't_worry** message. After this node gets through using the channel, it sends a **done** message to all other nodes on the network so that they could decrement their priority numbers and then take their turns (Fig. 3). This process continues until all nodes involved in the collision have used the communication channel. If there is a priority gap (nonconsecutive priority number was fetched by the next node to acquire the communication channel), all nodes involved in the collision will decrement their priority number if they do not receive a **don't_worry** message after a collision window time **c_w** (Fig. 3).

While the collision resolution is in progress, nodes that do not have their **c_inv_flg** on are aware of this through the status of their **c_stat_flg** (on) and therefore do not interfere or try to access the communication channel until they found the network continuously quiet for KT time (KT = CollisionWindow * MaxNode). This means that if they hear any transmission on the channel, they start counting the time again by resetting **swait** timeout. When KT time expires, the waiting node will acquire the channel, use it and finally broadcast an **end_col_res** (end of collision resolution) message so that all the nodes that still have their **c_stat_flg** on can turn them off and therefore resume the standard CSMA/CD protocol.

3.2 Detailed Description

To clarify the functionality of METHOD_X further, and to determine that it is deadlock free, the following four scenarios are presented, going through all possible paths of METHOD_X algorithm in Figures 1-3.

Scenario 1. The first path is entered in Figure 1 with both **c_inv_flg** and **c_stat_flg** set to false. This situation means there is neither a present collision nor a collision resolution in progress. As a result, this node will behave like a regular CSMA/CD node trying to access the communi-

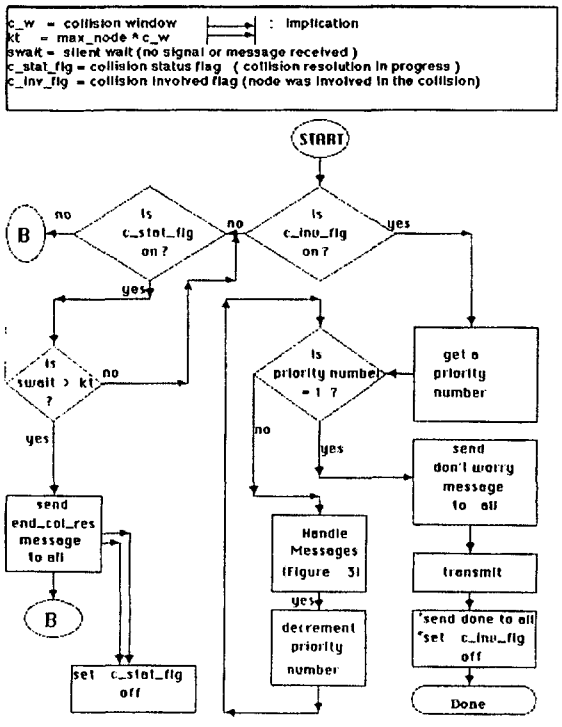


Fig. 1. METHOD X

cation channel as shown in Figure 2. This proves an extremely low overhead of METHOD_X imposed on the standard CSMA/CD, when no collision occurs.

Scenario 2. The second possible path is also entered in Figure 1 with both **c_inv_flg** and (automatically) **c_stat_flg** set to true. This situation means that a collision has just occurred and the node in question was involved in that collision. Actually the situation originates in Figure 2 when two or more nodes found the network idle and started their transmission. As a result of the collision, all **c_stat_flg**'s were set to true (on) and all collided nodes had their **c_inv_flg** set to true (on). The example in Figure 4 illustrates this situation using a sample configuration of nodes.

Figure 4 represents a network with a $MaxNode$ of 5, and with nodes A, B, C and D which have just collided and want to access the medium. Nodes A, B, C and D fetch 5, 3, 1 and 4 respectively as their priority numbers. At this point, each node checks its priority number to see if it is equal to 1 (highest priority).

On one hand, the node with the highest priority (node C) will acquire the network by first broadcasting a **don't_worry** message to all other nodes, followed by the transmission of its packet, and by finally broadcasting a **done** message to

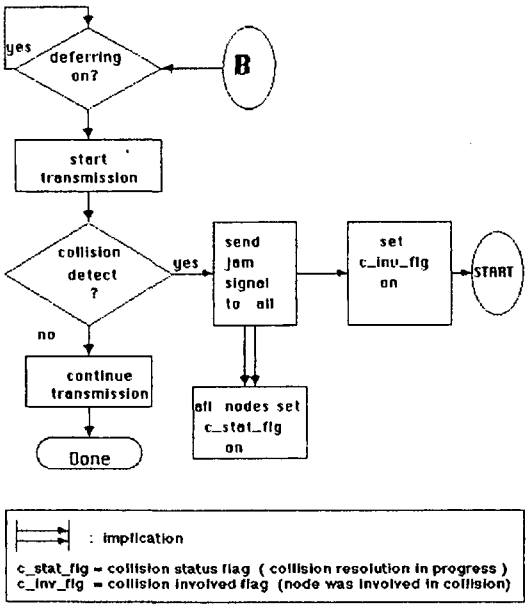


Fig. 2. METHOD_X (Ethernet Part)

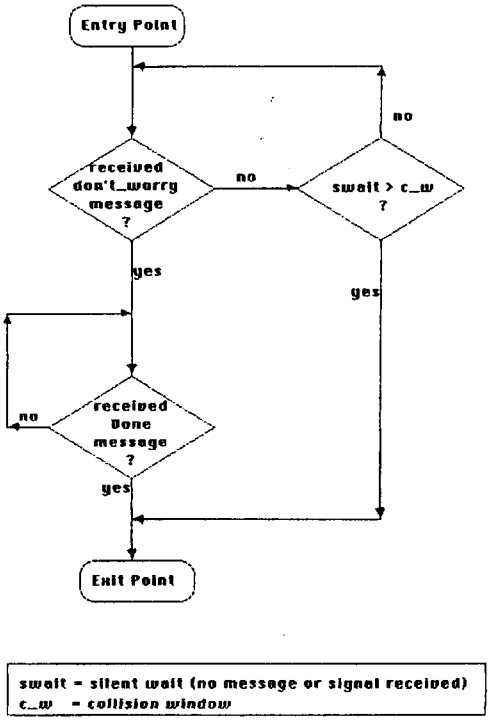


Fig. 3. Handle Messages

all nodes on the network (the rightmost path in Figure 1).

On the other hand, nodes with lower priorities will wait for the **don't_worry** message within the first collision window; otherwise they will decrement their priority numbers and continue the cycle until they have the highest priority (the middle path in Fig. 1). If these nodes receive the **don't_worry** message on time, they will unconditionally wait for a **done** message after which they will decrement their priority numbers and repeat the cycle until they have the highest priority.

Node C will then transmit its packet and broadcast a **done** message. Nodes A, B and D will decrement their priority numbers after receiving the **done** message. Since none of these nodes has the highest priority, they will not receive a **don't_worry** message within the first CollisionWindow time; this accounts for a priority gap. These nodes (A, B and D) will thus decrement their priority numbers to have 3, 1 and 2 respectively. Going through the same path, nodes B, D and A will transmit their packets respectively. At this moment, all **c_stat_flg**'s are still set to true (on).

Scenario 3. The third possible path is also entered in Figure 1 with **c_inv_flg** set to false (off) and **c_stat_flg** set to true (on). This situation implies the following: a collision resolution process is either in progress, or a collision has just been resolved. If the node in question finds the network continuously idle for KT time (the leftmost path in Figure 1), it assumes the collision resolution process is over and will thus broadcast an **end_col_res** (end of collision resolution) message to all other nodes (all **c_stat_flg** will be set to false), after which it behaves like a regular CSMA/CD node as shown in Figure 2.

Scenario 4. An interesting situation using the third path is also depicted in Figure 1 when two nodes enter this path at slightly different times, with their **c_inv_flg** set to false and **c_stat_flg** set to true. The first node to enter this path will consume the overhead of KT time mentioned in the preceding paragraph, after which it will broadcast an **end_col_res** message to all other nodes on the network (all **c_stat_flg** set to false) and resume the regular CSMA/CD algorithm in Figure 2. The next node to enter this path will soon find **c_stat_flg** to be false and will also resume the CSMA/CD algorithm in Figure 2. If both nodes involved collide, they use the METHOD_X resolution scheme, as discussed in Scenario 1.

Nodes	A	B	C	D	GAPS
Priority Numbers	5	3	1	4	
	4	2	Done	3	1 gap
	3	1		2	
	2	Done		1	
	1			Done	
	Done				

Fig. 4. Example for Scenario 2

4 Simulation Experiments

The standard CSMA/CD protocol and the protocol of METHOD_X are simulated to see how, upon collision, these two algorithms resolve the collision and attempt to transmit their packets. The main basis of performance comparison is the time it takes each algorithm to resolve the collision and transmit its packet. This is called $PacketTransmissionT$ time. In the rest of this section the procedures used to obtain results are discussed.

4.1 Simulation of CSMA/CD

In this simulation, the following network parameters are selected:

- packet size, 1526 bytes (maximum for the Ethernet)
- medium transmission rate, 10 Mbits/second
- CollisionWindow (round trip propagation delay), 64 bytes, to compute the PacketTransmissionT time as:
  $PacketTransmissionT = PurePacketT + DeferTime + (CollisionCount * CollisionWindow)$
  where
  - PurePacketT = (12208 bits)/(10 Mbits/second) = 1.220 microseconds
  - PacketSize = 1526 bytes * 8 bits/byte = 12208 bits
  - DeferTime, the binary exponential back_off used to compute the retransmission delay within a range $[0..(2 * *N) - 1]$, with $1 <= N < 1024$
  - CollisionWindow = (64 bytes * 8 bits/byte)/(10 Mbit/s) = 51.2 microseconds

4.2 METHOD_X

METHOD_X uses the same parameters as described above for CollisionWindow and PurePacketT, on the same physical medium. The total time (PacketTransmissionT) used to resolve a collision, called therefore the maximum collision resolution time, is computed differently from the CSMA/CD approach, and follows the algorithm paths from Figures 1-3:
$PacketTransmissionT = PurePacketT + CollisionWindow * 2 + CollisionT + GapT + MaxWaitT$
where

- CollisionT = CollisionCount * CollisionWindow
- CollisionCount = 1 (because there may be only one collision in METHOD_X)
- GapT = GapCount * CollisionWindow
- GapCount, total number of gaps (gap is a difference between the priority numbers of any two consecutive nodes that want to access the communication channel)
- MaxWaitT, time used by a regular node so as not to interfere with the collision resolution process in progress.

The formula to compute PacketTransmissionT for METHOD_X corresponds to Figures 1-3 as explained below.

PurePacketT. This is the time incurred by the actual transmission of the data packet on the medium. This time is only consumed by the packet size. In Figure 1, this time emanates from the **transmit** box (when **c_inv_flg** is true).

CollisionWindow*2. This is the overhead time incurred by broadcasting both, the **don't_worry** message and the **done** message. In Figure 1, boxes labeled **send_done_to_all**

and **send_don't_worry_message_to_all** represent this time.

CollisionT. This is the time used by the single collision which required the collision resolution process in question (Figure 2 when **collision_detect** is true).

GapT. This is the time used by priority gaps. These gaps emanate from the box labeled **Handle_Messages** in Figures 1 and 3 when a **don't_worry** message is not received within the first CollisionWindow time. There is a gap when none of the nodes involved in the collision has the highest priority equal one.

MaxWaitT. This is the time incurred by the first node that enters the algorithm in Figure 1 with **c_inv_flg** set to false and **c_stat_flg** set to true. The box labeled **is_swait_gt_KT** in Figure 1 is where this time is consumed.

5 Results and Discussion

This section examines the results generated by both CSMA/CD and METHOD_X simulations. These results are used to first analyze and validate the CSMA/CD simulation, next to analyze METHOD_X simulation, and then to proceed with a thorough interpretation and comparison of both methods, and finally to discuss the real-time implications.

5.1 Analysis and Validation of the CSMA/CD Simulation

The graph in Figure 5 represents an average time it takes collided nodes to resolve collision and to each transmit a data packet of Max_size (maximum size), 2/3 of Max_size and 1/3 of Max_size. Each line represents a different packet size as specified in the legend. It is consistent with known behavior of CSMA/CD. This graph also shows that, as smaller packet size (Max_size/3) doubles and even triples, their time merely increases, or does not increase proportionally to the increment in packet size. This observation confirms Hutchinson's point that the ratio of propagation delay to packet transmission time must be kept below 5% or else the collision frequency significantly degrade throughput performance [6].

Figure 6 represents the number of collisions that occurred during the collision resolution process of different packet sizes (Max_size, 2/3 Max_size and 1/3 Max_size). It appears that bigger packet sizes are involved in fewer collisions than smaller ones [11]. This is because shorter busy/idle transition period with smaller packet size results in more collisions. Figures 7 and 8 further confirm this point and support Hutchinson's observations [6].

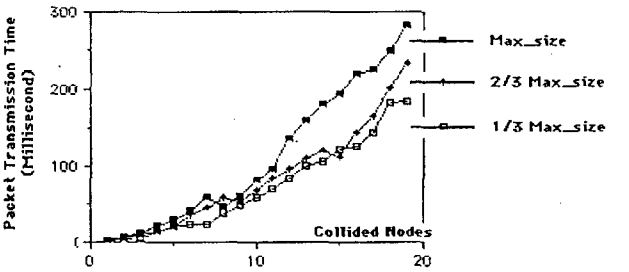


Fig. 5. Average Time to Transmit (CSMA/CD)

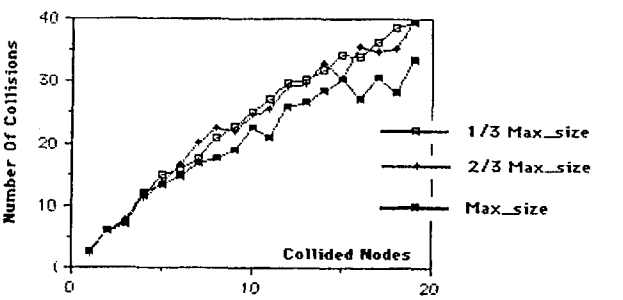


Fig. 6. Average Collisions Incurred (CSMA/CD)

Figure 7 represents the number of nodes that are involved in collisions after a collision resolution process, using different packet sizes. This figure indicates that on the average, more nodes with larger packet size are involved in a collision than those with smaller ones. This is due in part to the larger busy/idle transition period incurred with larger packet sizes.

Figure 8 represents the number of nodes that failed and were not able to transmit their packets due to excessive number of collisions (more than 10 collisions according the backoff method used by CSMA/CD). This figure reveals that nodes with larger packet size are more likely to fail than those with smaller ones. This fact can be explained as

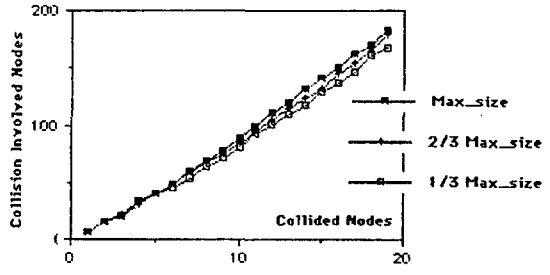


Fig. 7. Average Collision Involved Nodes (CSMA/CD)

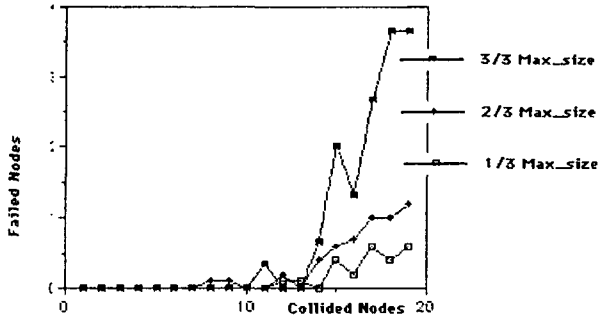


Fig. 8. Average Failed Nodes (CSMA/CD)

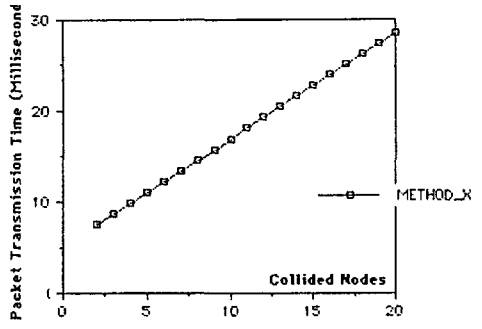


Fig. 9. Worst Case Time to Transmit (METHOD_X)

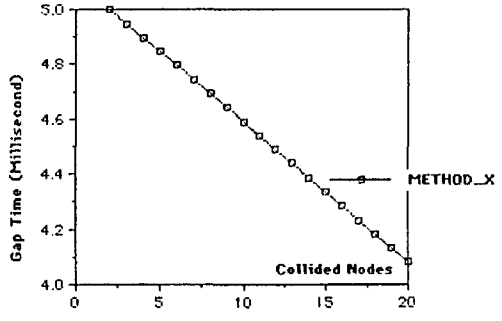


Fig. 10. Time Used for Gaps (METHOD_X)

follows: on the average, more nodes with larger packet size are involved in a collision than those with smaller ones because those with larger packet size have a longer busy/idle transition period on the network. A longer busy/idle transition period keeps more nodes waiting for that idle period after which they all collide.

The results of this simulation are consistent with observations of other authors [6, 7, 8] and thus validate the CSMA/CD simulation.

5.2 Analysis of METHOD_X Simulation

One particular aspect of METHOD_X is the fact that it uses a priority scheme to resolve collisions. The order of these priorities, as fetched by collided nodes to resolve collisions, can slow the collision resolution process as the number of unused priorities (gaps) increases. This simulation experiment takes such gaps into consideration and allows two types of situations: best case using no gaps and worst case using the maximum number of gaps possible.

Figure 9 represents the time it takes collided nodes to each transmit a Max_size packet using METHOD_X under worst case. All collided nodes are guaranteed to deliver their packets by the time indicated. Figure 10 represents the maximum overhead time incurred by gaps during the collision resolution process using METHOD_X. This overhead time decreases as the number of collided nodes increases, because the range of priorities is better filled with more nodes colliding.

Figure 11 shows the time it takes collided nodes to each transmit their maximum size packets under both best case (no gap) and worst case (maximum gaps) situations. It is noticeable that the difference between both cases is of the order of magnitude of the packet transmission time.

The graph in Figure 12 reports a time performance comparison of both simulation models (METHOD_X and CSMA/CD). This graph reveals a break-even point when about 4 nodes are involved in the collision. Below that break-even point, CSMA/CD has a better performance over

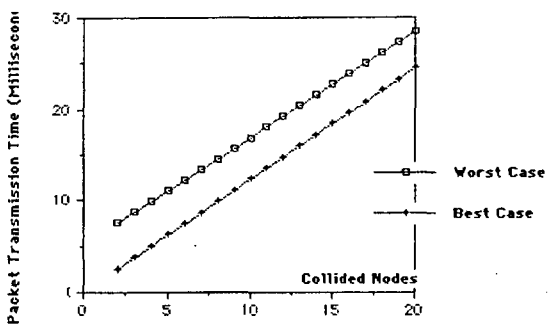


Fig. 11. Worst/Best Case Time to Transmit (METHOD_X)

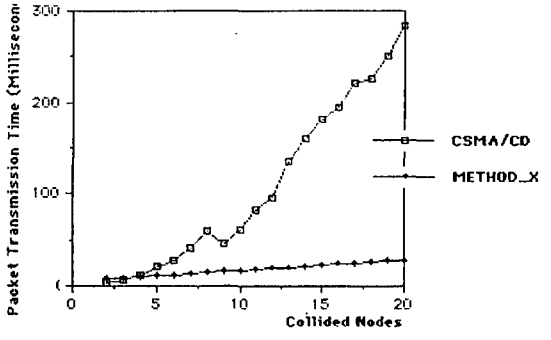


Fig. 12. Time to Transmit (CSMA/CD vs METHOD_X)

METHOD_X. This behavior can be explained by the fact that with just a few nodes colliding, the probability of repeated collision is quite small and have small retransmission intervals (CSMA/CD's backoff technique). Beyond the break-even point in Figure 12, CSMA/CD begins to deteriorate (its packet transmission time grows exponentially), while METHOD_X keeps the same pattern (straight line).

5.3 Real-Time Characteristics of METHOD_X

As mentioned in Section 3, upon a collision, METHOD_X resolves the collision, allowing no more collisions to happen until all nodes involved in that collision have successfully transmitted their packets. With this in mind, it becomes apparent that knowing the maximum number of nodes on the network, the time to resolve the collision that involves all nodes can be computed. This computed time would be the guaranteed packet delivery time for any node on the network using METHOD_X, from the time when the collision occurred.

It should be stressed that any external node not involved in the collision can not disrupt or affect the collision resolution process under METHOD_X, which is not the case with CSMA/CD. This non-interruptive aspect of METHOD_X is a key aspect of it being able to guarantee a maximum delivery time for its packets.

It can therefore be argued that METHOD_X is deterministic and can be used for real-time applications. For any node on the network, its packet delivery time should be less or equal to the maximum collision resolution time (specified by the formula in Section 4.2), plus the maximum busy/idle transition period (which is the time a node waits for the medium to be idle when it was previously busy).

6 Conclusion

This paper looked at a new medium access protocol to improve Ethernet's performance and also allow it to be deterministic. The approach used here, called METHOD_X was to isolate the problem of collisions in CSMA/CD and extend the protocol to resolve these collisions in a predictable way (to guarantee a maximum packet delivery time), by adding two flags and a priority counter to each node.

The following results were obtained from the simulation of performance of both methods:

- The CSMA/CD simulation revealed known behavior of the standard Ethernet and was successfully validated.
- Theoretical properties of METHOD_X were confirmed experimentally, which means that METHOD_X proves to be deterministic and deadlock-free.
- The final comparison showed that at very low traffic CSMA/CD performs slightly better, but beyond that point METHOD_X performance is far better than that of CSMA/CD.
- Although a disadvantage of METHOD_X is that worst case gap inflicts an overhead time of the order of packet transmission time, this overhead decreases as more nodes are involved in the collision.

References

[1] Alijani G. S., Morrison R. L., An Evaluation of IEEE 802 Protocols and FDDI in Real-Time Distributed Systems. Proc. 15th Conference on Local Computer Networks, pp. 334-342. IEEE, New York, 1990

[2] Armitage B., Dunlop G., Hutchison D., Yu S., Fieldbus: An Emerging Communication Standard. Microprocessors and Microsystems, Vol. 12, No. 12, pp. 555-562, December 1988

[3] Boudenant J., B. Feydel, P. Rolin, LYNX: An IEEE 802.3 Compatible Deterministic Protocol. Proc. INFOCOM '87 6th Ann. Conf. on Global Networks, pp. 573-579. IEEE Press, New York, 1987

[4] Court R., Real-Time Ethernet. Computer Communication, Vol. 15, No. 3, pp. 198-201, April 1992

[5] Hainich R., An Improved Ethernet for Real-Time Applications. Proc. Conf. on Real-Time Data Handling and Process Control, E.G. Kingham et al. (Eds.), pp. 293-301. North-Holland, Amsterdam, 1984

[6] Hutchison D., Merabti M., Ethernet for Real-time Application. IEE Proceedings, Vol. 134, Pt. E, No. 1, pp. 47-53, January 1987

[7] Kim C., Kim J., A Mean Value Analysis of The Ethernet Throughput. Information Processing Letters, Vol. 43, pp. 315-320, October 1992.

[8] van Oorschot J., A. Dekkers, Measuring and Simulating an 802.3 CSMA/CD LAN. Microprocessing and Microprogramming, Vol. 35, pp. 765-772, 1992

[9] Rzehak H., Abd E. E., Rudolf J., Analysis of Real-time Properties and Rules For Setting Protocols Parameters of MAP Networks, Real-Time Systems, Vol. 1, pp. 221-241, 1989

[10] Strosnider J.K., T. Marchok, J. Lechoczky, Advanced Real-Time Scheduling Using the IEEE 802.5 Token Ring. Proc. 9th IEEE Real-Time Systems Symposium, pp. 42-52. IEEE Computer Society Press, Los Alamitos, CA, 1988

[11] Uuspaa P.T., Ethernet in Real Time. Proc. Conf. Hardware and Software for Real-Time Process Control, J.Zalewski, W.Ehrenberger (Eds.), pp. 441-452. North-Holland, Amsterdam, 1989

[12] Venkatramani C., T. Chiueh, Supporting Real-Time Traffic on Ethernet. Proc. 15th IEEE Real-Time Systems Symposium, pp. 282-286. IEEE Computer Society Press, Los Alamitos, CA, 1994

Principles of a Formal Axiomatic Structure of the Informational

Anton P. Železnikar
Volaričeva ulica 8, 61111 Ljubljana, Slovenia
a.p.zeleznikar@ijs.si

Keywords: axiom, circularity, derivation, functionalism, general informational theory, inclusiveness, inference rule, parallelism, propositional vs. informational, serialism

Edited by: Vladimir Fomichov
Received: June 6, 1994 **Revised:** August 27, 1994 **Accepted:** September 15, 1994

The article deals with some basic problems of a formal axiomatic structure pertaining to the phenomenalism of the informational. In this way, solid philosophical and formal foundations of an emerging informational science are set from the strict informational point of view [9]. A general informational theory conjoins the so-called object theory and its metatheory, in contrariness to a narrower mathematical theory, where the metamathematical theory serves as an exterior means for proving of the object theory. The principles of informational axioms are treated from the dualistic point of view conjoining the axioms of the object theory and inference axioms of the metatheory. Inference rules become regular informational formulas which arise informationally as any other informational operands (entities).

1 Introduction

Axiomatic structure[1] of a general informational theory (GIT) is a problem per se, for it must be, for example, according to [9], self-contained in respect to the basic theory axioms on one side and the necessary initial inference rules (deduction, induction, abduction, modi of other possible inference) on the other side. Conception of GIT is certainly *logistic* [1] and *formalistic* [7], but not in the traditional mathematical sense. On contrary to the traditional mathematical theories, GIT can keep the inference rules within the theory itself while, in mathematics, deduction rules for example, remain outside the particular theory (e.g., in the so-called inferential metadomain, that is, metamathematics) as means by which a mathematician or machine can prove the correctness, logical consistency or non-contradictoriness of the theory.

Mathematics is not more rigorous than historiology, but only narrower, because the existential foundations relevant for it lie within a narrower range. (Heidegger [4], p. 195.) In mathematics, the metatheory by which an object theory is proved, lies outside of the object theory. It is mathematically unimaginable (uncommon) to join, for instance, an arithmetic theory (dealing with numbers) with the theory of deduction and induction, by which arithmetical theorems are proven and dealing with objects of logic of predicates. An object theory in mathematics is always meant as a narrower theory from which the metatheory is excluded.

Axiomatization in the described (informational) sense is a necessary step towards a sufficiently strict theory which can be applied as a constructing or designing tool for particular informational machines and programs. It is a sort of informational formalization [7] by which a calculus is introduced. On each step of formalization, there is certainly possible a look into the real world when formulas are *deformalized,* that is, made less formal through their interpretation (translation) in a less formal or object language (natural, picture, voice, signal, process language, etc.)

GIT is a theory of well-formed formulas of operands, operators, and parentheses pairs. It has a straightforward syntax where the formation

of its formulas depends on several informational views, possibilities, and principles (methods), giving the theory the so-called informationally arising (emerging, generating) character. For instance, some of the principles of informational arising concern procedures of formula and formula system decomposition and composition. Decomposition means, for example, deconstruction [3] in the sense of a particular semantic and pragmatic analysis of a formal item (operand, operator, formula, formula system) in the form of additional arising, enlarging, changing or modifying.

Several formal means have to be introduced before the axiomatization of the informational can begin. During our axiomatic discourse, we have the substantially different theoretical entities, the aim and purpose of which must be explained in a clear and definite manner. These entities are:

1. *Definitions* are a kind of preaxiomatic and pretheoretical determination entities which explain the introduced symbolism and symbolic structures (markers, variables, formulas, formula systems) used within the course of an informational theory advancing. Definitions are nothing else than transparent interpretations of formal symbolism for the reader in a natural language. They connect the emerging logistic and formalistic world ([1, 7]) with the natural one. They simultaneously enable the emerging of the formal theory world and its informational connection with a common (linguistic) individual consciousness. As such, definitions function like initiators beyond a particular axiomatized theory, linking the emerging formal world and the existing conscious world of the theoretically concluding mind.

2. *Theory axioms* are the essential origins of a theory obtained by an intuitive investigation of the basics by a theory setter (e.g., an expert of a scientific discipline). Informational axioms are formulas that commend themselves to general acceptance; they are informationally well-established and universally-concerned principles presenting the maxima of the possible, assignable degree of recognition.

3. *Inference axioms* are rules (laws) for deriving formulas from theory axioms and formulas obtained already through regular derivation. Inference axioms are the very initial rules for inferring, that is, for the drawing of conclusions from theory and inference axioms. Inference rules can be derived in the form of inference theorems, lemmas, consequences, etc. getting more complex and informationally interweaved inference rules. In this function, derived inference rules represent regular processes for drawing conclusions within an informational theory.

4. *Theorems, lemmas, conclusions* etc. are informationally derived theory formulas by means of inference axioms and inferred inference rules from axioms and already derived theorems, lemmas, conclusions, etc. They are "object-theoretical" (non-inferential) as well as "inferential".

5. *Proofs of theorems, lemmas, and conclusions* are procedures (informational processes) using inference axioms and generated rules with the aim to achieve certain results (in the form of theorems, lemmas, etc.) E.g., metamathematics can be understood to be a proof theory (D. Hilbert, see, for instance, [7]). In formally loose theories, the process of proving becomes an art instead of a formal procedure.

6. *Analysis of theory axioms, theorems, and proofs* occurs after the process of proving a theorem, lemma, etc. to see what could be improved, complemented, and added for the sake of a more complete and powerful theory. Thus, one can glance at induced axioms, derived theorems, and accomplished proofs.

The enumerated theory entities (definitions, theory axioms, inference axioms, theorems, proofs, and analyses) constitute a spontaneous and circular discourse in the following sense:

A. *Construction of definitions and theory axioms* is an informational approach by which the theory designer is getting his/her *master* for the emerging formal theory. The inference of further axioms and their notional improvement is on the way to the theory-axiomatic consolidation (fortifying, strengthening).

B. Construction of the inference axioms belongs to the functioning of the master and without them there would not be possible to deduce (prove) new axioms and the initial theorems, lemmas, and consequences. In case of an informational theory, inference axioms are parts of the particular informational theory and are not excluded from the object theory as it is the case in mathematics ([8], p. 30). Thus the entire informational master domain which governs the emerging of a theory uses definitions, theory axioms, and inference axioms for the *mastering* of the informational arising (development) of the theory.

C. *Construction of theorems lemmas, consequences,* etc. brings to the surface the so-called *university* or *teaching discourse.* Theorems, lemmas, consequences, etc. can now be taught as a theory truth concerning the theory relevant entities. Derived theorems can be used in the same way as axioms together with the derived inference rules. But, new theorems have to be proved in a consequent manner, so that the teaching domain obtains the theory legacy.

D. *Construction of proofs* can become a questionable task because someone *wishes* to prove a certain theorem which was constructed in advance, with the one's intention for some particular purpose, that is, intuitively. In this respect, proving of theorems can constitute the so-called *histeric's discourse.* In mathematics, the object domain (a theory) and the metadomain (metamathematics as a proof theory) are separated. A mathematician, proving his/her theory uses the metamathematical principles intuitively, for instance, in the mixed form of a natural and a formal mathematical language.

E. *Construction of analyses of the arisen theory* constitutes the so-called *analyst's discourse.* Analysis governs the arising of cycles **A**, ···, **D** and constitutes also the long cycle of a theory design, that is, **A**, ···, **E**, **A**, ···.

The mentioned names (markers) of discourses have been invented by Lacan [14] and constitute a theoretical, cyclically and subcyclically structured discourse in its entirety.

2 Introducing Informational Operands and Operators

Informational operands and operators are, together with parenthesis pairs, the basic entities of informational formulas. It must be determined definitely, what these entities are, how they are structured and which kind of symbolism is used for their presentation.

2.1 Informational Operands

Informational operands are simple and complex entities in the most general sense. They have active and passive informational properties, when we say that they inform and are informed. In this manner, active components of entities can be explicated by two usual forms: as informational operands and as included informing entities within entities themselves. The included entities perform again as regular informational entities.

2.1.1 An Introduction to Informational Operands as Informing Entities

Informational philosophy says that, irrespective of their physical, mental, social, individual nature, entities inform and are informed. This statement has the meaning of the fact that entities impact entities and themselves, and are impacted by entities and themselves in an phenomenal way, that is, according to the entities' proprietary possibilities of entities-concerning phenomena (e.g., physicalism, biologicism, mentalism, linguisticism, or any specific phenomenalism).

On the abstract or any informational level, phenomena concerning things can be marked (specifically encoded) and structured into formulas and formula systems. Usually, an entity is informationally represented by an adequate formula system in which phenomenal components of the entity occur as entities, that is informational operands, constituting together with informational operators and parenthesis pairs the so-called formula system. Markers, formulas, and formula systems are operands in the sense of informational variables if compared to adequately constructed mathematical entities.

An informational operand, representing (modeling, phenomenalizing) a real entity, is informationally structured, irrespective of the instantaneous

possibilities of revealment or hiding of its informational nature. The operand structure can come to the surface through a stepwise, informationally consistent, and consequent decomposition, which is nothing else than a process of deconstruction [3] in the sense of semantic and pragmatic analysis and synthesis of the entity-structural case. This procedure of decomposition is carried of by principles (axioms, inference rules) of informational decomposition which is a part of the so-called counterinformational phenomenon of the entity itself and it impacting environment.

Operands are representatives of simple, composed and the most complex entities of the world. In this respect, operands are operand markers, informationally well-formed formulas, and formula systems. They represent informationally arising entities in the sense of informational spontaneity and circularity, according to the principles of information [9] called *informational principles.*

2.1.2 Operands Marking Informational Entities

Simple informational entities, marked as simple operands, are the beginning of something or something which is not informationally decomposed yet. On the other hand, arbitrary complex formulas and systems can be represented by simple markers.

Definition 1 [Operands Representing Informational Entities] *Informational entities are distinguished markers for simple operands, informational formulas and informational formula systems represented as operands. The so-called* informational entities *which hide the entity's informing are marked by small letters of the Greek or Fraktur alphabet, which can be subscribed and superscribed and written in a functional form. E.g.,*

$$\alpha, \beta, \gamma, \cdots, \omega; \mathfrak{a}, \mathfrak{b}, \mathfrak{c}, \cdots, \mathfrak{z};$$
$$\alpha_1, \beta_{\text{belief}}, \gamma_\alpha, \cdots, \omega_{\mathcal{I}};$$
$$\mathfrak{a}^1, \mathfrak{b}^{\text{belief}}, \mathfrak{c}^\alpha, \cdots, \mathfrak{z}^{\mathcal{I}};$$
$$\alpha(\beta), \beta_\alpha(\gamma), \gamma_{\text{consciousness}}(C), \cdots, \omega_{\mathcal{Z}}(\alpha)$$

are examples of single simple, subscribed, superscribed and functional operands, respectively. □

2.1.3 Informational Operands Marking the Informing of Entities

Informing of an entity is meant to be an active component of (within) the entity, characterizing the entity's informational properties which can be observed by another entity or the entity itself. Informing of an entity is expressive (informational externalism) and impressive (informational internalism). Or said by other words: informing of an entity is distinguished in two characteristic ways that belong to the basic verbal forms which are *to inform* and *to be informed.*

There is no conceptual difference between informational operands as entities and informational operands as informings of entities. They are merely marked differently to distinguish them clearly between each other.

Definition 2 [Operands Representing Informing Entities of Informational Entities] *Informing entities of informational entities are distinguished markers for simple informing operands, informational formulas and informational formula systems represented as operands belonging to the informings of entities. The so-called* informings of informational entities *can hide other informational entities and their informings and are marked by capital calligraphic or Fraktur letters, which can be subscribed and superscribed and written in a functional form. E.g.,*

$$\mathcal{A}, \mathcal{B}, \cdots, \mathcal{Z};$$
$$\mathfrak{A}, \mathfrak{B}, \mathfrak{C}, \cdots, \mathfrak{Z};$$
$$\mathcal{A}_1, \mathcal{B}_{\text{believe}}, \mathcal{C}_\alpha, \cdots, \mathcal{Z}_{\mathcal{I}};$$
$$\mathfrak{A}^1, \mathfrak{B}^{\text{believe}}, \mathfrak{C}^\alpha, \cdots, \mathfrak{Z}^{\mathcal{I}};$$
$$\mathcal{A}(\beta), \mathfrak{B}_{\mathfrak{a}}(\mathfrak{c}), \mathcal{C}_{\text{be\_conscious}}(\mathfrak{C}), \cdots, \mathfrak{Z}_{\mathcal{Z}}(\alpha)$$

are examples of single simple, subscribed, superscribed and functional operands of informing of entities, respectively. □

Operands of informing explicate the operational properties (like informational operators) to them belonging or they including informational entities.

2.1.4 Functional Informational Operands

Functional operands express the informational functionalism which is an extreme generalization of function. The notion of mathematical function on the other (lower) side is a simple, reductionistic notion. In the last consequence, it represents

an algorithm (mathematical definition) by which an argument set is mapped onto or into a value set, where arguments and values can be any abstractly determined objects. On the other side, informational functions are arbitrary formulas or formula systems being informationally dependent in a complex manner, e.g. as defined recursively and mutually-informingly in [13].

Definition 3 [OPERANDS REPRESENTING INFORMATIONAL FUNCTIONS] *There are two equivalent forms of functional notation, $\varphi(\xi)$ and $\varphi^*\xi$. The first form follows the mathematical notation convention while the second one is more transparent in cases where φ and ξ are arbitrarily complex formulas or formula systems. The second form shows already the substantial (structural) difference between a mathematical and an informational function. For instance, an informational function of the form*

$$(\alpha \models \beta)^*(\gamma \models \delta)$$

where $\models$ is an informational operator, has not an adequate notional equivalent in mathematics. According to [13], the following definition of an informational function is senseful:

$$\varphi(\xi) \rightleftharpoons_{\text{def}} \begin{pmatrix} \varphi \models_{\text{of}} \xi; \\ \xi \models \varphi; \\ (\varphi \models_{\text{of}} \xi) \subset \varphi; \\ (\xi \models \varphi) \subset_{\text{of}} \varphi \end{pmatrix}$$

In this formula, operator $\rightleftharpoons_{\text{def}}$ means means by definition, *$\models_{\text{of}}$ means* is a function of *or* depends informationally on, *etc. Operators of informational inclusion $\subset$ and $\subset_{\text{of}}$ are determined recursively, in this particular case, by*

$$((\varphi \models_{\text{of}} \alpha) \subset \varphi) \rightleftharpoons \begin{pmatrix} \varphi \models (\varphi \models_{\text{of}} \alpha); \\ (\varphi \models_{\text{of}} \alpha) \models \varphi; \\ (\varphi \models (\varphi \models_{\text{of}} \alpha)) \subset \varphi; \\ ((\varphi \models_{\text{of}} \alpha) \models \varphi) \subset \varphi \end{pmatrix}$$

where operator $\rightleftharpoons$ is read means *and, for the second informational includedness, according to [12],*

$$((\alpha \models \varphi) \subset_{\text{of}} \varphi) \rightleftharpoons \begin{pmatrix} \varphi \models_{\text{of}} (\alpha \models \varphi); \\ (\alpha \models \varphi) \models_{\text{of}} \varphi; \\ (\varphi \models_{\text{of}} (\alpha \models \varphi)) \subset_{\text{of}} \varphi; \\ ((\alpha \models \varphi) \models_{\text{of}} \varphi) \subset_{\text{of}} \varphi \end{pmatrix}$$

Operators as informational entities will be defined in the next subsection. □

2.1.5 Other Informational Operands

Other informational operands concern special arrays of formulas as a consequence of, for example, informational decomposition, composition, gestalt, etc.

Definition 4 [OPERANDS REPRESENTING INFORMATIONAL DECOMPOSITION, COMPOSITION, GESTALT, ETC.] *Special informational operands are distinguished markers for simple operands, informational formulas and informational formula systems which represent arrays of formulas being a consequence of informational decomposition, composition and gestalt structuring. The so-called* special informational operands *hide systematically (e.g. metaphysically, syntactically, etc.) derived formulas and are marked by the distinguished capital letters of the Greek alphabet [11], that is $\Gamma, \Delta, \Theta, \Lambda, \Xi, \Pi, \Sigma, \Upsilon, \Phi, \Psi, \Omega$, which can be subscribed and superscribed and written in a functional form. E.g.,*

$$\begin{aligned} &\Gamma, \Delta, \Theta, \cdots, \Omega; \\ &\Gamma_{\text{composition}}, \Gamma_{\text{gestalt}}, \Delta_{\text{decomposition}}, \cdots, \Omega_{\alpha}; \\ &\Gamma(\alpha), \Delta_{\text{serial}}(\alpha), \Delta^{\text{metaphysical}}_{\text{serial-parallel}}(\alpha), \cdots, \Omega_{\mathcal{I}}; \\ &\Gamma_{\text{serial}}(\alpha_1, \alpha_2, \cdots, \alpha_n), \cdots, \\ &\qquad \Omega^{\text{metaphysical}}_{\text{gestalt}}(\psi_1, \cdots, \psi_n) \end{aligned}$$

are examples of single simple, subscribed, superscribed and functional special operands, respectively [13]. □

Definition 5 [OPERANDS REPRESENTING INFORMATIONAL INFERENCE RULES, PREMISES, CONCLUSIONS, AND OTHER ENTITIES] *Many other operand symbols can be introduced marking special informational entities or to them belonging formulas. For instance, for inferential rules, their premises and conclusions, the various alphabets of small and capital letters, e.g.*

$$\begin{aligned} &\mathbb{A}, \mathbb{B}, \mathbb{C}, \cdots, \mathbb{Z}, \\ &\mathfrak{a}, \mathfrak{b}, \mathfrak{c}, \cdots, \mathfrak{z}, \\ &\mathsf{A}, \mathsf{B}, \mathsf{C}, \cdots, \mathsf{Z} \end{aligned}$$

can be introduced. Thus,

$$\mathbb{R}^{\text{rule},i}_{\text{inferential}}(\mathsf{A}_i, \mathsf{B}_i) \rightleftharpoons \frac{\mathbb{P}^i_{\text{premise}}(\mathsf{A}_i, \mathsf{B}_i)}{\mathbb{C}^i_{\text{conclusion}}(\mathsf{B}_i)}$$

where A_i and B_i are variables of the premise and the conclusion function, respectively. □

Other specific operand symbols can be used to make entities clearly (characteristically) distinguished from each other.

2.2 Informational Operators

Informational operators inform the properties of the entities to which they belong. In this sense, they are dualistic entities in regard to the informing of entities [e.g. marked by $\mathcal{I}(\alpha)$ or $\mathcal{I}_\alpha$]. Like informing of an entity, the corresponding informational operator expresses the entity's property in an active informational manner. This correspondence is twofold: the informingness of the entity (informational externalism) and the informedness of the entity (informational internalism). In principle, various sorts of operators belong to an entity's externalistic and internalistic informing.

2.2.1 An Introduction to the Notion of Informational Operator

An informational operator expresses the general property of an entity in the form of entity's informing. It does not mean that by an operator the entire operational possibility of an operand is exhausted. A general case operator can be particularized and universalized in many ways, depending on the happening of an operand as informer and observer. We introduce the most general operator and its possible particularizations and universalizations by the following definitions.

Definition 6 [General Informational Operator as a Unique Operational Joker] *The general informational operator, marked by* $\models$, *expresses the most general property of an entity, represented by an informational operand in a simple (marker) or a complex form (formula system). Although this operator is from-the-left-to-the-right-oriented, to enable its reading in the form* inform(s) *and* are (is) informed, *it does not mean that operator* $\models$ *does not possesses, within its generality, the potentiality of being from-the-right-to-the-left-oriented. Thus, any particularized, universalized or direction-concerning informational operator has its ground in* $\models$ *and represents nothing less and nothing more than a special case of this operator. Operator* $\models$ *performs as an informational joker which can act as a substitute and (mutual) replacement of any operational phenomenon.* □

A general informational operator embraces everything which can be imagined as operator, as an informational activity (informing and informed property) of the operand to which it belongs, to which it is attributed. Introducing the operational joker has the notional roots in the potentiality for leaving open any possibility of informing and determining the joker just in case when the property of an operand (informational entity) becomes (arises, emerges as) clearly identified. Thus, general informing means informing in a free and unforeseeable way, to guarantee the phenomenalism of informational spontaneity and circularity of the entity. The operational joker implicitly expresses just this informational situation and attitude of an entity which informs and is being informed.

2.2.2 General Informational Operator and Its Particularization and Universalization

Everything which is not a general informational operator, that is, $\models$, can be understood to be particularized or universalized through a meaning attributed to the operator. Operator particularization and universalization concerns a semantical content belonging to the operator as a consequence of the operand to which the operator is bound in an informing (externalistic) or informed (internalistic) manner.

The difference between particularization and universalization is merely semantic. In fact, both mean a specialization or concretization of the operational joker. On the other hand, a particularized operator can be meaningly universalized (replaced) to some extent and up to the joker itself.

Definition 7 [Generally Particularized and/or Universalized Operational Joker] *One can introduce, together with the operational joker, four groups of four operators in the following way:*

- *Symbols* $\models, \not\models, \dashv, \not\dashv$ *are operators of informing, non-informing, alternative informing, and alternative non-informing, respectively. The alternative operators are read (from the left to the right) as* is (are) informed *and* is (are) not informed.

- *Symbols* $\|\models, \|\not\models, =\|, \not=\|$ *represent operators of parallel, non-parallel, alternative parallel, and alternative non-parallel informing, respectively. They are read in the following sense: inform(s), do(es) not inform, alternatively inform(s), alternatively do(es) not inform in parallel.*

- *Symbols* $\vdash$, $\not\vdash$, $\dashv$, $\not\dashv$ *represent operators of circular (cyclical, loop-like) informing, non-informing, alternative informing, and alternative non-informing, respectively. They are read in the following way: inform(s), do(es) not inform, alternatively inform(s), alternatively do(es) not inform circularly.*

- *Symbols* $\|\vdash$, $\|\not\vdash$, $\dashv\|$, $\not\dashv\|$ *represent operators of parallel-circular (parallel-cyclical, parallel-loop-like) informing, non-informing, alternative informing, and alternative non-informing, respectively. They are read in the following way: inform(s), do(es) not inform, alternatively inform(s), alternatively do(es) not inform parallel-circularly.*

□

Although, according to informational principles [9], informational entities inform in a circular way, the circular non-informing represents a particular (abstract) situation where circularity is excluded (e.g., mathematical formulas).

Definition 8 [SEMANTICALLY PARTICULARIZED AND/OR UNIVERSALIZED OPERATIONAL JOKER] *Arbitrary informational subscripts and superscripts for informational operators can be used. The following examples demonstrate such possibilities:*

$$\models_{\alpha}, \not\models_{\alpha}, \models^{\text{alternatively}}_{\alpha}, \not\models^{\text{alternatively}}_{\alpha},$$
$$\models^{\text{in\_parallel}}_{\alpha}, \not\models^{\text{in\_parallel}}_{\alpha}, \models^{\text{in\_parallel}}_{\text{alternatively},\alpha},$$
$$\models_{\text{circularly}}, \models^{\text{circularly}}_{\text{in\_parallel},\alpha},$$
$$\models_{\text{inferentially}}, \models^{\text{inferentially}}_{\text{by\_modus\_ponens}}$$

etc. □

Particularization and universalization of informational operators can be chosen pragmatically, according to a language convention.

2.2.3 Informational Semicolon as an Operator of Parallelism

Parallelism of phenomena belong to the most general happening of the informational. Informational semicolon, marked by ';', has the meaning of parallelism of formulas between which it is set. It can be interpreted operator-rigorously by the use of parenthesis pairs. But usually, the parenthesis pairs are omitted.

Definition 9 [SEMICOLON REPRESENTING INFORMATIONAL PARALLELISM] *A semicolon between two markers, formulas or formula systems α and β means that these operands inform in parallel irrespective of their mutual informational connection. In traditional logic, parallelism means a conjunction of logical operands, e.g. $\alpha\&\beta$ or $\alpha \wedge \beta$. Informationally, operator $\|$ could be used. The precise definition is*

$$(\alpha;\beta) \rightleftharpoons_{\text{def}} (\alpha \models_{\text{in\_parallel}} \beta)$$

Thus, operator ';' can represent any of operators $\models_{\text{in\_parallel}}$, $\&$, $\wedge$, $\|\models$ *(*$=\|$ *for an alternatively parallel case) and* $\|$. □

Circularly parallel operators describe circularly perplexed parallel cases.

2.2.4 Operator of Informational Implication

Informational implication differs essentially from the logical implication. As an operator, it appropriates the most general linguistic meaning of the verb *to imply*. For instance [15, 16],

- to enfold, enwrap, entangle, involve;
- to involve or comprise as a necessary logical consequence;
- to involve the truth or *existence* of (something not expressly asserted or maintained);
- to involve as a necessary circumstance: *informer entity implies an informed entity (informational observer);*
- to indicate or suggest as something naturally to be inferred, without express statement;
- to involve by signification or import;

- to signify, import, mean;
- to signify as much as, to be equivalent to, to mean or intend for;
- to express indirectly, to insinuate, hint at;
- to assume, include (synonymously); and
- to ascribe, attribute

are cases of a semantic correspondence. On the other hand, *to imply informationally* can simply appropriate the meanings as

- to interweave, interwine, interlace informationally; and
- to embrace, involve informationally.

Definition 10 [Operator of Informational Implication] *Operator of informational implication, marked by* $\Longrightarrow$, *is a particularized form of the general informational operator* $\models$, *e.g.* $\models_{\Longrightarrow}$, $\models_{\text{implicatively}}$, $\models_{\text{involvingly}}$, *etc. The informationally obvious reading of operator* $\Longrightarrow$ *is 'implies informationally' or 'informs implicatively' (from the left to the right side of formula).* □

Similarly as in the traditional logic, informational implication is one of the keystones of the informational reasoning and inference, by which various informational derivations can come into existence.

2.2.5 Informational Operator of Inference

Informational operator of detachment in an inference rule has usually the form of a fraction line and a specific meaning.

Definition 11 [Operator of Informational Inference] *In an expression (informational formula) of the form* $\frac{\alpha}{\beta}$, *where formula* α *is called the premise and formula* β *the conclusion, the fraction line* $(\frac{\cdots}{\cdots})$ *is an operator of informational inference which reads 'inform(s) inferentially' (or detachably). Thus,*

$$\frac{\alpha}{\beta} \rightleftharpoons (\alpha \models_{\text{inferentially}} \beta)$$

Premise α *is marked by* $\mathbb{P}$ *and is a function of at least two operands, e.g.,* A *and* B, *that is,* $\mathbb{P}(\mathsf{A}, \mathsf{B})$. *Conclusion* β *is marked by* $\mathbb{C}$ *and is a function of* B, *that is,* $\mathbb{C}(\mathsf{B})$. *Thus, instead of the inferential rule* $\mathbb{R}$ *in the form*

$$\mathbb{R}(\mathsf{A}, \mathsf{B}) \rightleftharpoons \frac{\mathbb{P}(\mathsf{A}, \mathsf{B})}{\mathbb{C}(\mathsf{B})}$$

there is

$$\mathbb{R}(\mathsf{A}, \mathsf{B}) \rightleftharpoons (\mathbb{P}(\mathsf{A}, \mathsf{B}) \models_{\text{inferentially}} \mathbb{C}(\mathsf{B}))$$

□

There are many "standard" forms of inference rules with characteristic premise and conclusion entities; they will be informationally examined in Section 6.

2.2.6 Operator of Informational Being-in—Informational Inclusivism

Informational inclusion is a recursive concept which brings to the surface the informational connectedness or interweavement of informational entities.

Definition 12 [Operator of Informational Inclusion] *Operator of informational inclusion or informational Being-in is marked by* $\subset$. *In the context of operands* α *and* β, *it is defined by the following recursive (circular) informational formula [12]:*

$$(\alpha \subset \beta) \rightleftharpoons_{\text{Def}} \begin{pmatrix} \beta \models \alpha; \\ \alpha \models \beta; \\ \Xi(\alpha \subset \beta) \end{pmatrix}$$

where for the extensional part $\Xi(\alpha \subset \beta)$ *of the includedness* $\alpha \subset \beta$, *there is,*

$$\Xi(\alpha \subset \beta) \in \mathcal{P}\left(\left\{\begin{matrix} (\beta \models \alpha) \subset \beta, \\ (\alpha \models \beta) \subset \beta, \\ (\beta \models \alpha) \subset \alpha, \\ (\alpha \models \beta) \subset \alpha \end{matrix}\right\}\right)$$

The most complex element of power set $\mathcal{P}$ *is denoted by*

$$\Xi^{\beta,\alpha}_{\beta,\alpha}(\alpha \subset \beta) \rightleftharpoons \begin{pmatrix} (\beta \models \alpha) \subset \beta, \alpha; \\ (\alpha \models \beta) \subset \beta, \alpha \end{pmatrix}$$

Cases, where $\Xi(\alpha \subset \beta) \rightleftharpoons \emptyset$, *and* $\emptyset$ *denotes an empty entity (informational nothing), are exceptional (reductionistic).* □

2.2.7 Operator of Informational Being-of—Informational Functionalism

Informational function, as defined by Definition 3 (the informational Being-of), is a recursive concept of informational dependence between informational entities and represents a generalization of the concept of a function known in mathematics. Within a functional notation, e.g. $\varphi(\xi)$, the operator of functionality remains hidden, that is, not explicitly visible. That which happens between the functional formula φ and its argument formula ξ, is ladled by Definition 3. We can understand that informational structure, if instead $\varphi(\xi)$ the notation $\varphi^*\xi$ or even $(\varphi)^*(\xi)$ is used, where both informational parts are clearly operationally distinguished. Thus, operator * functions as a complex operator according to Definition 3.

On the other hand, operator $\models_{\text{of}}$ is a narrower functional operator, expressing only a part of the functional concept. This operator can be determined into further details, symbolizing the informational dependence of functional entity φ on argumentative entity ξ.

Definition 13 [Operators of Informational Functionalism] *Symbols, which mark informational functionality of a broader and a narrower sense, are*

$$\varphi(\xi),\ ^*,\ \models_{\text{be\_a\_function\_of}},\ \models_{\text{of}},\ \cdots$$

They can be variously defined in a concrete informational manner. □

2.2.8 Composition and Decomposition of Informational Operators

Informational operators can be composed and decomposed. Composition is a process of operational design where distinguished informational operators are composed into bigger operator units. Decomposition of an operator means to deconstruct it by means of an adequate formula part introducing the so-called informational gestalts [13] into a formula with certain operands.

Definition 14 [Composed and Decomposed Informational Operators] *Two informational operators, $\models_\alpha$ and $\models_\beta$ can be composed into a new operator, applying the special symbol $\circ$, that is, $\models_\alpha \circ \models_\beta$. More complex operator compositions of operators $\models_{\alpha_1}, \models_{\alpha_2}, \cdots, \models_{\alpha_n}$ must be properly parenthesized, e.g.*

$$(\cdots((\models_{\alpha_1} \circ \models_{\alpha_2}) \circ \models_{\alpha_3}) \circ \cdots \models_{\alpha_{n-1}}) \circ \models_{\alpha_n},$$
$$\cdots$$
$$\models_{\alpha_1} \circ (\models_{\alpha_2} \circ (\models_{\alpha_3} \cdots \circ (\models_{\alpha_{n-1}} \circ \models_{\alpha_n}) \cdots))$$

To decompose an informational operator $\models$, for instance in a formula $\alpha \models \beta$, means to put between operands α and β a part of formula, that is an informational frame ([13], Definition 12), where the original formula $\alpha \models \beta$ becomes $\Subset\alpha \Rrightarrow \beta\Supset$ or, formally,

$$(\alpha \models \beta) \models_{\text{by\_decomposition}} (\Subset\alpha \Rrightarrow \beta\Supset)$$

There exist infinitely many possibilities of an operator $\models$ decomposition. □

2.2.9 Other Informational Operators

Other informational operators can be introduced pragmatically considering a language convention and the appropriateness of operator symbols. In this sense, direct (clearly symbolical), particularized and universalized operators can be introduced. For example, $\models$, $\in$, $\rightarrow$,* in $\varphi^*\xi$, $\subset$, $\Longrightarrow$, $\rightleftharpoons$, etc. belong to the class of direct informational operators. Particularized operators, e.g. $\models_{\text{inferentially}}$, express a special, narrower properties of informing entities, while universalized operators express broader properties of entities which inform and are informed. There is a hierarchy of operators in the sense of the particular towards the general, which is particular-universal-general.

3 Concept of Informational Formula

Informational formula is a well-formed sequence of informational operands, operators and parenthesis pairs. The well-formedness of informational formulas is determined recursively. A formula represents an informationally arising informational entity and behaves by itself as an arising informational entity. From the philosophical point of view, a formula is nothing else than a result of an observer's informational process, which represents the observed entity at the site and through the view of the observer.

Informational formula is a model of the informing entity which is being analyzed, deconstructed and decomposed. This process of the informational identification of an entity through an informational formula or formula system can be continued to new forms, facts, constructions, designs, etc. according to the abilities of the observing entity. The observing entity can observe itself and perform informational changes on itself, so, this principle leads to the circular informing of an entity, that is, to the circular structure of an entity representing formula.

3.1 A General Syntax of Informational Formulas (Operands and Operators)

Informational formula is a general term including also the system of informational formulas. A well-formed informational formula acts as an informational operand. E.g. informational markers are formulas which mark complex, composed formulas.

Definition 15 [INFORMATIONAL FORMULA SYNTAX] *Let α mark different informational operands α, β, $\cdots$, ω, $\mathcal{A}$, $\mathcal{B}$, $\cdots$, $\mathcal{Z}$, $\mathfrak{A}, \mathfrak{B}, \cdots, \mathfrak{Z}$, Γ, Δ, $\cdots$, Ω, $\mathbb{A}$, $\mathbb{B}$, $\cdots$, $\mathbb{Z}$, $\cdots$ and let $\models$ be the most general informational operator which can represent any operational particularization and universalization. Then, an informational formula (IF for short) is informationally well-formed, if it is constructed by the following syntactic rules, where operator $\leftarrow$ has the meaning 'becomes (gets, is replaced by)':*

1. *Operand α (as a marker) is IF.*
2. *Rule $\alpha \leftarrow (\alpha)$ says that operand α, representing a marker, formula or formula system, can be put into parentheses. Expression (α) is IF.*
3. *Rule $\alpha \leftarrow (\alpha_1, \alpha_2, \cdots, \alpha_n)$ permits the replacement of α by a list (of mutually non-informing) operands α_1, α_2, $\cdots$, α_n. Such list of operands is IF.*
4. *Rule $\alpha \leftarrow \begin{pmatrix}\alpha;\\ \alpha\end{pmatrix}$ says that α can be parallelized by α where α's can represent different entities informing in parallel. The parentheses can be omitted. Expression $\begin{pmatrix}\alpha;\\ \alpha\end{pmatrix}$ is IF.*
5. *Rule $\alpha \leftarrow (\alpha \models)$ allows the replacement of α by $\alpha \models$. Expression $\alpha \models$ is IF.*
6. *Rule $\alpha \leftarrow (\models \alpha)$ enables the replacement of α by $\models \alpha$. Expression $\models \alpha$ is IF.*
7. *Rule $\alpha \leftarrow (\alpha \models \alpha)$ says that operand α can be replaced by $\alpha \models \alpha$ (where the parentheses are omitted), and the first and the second operand α can differ arbitrarily. Expression $\alpha \models \alpha$ is IF.*

This list of syntactic rules can be broadened if necessary. □

Other syntactic structures are already deduced by the defined list of syntactic rules. For instance, function $\varphi(\xi)$ is nothing else than an expression $\varphi^*\xi$, where * is informational operator, that is, $\varphi(\xi) \rightleftharpoons (\varphi \models_{\text{functionally\_on}} \xi)$.

Definition 16 [INFORMATIONAL OPERATOR COMPOSITIONS] *Compositions of informational operators (IO for short), where $\circ$ marks the operator composition, underlie the following operator syntactic rules:*

1. *Symbol $\models$ represents the general IO.*
2. *Operator rule $\models \leftarrow (\models_\alpha \circ \models_\beta)$ says that operator $\models$ can be replaced by a meaningly adequate composition $\models_\alpha \circ \models_\beta$ of operators $\models_\alpha$ and $\models_\beta$. An operator composition is IO.*
3. *If in an operator composition there are more than two operators, they must be adequately parenthesized, e.g.*

   $$\models_\alpha \circ (\models_\beta \circ \models_\gamma) \quad \text{or} \quad (\models_\alpha \circ \models_\beta) \circ \models_\gamma$$

   etc., where complex compositions are IO's.
4. *Operator rule*

   $$\models \leftarrow (\models_{\text{particularly}} \vee \models_{\text{universally}} \vee \models_{\text{directly}})$$

   where $\vee$ means 'is alternative to', says that IO $\models$ can be replaced by operators $\models_{\text{particularly}}$ or $\models_{\text{universally}}$ or $\models_{\text{directly}}$ which are IO's. Operator $\models_{\text{directly}}$ represents the so-called directly expressed operators, e.g. $\subset$, $\rightleftharpoons$, $\Longrightarrow$, *, $\leftarrow$, etc.

Operator compositions follow a conceptual semantics of the designing and designed entity and their syntax (parenthesizing) is determined by Definition 14. □

The presented informational syntax is in no way a final case. The syntactic concepts can be refined or detailed according to informational circumstances.

3.2 Equivalence of Informational Formulas

There does not exist an informaional equivalence of different informational formulas. But, equivalence relations between different formalized expressions can be introduced on an abstract and reductionistic level, e.g. in mathematics. Within GIT, it is possible to observe different formulas with similar informational meaning (semantics, pragmatics). Thus, for example, formulas α and $\alpha \models \alpha$ are not equivalent because the second formula is a derivation of the first formula in the sense of a consecutive application of modus ponens. The meaning of α is a marker, the meaning of $\alpha \models \alpha$ points to an inner circular (metaphysical, deconstructive, decompositional) structure of entity represented by α.

3.3 Implicitness and Explicitness of Informational Formulas

In concern to the discussion in the previous subsection, formula α as a marker is entirely implicit as long as its meaning is determined on some other place by some other formula. We say, that irrespective of the existence of such other, meaningly determined formulas, formula α hides the implicitness of its informational potentiality. This means that α as any other, regular informational entity, can be decomposed into more details, determining its structure which, through decomposition, becomes more and more complex, e.g. serially as well as in a parallel manner. For example, $\alpha \models \alpha$ is the first (although formally trivial) step on the way of informational decomposition. In this sense, formula $\alpha \models \alpha$ informs more explicitly than does formula α. The possibility for a further explicitness of a formula does always exist.

3.4 Formula Parallelism

The possibilities of parallelism of informational formulas do always exist. The syntactic rule

$$\alpha \leftarrow \begin{pmatrix} \alpha; \\ \alpha \end{pmatrix}$$

is simultaneously a regular principle of an entity parallel decomposition. By this rule, parallel formulas, concerning entity α can be generated ad infinitum.

Informational parallelism is straightforward and cyclic, depending on the structure of parallel formulas.

3.5 Formula Serialism

The possibilities of serialism of informational formulas do always exist. The syntactic rule

$$\alpha \leftarrow (\alpha \models \alpha)$$

assures the arising of serial formulas, which can be straightforward, circular or metaphysical [10]. A straightforward serial formula is, for example,

$$(\cdots(\alpha \models \beta) \models \cdots\psi) \models \omega$$

and all other formulas obtained by the well-formed displacement of the parenthesis pairs. A cyclical serial formula is, for instance,

$$((\cdots(\alpha \models \beta) \models \cdots\psi) \models \omega) \models \alpha$$

and all other formulas obtained by the well-formed displacement of the parenthesis pairs. A metaphysical serial formula is, for example,

$$(((((\alpha \models \mathfrak{I}_\alpha) \models \mathfrak{C}_\alpha) \models \mathfrak{c}_\alpha) \models \mathfrak{E}_\alpha) \models \mathfrak{e}_\alpha) \models \alpha$$

where $\mathfrak{I}_\alpha$ is informing, $\mathfrak{C}_\alpha$ is counterinforming, $\mathfrak{c}_\alpha$ is counterinformational entity, $\mathfrak{E}_\alpha$ is informational embedding, and $\mathfrak{e}_\alpha$ is embedding informational entity of informational entity α. All other metaphysical formulas concerning α can be obtained by the well-formed displacement of the parenthesis pairs.

3.6 Parallel and Serial Circularity of Informational Formulas

The circularity of formulas can become very complex, for example, parallel-serial and serial-parallel.

A parallel-serial circularity is given by a set of parallel formulas which are structured in such a manner that a certain transitivity of occurring operands through these parallel formula takes place. A trivial example of a parallel-serial scheme would be a formula system, marked by $\psi^{\|}$, that is,

$$\psi^{\|} \rightleftharpoons \begin{pmatrix} \alpha \models \alpha_1; \\ \alpha_1 \models \alpha_2; \\ \cdots \\ \alpha_{n-1} \models \alpha_n; \\ \alpha_n \models \alpha \end{pmatrix}$$

A serial-parallel circularity is obtained if in a serial formula parallel subformulas appear, for instance, in the form

$$(\cdots((\alpha \models \alpha_1^{\|}) \models \alpha_2^{\|}) \models \cdots \alpha_n^{\|}) \models \alpha$$

where $\alpha_1^{\|}$, $\alpha_2^{\|}$, $\cdots$, $\alpha_n^{\|}$ are parallel arrays of formulas.

The reader can imagine how this basic example can become more and more complicated.

3.7 The Case of Formula $\alpha \models \beta$

The case of formula $\alpha \models \beta$ offers a unique opportunity for clarification of the problem existing between α as informer and β as observer of α

Definition 17 [The Informer and Observer Problem Concerning Formula $\alpha \models \beta$] *Considering the concept of operator composition in Definition 14, one has the following definition:*

$$(\alpha \models \beta) \rightleftharpoons_{\text{def}} (\alpha \models_\alpha \circ \models_\beta \beta)$$

Operator composition $\models_\alpha \circ \models_\beta$ *performs as an informational transition filter between entities represented by operands* α *and* β. □

This definition explains how observer β can be informed about α only to the extent within which α informs in an α's specific way and β is capable to be informed in a β's specific way.

It seems senseful to explain the nature of informingness $\alpha \models$ and informedness $\models \beta$ additionally. The first case belongs to informational externalism and means that entity represented by an operator marked by α informs strictly within the informing abilities of α, that is, α-characteristically, or

$$\alpha \models_\alpha \circ \models$$

The occurring operator composition $\models_\alpha \circ \models$ demonstrates that the informing of α happens openly to the entire informational domain (field, space, also realm) through operator $\models$ at the right end of the operator composition.

In the second case we have to do with informational internalism, which means that entity represented by operator marked by β is informed (in fact, can be informed) strictly in the framework of the informing abilities (informedness) of β, that is, β-characteristically, or

$$\models \circ \models_\beta \beta$$

The occurring operator composition $\models \circ \models_\beta$ demonstrates that the informedness of β happens openly to the entire informational domain through operator $\models$ at the left beginning of the operator composition.

Within formula $\alpha \models \beta$ the described informational openness of the left and the right operand is blurred (however, implicitly present). Thus, possible complete meanings of the formula would be

$$\begin{array}{ll} \alpha\,(\models_\alpha \circ \models) \circ (\models \circ \models_\beta)\,\beta & \text{or} \\ \alpha\,((\models_\alpha \circ \models) \circ \models) \circ \models_\beta \beta & \text{or} \\ \alpha \models_\alpha \circ (\models \circ (\models \circ \models_\beta))\,\beta & \end{array}$$

where in the basic form $\alpha \models \beta$ the characteristic operator parts $\models_\alpha$ and $\models_\beta$ are implicit (invisible) and in the compositional form $\alpha \models_\alpha \circ \models_\beta$ the general operator (joker) $\models$ is superfluous.

3.8 Inferential Informational Formulas

An inferential informational formula or inference in short has a general form

$$\frac{\alpha}{\beta} \quad \left(\text{or} \quad \frac{\mathbb{P}}{\mathbb{C}}\right)$$

where α marks the *premise* (marked also by $\mathbb{P}$) and β the *conclusion* (marked also by $\mathbb{C}$). Thus, by $\frac{\alpha}{\beta}$ (or $\mathbb{P}/\mathbb{C}$), there is an *inferring* (informing in an inferential manner) from α to β (or $\mathbb{P}$ to $\mathbb{C}$). What stands under the inferential line (informational operator of inference) is always a conclusion (operand marker, formula or formula system) and above of it, a premise (operand marker, formula or formula system). Premise means assumption,

postulate, hypothesis, axiom, principle, and the like. E.g., a postulate is something (informational formula) taken as self-evident or assumed without proof as a basis for reasoning. Thus, a postulate within a premise performs as an axiom or as an already derived operand, theorem, formula, formula system, etc.

E.g., one cannot say that α infers β; but, one can always observe that α *informs inferentially* β, for instance in the sense, that the occurrence of α calls for an inference to β or that from α, there can be inferred to β, etc. In this manner, the informing of something expresses the capability or characteristics of the informing entity in respect to the informed entity.

In case $\frac{\alpha}{\beta}$ we have the situation which must not be forgotten:

$$\left(\frac{\alpha}{\beta}\right) \rightleftharpoons (\alpha \models_{\alpha,\,\text{inferentially}} \circ \models_{\beta,\,\text{inferentially}} \beta)$$

If $\alpha \models_{\alpha,\,\text{inferentially}}$, there can be not only $\models_{\beta,\,\text{inferentially}} \beta$, but any other kind of conclusion, say, $\models_{\gamma,\,\text{inferentially}} \gamma$, with another, informationally different (logical) structure of γ in comparison to β. Rules of inference can arise as any other regular informational formula. If one proceeds from standard inferential rules (e.g., tertium non datur, modus ponens, modus tollens, etc.), it does not mean that arbitrary inferential rules cannot come to the theoretical surface or cannot emerge during the theoretical discourse.

Further, it must be clarified what can the application of an inferential rule ρ (or $\mathbb{R}$) upon an informationally approved formulas α and β (acting as a premise $\mathbb{P}$ and conclusion $\mathbb{C}$) mean, for instance, in the form of the informational Being-of or functionalism $\rho(\alpha, \beta)$ [or $\mathbb{R}(\mathbb{P}, \mathbb{C})$]. In this case, inferential rule ρ becomes an informational function over formulas (formula systems) marked by α and β.

4 The Propositional and the Predicate versus the Informational

4.1 Traditional and Informational Logic

Theory of logical propositions and predicates (for instance, [5, 8]) introduces propositions and predicates (logical functions concerning elements as functional arguments belonging to arbitrary sets) in the value domain of truth and falseness (untruth). Informational entities and informational functions concern formulas which, within them and in parallel, can produce formulas as results or "values". Let us demonstrate the difference between both approaches on the level of existence of something, truth of predicates, and informing of something.

Something, marked by α, certainly has the property of existence. The framed expression $\boxed{\text{something exists}}$ is a formula which transits in a predicate form $\boxed{\boxed{\text{something exists}}\ \text{is true}}$. In a formalized way, the predicate form

$$\boxed{\boxed{\alpha\ \text{exists}}\ \text{is true}}$$

corresponds to the predicate $E(\alpha)$. A predicate is always understood to be a matter of the observer, e.g. mathematician. On the other hand, informational formula which expresses the fact 'α exists' or, more precisely, 'α informs to exist', or 'α informs existingly', that is, $\alpha \models_{\text{exist}}$, belongs not to the observer, but to the informer α as a property of its informing. The predicate form $E(\alpha)$, expressed in an informational form, would be

$$(\alpha \models_{\text{exist}}) \models_{\text{true}}$$

The truth of a predicate concerns the predicate and not the entity as an argument of the predicate. Thus, the framed expression

$$\boxed{\boxed{\text{entity exists}}\ \text{informs true}}$$

can be understood as a predicate $E(\alpha)$ with an implicit (assumed) faculty of trueness on one side and as an informational formula $(\alpha \models_{\text{exist}}) \models_{\text{true}}$ which transparently (expressively) informs the faculty of the entity itself, for example, in the form as: *entity is, as it does exist, and as it does exist in a true way.*

The other (contrary) case is

$$\boxed{\boxed{\text{entity does not exist}}\ \text{informs true}}$$

which corresponds to the predicate form $\overline{E(\alpha)}$ and to the informational form

$$(\alpha \not\models_{\text{exist}}) \models_{\text{true}}$$

In this point, one has to clarify how does entity α (in German, das Seiende α) not exist. The answer is: *in a certain way!* Informational operator $\not\models_{\text{exist}}$ is in respect to operator $\models_{\text{exist}}$ nothing else than a particularized operator of the type $\models_{\text{exist}}$. This is uniquely not clear in case of $\overline{E(\alpha)}$.

The difference which can now be drawn between the predicate view and the informational one is the following: the predicate view concerns implicitly the observer of an entity while, on contrary, the informational view concerns explicitly the informer and the observer and where both can perform in one and/or another way, that is, informer as informer and observer, and observer as informer and observer, simultaneously. In the informational case, the observer must be explicitly present (marked) and must inform and be informed explicitly, through concrete, particularized or universalized informational operators. While the predicate case concentrates on the observer and the informer (the argument or variable of the predicate) is only an object of the observer, the informational view distributes the informing between both the observer (the informedness) and informer (the informingness).

Axiomatically, the informer stands before the observer and the observed (the informing entity, that is, informer) can only be that which informs. Both are informationally active and passive entities (subject and object, simultaneously) and explicitly present (informationally determined).

Propositional and predicate logic stress the observer's view, that is, the so-called *informational internalism.* Informational logic unites the so-called *informational externalism* and *internalism* in the framework of informational metaphysicalism and phenomenalism. And, in this kind of view lies the novelty and the power of informational arising as a spontaneous and circular phenomenon, within the discourse which is on the way and which follows.

4.2 The "Value" of an Informational Formula

The concept of value belongs to the basic mathematical concepts. In mathematics [15, 16], value is the precise number or amount represented by figure, quantity, etc. For instance, numbers, elements of sets, truth and falsity, magnitude, a point in the range of a function, the value of a word, etc. are values for variables and functions. A similar question can be reasonably put to the surface in case of the informational: which are the values of informational operands as variables, markers, formulas and formula systems? How can informational values be achieved (accessed) and what do they represent as informational formulas?

In music, value is the relative length or duration of a tone signified by a note. In painting, it is due or proper effect or importance; relative tone of colour in each distinct section of a picture; a patch characterized by a particular tone. In philosophy, value means axiology.

An informing formula produces formula-like results. A result can be understood as a part, piece of formula, as an arising parallel formula or formula system, which value is a semantically and pragmatically converted, transverse sort of information, e.g. text, picture, voice, signal, etc. The same principle can be used for the domain of input operands, that is, informational variables, formulas, etc. as informing entities.

Informational formulas are, in respect to the natural means (languages, pictures, voices, signals, etc.), adequately informationally encoded entities which, in any state or position, can be understandingly decoded as values, in a "natural" form. Informational encoding and decoding can use any formalized (mathematized, systemized, procedural, etc.) means, methods, concepts, algorithms, apparatuses, approaches, formulas, etc. as well as those of the informational view, science, theory, systems, etc.

4.3 Logical and Informational Examples

Tautology and (informational) circularity are the focal problems of traditional mathematical and informational logic. We will show how a syntactically circular formula in mathematical logic is never comprehended as a circular (tautological) scheme while within an informational logic just this type of syntactic expression is considered to be circular. Thus, traditionally, the formula circularity (tautology) is pushed off the conscious horizon, while informationally circularity is considered as a cyclically operating kind of informing. To get a clear picture of such phenomenalism, we will use examples concerning the so-called foundations of mathematics, that is, its metatheory.

Example 1 [IMPLICATIVE AXIOMS FOR PROPOSITIONAL CALCULUS] *Formalizing the logical reasoning (inference) in propositional calculus ([5] p. 66), Hilbert lists a (geometrical) group of his axiom formulas of implication:*

$$\begin{array}{rcl} A & \rightarrow & (B \rightarrow A), \\ (A \rightarrow (A \rightarrow B)) & \rightarrow & (A \rightarrow B), \\ (A \rightarrow B) & \rightarrow & ((B \rightarrow C) \rightarrow (A \rightarrow C)) \end{array}$$

Informationally, these implicative formulas exert a sort of circularity regarding propositional operands A, B and C. □

The first formula says that proposition A, if not an axiom, has its logical cause in a proposition B. It simply means the following: *Something implies that it is implied by something other.* The second formula stresses that if proposition A implies an implication $A \rightarrow B$, then A implies B. It can be interpreted as: *If something implies that it implies something other then something implies something other.* The third formula says: if A implies B then the implication $B \rightarrow C$ implies also the implication $A \rightarrow C$. Said by other words, there is: *Something implies something other implies the following: if something other implies something third then something implies something third.*

The listed axioms are in a certain accord with common sense. All of them are identically true logical formulas, which can be easily verified by

$$\begin{array}{rcl} \overline{A} & \vee & \overline{B} \vee A, \\ (A \wedge \overline{B}) & \vee & \overline{A} \vee B, \\ (A \wedge \overline{B}) & \vee & (B \wedge \overline{C}) \vee \overline{A} \vee C \end{array}$$

respectively.

Reading the original formulas, a mathematician does not observe the circular structure of the listed axiomatic formulas. Implication seems to be such a kind of the logical operator which does not evoke the 'feeling' of circularity although the markers of one and the same kind are used several times in the implicative expression (e.g. in implication of implication). This fact becomes informationally true if in original formulas the logic implication operator is replaced (universalized) by the informational joker and operands are adequately marked by α_A, β_B and γ_C that is,

$$\begin{array}{l} \alpha_A \models (\beta_B \models \alpha_A); \\ (\alpha_A \models (\alpha_A \models \beta_B)) \models (\alpha_A \models \beta_B); \\ (\alpha_A \models \beta_B) \models ((\beta_B \models \gamma_C) \models (\alpha_A \models \gamma_C)) \end{array}$$

The first formula is circular in α_A, the second one in α_A and β_B, and the third one in α_A, β_B and γ_C. All together form a parallel informational system (operator ';').

Example 2 [IMPLICATIVE AXIOMS FOR INFORMING OF INFORMATIONAL ENTITIES (OPERANDS)] *An instructive, now informational case with informational implication operator $\Longrightarrow$ is*

$$\begin{array}{rcl} \alpha & \Longrightarrow & (\beta \Longrightarrow \alpha); \\ (\alpha \Longrightarrow (\alpha \Longrightarrow \beta)) & \Longrightarrow & (\alpha \Longrightarrow \beta); \\ (\alpha \Longrightarrow \beta) & \Longrightarrow & ((\beta \Longrightarrow \gamma) \Longrightarrow \\ & & (\alpha \Longrightarrow \gamma)) \end{array}$$

which leads to the basic informational axioms, for example, of the form

$$\begin{array}{rcl} (\alpha \models) & \Longrightarrow & (\alpha \Longrightarrow (\alpha \models)); \\ (\alpha \Longrightarrow (\alpha \Longrightarrow (\alpha \models))) & \Longrightarrow & (\alpha \Longrightarrow (\alpha \models)); \\ (\alpha \Longrightarrow (\alpha \models)) & \Longrightarrow & (((\alpha \models) \Longrightarrow \\ & & (\models \alpha)) \Longrightarrow \\ & & ((\alpha \Longrightarrow (\models \alpha))) \end{array}$$

The last example shows how the "global" Hilbert's implicative axioms can reasonably be applied in the informational case where the traditional Truth of Propositions *is replaced by the* Informing of Informational Formulas. □

5 Phenomenalistic Axioms of the Informational

> Zur Erleichterung soll bei den ersten Axiomen die sprachliche Fassung hinzugefügt werden.
>
> —D. Hilbert und P. Bernays [5] 5

The so-called phenomenalistic axioms of the informational are meant to be the axioms of the object and metatheory, and the inference axioms (initial rules for informational inference) underlie the general informational phenomenalism. General informational theory is, namely, a unit of the formal object theory and the formal metatheory, that is, the theory of informational inferring (proving, causing, concluding—deriving). As stated in the quotation of this section, the natural language comprehension cannot be avoided at the very beginning of the presented informational axiomatization.

In this section the basic axioms will be presented in an aprioristic and postprioristic manner. The independence of axioms will not be considered. Later on, it will become clear that only one informational axiom can be chosen, however, by the use of informational inference rules, other axioms can be derived (deduced).

5.1 Informational Externalism

Let us try to state which sort of axiom could be quite on the top of the informational. Already in Example 2 we have applied Hilbert's axioms [5] for the informational case.

Axiom 1 [INFORMATIONAL EXTERNALISM] *Aprioristically (commonsensically, trivially, intuitively [6]) at the top of the informational (system) has to be something which deepeningly (most essentially) concerns an informational entity α. So, let it be an informational implication of the form*

$$(\alpha \Longrightarrow (\alpha \Longrightarrow (\alpha \models))) \Longrightarrow (\alpha \Longrightarrow (\alpha \models))$$

This axiomatic formula says: "If informational entity (represented by operand α) implies that it implies its informing(ness), then the entity implies that it informs." The next, substantial axiomatic rule of informational externalism (according to Example 2 [5]) is

$$(\alpha \models) \Longrightarrow (\alpha \Longrightarrow (\alpha \models))$$

Informing(ness) of α implies that α itself is the cause of its informing(ness). □

According to the last axiomatic formulas, everything informational, irrespective of its informational structure or complexity, informs. Both formulas are informationally (and traditional-logically) consistent, that is, informationally (logically) noncontradictory. In traditional logic, pertaining to truth, it would mean, that both formulas are true (even identically true) or, expressed informationally, $(\alpha \Longrightarrow (\alpha \models)) \models_{\text{true}}$.

Example 3 [EXTERNALISM OF NON-INFORMING] *Informing and non-informing of an entity α are parallel phenomena. Non-informing may be comprehended as a particular phenomenon of informing. Thus, operator $\not\models$ which reads 'does not inform', is a particular case of operator $\models$.*

$\alpha \models$ means that α informs in a specific manner, that is α-characteristically. $\alpha \not\models$ means that α does not inform in a certain way, that is, it informs α-non-characteristically. Thus,

$$(\alpha \models) \rightleftharpoons (\alpha \models_{\alpha}) \quad \text{and} \quad (\alpha \not\models) \rightleftharpoons (\alpha \not\models_{\alpha})$$

Operator $\models_{\neq\alpha}$ would mean informs differently in comparison to α-characteristically.

According to Axiom 1, for a particular case concerning α, there is $\alpha \Longrightarrow (\alpha \models_{\text{particularly}})$. According to this principle, also

$$\alpha \Longrightarrow (\alpha \not\models)$$

holds. This implication will become significant in our further discussion. □

5.2 Informational Internalism

Informational internalism is a dualistic concept in regard to informational externalism. Axiomatically, the question arises, which of both phenomena is the primary one and which is the consequence of the other. Thus, quite at the beginning of axiomatization, the next axiom could also be accepted.

Axiom 2 [INFORMATIONAL INTERNALISM] *Aprioristically (commonsensically, trivially) at the top of the informational could also be something which deepeningly (most essentially) concerns an informational entity α in the sense of its informedness. So, let introduce an informational implication of the form*

$(\alpha \Longrightarrow (\alpha \Longrightarrow (\models \alpha))) \Longrightarrow (\alpha \Longrightarrow (\models \alpha))$

This axiomatic formula says: "If informational entity (represented by operand α) implies that it implies its informedness, then the entity implies that it is informed." The next, substantial axiomatic rule of internalism is, for instance,

$$(\models \alpha) \Longrightarrow (\alpha \Longrightarrow (\models \alpha))$$

Informedness of α implies that α itself is the cause of its informedness. □

According to the last axiomatic formulas, everything informational, irrespective of its informational structure or complexity, is informed. Both formulas are informationally (and traditional-logically) consistent, that is, informationally (logically) noncontradictory. In traditional logic, pertaining to truth, it would mean,

that both formulas inform true (are identically true), or expressed informationally, $(\alpha \Longrightarrow (\models \alpha)) \models_{\text{true}}$.

5.3 Informational Metaphysicalism

Informational metaphysicalism is a general and entity specific way of circular informing. In general, it proceeds from the initial circular form $\alpha \models \alpha$ which is trivially circular, but becomes structurally circular by decomposition. Specifically, the decomposition of this form can be standardized to some extent, introducing explicitly the components of informing, counterinforming and informational embedding as entities which inform within an informational entity [10].

Axiom 3 [INFORMATIONAL METAPHYSICALISM] *Aprioristically (commonsensically, trivially, intuitively) at the top of the informational could also be something which deepeningly (most essentially) concerns an informational entity α in itself, as its inner informing or informational arising, called metaphysicalism. So, we can introduce an informational implication of the form*

$$(\alpha \Longrightarrow (\alpha \Longrightarrow (\alpha \models \alpha))) \Longrightarrow (\alpha \Longrightarrow (\alpha \models \alpha))$$

This axiomatic formula says: "If informational entity (represented by operand α) implies that it implies its metaphysicalism, then the entity implies that it informs and is informed circularly." The next, essential axiomatic rule of metaphysicalism is, for instance,

$$(\alpha \models \alpha) \Longrightarrow (\alpha \Longrightarrow (\alpha \models \alpha))$$

Circular informing within α itself implies that α itself is the cause (phenomenon) of its metaphysicalism. □

According to the last axiomatic formulas, everything informational, irrespective of its informational structure or complexity, informs and is informed in a metaphysical manner.

5.4 Informational Phenomenalism

Informational phenomenalism means a parallelism of informational externalism and internalism regarding an informational entity α. By this, an entity is open as an informer and observer to its environment and to itself (metaphysicalism). Informational phenomenalism is the most general concept of informing of entities. This belief can lead to the axiom which follows.

Axiom 4 [INFORMATIONAL PHENOMENALISM] *Aprioristically (intuitively, commonsensically, trivially) at the top of the informational could also be something which deepeningly (most essentially) concerns an informational entity α toward the outside (outward, externally), toward the inside (inward) and in itself, as its entire informing or informational arising, called phenomenalism. So, we can introduce an informational implication of the form*

$$\left(\alpha \Longrightarrow \left(\alpha \Longrightarrow \left(\begin{matrix}\alpha \models;\\ \models \alpha\end{matrix}\right)\right) \Longrightarrow \left(\alpha \Longrightarrow \left(\begin{matrix}\alpha \models;\\ \models \alpha\end{matrix}\right)\right)\right.$$

This axiomatic formula says: "If informational entity (represented by operand α) implies that it implies its phenomenalism, then the entity implies that it informs externalistically and is informed internalistically." The next, essential axiomatic rule of phenomenalism is, for instance,

$$\left(\begin{matrix}\alpha \models;\\ \models \alpha\end{matrix}\right) \Longrightarrow \left(\alpha \Longrightarrow \left(\begin{matrix}\alpha \models;\\ \models \alpha\end{matrix}\right)\right)$$

Phenomenal informing of α implies that α itself is the cause (phenomenon) of its phenomenalism. □

According to the last axiomatic formulas, everything informational, irrespective of its informational structure or complexity, informs and is informed in a phenomenal manner.

It will be shown how Axioms 2, 3 and 4 can be derived form Axiom 1 if the axiomatic inference rule of informational modus ponens is adopted.

6 Axioms Related to Informational Rules of Inference

Inference rules of a theory pertain to the theory's metatheory, which performs as a theory of theory. In this function, a metatheory concerns the proving, founding, logicism and formalism of a theory, that is, in regard to metatheory, the object

theory. Separation between the object and metareasoning is traditional and roots in mathematics, in its platonistic (tautological) approach with the intention to make theories function in a noncontradictory, logically consistent and reductionistic way.

6.1 The True versus the Informational

Truth is the central concept of any mathematical theory and of mathematics as such. Everything derived from axioms by rules of inference must be true. Through mathematical proofs, the truth of theorems or derived consequences must be verified. Otherwise, the derived results are not mathematically correct. The basic question is how this traditional approach could be diversified in such a way that the mathematical truth becomes only a particular informational entity (operand) or an entity's property (operator)?

In Subsection 4.3 we have shown a possible difference between the true and the informational. The difference can exist in the following different manners:

N	The true	The informational
1	Logicism	Informationalism
2	Particularization	Generalization
3	Tertium non datur	Various informing
4	A is true or false	α is informational
5	A informs true	α informs
6	$A \models_{\text{true}}$	$\alpha \models$
7	A informs false	α does not inform in a way
8	$A \models_{\text{false}}$ or $A \not\models_{\text{true}}$	$\alpha \not\models$
9	A is informed true	α is informed
10	$\models_{\text{true}} A$	$\models \alpha$
11	A is not informed true	α is not informed
12	$\not\models_{\text{true}} A$ or $\models_{\text{false}} A$	$\not\models \alpha$

The last list of differences illustrates only the initial possibilities; so the reader can continue to list further imaginable differences.

6.1.1 Identical Truth of Propositions and Predicates

Propositional formulas (which are propositions representing logically connected propositions) can be constructed in such a way that they do not depend on the true and false values of their operands. Such formulas are said to be *identically true* or identically false. For instance, propositional formula $A \rightarrow (B \rightarrow A)$ is identically true, while formula (its negation) $\overline{A \rightarrow (B \rightarrow A)}$ is identically false.

The triviality of logical axioms and rules of inference lies in their identical trueness. For instance, the pure implicative axioms of logic ([5], p. 66) are identically true, that is, they do not depend on the values of their propositional arguments. The same is valid for the derivation (deduction) rules of the type modus ponens and modus tollens, which can be logically transcribed into $(A \wedge (A \rightarrow B)) \rightarrow B$ and $((A \rightarrow B) \wedge \overline{B}) \rightarrow \overline{B}$, respectively. In this way, something hidden (unrevealed, intuitive and, also, tautological) remains in the background of these rules of inference.

6.1.2 The Value Domain of the Logical

Following the principle of tertium not datur, there are only two values of propositions and predicates in traditional logic, that is—true and false. In multivalued logic, more than two values are permitted and gradations between true and false value are possible. But the nature of the principles of trueness remains preserved in various manners (e.g., probabilistically, modally [2], etc.). A *valuation* is an assignment of truth values ($\top$ and $\bot$) to the proposition sentences after their semantic analysis.

6.1.3 The Formula Value Domain of the Informational

The informational formula value domain is formula-like. Arguments and values of informational formulas are formulas as inputs and outputs. The difference to the traditional-logical is that arguments can influence the entity in question to an informational extent (in trivial cases also to an 'entire' extent) and that the so-called values (results) are 'produced' (influenced) only to some informational extent (trivially, to a full extent).

Informational formulas simply absorb the propositional and predicate power of a traditional-logic apparatus (calculus).

'To influence to an informational extent' means to impact something informationally not as a product but as an already existing entity; and the similar concerns the informational impactedness, where something performs its influence on the entity impacted by it.

6.2 Mathematical Implication versus Informational Implication

As the mindful reader can observe, the mathematical implication is not only logical. In fact, the implication is a metamathematical (philosophical, intuitive) connective of arguments, closely tied to the semantics of each argument and the implication as a semantic structure in particular.

Informational implication approaches to various dictionary (informational) concepts of the word 'implication'. It certainly absorbs the concept of mathematical implication.

6.3 Rules (Axioms) of Informational Modus ponens

Modus ponens (MP for short) belongs to the most popular rules of inference as a mechanism for mathematical deduction of formulas from an object theory axioms and already deduced particular formulas, called theorems. Simultaneously, modus ponens is the main instrument in the proof procedures which are nothing else than just deduction processes as described. It considers the conveyance (an old law) and is the way in which anything deduced is obtained. MP is a mode of deductive operation (e.g. modus operandi or modus agendi).

The Latin verb *pono* (posui, positum) means to set down, before; to lay out, put out at interest in the sense to lay down as true, assert, assume, etc. MP is a mood that affirms (in German, bejahender Modus). It is a rule by which from *if* p *then* q together (and) with p, the operand q may be inferred. The full meaning in Latin is modus ponendo ponens or law of detachment (in German, Abtrennungsregel), written in the form $(p, p \to q) \longrightarrow q$. The meaning is: if, simultaneously (in German, sowohl), p and also 'if p then q' is valid (true), then also q is valid (true). Because MP is a rule, the rule arrow $\longrightarrow$ has to be used as a communication sign for an action (operation) instruction.

6.3.1 Interpretation of Logical Modus ponens

In traditional logic, modus ponens (the rule of detachment) has the form

$$\frac{p, p \to q}{q}$$

where the inferring line is the operator of detachment. As already shown, this rule (when neglecting its communication role) represents an identically true formula in the form $(p \wedge (p \to q)) \longrightarrow q$.

The uttermost informational interpretation of the above formula regarding the truth as the only relevant logical value could be the following:

$$\left(\frac{(p \models_{\text{true}}; (p \to q) \models_{\text{true}}) \models_{\text{true}}}{q \models_{\text{true}}} \right) \models_{\text{true}}$$

In the last case, by the entering into formula of MP, the truth of q has to be verified. If q is a true theorem, then in the premise, p and $p \to q$ are assumed to be valid, that is, true. Such an understanding of MP seems to be commonsensical. However, MP informs true without regard to the truth of the constituting components (p and $p \to q$).

6.3.2 Informational Modus ponens and Its Possible Interpretations

Together with the fundamental axiom of the object informational theory, that is, $\alpha \Longrightarrow (\alpha \models)$, we need a fundamental inference axiom, by which from the initial informational entity α the result $\alpha \models$ can be derived (deduced).

Inference Axiom 1 [INFORMATIONAL MODUS PONENS] *We adopt the following basic inference axiom for informational derivation:*

$$\frac{\alpha; (\alpha \Longrightarrow \beta)}{\beta}$$

By this rule, marked by $\mathbb{R}_{\text{mp}}(\alpha, \beta)$*, where 'mp' in the subscript stands for 'modus ponens', operand (formula, formula system)* β *will be derived from operand* α *(formula, formula system), that is,*

$$\alpha \twoheadrightarrow_{\mathrm{mp}} \beta \quad \text{or, simply,} \quad \alpha \twoheadrightarrow \beta$$

Formula $\alpha \twoheadrightarrow \beta$ is called derivation (by modus ponens) from α to β. □

By means of the last inference axiom from the first object axiom (e.g., Axiom 1) a theorem can be proved which follows.

Theorem 1 [EXTERNALISTIC INFORMING OF AN INFORMATIONAL ENTITY] *If α is an informational formula, then $\alpha \models$ is an informationally regular (α-equivalent, α-replaceable) formula. It means that in decompositions of α (serial, parallel, circular, metaphysical or whichever deconstruction), formula $\alpha \models$ performs as another phenomenon of formula α. There is, certainly, $\alpha \twoheadrightarrow (\alpha \models)$.* □

Proof 1 [FORMULA $\alpha \models$ AS A REGULAR OCCURRENCE OF α] *The initial 'axiom' of an informational operand α is the operand itself. We must prove, that formula $\alpha \models$ is derivable from α. Axiom 1 offers the informational validity of formula $\alpha \Longrightarrow (\alpha \models)$. In this way, we dispose with elements of the premise necessary for modus ponens. Finally,*

$$\frac{\alpha; (\alpha \Longrightarrow (\alpha \models))}{\alpha \models}$$

In fact, this is a trivial (aprioristic) proof of the informational existence of $\alpha \models$ if the existence of operand α was axiomatized. Thus, the existence of derivation $\alpha \twoheadrightarrow (\alpha \models)$ is proved. □

Theorem 2 [INTERNALISTIC INFORMING OF AN INFORMATIONAL ENTITY] *If α is an informational formula, then $\models \alpha$ is an informationally regular (α-equivalent, α-replaceable) formula. It means that in decompositions of α (serial, parallel, circular, metaphysical or whichever deconstruction), formula $\models \alpha$ performs as another phenomenon of formula α. There is, certainly, $\alpha \twoheadrightarrow (\models \alpha)$.* □

Proof 2
[FORMULA $\models \alpha$ AS A REGULAR OCCURRENCE OF α] *We must prove, that formula $\models \alpha$ is derivable from α. Axiom 2 offers the informational validity of formula $\alpha \Longrightarrow (\models \alpha)$. Let us show cases of proving the derivability of $\models \alpha$.*

The first possible case (Axiom 2) is

$$\frac{\alpha; (\alpha \Longrightarrow (\models \alpha))}{\models \alpha}$$

The second possible case considers the axiomatic fact $(\alpha \models) \Longrightarrow (\models \alpha)$ as a necessity which says that if something informs, something must be informed. Thus,

$$\frac{(\alpha \models); ((\alpha \models) \Longrightarrow (\models \alpha))}{\models \alpha}$$

At the end of the proof, let us show informationally three axiomatic implications which follow according to Example 1, the third rule:

$$(\alpha \Longrightarrow (\alpha \models)) \Longrightarrow \\ (((\alpha \models) \Longrightarrow (\models \alpha)) \Longrightarrow (\alpha \Longrightarrow (\models \alpha)));$$

$$(\alpha \Longrightarrow (\models \alpha)) \Longrightarrow \\ (((\models \alpha) \Longrightarrow (\alpha \models)) \Longrightarrow (\alpha \Longrightarrow (\alpha \models)));$$

$$((\alpha \models) \Longrightarrow (\models \alpha)) \Longrightarrow \\ (((\models \alpha) \Longrightarrow \alpha) \Longrightarrow ((\alpha \models) \Longrightarrow \alpha))$$

The implicative circularity of entities α, $\alpha \models$ and $\models \alpha$ is complete.

Thus, the existence of derivation $\alpha \twoheadrightarrow (\models \alpha)$ is proved. □

Theorem 3 [PHENOMENALISTIC INFORMING OF AN INFORMATIONAL ENTITY] *If α is an informational formula, then formula system $\alpha \models; \models \alpha$ is an informationally regular (α-equivalent, α-replaceable) formula. It means that in decompositions of α (serial, parallel, circular, metaphysical or whichever deconstruction), formula system $\alpha \models; \models \alpha$ performs as another phenomenon of formula α. There is, certainly, $\alpha \twoheadrightarrow (\alpha \models; \models \alpha)$.* □

Proof 3 [FORMULA SYSTEM $\alpha \models; \models \alpha$ AS A REGULAR OCCURRENCE OF α] *A consequence of the previous axioms and theorems is formula*

$$\alpha \Longrightarrow (\alpha \models; \models \alpha)$$

By informational modus ponens, there is,

$$\frac{\alpha; (\alpha \Longrightarrow (\alpha \models; \models \alpha))}{\alpha \models; \models \alpha}$$

This proves $\alpha \twoheadrightarrow (\alpha \models; \models \alpha)$. □

Theorem 4 [METAPHYSICALISTIC INFORMING OF AN INFORMATIONAL ENTITY] *If α is an informational formula, then formula $\alpha \models \alpha$ is an informationally regular (α-equivalent, α-replaceable) formula. It means that in decompositions of α (serial, parallel, circular, metaphysical or whichever deconstruction), formula $\alpha \models \alpha$ performs as another phenomenon of formula α. There is, certainly, $\alpha \twoheadrightarrow (\alpha \models \alpha)$.* □

Proof 4 [FORMULA $\alpha \models \alpha$ AS A REGULAR OCCURRENCE OF α] *A consequence of the previous axioms and theorems is formula*

$$(\alpha \models; \models \alpha) \Longrightarrow (\alpha \models \alpha)$$

By informational modus ponens, there is,

$$\frac{(\alpha \models; \models \alpha); ((\alpha \models; \models \alpha) \Longrightarrow (\alpha \models \alpha))}{\alpha \models \alpha}$$

(See Definition 15 for $\alpha \models \alpha$.) This proves $\alpha \twoheadrightarrow (\alpha \models \alpha)$. □

Consequence 1 [REPLACEMENT POSSIBILITIES FOR AN INFORMATIONAL OR INFORMING ENTITY] *Let us have*

$$\mathfrak{a}, \mathfrak{b} \in \{\alpha,\ \alpha \models,\ \models \alpha,\ \alpha \models \alpha,\ (\alpha \models; \models \alpha\}$$

Then, for $\mathfrak{a} \neq \mathfrak{b}$, there is $\mathfrak{a} \twoheadrightarrow \mathfrak{b}$ and, so, $\mathfrak{a} \leftarrow \mathfrak{b}$. □

Consequence Proof 1 [PROVING $\mathfrak{a} \leftarrow \mathfrak{b}$ FOR ENTITIES AND THEIR INFORMING] *The last consequence means the possibilities of replacements in decomposition (deconstruction) procedures, that is,*

$$\alpha \leftarrow \begin{pmatrix} \alpha \models, \\ \models \alpha, \\ \alpha \models \alpha, \\ \begin{pmatrix} \alpha \models; \\ \models \alpha \end{pmatrix} \end{pmatrix}; \quad (\alpha \models) \leftarrow \begin{pmatrix} \alpha, \\ \models \alpha, \\ \alpha \models \alpha, \\ \begin{pmatrix} \alpha \models; \\ \models \alpha \end{pmatrix} \end{pmatrix};$$

$$(\models \alpha) \leftarrow \begin{pmatrix} \alpha, \\ \alpha \models, \\ \alpha \models \alpha, \\ \begin{pmatrix} \alpha \models; \\ \models \alpha \end{pmatrix} \end{pmatrix}; \quad (\alpha \models \alpha) \leftarrow \begin{pmatrix} \alpha, \\ \alpha \models, \\ \models \alpha, \\ \begin{pmatrix} \alpha \models; \\ \models \alpha \end{pmatrix} \end{pmatrix};$$

$$\begin{pmatrix} \alpha \models; \\ \models \alpha \end{pmatrix} \leftarrow \begin{pmatrix} \alpha, \\ \alpha \models, \\ \models \alpha, \\ \alpha \models \alpha \end{pmatrix}$$

The consequence can be proved by considering the previous axioms and theorems. □

According to Subsubsection 6.3.1 where the logical modus ponens was informationally interpreted, we can interpret the informational modus ponens in several informational manners. The first possible and informationally consequent interpretation of MP is externalistic and yields

$$\frac{(\alpha \models; (\alpha \Longrightarrow \beta) \models) \models}{\beta \models} \models$$

where α and β can be arbitrarily complex formulas. As we see, its components (three of them in the premise, one in the conclusion) and the entire rule of MP inform in an externalistic manner. The exact meaning of this interpretation could be the following: if α informs its informational existence and, in parallel, the implication α implies β informs its informational existence, then β informs its informational existence. Under these conditions, the respective formula of MP informs its informational existence.

Other interpretations of informational MP could be internalistic, metaphysicalistic and phenomenalistic. The consequent internalistic interpretation of MP is

$$\models \frac{\models (\models \alpha; \models (\alpha \Longrightarrow \beta))}{\models \beta}$$

The consequent metaphysicalistic interpretation becomes pretty cumbersome, that is,

$$\frac{\begin{array}{c}(\alpha \models \alpha; (\alpha \Longrightarrow \beta) \models (\alpha \Longrightarrow \beta)) \models \\ (\alpha \models \alpha; (\alpha \Longrightarrow \beta) \models (\alpha \Longrightarrow \beta))\end{array}}{\beta \models \beta} \models$$

$$\frac{\begin{array}{c}(\alpha \models \alpha; (\alpha \Longrightarrow \beta) \models (\alpha \Longrightarrow \beta)) \models \\ (\alpha \models \alpha; (\alpha \Longrightarrow \beta) \models (\alpha \Longrightarrow \beta))\end{array}}{\beta \models \beta}$$

where metaphysicalism must be considered on all five components of the inference rule (metaphysicalism of two components of the premise; the premise as a whole; the conclusion; and the rule as a whole).

The consequent phenomenalistic interpretation is cumbersome too, which becomes evident from the two equivalent inference rules, where the first one informs and the second one is informed:

$$\left(\begin{array}{l}(\alpha \models; \models \alpha);\\ \left(\begin{array}{l}((\alpha \Longrightarrow \beta) \models; \models (\alpha \Longrightarrow \beta)) \models;\\ \models ((\alpha \Longrightarrow \beta) \models; \models (\alpha \Longrightarrow \beta))\end{array}\right)\end{array}\right) \models;$$

$$\models \frac{\left(\begin{array}{l}(\alpha \models; \models \alpha)\\ \left(\begin{array}{l}((\alpha \Longrightarrow \beta) \models; \models (\alpha \Longrightarrow \beta)) \models;\\ \models ((\alpha \Longrightarrow \beta) \models; \models (\alpha \Longrightarrow \beta))\end{array}\right)\end{array}\right)}{\beta \models; \models \beta} \models;$$

$$\left(\begin{array}{l}(\alpha \models; \models \alpha);\\ \left(\begin{array}{l}((\alpha \Longrightarrow \beta) \models; \models (\alpha \Longrightarrow \beta)) \models;\\ \models ((\alpha \Longrightarrow \beta) \models; \models (\alpha \Longrightarrow \beta))\end{array}\right)\end{array}\right) \models;$$

$$\models \frac{\left(\begin{array}{l}(\alpha \models; \models \alpha);\\ \left(\begin{array}{l}((\alpha \Longrightarrow \beta) \models; \models (\alpha \Longrightarrow \beta)) \models;\\ \models ((\alpha \Longrightarrow \beta) \models; \models (\alpha \Longrightarrow \beta))\end{array}\right)\end{array}\right)}{\beta \models; \models \beta}$$

The reader can imagine how mixed externalistic, internalistic, metaphysicalistic and phenomenalistic interpretations are possible. In this way an informational explosion of MP possibilities exists.

6.4 A Generalization of Informational Inference Rule Interpretation

The case of informational MP interpretation calls for a generalization principle in the following sense.

Inference Axiom 2 [INFERENCE RULE PHENOMENALISM] *An inference rule*

$$\mathbb{R}(\mathsf{A},\mathsf{B}) \rightleftharpoons \frac{\mathbb{P}(\mathsf{A},\mathsf{B})}{\mathbb{C}(\mathsf{B})}$$

is by itself an informational formula which underlies the principles of informational phenomenalism. Cases of the externalistic, internalistic, metaphysicalistic and phenomenalistic forms of inference rules, respectively, can be understood as permissive replacements, that is as transformations of the initial rule $\mathbb{R}(\mathsf{A},\mathsf{B})$. *Thus,*

$$\mathbb{R}(\mathsf{A},\mathsf{B}) \leftarrow \left(\begin{array}{l}\mathbb{R}(\mathsf{A},\mathsf{B}) \models,\\ \models \mathbb{R}(\mathsf{A},\mathsf{B}),\\ \mathbb{R}(\mathsf{A},\mathsf{B}) \models \mathbb{R}(\mathsf{A},\mathsf{B}),\\ (\mathbb{R}(\mathsf{A},\mathsf{B}) \models; \models \mathbb{R}(\mathsf{A},\mathsf{B}))\end{array}\right)$$

Concerning premise $\mathbb{P}(\mathsf{A},\mathsf{B})$ *and conclusion* $\mathbb{C}(\mathsf{B})$ *of a rule* $\mathbb{R}(\mathsf{A},\mathsf{B})$, *the following double-phenomenal cases are convenient, called* ex-externalism, in-internalism, meta-metaphysicalism *and* pheno-phenomenalism *of inferential rules, respectively:*

$$\frac{\mathbb{P}(\mathsf{A},\mathsf{B}) \models}{\mathbb{C}(\mathsf{B}) \models} \models;$$

$$\models \frac{\models \mathbb{P}(\mathsf{A},\mathsf{B})}{\models \mathbb{C}(\mathsf{B})};$$

$$\frac{\mathbb{P}(\mathsf{A},\mathsf{B}) \models \mathbb{P}(\mathsf{A},\mathsf{B})}{\mathbb{C}(\mathsf{B}) \models \mathbb{C}\mathsf{B})} \models \frac{\mathbb{P}(\mathsf{A},\mathsf{B}) \models \mathbb{P}(\mathsf{A},\mathsf{B})}{\mathbb{C}(\mathsf{B}) \models \mathbb{C}\mathsf{B})};$$

$$\left(\begin{array}{l}\dfrac{\mathbb{P}(\mathsf{A},\mathsf{B}) \models; \models \mathbb{P}(\mathsf{A},\mathsf{B})}{\mathbb{C}(\mathsf{B}) \models; \models \mathbb{C}(\mathsf{B})} \models;\\ \models \dfrac{\mathbb{P}(\mathsf{A},\mathsf{B}) \models; \models \mathbb{P}(\mathsf{A},\mathsf{B})}{\mathbb{C}(\mathsf{B}) \models \mathbb{C}\mathsf{B})}\end{array}\right)$$

Phenomenalism of arguments A *and* B *depends on the structure of premise* $\mathbb{P}$ *and conclusion* $\mathbb{C}$. □

6.5 Rules of Informational Modus tollens

Modus tollens (MT for short), in full modus tollendo tollens, belongs to the mood that denies and is the rule that from *if p then q* together with *not-q, not-p* may be inferred. An inference in modus tollendo tollens yields the contrary of the original contrary hypothesis. It is the principle that, if a conditional holds and also the negation of its consequent, then the negation of its antecedent holds [15]. MT is a mode of deductive operation.

The Latin verb *tollo* (sustuli, sublatum) means to lift or take up; to take away, remove, take or carry off, make away with, destroy; to annul, cancel, abolish. MT is a mood that denies (in German, verneinender Modus).

6.5.1 Interpretation of Logical Modus tollens

In logic, modus tollens has the form

$$\frac{p \rightarrow q, \overline{q}}{\overline{p}}$$

where the inferring line is the operator of detachment. This rule (when neglecting its communication role) represents an identically true formula in the form $((p \rightarrow q) \wedge \overline{q}) \rightarrow \overline{p}$.

The uttermost informational interpretation of the above formula regarding the truth as the only relevant logical value is the following:

$$\left(\frac{((p \rightarrow q) \models_{\text{true}}; \overline{q} \models_{\text{true}}) \models_{\text{true}}}{\overline{p} \models_{\text{true}}}\right) \models_{\text{true}}$$

6.5.2 Informational Modus tollens and Its Possible Interpretations

Together with the fundamental axioms of the object informational theory, we need an inference axiom, by which from the initial informational entity α the result $\alpha \not\models$ can be derived (deduced).

Inference Axiom 3 [INFORMATIONAL MODUS TOLLENS] *We adopt the following basic inference axiom for informational derivation:*

$$\frac{(\alpha \Longrightarrow \beta); \beta \not\models}{\alpha \not\models}$$

By this rule, marked by $\mathbb{R}_{\rm mt}(\alpha, \beta)$, where 'mt' in the subscript stands for 'modus tollens', operand (formula, formula system) $\alpha \not\models$ will be derived from operand α (formula, formula system), that is,

$$\alpha \twoheadrightarrow_{\rm mt} (\alpha \not\models) \quad \text{or, simply,} \quad \alpha \twoheadrightarrow (\alpha \not\models)$$

Formula $\alpha \twoheadrightarrow (\alpha \not\models)$ is called derivation from α to $\alpha \not\models$ (by informational modus tollens). □

By means of the last inference axiom from the first object axiom (e.g., Axiom 1) a theorem can be proved which follows.

Theorem 5 [EXTERNALISTIC NON-INFORMING OF AN INFORMATIONAL ENTITY] *If α is an informational formula, then $\alpha \not\models$ is an informationally regular (α-equivalent, α-replaceable) formula. It means that in decompositions of α (serial, parallel, circular, metaphysical or whichever deconstruction), formula $\alpha \not\models$ performs as another phenomenon of formula α. There is, certainly, $\alpha \twoheadrightarrow (\alpha \not\models)$.* □

Proof 5 [FORMULA $\alpha \not\models$ AS A REGULAR OCCURRENCE OF α] *The initial 'axiom' of an informational operand α is the operand itself. We must prove, that formula $\alpha \not\models$ is derivable from α. Example 3 explains the informational validity of formula $\alpha \Longrightarrow (\alpha \not\models)$. In this way, we dispose with elements of the premise necessary for both modus ponens and modus tollens. Firstly, by MP,*

$$\frac{\alpha; (\alpha \Longrightarrow (\alpha \not\models))}{\alpha \not\models}$$

and secondly, by MT,

$$\frac{(\alpha \Longrightarrow (\alpha \models)); \alpha \not\models}{\alpha \not\models}$$

In fact, these are trivial (aprioristic) proofs of the informational existence of $\alpha \not\models$ at the given (already derived) formula α. Thus, the existence of derivation $\alpha \twoheadrightarrow (\alpha \not\models)$ is proved. □

We can join the theorems concerning the internalistic, phenomenalistic and metaphysicalistic noninforming in the following manner.

Theorem 6 [INTERNALISTIC, PHENOMENALISTIC AND METAPHYSICALISTIC NON-INFORMING OF AN INFORMATIONAL ENTITY] *If α is an informational formula, then $\not\models \alpha$, $(\alpha \not\models; \not\models \alpha)$ and $\alpha \not\models \alpha$ are informationally regular (α-equivalent, α-replaceable) formulas. It means that in decompositions of α (serial, parallel, circular, metaphysical or whichever deconstruction), these formulas perform as distinguished phenomena of formula α. There is, certainly, $\alpha \twoheadrightarrow (\not\models \alpha)$, $\alpha \twoheadrightarrow (\alpha \not\models; \not\models \alpha)$ and $\alpha \twoheadrightarrow (\alpha \not\models \alpha)$.* □

Proof 6 [FORMULAS $\not\models \alpha$, $(\alpha \not\models; \not\models \alpha)$ AND $\alpha \not\models \alpha$ AS REGULAR OCCURRENCES OF α] *The initial formula (axiom, theorem) of an informational operand α is represented by the operand itself. We must prove, that formulas $\not\models \alpha$, $(\alpha \not\models; \not\models \alpha)$ and $\alpha \not\models \alpha$ are derivable from α. Adequately as in Proof 5 we can infer by MP and MT for the internalistic case,*

$$\frac{\alpha; (\alpha \Longrightarrow (\not\models \alpha))}{\not\models \alpha}; \frac{(\alpha \Longrightarrow (\models \alpha)); \not\models \alpha}{\not\models \alpha}$$

for the phenomenalistic case,

$$\frac{\alpha; (\alpha \Longrightarrow (\alpha \not\models; \not\models \alpha))}{\alpha \not\models; \not\models \alpha};$$

$$\frac{(\alpha \Longrightarrow (\alpha \models; \models \alpha)); (\alpha \not\models; \not\models \alpha)}{\alpha \not\models; \not\models \alpha}$$

and for the metaphysicalistic case,

$$\frac{\alpha; (\alpha \Longrightarrow (\alpha \not\models \alpha))}{\alpha \not\models \alpha}; \frac{(\alpha \Longrightarrow (\alpha \models \alpha)); (\alpha \not\models \alpha)}{\alpha \not\models \alpha}$$

Thus, the existence of derivations $\alpha \twoheadrightarrow (\not\models \alpha)$, $\alpha \twoheadrightarrow (\alpha \not\models; \not\models \alpha)$ and $\alpha \twoheadrightarrow (\alpha \not\models \alpha)$ is proved. □

Cases of (with, through, in, by) *informational arising* can be illuminated through simultaneous informing and particular non-informing of an informational entity. These cases may be seen as phenomena belonging to the realm of informational spontaneity. So, the following mixed externalistic, internalistic, metaphysicalistic and phenomenalistic occurrences are possible:

$$(\alpha \models; \alpha \not\models);\ (\models \alpha; \not\models \alpha);\ (\alpha \models \alpha; \alpha \not\models \alpha);$$
$$(\alpha \models; \not\models \alpha);\ (\alpha \not\models; \models \alpha);\ (\alpha \not\models; \models \alpha; \not\models \alpha)$$

etc., infinitely.

6.6 Rules of Informational Modus rectus

Modus rectus (MR for short) represents a direct inference orientation to an experienced reality (e.g. intention). It is a hidden, yet unrevealed informational impacting governing an informing entity in an informingly specific manner (e.g., ideologically, cynically, demagogically, sociologically). The informational hidenness of something β in something α concerns the so-called informational Being-in (includedness) [12], that is $\beta \subset \alpha$. By modus rectus, the yet-hidden component β in α is inferred, that is, derived in the form $\alpha \twoheadrightarrow_{\mathrm{mr}} \beta$.

In music, in a fugal composition, rectus has the meaning of the version of a theme performed in the basic or original, as opposed to the reversed or inverted, order [15].

In Latin, *rector* means controller, director, governor, steersman, tutor, etc. By MR the controlling, directing, governing, steering, tutoring informational component is detached out of some informing entity. The Latin adjective *rectus* means straight; upright, erect; right, correct, proper, appropriate, suitable, due; plain, simple, natural, etc.

Inference Axiom 4 [INFORMATIONAL MODUS RECTUS] *We can adopt several inference axioms for informational modus rectus:*

$$\frac{(\alpha; (\iota \subset \alpha)); \iota}{\iota};\quad \frac{(\alpha; (\alpha \Longrightarrow \iota)); (\iota \subset \alpha)}{\iota};$$

$$\frac{(\alpha; (\alpha \Longrightarrow \iota); (\iota \subset \alpha)); \iota}{\iota};\quad \frac{\iota \subset \alpha}{\iota}$$

etc. By these rules, marked by $\mathbb{R}^i_{\mathrm{mr}}(\alpha, \iota)$, *where 'mr' in the subscript stands for 'modus rectus', operand (formula, formula system)* $\iota \rightleftharpoons$ $(\iota \models_{\text{intentionally}}; \models_{\text{intentionally}} \iota)$ *will be derived from operand* α *(formula, formula system), that is,*

$$\alpha \twoheadrightarrow^i_{\mathrm{mr}} (\iota) \quad \text{or, simply,} \quad \alpha \twoheadrightarrow \iota$$

Formula $\alpha \twoheadrightarrow \iota$ *is called derivation from* α *to* ι. □

Formula $\iota \subset \alpha$ is recursively defined in [12]. Various theorems concerning modus rectus can be derived according to concrete situations.

6.7 Rules of Informational Modus obliquus

Modus obliquus (MO for short) represents an oblique, devious, indirect, evasive (winding) inference orientation which appears simultaneously with a direct orientation (e.g. intention). It is a hidden, also contradictory, yet unrevealed informational impacting governing the background of an informing entity in an informingly specific manner (e.g., obliquely, trickily; cunningly, slyly, guilefully, artfully; craftily; astutely; wile-likely).

The informational obliquity (divergence, perversity) of something β in something α concerns the so-called informational Being-in (includedness) [12] and Being-of (functionalism) [13], that is $\beta \subset \alpha$ and $\beta(\alpha)$ or $\beta^*\alpha$. By modus obliquus, the obliquely informing component β in α is inferred, that is, derived in the form $\alpha \twoheadrightarrow_{\mathrm{mo}} \beta$.

The figurative meaning of the adjective *oblique* is not taking the straight or direct course to the end in view; not going straight to the point; indirectly stated or expressed; resulting or arising indirectly; deviating from right informing or thought; informationally one-sided or perverse.

In Latin, *obliquo* means to turn sideways or aside, turn awry. *Obliquus* means slanting, sideways, oblique; indirect, covert; envious.

Inference Axiom 5 [INFORMATIONAL MODUS OBLIQUUS] *We can adopt several inference axioms for informational modus obliquus:*

$$\frac{(\alpha; (o \subset \alpha)); o}{o};\quad \frac{(\alpha; (\alpha \Longrightarrow o)); o(\alpha)}{o};$$

$$\frac{(\alpha; (\alpha \Longrightarrow o); (o \subset \alpha); o(\alpha)); o}{o}$$

etc. By these rules, marked by $\mathbb{R}^i_{\mathrm{mo}}(\alpha, o)$, *where 'mo' in the subscript stands for 'modus obliquus', operand (formula, formula system)* $o \rightleftharpoons$

$(o \models_{\text{obliquely}}; \models_{\text{obliquely}} o)$ *will be derived (detached) from operand* α *(formula, formula system), that is,*

$$\alpha \twoheadrightarrow^{j}_{\text{mo}} (o) \quad \text{or, simply,} \quad \alpha \twoheadrightarrow o$$

Formula $\alpha \twoheadrightarrow o$ *is called derivation from* α *to* o *(by informational modus obliquus).* □

Formulas $o \subset \alpha$ and $o(\alpha)$ are recursively defined in [12] and [13], respectively. Various theorems concerning modus obliquus can be derived according to concrete situations. As premises of modus obliquus, various affirmative, negatory, contrary, subalternate, contradictory, absurd and other informational entities can be conjoined. Such a premise structure can cause a parallel set of conclusions by which the so-called zigzag effects of the oblique discourse are coming into existence.

6.8 Informing of Informational Inference Rules

Informational inference rules of the form $\mathbb{R}(\alpha, \beta)$ inform as any other regular informational entity, that is, by the entirely possible informational phenomenalism. This principle does not coincide with the traditional metamathematical inference rules which are fixed once for all. Thus, an IIR can become not only as complex as possible but also as unique (individual) as possible. Such a principle enables the emerging of formulas, their informational development in the sense of informational spontaneity and circularity.

7 Axioms of Informational Operand Decomposition

Decomposition of informational operands (markers, formulas, formula systems) roots in particular (particularized) inference rules by which informational items (parts, subformulas, informational frames, gestalts) are informationally adequately composed, added, connected [in]to existing formal (symbolically identified) entities.

An operand decomposition applies serialization (deconstruction) of formulas and their parallelization according to some analytical criteria, enlarging the initial formula system. The philosophy of an informational operand decomposition calls for a separate exhaustive presentation since it is one of the main informational phenomena of informational arising in the sense of spontaneity and circularity.

8 Axioms of Informational Operator Decomposition

It is possible to make a distinction between the so-called operand decomposition and operator decomposition. It depends from the view of the observer which kind of decomposition will be preferred in a case of analytical investigation. In case of operator decomposition we are primarily confronted with the so-called informational frames, frame pairs or frame triplets (the left-, middle- and the right-positioned frame) which constitute a certain operator decomposition.

The advantage of operator decomposition lies in the independence of an operand position within a formula. This means that between any two, arbitrarily positioned operands in a well-structured formula, an adequate, in a decomposing way structured frame, frame pair or frame triplet can be positioned. We have shown some characteristic possibilities of operator framing in [13].

Informational operator decomposition is a new discipline being not anchored in the traditional mathematics (metamathematics) or elsewhere. The philosophy of an informational operator decomposition calls for an original and exhaustive analysis and discussion since it belongs to the main informational phenomena of informational particularization (and universalization) in the sense of spontaneity and circularity.

9 Conclusion

At the end, it is significant to stress that the key to a theoretical and machine-oriented usage lies in the axiomatization of the informational. There are still some philosophical and formal-theoretical obstacles on the way to a well-formed axiomatization, for instance, covering the axiomatic principles already known in metamathematics (see, for example, at [5, 8]).

On the other hand, the contemporary informational mind is aware of the syllogistic, trivial, intuitive, tautological but also contradictory and

absurd nature of the mathematical art and philosophy of axiomatization within metamathematics. The author recommends the reading of Lakatos' papers (for instance, [6]). It becomes evident that there do not exist entirely (universally) axiomatized theories being completely free from contradiction and that problems as stated within the different fundamental mathematical programs have been set on an idealistic or Platonic ground through the history from the ancient Greek era on. However, in spite of these philosophical faultinesses and deficiencies, man has constructed computing systems as successful tools in different areas of his methodology and technology.

The time of sobering and disillusion has dawned much prior to the appearance of the consciousness of the informational. Thus, a general informational theory does not search anymore for an idealistic (non-contradictory, decisive, algorithmic) systems of information.

There are certainly substantial philosophical differences existing between metamathematics and GIT. For example, logical quantifiers $\forall$ and $\exists$ reduce in ordinary informational operators. Irrespective of their nature, verbs are treated as operators and the verb *to exist* has not a specific informational advantage as in logic, where it is treated as a quantifying entity. In the informational, the verb to exist means to inform the existence of something informational and nothing else.

The program of the informational axiomatization continues into new directions and the discussion shown in this article is merely a beginning. Finally, informational axioms have to be developed to a satisfactory step of recognition—enabling the general informational theory to become a solid fundament for the development of new informational (intelligent) methodologies, tools, calculuses, apparatuses, machines, etc.

References

[1] R. Carnap: *Die logizistische Grundlegung der Mathematik.* Erkenntnis, 2. Band (1931) 91–106.

[2] B.F. Chellas: *Modal Logic.* (An Introduction) Cambridge University Press, Cambridge, 1980.

[3] J. Derrida: *La Voix et le Phénomène.* Presses Universitaires de France, Paris, 1967.

[4] M. Heidegger: *Being and Time.* Harper & Row, Publishers, New York, 1962.

[5] D. Hilbert und P. Bernays: *Grundlagen der Mathematik.* Erster Band. Die Grundlagen der mathematischen Wissenschaften in Einzeldarstellungen, Band XL. Verlag von Julius Springer, Berlin, 1934.

[6] I. Lakatos: *Infinite Regress and Foundations of Mathematics.* In I. Lakatos: *Mathematics, Science and Epistomology.* Vol. 2, pp. 3–24, Cambridge University Press, Cambridge, 1978.

[7] J. von Neuman: *Die formalistische Grundlegung der Mathematik.* Erkenntnis, 2. Band (1931) 116–122.

[8] П.С. Новиков: *Элементы математической логики.* Государственное издательство физико-математической литературы (Физматгиз), Москва, 1959.

[9] A.P. Železnikar: *Principles of Information.* Cybernetica **31** (1988) 99–122.

[10] A.P. Železnikar: *Metaphysicalism of Informing.* Informatica **17** (1993) 65–80.

[11] A.P. Železnikar: *Logos of the Informational.* Informatica **17** (1993) 245–266.

[12] A.P. Železnikar: *Informational Being-in.* Informatica **18** (1994) 1–24.

[13] A.P. Železnikar: *Informational Being-of.* Informatica **18** (1994) 277–298.

[14] S. Žižek, Ed.: *Domination, Education, Analysis* (In Slovene). A Collection of Texts of the Lacan School of Psychoanalysis, DDU Univerzum, Ljubljana, 1983.

[15] *The Oxford English Dictionary.* Second Edition (on compact disc), Oxford University Press, Oxford, 1992.

[16] *Webster's Encyclopedic Unabridged Dictionary of the English Language.* Portland House, New York, 1989.

MINDS AND MACHINES

Minds and Machines is a journal for artificial intelligence, philosophy, and cognitive science. The editor is James H. Fetzer and the book review editor is William J. Rapaport. The editorial board members are: Jon Barwise, Andy Clarc, Robert Cummins, Fred Dretske, Jerry Fodor, Clarc Glymour, Stevan Harnad, John Haugeland, Jaakko Hintikka, David Israel, Philip Johnson-Laird, Frank Keil, Henry Kyburg, John McCarthy, Donald Nute, Zenon Pylyshyn, Barry Richards, David Rumelhart, Roger C. Schank, John Searl, Brian Cantwell Smith, Paul Smolensky, Stephen Stich, and Terry Winograd.

Minds and Machines is published in 4 issues a year and, in 1995, Volume 5 is on the way. Subscription price, per volume is NLG 424 (or USD 221.50) including postage. Subscriptions should be sent to KLUWER ACADEMIC PUBLISHERS GROUP, P.O. BOX 322, 3300 AH DORDRECHT, THE NETHERLANDS, or at P.O. BOX 358, ACCORD STATION, HINGHAM, MA 02018–0358, U.S.A., or to any subscription agent. Private subscriptions should be sent direct to the publishers. A special rate is available. For more information, write to Professor James H. Moor, Department of Philosophy, 6035 Thornton Hall, Dartmouth College, Hanover, NH 03755–392, U.S.A. or e-mail to `james.moor@dartmouth.edu`.

Aims and Scope

Minds and Machines affords an international forum for discussion and debate of important and controversial issues concerning significant developments within its areas of editorial focus. Well-reasoned contributions from diverse theoretical perspectives are welcome and every effort will be made to ensure their prompt publication. Among the features that are intended to make this journal distinctive within the field are these:

- Strong stands on controversial issues are especially encouraged;
- Important articles exceeding normal journal length may appear;
- Special issues devoted to specific topics will be a regular feature;
- Review essays discussing current problem situations will appear;
- Critical responses to previously published pieces are also invited.

This journal is intended to foster a tradition of criticism within the AI and philosophical communities on problems and issues of common concern. Its scope explicitly encompasses philosophical aspects of computer science. All submissions will be subject to review.

Information for Authors

Four copies of the manuscript should be sent directly to the editor: James H. Fetzer, Department of Philosophy, University of Minnesota, 10 University Drive, Duluth, MN 55812, U.S.A.

Use double spacing and leave wide margins. Include an abstract of 100–200 words and a list of key-words for use in indexing.

The author receives two sets of first page proofs together with the manuscript. 25 offprints free of charge will be supplied. No page charges are levied on authors or their institutions.

Consent to publish in this journal entails the author irrevocable and exclusive authorisation of the publisher to collect any sums or considerations for copying or reproduction payable by third parties.

Microfilm and microfiche editions of this journal are available from University Microfilm International, 300 North Zeeb Road, Ann Arbor, MI 48106, U.S.A.

Minds and Machines is surveyed by *Current Contents, Information Technology and the Law, INSPEC, Psychological Abstracts, PsycLIT, PsycINFO online database, Computer Abstracts, Engineering Index, Ei Page One, Compendex Plus.*

A Look into the Journal

For the reader, the first look into a journal may be essential and challenging. Such a look into No. 3 (August 1994) discovers the following contents: —CRITICAL EXCHANGE: Intentionality, Qualia, and Mind/Brain Identity (Paul Schweizer); Thought and Qualia (David Cole);

—GENERAL ARTICLES: The Secret Operations of the Mind (Saul Traiger); Representational Trajectories in Connectionist Learning (Andy Clark); Can Computers Carry Content 'Inexplicitly'? (Paul G. Skokowski);
—DISCUSSION REVIEW: James H. Fetzer, *Philosophy and Cognitive Science,* in Jay L. Garfield (ed.), *Foundation of Cognitive Science: The Essential Readings* (Robert L. Causey);
—BOOK REVIEWS: Hubert L. Dreyfus, *Being-in-the-World: A Commentary on Heidegger's* Being and Time (Beth Preston); Daniel C. Dennett, *Consciousness Explained* (Matthew Elton); Andy Clark, *Microcognition: Philosophy, Cognitive Science, and Parallel Distributed Processing* (Michael Losonsky); Leonard Angel, *How to Build a Conscious Machine* (Saul Traiger); Geoffrey Brown, *Brains and Machines* (Randall R. Dipert); David M. Rosenthal (ed.), *The Nature of Mind* (Jerome A. Shaffer).

Citations from Minds and Machines

Let us show some interesting citations from *Minds and Machines,* vol. 4 (1994), No.3, for the readers of *Informatica.*

—(259, P. Schweizer) The two most important distinguishing characteristics of the mind are often taken to be intentionality and the experience of subjective presentation or 'qualia'. Genuine cognitive states are purported to possess a unique and intrinsic property of 'aboutness' or 'directedness', and, in the tradition of Brentano, this intentional aspect is held to be of central importance in distinguishing the mental from the nonmental.

—(265, P. Schweizer) Subjective experience supplies the starting point from which the objective principles of science are gradually inferred, and the resulting system of inferred principles is not a sufficient basis from which to move in the reverse direction and *deduce* the nature of subjective experience. Only sentences are deducible within the framework of a scientific/mathematical formalism, and the formalism alone cannot yield an interpretation of these sentences.

—(293, D. Cole) The qualia are the internal representations. All of their phenomenal properties, the subjective character of the experience of thinking a thought, may be accounted for by the functional role of the linguistic representation. But it is not primarily the semantic representation that is important here. It is the qualitative representation.

—(300–301, D. Cole) Computationalists such as myself take qualia seriously. Having qualia *is* information processing. So having qualia is not epiphenomenal; it is *essential* for human mentality. It is required to account for human behavior. It seems to me that other accounts *either* treat having qualia as epiphenomenal *or* head off towards a mysterious dualism. Having qualia is a brain process, but it cannot perspicuously be understood at the neural level—one can't see why there are qualia, even given a complete neurophysiological description of their activity (Leibniz's Mill).

—(303, P. Traiger) It is a common practice among philosopher of psychology to trace the origins of functionalism, and cognitive science more generally, to texts deep within the history of philosophy. Plato, for example, is described by Hubert Dreyfus as a "knowledge engineer" for the view he develops in the *Euthyphro* of expertise as the mastery of explicit rules and for the doctrine of recollection in the *Mono* ...

—(319, A. Clark) One way of solving a learning problem is, in effect, to give up on it. Thus it could be argued that certain features are simply unlearnable, by connectionist means, on the basis of certain bodies of training data ...

—(369, M. Elton) ...if you take care of intentionality, consciousness will take care of itself. ...Arguments for reducing the problem of consciousness to the problem of intentionality would be of interest to the many philosophers who have claimed that the phenomenon of consciousness is a special challenge for functionalist theories of mind.

The undersigned believes that information concerning *Minds and Machines* is instructive for the readers and authors of *Informatica* in the sense of a journal aims, scope, possibilities, and contents which concern philosophy, AI, computer science, and information technology.

A.P. Železnikar

First Call for Papers

The Eighth Australian Joint Conference on Artificial Intelligence (AI'95) 13 – 17 November 1995

Hosted by
Department of Computer Science
University College, The University of New South Wales
Australian Defence Force Academy, Canberra, ACT 2600, Australia

About AI'95

AI'95 is the Eighth Australian Joint Conference on Artificial Intelligence. The last two conferences were held in Melbourne, Victoria (AI'93), and Armidale, New South Wales (AI'94). This annual conference is the largest Australian AI conference and also attracts many overseas participants. Over 45% of submitted papers to AI'93 and AI'94 came from overseas. The proceedings of previous conferences were published by World Scientific Publishing Co. Ltd.

The main theme of AI'95 is "*bridging the gaps*," i.e., bridging the gap between the classical symbolic approach and other subsymbolic approaches, such as artificial neural networks, evolutionary computation and artificial life, to AI, and bridging the gap between the AI theory and real world applications. The goals of the conference are to promote cross-fertilisation among different approaches to AI and provide a common forum for both researchers and practitioners in the AI field to exchange new ideas and share their experience. Tutorials and workshops on various topics of AI will be organised before the main conference. A separate call for proposals for tutorials and workshops will be distributed.

Paper Submission

Authors are invited to submit papers describing both theoretical and practical work in any areas of artificial intelligence. (Papers accepted or under review by other conferences or journals are not acceptable.) Topics of interest include, but are not limited to:

Adaptive Behaviours
Artificial Life
Artificial Intelligence Applications
Automated Reasoning
Autonomous Intelligent Systems
Bayesian and Statistical Learning Methods
Cognitive Modelling Computer Vision
Distributed Artificial Intelligence
Evolutionary Learning
Evolutionary Optimisation
Fuzzy Systems
Group Decision Support Systems
Hybrid Systems
Image Analysis and Understanding
Intelligent Decision Support Systems
Knowledge Acquisition
Knowledge-Based Systems
Knowledge Representation
Machine Learning
Natural Language Processing
Neural Networks
Pattern Recognition
Philosophy of AI Planning and Scheduling
Robotics
Speech Recognition

Five hard copies of the completed paper must be *received* by the conference programme committee chair before or on **9 June 1994**. Fax and electronic submission are not acceptable. Papers received after 9 June 1994 will be returned unopened. A **best student paper award** will be given at the conference. The first author of the paper must be a full-time student, e.g., a PhD, MSc, or Honours student. A letter from the head of the student's department, confirming the status of the student, must be submitted along with the paper in order to be considered for the best student paper award. The award includes a $200 cheque and a certificate issued by the AI'95 Programme Committee. Send all paper submissions to:

Dr X. Yao
AI'95 Programme Committee Chair
Department of Computer Science
University College,
The University of New South Wales
Australian Defence Force Academy
Canberra, ACT 2600, Australia
Email: xin@csadfa.cs.adfa.oz.au
Phone: +61 6 268 8819
Fax: +61 6 268 8581

Preparation of Manuscript

All five hard copies must be printed on 8.5×11 (inch^2) or *A*4 paper using 12 point *Times*. The left and right margin should be 25mm each. The top and bottom margin should be 35mm each. Each submitted paper must have a separate title page and a body. The title page must include a title, a $300 - 400$ word abstract, a list of keywords, the names and addresses of all authors, their email addresses, and their telephone and fax numbers. The body must also include the title and abstract, but the author information must be excluded. The length of submitted papers (excluding the title page) must be no more than 8 single-spaced, single-column pages including all figures, tables, and bibliography. Papers not conforming to the above requirements may be rejected without review.

Programme Committee

Dr. Xin Yao (Chair), UNSW/ADFA
Prof. Zeungnam Bien, KAIST, Korea
Mr. Phil Collier, University of Tasmania
A/Prof. Paul Compton, UNSW
Dr. Terry Dartnall, Griffith University
Prof. John Debenham, University of Technology, Sydney
Dr. David Dowe, Monash University
Dr. David Fogel, Natural Selection, Inc., USA
A/Prof. Norman Foo, University of Sydney
A/Prof. Matjaz Gams, Jozef Stefan Institute, Slovenia
Prof. Ray Jarvis, Monash University
A/Prof. Jong-Hwan Kim, KAIST, Korea
Prof. Guo-Jie Li, NCIC, PRC
Dr. Dickson Lukose, University of New England
Dr. Bob McKay, UNSW/ADFA
Prof. Zbigniew Michalewicz, UNC-Charlotte, USA
Mr. Chris Rowles, Telecom Research Laboratories
Dr. John Slaney, Australian National University
Prof. Rodney Topor, Griffith University
Dr. Chi Ping Tsang, University of Western Australia
Dr. Olivier de Vel, James Cook University of North Queensland
Dr. Geoff Webb, Deakin University
Dr. Wilson Wen, Telecom Research Laboratories
A/Prof. Kit Po Wong, University of Western Australia
Dr. Chengqi Zhang, University of New England

Organising Committee

Dr. Bob McKay (Chair), UNSW/ADFA
Dr. Jennie Clothier, Defence Science and Technology Organisation
Dr. Richard Davis, Commonwealth Scientific and Industrial Research Organisation (CSIRO)
Mr. Warwick Graco, Health Insurance Commission
Dr. Tu Van Le, University of Canberra
Mr. John O'Neill, Defence Science and Technology Organisation
Mr. Peter Whigham, UNSW/ADFA
Dr. Graham Williams, CSIRO
Dr. Xin Yao, UNSW/ADFA

Conference Location

AI'95 will be held at ADFA (Australian Defence Force Academy) in Canberra, the capital city of Australia. ADFA is located less than 5km from the CBD of Canberra.

Proposal Submission

Tutorials on various AI related topics will be organised on 13 and 14 November 1995 in parallel with pre-conference workshops. The length of each tutorial is 3 hours. Proposals dealing with any AI related topics are solicited. Application-oriented tutorials are particularly welcome, espe-

cially those relating to topics of interest to Canberra's large administrative sector. Topics of interest include, but are not limited to:

Three hard copies of the tutorial proposal must be *received* by the tutorial/workshop coordinator at the following address before or on **7 April 1995**.

Dr. J. R. Davis
AI'95 Tutorial/Workshop Coordinator
CSIRO Division of Water Resources
P O Box 1666
Canberra City 2601, Australia
Email: richardd@cbr.dwr.csiro.au
Phone: +61 6 246 5706
Fax: +61 6 246 5800

Further Information

Further information about AI'95 can be obtained by emailing the following address (preferred):

ai95@adfa.edu.au

or by contacting the organising committee chair Dr. Bob McKay at the following address:

Dr Bob McKay
AI'95 Organising Committee Chair
Department of Computer Science
University College, The University of New South Wales
Australian Defence Force Academy
Canberra, ACT 2600, Australia
Email: rim@csadfa.cs.adfa.oz.au
Phone: +61 6 268 8169
Fax: +61 6 268 8581

Workshop Proposal Submission

AI'95 continues the tradition of organising high standard pre-conference workshops. Some of previous workshops have resulted in either journal special issues or published proceedings. However, workshops which focus on informal talks and discussions are equally welcome. We invite potential workshop organisers to submit their proposals for pre-conference workshops to AI'95. The workshop organisers are only responsible for technical issues such as sending out their CFPs, reviewing submitted papers, inviting speakers, and preparing the programme. The conference organising committee will look after all the organisational issues such as venue booking, registration, accommodation, etc.

All workshops will be held on 13 and 14 November 1995 in parallel with tutorials. The length of a workshop should be at least half a day and at most two days. Three copies of the proposal must be received by the tutorial/workshop coordinator Dr. J.R. Davis before or on **7 April 1995**.

Important Dates

7 April 1995 Deadline for Workshop Proposals.
5 May 1995 Notification of Proposal Acceptance.
9 June 1995 Deadline for Paper Submission.
21 July 1995 Notification of Acceptance.
18 August 1995 Camera Ready Copy.
9 October 1995 Camera Ready Copy of Workshop Papers.
13–14 November 1995 Tutorials/Workshops.
15–17 November 1995 Conference Sessions.

This is the 2nd Call For Papers for a special journal issue of INFORMATICA on the topic:

MIND <> COMPUTER

[i.e. Mind NOT EQUAL Computer]

In this special issue we want to reevaluate the soundness of current AI research positions (especially the heavily disputed strong-AI paradigm) as well as pursue new directions aimed at achieving true intelligence. This is a brainstorming special issue about core ideas that will shape future AI. We are interested in critical papers representing all positions on the issues.

The first part of this special issue will be a small number of invited papers, including papers by Winograd, Dreyfus, Michie, McDermott, Agre, Tecuci etc. Here we are soliciting additional papers on the topic.

TOPICS: Papers are invited in all subareas and on all aspects of the above topic, especially on:

- the current state, positions, and advancements achieved in the last 5 years in particular subfields of AI,
- the trends, perspectives and foundations of natural and artificial intelligence,
- strong AI versus weak AI and the reality of most current "typical" publications in AI,
- new directions in AI.

TIME TABLE AND CONTACTS: Papers in 5 hard copies should be received by May 15, 1995 at one of the following addresses (please, no e-mail/FAX submissions):

North & South America:
Marcin Paprzycki
paprzycki_m@gusher.pb.utexas.edu
Department of Mathematics and
Computer Science
University of Texas of the Permian Basin
Odessa, TX 79762, USA

Asia, Australia:
Xindong Wu
xindong@insect.sd.monash.edu.au
Department of Software Development,
Monash University
Melbourne, VIC 3145, Australia

Europe, Africa:
Matjaz Gams
matjaz.gams@ijs.si
Jozef Stefan Institute, Jamova 39
61000 Slovenia, Europe

E-mail information about the special issue is available from the above 3 contact editors.

The special issue will be published in late 1995.

FORMAT AND REVIEWING PROCESS: Papers should not exceed 8,000 words (including figures and tables but excluding references. A full page figure should be counted as 500 words). Ideally 5,000 words are desirable.

Each paper will be refereed by at least two anonymous referees outside the author's country and by an appropriate subset of the program committee.

When accepted, the authors will be asked to transform their manuscripts into the Informatica LaTeX style (available from ftp.arnes.si; directory /magazines/informatica).

More information about Informatica and the Special Issue can be accessed through URL: ftp://ftp.arnes.si/magazines/informatica.

Call For Papers: KDD-95
The First International Conference on Knowledge Discovery and Data Mining

Knowledge Discovery in Databases (KDD) and Data Mining are areas of common interest to researchers in machine learning, machine discovery, statistics, intelligent databases, knowledge acquisition, data visualization, high performance computing, and expert systems. The rapid growth of data and information created a need and an opportunity for extracting knowledge from databases, and both researchers and application developers have been responding to that need. KDD applications have been developed for astronomy, biology, finance, insurance, marketing, medicine, and many other fields. Core Problems in KDD include representation issues, search complexity, the use of prior knowledge, statistical inference, and algorithms for the analysis of massive amounts of data both in size and dimensionality.

Due to strong demand for participation and the growing demand for formal proceedings, it has become necessary to change the format of the previous KDD workshops to a conference with open attendance. This conference will continue in the tradition of the 1989, 1991, 1993, and 1994 KDD workshops by bringing together researchers and application developers from different areas, and focusing on unifying themes such as the use of domain knowledge, managing uncertainty, interactive (human-oriented) presentation, and applications. The topics of interest include:

- Foundational Issues and Core problems in KDD
- Database Mining Tools and Applications
- Computationally Efficient Search for Structure in Data
- Interactive Data Exploration and Discovery
- Knowledge Representation Issues in KDD
- Data and Knowledge Visualization
- Data and Dimensionality Reduction
- Prior Domain Knowledge and Re-use of Discovered Knowledge
- Statistical and Probabilistic Aspects of KDD
- Dependency Models and Inference
- Machine Learning/Discovery Algorithms for Large Databases
- Managing Model Selection and Model Uncertainty
- Assessment of Model Predictive Performance
- Integrated Discovery Systems and Theories
- Parallel techniques for data management and search
- Security and Privacy Issues in Machine Discovery

This list of topics is not intended to be exhaustive but an indication of typical topics of interest. Prospective authors are encouraged to submit papers on any topics of relevance to Knowledge Discovery and Data Mining. We also invite working demonstrations of discovery systems. The conference program will include invited talks, a demo and poster session, and panel discussions. Active discussion format will be encouraged to maintain the workshop feel that previous participants found valuable and constructive. The conference proceedings will be published by AAAI. As in previous KDD Workshops, a selected set of KDD-95 papers will be considered for publication in journal special issues and as chapters in a book.

Please submit 5 *hardcopies* of a short paper (a maximum of 9 single-spaced pages not including cover page but including bibliography, 1 inch margins, and 12pt font) by March 3, 1995. A cover page must include author(s) full address, E-MAIL, a 200 word abstract, and up to 5 keywords. This cover page must accompany the paper. IN ADDITION, an electronic version of the cover page MUST BE SENT BY E-MAIL to kdd95@aig.jpl.nasa.gov by March 3, 1995.

Please mail the papers to :
KDD-95
AAAI
445 Burgess Drive
Menlo Park, CA 94025-3496
U.S.A.

send e-mail queries regarding submissions logistics to: kdd@aaai.org

Important Dates

Submissions Due:	March 3, 1995
Acceptance Notice:	April 10, 1995
Camera-ready paper due:	May 12, 1995

Conference Co-Chairs:

Usama M. Fayyad (Jet Propulsion Lab, California Institute of Technology)

Ramasamy Uthurusamy (General Motors Research Laboratories)

Program Committee

Rakesh Agrawal (IBM Almaden Research Center, USA)
Tej Anand (AT&T Global Information Solutions, USA)
Ron Brachman (AT&T Bell Laboratories, USA)
Leo Breiman (University of California, Berkeley, USA)
Wray Buntine (NASA AMES Research Center, USA)
Peter Cheeseman (NASA AMES Research Center, USA)
Greg Cooper (University of Pittsburgh, USA)
Brian Gaines (University of Calgary, Canada)
Clark Glymour (Carnegie-Mellon University, USA)
David Heckerman (Microsoft Corporation, USA)
Se June Hong (IBM T.J. Watson Research Center, USA)
Larry Jackel (AT&T Bell Labs, USA)
Larry Kerschberg (George Mason University, USA)
Willi Kloesgen (GMD, Germany)
David Madigan (University of Washington, USA)
Chris Matheus (GTE Laboratories, USA)
Heikki Mannila (University of Helsinki, Finland)
Gregory Piatetsky-Shapiro (GTE Laboratories, USA)
Daryl Pregibon (AT&T Bell Laboratories, USA)
Arno Siebes (CWI, Netherlands)
Evangelos Simoudis (Lockheed Research Center, USA)
Andrzej Skowron (University of Warsaw, Poland)
Padhraic Smyth (Jet Propulsion Laboratory, USA)
Alex Tuzhilin (NYU Sloan School, USA)
Xindong Wu (Monash University, Australia)
Wojciech Ziarko (University of Regina, Canada)
Jan Zytkow (Wichita State University, USA)

Publicity Chair:

Padhraic Smyth, Jet Propulsion Laboratory

Industry Liason:

Gregory Piatetsky-Shapiro, GTE Laboratories

Contact Information

Please send KDD-95 conference registration and related inquiries to:

KDD-95
American Association for Artificial Intelligence (AAAI)
445 Burgess Drive Menlo Park
CA 94025-3496, U.S.A.
Phone: (+1 415) 328-3123;
Fax: (+1 415) 321-4457
Email: kdd@aaai.org

Please send technical program related queries to Program Co-Chairs:
(Email: kdd95@aig.jpl.nasa.gov)

Usama M. Fayyad
Machine Learning Systems Group
Jet Propulsion Lab M/S 525-3660
California Institute of Technology
Pasadena, CA 91109
U.S.A.
(+1 818) 306-6197 Phone
(+1 818) 306-6912 FAX

Ramasamy Uthurusamy
Computer Science Department, AP/50
General Motors Research, Bldg 1-6
30500 Mound Road, Box 9055
Warren, MI 48090-9055
U.S.A.
(+1 810) 986-1989 Phone
(+1 810) 986-9356 Fax

Please send KDD-95 Publicity and related inquiries to:

Padhraic Smyth (KDD-95)
Email: kdd95@aig.jpl.nasa.gov
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109 U.S.A.
Phone: (+1 818) 306-6422
Fax: (+1 818) 306-6912

Inquiries about KDD-95 sponsorship and industry participation to:

Gregory Piatetsky-Shapiro
GTE Laboratories, MS-45
40 Sylvan Road
Waltham MA 02154-1120 USA
e-mail: gps@gte.com
tel: 617-466-4236
fax: 617-466-2960
URL: http://info.gte.com/ kdd/

CALL FOR PAPERS
International Conference on Software Quality
ICSQ '95
November, 6 - 9 1995
Maribor, Slovenia

Organised by:
University of Maribor
Faculty of Electrical Engineering and Computer Science,
Faculty of Business and Economics,
Slovenia Section IEEE,
Association of Economics Maribor,
Slovene Society Informatika

Objectives

The aim of ICSQ '95 is to provide a platform for technology and knowledge transfer between academia, industry and research institutions in the software quality field, by:

- the introduction and discussion new research results in software quality,
- offering the practising quality engineers an insight into the results of ongoing research,
- acquainting the research community with the problems of practical application.

Topics

Some important topics for the Conference include, but are not limited to the following:

- quality management systems (QMS),
- metrics,
- process improvement,
- risk Management,
- methodologies,
- verification & validation methods,
- quality planning,
- QMS tools,
- total quality management (TQM),
- audits systems,
- human factors in quality management,
- standards.

Instructions for Authors

Four copies (in English) of the original work, not longer than 4000 words (10 pages), should be submitted to the Scientific Conference Secretariat before May 15th, 1995. Papers should include a title, a short abstract and a list of keywords, the author's name, address and title should be on a separate page. All papers received will be refereed by the International Program Committee. The accepted papers will be published in the Conference Proceedings and will be available to the delegates at the time of registration. The language of the conference will be English.

Important Dates

May 15th, 1995	Full paper
June 30th, 1995	Notification of final acceptance
October 1st, 1995	Camera Ready Copy

Conference Location

The town Maribor was founded in the 12th century. Today it is the second largest town of Slovenia, located in its North, close to the Austrian border. Many businessmen and tourists enjoy the variety of cultural, sports and gastronomic possibilities of the town. Maribor is surrounded by vineyards and has one of the largest wine-cellars

in this part of Europe. Maribor is easily accessible by international air lines from the airport of Brnik (Slovenia) and from Graz (Austria).

Conference Organisation

Organising Chairperson:
Marjan Pivka
Faculty of Business and Economics
Razlagova 14, Maribor 62000
Slovenia
Tel.: +386 62 224 611
Fax.: +386 62 227 056
Email: pivka@uni-mb.si

Programme Chairperson:
Ivan Rozman
Faculty of Electrical Engineering and Computer Science
Smetanova 17, Maribor 62000
Slovenia
Tel.: +386 62 25 461, +386 62 221 112
Fax: +386 62 227 056
Email: i.rozman@uni-mb.si

Conference Secretariat:
Miss Cvetka Rogina
Association of Economics Maribor
Cafova ulica 7, 62000 Maribor
Slovenia
Tel.: +386 62 211 940
Fax.: +386 62 211 940

Programme Committee

Boris I. Cogan, Institute for Automation and Control, Vladivostok (Russia)
Sasa Dekleva, DePaul University (USA)
Matjaz Gams, J. Stefan Institute, Ljubljana (Slovenia)
Hannu Jaakkola, Tampere University of Technology (Finland)
Marjan Pivka, University of Maribor (Slovenia)
Heinrich C. Mayer, University of Klagenfurt (Austria)
Erich Ortner, University of Konstanz (Germany)
Ivan Rozman, University of Maribor (Slovenia)
Franc Solina, University of Ljubljana (Slovenia)
Stanislaw Wrycza, University of Gdansk (Poland)
Joze Zupancic, University of Maribor (Slovenia)

Machine Learning List

The Machine Learning List is moderated. Contributions should be relevant to the scientific study of machine learning. Mail contributions to ml@ics.uci.edu. Mail requests to be added or deleted to ml-request@ics.uci.edu. Back issues may be FTP'd from ics.uci.edu in pub/ml-list/V<X>/<N> or N.Z where X and N are the volume and number of the issue; ID: anonymous PASSWORD: <your mail address> URL- http://www.ics.uci.edu/AI/-ML/Machine-Learning.html

The Fourteenth International Conference on
Object-Oriented & Entity Relationship Modelling
(Formerly the Entity-Relationship Conference)

Application of Entity-Relationship & Object-Oriented Technology to Information Systems Modelling.

December 13-15, 1995
Bond University, Gold Coast, Queensland,
Australia

The Conference

The objective of the Object-Oriented Entity-Relationship (O-O ER) Conference is is to provide a forum for researchers and practitioners in the area of conceptual modelling to interact, present existing results and explore directions that will affect the current and future generation of information systems.

The conference has been renamed to encompass current technological thrusts and directions in the area of conceptual modelling and to provide a broader forum for researchers and practitioners to exchange ideas and report on progress.

This year's theme will be dedicated to the Application of Object-Oriented/Entity-Relationship Technologies to Information Systems Modelling.

The Entity-Relationship approach has been extensively used in many database system and information system design methodologies. Recently, Object-Oriented Technology has drawn tremendous interest not only from the research community but it has also moved into mainstream industrial software design and development.

The O-O ER conference provides an opportunity towards integrating these two technologies and opens new opportunities for modelling by promoting better understanding of applications, cleaner design practices, more updatable and maintainable systems and provides a basis for reusing and retrofitting existing systems and technology.

The topic of the conference is of tremendous interest to both academia and industry and one where technological advances in conceptual modelling can have a profound impact on how organisations will model and meet future business objectives and cope with an evolving technology.

Topics of Interest (not limited to:)

Original papers are solicited on, and should clearly emphasize, describe the role of modelling tools and methodologies based on the O-O and/or ER approaches. The topics of interest include:

Integrating the ER & O-O technologies, Comparing the power and methodologies of O-O and ER modelling, Design Methodologies for Object-Oriented Information Systems, Re-Engineering of Database Systems, Business Process Modelling, Enterprise Modelling, Active Databases, Workflows & Flexible Transaction Models, Intelligent Object-Oriented Systems, View Mechanisms, Object Dynamics, Temporal Databases, Geographic Information Systems, Secure Databases, Schema evolution, Interoperable Information Systems, Object-oriented Multi-media Databases, Advanced Query Interfaces, CASE Environments, Expert Systems and Applications.

For the purposes of O-O ER'95 modelling will be considered in a broad sense and can cover any theoretical as well as practical issue. Practitioner's papers reporting on actual experience are particularly welcome and will be reviewed in a separate category.

Information for Authors

Five copies of original and compelling unpublished papers up to 5000 words that are not under consideration for publication elsewhere during the reviewing period should be sent to the Program Committee Chair. Submissions must include contact information (contact name, postal and e-mail address, and phone number), a 100-word abstract,

and explicitly indicate the paper area.

The edited proceedings of O-O ER'95 will be published by Springer-Verlag as part of the Lecture Notes in Computer Science (LNCS) series.

Important Dates

Paper, Tutorial & Panel Submission: 21 April, 1995.
Notification of Acceptance: 26 June, 1995.
Camera Ready Papers due: 27 August, 1995.

General Conference Chair

Fred Lochovsky,
Dept. of Computer Science,
Hong-Kong Univ. of Science & Technology,
Clear Water Bay,
Kawloon,
Hong-Kong
tel. +852- 358-6996
fax. +852- 358-1477
e-mail: fred@cs.ust.hk

Program Committee Chair

Mike Papazoglou,
Queensland Univ. of Technology,
School of Information Systems,
GPO Box 2434,
Brisbane 4001,
Australia
tel. +61-7-864 1972,
fax. nr. +61-7-864 1969.
e-mail: mikep@icis.qut.edu.au

Organizing Chair

Zahir Tari
Queensland Univ. of Technology,
School of Information Systems,
GPO Box 2434,
Brisbane 4001,
Australia
tel. +61-7-864 1945,
fax. nr. +61-7-864 1969.
e-mail: zahirt@icis.qut.edu.au

Tutorial Chair

Makoto Takizawa (Tokyo Denki Univ.)

Panel Chair

Leszek Maciaszek (Macquarie Univ., Sydney)

Program Committee

Peter Apers (Twente Univ., Holland), Janis Bubenko (SISU, Sweden), Athman Bouguettaya (QUT, Australia), Tiziana Catarci (Univ. of Rome, Italy), Sang Cha (Seoul National University, Korea), Chin-Wan Chung (KAIST, Korea), David Edmond (QUT, Australia), Ramez ElMasri (Univ. of Texas, Arlington, USA), Opher Etzion (Technion, Israel), Joseph Fong (City Polytechnic of Hong-Kong), Terry Halpin (Univ. of Queensland, Australia), Jean-Luc Hainaut (Univ. of Namur, Belgium), Igor Hawryszkiewycz (Univ. of Technology, Sydney), Yahiko Kambayashi (Koyoto Univ., Japan), Dimitris Karagiannis (Univ. of Vienna, Austria), Roger King (Univ. of Colorado, USA), Qing Li (HKUST, Hong-Kong), Tok Wang Ling (NUS, Singapore), Peri Loucopoulos (UMIST, UK), Robert Meersman (Univ. of Tilburg, Holland), John Mylopoulos (Univ. of Toronto, Canada), Erich Neuhold (GMD-IPSI, Germany), Anne Ngu (UNSW, Australia), Oscar Nierstrasz (Bern Univ., Switzerland), Marian Nodine (Brown Univ., USA), Christine Parent (Univ. of Burgundy, France), Niki Pissinou (Univ. of SouthWestern Louisiana, USA), Sudha Ram (Univ. of Arizona, USA), Gunter Schlageter (Fern Univ. Hagen, Germany), Arie Segev (Berkeley Univ., USA), Graeme Shanks (Monash Univ., Australia), Amit Sheth (Univ. of Georgia, USA), Arne Solvberg (Univ. of Trondheim, Norway), Stefano Spaccapietra (EPFL, Switzerland), A. Min Tjoa (Technical Univ. of Vienna, Austria), Kazumasa Yokota (ICOT, Japan), Kyu Whang (KIST, Korea), Carson Woo (Univ. of British Columbia), John Zeleznikow (La Trobe Univ., Australia)

3rd International Conference on **Computer Aided Engineering Education**

Slovak Technical University in Bratislava
Faculty of Electrical Engineering and Information Technology

13 - 15 September 1995
Bratislava, Slovakia

Introduction

The International Conference on Computer Aided Engineering Education CAEE'95 is the 3rd in the series of biennial CAEE conferences. It follows CAEE'91 in Prague and CAEE'93 in Bucharest, and also the conferences on Computer Aided Learning and Instruction in SCience and Engineering CALISCE'91 held in Lausanne and CALISCE'94 in Paris. Since 1994, the conferences CAEE and CALISCE have been twinned and are organized alternately. The aim of the Conference is to bring together people involved in the theory, design and exploitation of Computer Aided Learning/ Instruction methods and tools (CAL, CAI, multimedia, simulations) in learning of fundamental and technical sciences. The aspiration of the Conference is to promote existing hardware and software products, to encourage mutual exchange of expertise in higher scientific and technical education, and to create an opportunity for establishing new professional contacts.

Scopes of the conference

The scopes of the Conference include the theory of educational and learning software, which opens new ways towards more efficient interactions between students and computers, mainly through the use of multimedia and hypermedia. Much attention will be paid to the development of new educational software (courseware) and to its better and more productive exploitation. Finally, experience will be exchanged and practical training methods will be discussed and/or demonstrated, particularly those based on modelling and simulation.

Organizer of the conference

The Conference is organized by the Slovak Technical University (STU) in Bratislava. The main mission of STU is to provide a broad spectrum and various forms of technical education by unifying the educational and research activities. At the same time, STU is one of the most important centres of science, fundamental and applied research in Slovakia. The Conference will be held in the premises of the Faculty of Electrical Engineering and Information Technology.

The Conference is organized in co-operation with Apple Computer, Sweden.

Scientific programme

The invited lectures will be given by internationally renowned experts in currently important topics.

The preliminary list of invited speakers includes:
A. DRESLING (Aalborg University, Denmark)
E. FORTE (EPFL Lausanne, Switzerland)
P. J. HICKS (UMIST Manchester, UK)
M. F. ISKANDER (University of Utah, USA)
J. KORVINK (ETH Zurich, Switzerland)
J. de SOUSA PIRES (Apple Computer, Stockholm, Sweden).

The scope of the Conference is very wide and it is assumed that the presented lectures will contribute to the development of education in numerous branches of science. Nevertheless, particular attention will be paid to electronics, electrical engineering and information technology. The programme of the Conference will be divided into parallel sections, in accordance with discussed topics. Sufficient space will be given to presentations of new software, practical training, and exchange of software tools for educational purposes.

International programme committee

J. BREZA (STU Bratislava, Slovakia)
I. F. de CASTRO (University of S. Sebastian, Spain)
M. CHRZANOWSKI (Polytechnic Krakow, Poland)
F. de COULON (EPFL Lausanne, Switzerland)
J. L. DESSALLES (Telecom Paris, France)
D. DONOVAL (STU Bratislava, Slovakia)
E. FORTE (EPFL Lausanne, Switzerland)
N. FRISTACKY (STU Bratislava, Slovakia)
P. J. HICKS (UMIST Manchester UK)
D. IOAN (University of Bucharest, Romania)
G. KARLSSON (KTH Stockholm, Sweden)
J. KORVINK (ETH Zurich, Switzerland)
K. KVETON (CTU Praha, Czech Republic)
L. MACARI (MEDC Paisley, UK)
S. MEDHAT (University Bournemouth, UK)
J. MICHEL (ENPCH Paris, France)
J. MURGAS (STU Bratislava, Slovakia)
J. MURIN (STU Bratislava, Slovakia)
P. NAVRAT (STU Bratislava, Slovakia)
D. PONTA (University of Genoa, Italy)
J. de SOUSA PIRES (Apple Computer Stockholm, Sweden)
J. VANNEUVILLE (KIHWV Oostende, Belgium)
G. WACHUTKA (TU Munich, Germany)
M. WALD (HAP Hamburg, Germany)

Registration fee

The Conference fee is 500,- DM or equivalence in the local currency. The fee includes admission to all scientific sections, a copy of the proceedings, luncheons and the conference banquet. The student rate is 200,- DM. To qualify for the student rate a letter of recommentation from the supervisor should be submitted along with the application form.

Official language

The language of the Conference is English, neither translation nor interpretation will be provided.

Location

Bratislava, the capital of Slovakia, lies on the Danube, near the border of three countries - Slovakia, Austria, and Hungary. The city is nicely located on the southern slopes of the Small Carpathian Mountains, in an area of excellent and famous wines. It can be reached easily by air, train or road (or even by regular ship connections from Vienna or Budapest). The visitors are advised to use the services of the Airport in Bratislava, or they can fly to the Airport in Vienna-Schwechat, which lies approximately 50 km west of Bratislava. Regular bus or taxi transport from the Airport in Schwechat is provided.The most convenient road and train links lead from Vienna (60 km), Prague (350 km) or Budapest (200 km).

Social programme

In the course of the Conference, short guided trips and sightseeing tours through the city of Bratislava and the closest vicinity will be organized for accompanying persons. In the weekend following the Conference, the organizers are ready to prepare a longer trip through Slovakia for all interested participants.

Further information can be obtained and all correspondence should be addressed to:

CAEE '95
Conference Secretariat
Slovak Technical University
Microelectronics Department
Ilkovicova 3
SK - 812 19 Bratislava
SLOVAKIA

phone: +42-7-723486
fax: +42-7-723480
e-mail: caee95@elf.stuba.sk

Announcement and Call For Papers

GLOCOSM
Global Conference on Small & Medium Industry & Business

3-5 January 1996
Bangalore, India

Organized by
SDM Institute for Management Development, India
and Indiana University Purdue University Fort Wayne, U.S.A.

Educators, executives and government officials from around the world are invited to participate in this unique conference whose theme is "Small & Medium Industry & Business in the New Global Environment: Prospects & Problems." Papers, abstracts or symposium/workshop proposals related to the theme are solicited. Submissions in accounting, commerce, economics, finance, human resources management/organizational behaviour, information systems, operations management, quantitative methods/statistics, strategy, environmental/ethical/legal issues, public sector management, social/cultural issues, technology management and other topics relevant to the global community of management professionals scholars are also welcome.

Deadline for submission of contribution: 15 April 1995.

The Address of General Chair

Dr. B.P. Lingaraj
General Chair - GLOCOSM
Department of Management & Marketing
Indiana University Purdue University
Fort Wayne, IN 46805, U.S.A
E-mail: lingaraj@cvax.ipfw.indiana.edu

For other details (instructions for the authors etc.) please contact the member of the Programme Committee:

Professor Ludvik Bogataj
University of Ljubljana
Faculty of Economics
Kardeljeva ploščad 17, P.O.Box 103
61000 Ljubljana
Slovenia
Phone: +386 (061) 168-33 -33
Fax: +386 (061) 301-110
E-Mail: ludvik.bogataj@uni-lj.si

THE MINISTRY OF SCIENCE AND TECHNOLOGY OF THE REPUBLIC OF SLOVENIA

Address: Slovenska 50, 61000 Ljubljana, Tel.: +386 61 1311 107, Fax: +386 61 1324 140.
WWW:http://www.mzt.si
Minister: Prof. Rado Bohinc, Ph.D.
State Secretary for Int. Coop.: Rado Genorio, Ph.D.
State Secretary for Sci. and Tech.: Ciril Baškovič
Secretary General: Franc Hudej, Ph.D.

The Ministry also includes:
The Standards and Metrology Institute of the Republic of Slovenia
Address: Kotnikova 6, 61000 Ljubljana, Tel.: +386 61 1312 322, Fax: +386 61 314 882., and
The Industrial Property Protection Office of the Republic of Slovenia
Address: Kotnikova 6, 61000 Ljubljana, Tel.: +386 61 1312 322, Fax: +386 61 318 983.

Scientific Research and Development Potential. The statistical data for 1993 showed that there were 180 research and development institutions in Slovenia. Altogether, they employed 10,400 people, of whom 4,900 were researchers and 3,900 expert or technical staff.

In the past ten years, the number of researchers has almost doubled: the number of Ph.D. graduates increased from 1,100 to 1,565, while the number of M.Sc.'s rose from 650 to 1,029. The "Young Researchers" (i.e. postgraduate students) program has greatly helped towards revitalizing research. The average age of researchers has been brought down to 40, with one-fifth of them being younger than 29.

The table below shows the distribution of researchers according to educational level and sectors (in 1993):

Sector	Ph.D.	M.Sc.
Business enterprises	51	196
Government	482	395
Private non-profit organizations	10	12
Higher education organizations	1022	426
Total	1,565	1,029

Financing Research and Development. Statistical estimates indicate that US$ 185 million (1,4% of GDP) was spent on research and development in Slovenia in 1993. More than half of this comes from public expenditure, mainly the state budget. In the last three years, R&D expenditure by business organizations has stagnated, a result of the current economic transition. This transition has led to the financial decline and increased insolvency of firms and companies. These cannot be replaced by the growing number of mainly small businesses. The shortfall was addressed by increased public-sector spending: its share of GDP nearly doubled from the mid-seventies to 0,86% in 1993.

Income of R&D organizations spent on R&D activities in 1993 (in million US$):

Sector	Total	Basic res.	App. res.	Exp. dev.
Business ent.	83,9	4,7	32,6	46,6
Government	58,4	16,1	21,5	20,8
Private non-p.	1,3	0,2	0,6	0,5
Higher edu.	40,9	24,2	8,7	8
Total	184,5	45,2	63,4	75,9

The policy of the Slovene Government is to increase the percentage intended for R&D in its budget. The Science and Technology Council of the Republic of Slovenia is preparing the draft of a national research program (NRP). The government will harmonize the NRP with its general development policy, and submit it first to the parliamentary Committee for Science, Technology and Development and after that to the parliament. The parliament approves the NRP each year, thus setting the basis for deciding the level of public support for R&D.

The Ministry of Science and Technology is mainly a government institution responsible for controlling expenditure of the R&D budget, in compliance with the NRP and the criteria provided by the Law on Research Activities. The Ministry finances research or co- finances development projects through public bidding, partially finances infrastructure research institutions (national institutes), while it directly finances management and top-level science.

The focal points of R&D policy in Slovenia are:

- maintaining the high level and quality of research activities,
- stimulating collaboration between research and industrial institutions,
- (co)financing and tax assistance for companies engaged in technical development and other applied research projects,
- research training and professional development of leading experts,
- close involvement in international research and development projects,
- establishing and operating facilities for the transfer of technology and experience.

JOŽEF STEFAN INSTITUTE

Jožef Stefan (1835-1893) was one of the most prominent physicists of the 19th century. Born to Slovene parents, he obtained his Ph.D. at Vienna University, where he was later Director of the Physics Institute, Vice-President of the Vienna Academy of Sciences and a member of several scientific institutions in Europe. Stefan explored many areas in hydrodynamics, optics, acoustics, electricity, magnetism and the kinetic theory of gases. Among other things, he originated the law that the total radiation from a black body is proportional to the 4th power of its absolute temperature, known as the Stefan–Boltzmann law.

The Jožef Stefan Institute (JSI) is the leading independent scientific research in Slovenia, covering a broad spectrum of fundamental and applied research in the fields of physics, chemistry and biochemistry, electronics and information science, nuclear science technology, energy research and environmental science.

The Jožef Stefan Institute (JSI) is a research organisation for pure and applied research in the natural sciences and technology. Both are closely interconnected in research departments composed of different task teams. Emphasis in basic research is given to the development and education of young scientists, while applied research and development serve for the transfer of advanced knowledge, contributing to the development of the national economy and society in general.

At present the Institute, with a total of about 700 staff, has 500 researchers, about 250 of whom are postgraduates, over 200 of whom have doctorates (Ph.D.), and around 150 of whom have permanent professorships or temporary teaching assignments at the Universities.

In view of its activities and status, the JSI plays the role of a national institute, complementing the role of the universities and bridging the gap between basic science and applications.

Research at the JSI includes the following major fields: physics; chemistry; electronics, informatics and computer sciences; biochemistry; ecology; reactor technology; applied mathematics. Most of the activities are more or less closely connected to information sciences, in particular computer sciences, artificial intelligence, language and speech technologies, computer-aided design, computer architectures, biocybernetics and robotics, computer automation and control, professional electronics, digital communications and networks, and applied mathematics.

The Institute is located in Ljubljana, the capital of the independent state of Slovenia (or S♡nia). The capital today is considered a crossroad between East, West and Mediterranean Europe, offering excellent productive capabilities and solid business opportunities, with strong international connections. Ljubljana is connected to important centers such as Prague, Budapest, Vienna, Zagreb, Milan, Rome, Monaco, Nice, Bern and Munich, all within a radius of 600 km.

In the last year on the site of the Jožef Stefan Institute, the Technology park "Ljubljana" has been proposed as part of the national strategy for technological development to foster synergies between research and industry, to promote joint ventures between university bodies, research institutes and innovative industry, to act as an incubator for high-tech initiatives and to accelerate the development cycle of innovative products.

At the present time, part of the Institute is being reorganized into several high-tech units supported by and connected within the Technology park at the Jožef Stefan Institute, established as the beginning of a regional Technology park "Ljubljana". The project is being developed at a particularly historical moment, characterized by the process of state reorganisation, privatisation and private initiative. The national Technology Park will take the form of a shareholding company and will host an independent venture-capital institution.

The promoters and operational entities of the project are the Republic of Slovenia, Ministry of Science and Technology and the Jožef Stefan Institute. The framework of the operation also includes the University of Ljubljana, the National Institute of Chemistry, the Institute for Electronics and Vacuum Technology and the Institute for Materials and Construction Research among others. In addition, the project is supported by the Ministry of Economic Relations and Development, the National Chamber of Economy and the City of Ljubljana.

Jožef Stefan Institute
Jamova 39, 61000 Ljubljana, Slovenia
Tel.:+386 61 1773 900, Fax.:+386 61 219 385
Tlx.:31 296 JOSTIN SI
WWW: http://www.ijs.si
E-mail: matjaz.gams@ijs.si
Contact person for the Park: Iztok Lesjak, M.Sc
Public relations: Natalija Polenec

REVIEW REPORT

Basic Instructions

Informatica publishes scientific papers accepted by at least two referees outside the author's country. Each author should submit three copies of the manuscript with good copies of the figures and photographs to one of the editors from the Editorial Board or to the Contact Person. Editing and refereeing are distributed. Each editor can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. The names of the referees should not be revealed to the authors under any circumstances. The names of referees will appear in the Refereeing Board. Each paper bears the name of the editor who appointed the referees.

It is highly recommended that each referee writes **as many remarks as possible directly on the manuscript**, ranging from typing errors to global philosophical disagreements. The chosen editor will send the author copies with remarks, and if accepted also to the Contact Person with the accompanying completed Review Reports. The Executive Board will inform the author that the paper is accepted, meaning that it will be published in less than one year after receiving original figures on separate sheets and the text on an IBM PC DOS floppy disk or through e-mail – both in ASCII and the Informatica LaTeX format. Style and examples of papers can be obtained by e-mail from the Contact Person or from FTP or WWW (see the last page of Informatica).

Date Sent:

Date to be Returned:

Name and Country of Referee:

Signature of Referee:

Name of Editor:

Title:

Authors:

Additional Remarks:

All boxes should be filled with numbers 1-10 with 10 as the highest rated.

The final mark (recommendation) consists of two orthogonal assessments: scientific quality and readability. The readability mark is based on the estimated perception of average reader with faculty education in computer science and informatics. It consists of four subfields, representing if the article is interesting for large audience (interesting), if its scope and approach is enough general (generality), and presentation and language. Therefore, very specific articles with high scientific quality should have approximately similar recommendation as general articles about scientific and educational viewpoints related to computer science and informatics.

☐ SCIENTIFIC QUALITY

☐ Originality

☐ Significance

☐ Relevance

☐ Soundness

☐ Presentation

☐ READABILITY

☐ Interesting

☐ Generality

☐ Presentation

☐ Language

☐ FINAL RECOMMENDATION

☐ Highly recommended

☐ Accept without changes

☐ Accept with minor changes

☐ Accept with major changes

☐ Author should prepare a major revision

☐ Reject

INFORMATICA

AN INTERNATIONAL JOURNAL OF COMPUTING AND INFORMATICS

INVITATION, COOPERATION

Submissions and Refereeing

Please submit three copies of the manuscript with good copies of the figures and photographs to one of the editors from the Editorial Board or to the Contact Person. At least two referees outside the author's country will examine it, and they are invited to make as many remarks as possible directly on the manuscript, from typing errors to global philosophical disagreements. The chosen editor will send the author copies with remarks. If the paper is accepted, the editor will also send copies to the Contact Person. The Executive Board will inform the author that the paper has been accepted, in which case it will be published within one year of receipt of the original figures on separate sheets and the text on an IBM PC DOS floppy disk or by e-mail – both in ASCII and the Informatica LaTeX format. Style and examples of papers can be obtained by e-mail from the Contact Person or from FTP or WWW (see the last page of Informatica).

Opinions, news, calls for conferences, calls for papers, etc. should be sent directly to the Contact Person.

QUESTIONNAIRE

☐ Send Informatica free of charge

☐ Yes, we subscribe

Please, complete the order form and send it to Dr. Rudi Murn, Informatica, Institut Jožef Stefan, Jamova 39, 61111 Ljubljana, Slovenia.

Since 1977, Informatica has been a major Slovenian scientific journal of computing and informatics, including telecommunications, automation and other related areas. In its 16th year (more than two years ago) it became truly international, although it still remains connected to Central Europe. The basic aim of Informatica is to impose intellectual values (science, engineering) in a distributed organisation.

Informatica is a journal primarily covering the European computer science and informatics community - scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the Refereeing Board.

Informatica is free of charge for major scientific, educational and governmental institutions. Others should subscribe (see the last page of Informatica).

ORDER FORM – INFORMATICA

Name: .....

Title and Profession (optional): .....

.....

Home Address and Telephone (optional): .....

.....

Office Address and Telephone (optional): .....

.....

E-mail Address (optional): .....

Signature and Date: .....

EDITORIAL BOARDS, PUBLISHING COUNCIL

Informatica is a journal primarily covering the European computer science and informatics community; scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor from the Editorial Board can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the Refereeing Board. Each paper bears the name of the editor who appointed the referees. Each editor can propose new members for the Editorial Board or Board of Referees. Editors and referees inactive for a longer period can be automatically replaced. Changes in the Editorial Board and Board of Referees are confirmed by the Executive Editors.

The coordination necessary is made through the Executive Editors who examine the reviews, sort the accepted articles and maintain appropriate international distribution. The Executive Board is appointed by the Society Informatika. Informatica is partially supported by the Slovenian Ministry of Science and Technology.

Each author is guaranteed to receive the reviews of his article. When accepted, publication in Informatica is guaranteed in less than one year after the Executive Editors receive the corrected version of the article.

Volume 19 Number 1 February 1995 ISSN 0350-5596

Informatica

An International Journal of Computing and Informatics

Contents:

Profile: Haneef Fatmi		1
Introduction		3
Supporting the Evolution of Distributed, Non-stop, Mission and Safety Critical Systems	C.W. McKay C. Atkinson	7
Loose Specification of Real Time Systems	Jan van Katwijk Hans Toetenel	25
An Object-Oriented Approach for Modeling and Analysis of Safety-Critical Real-time Systems	J. Lin D.C. Kung P. Hsia	43
Supporting High Integrity and Behavioural Predictability of Hard Real-Time Systems	M. Colnarič D. Verber W.A. Halang	59
A Novel Approach to Off-line Scheduling in Real-Time Systems	G. Yu L.R. Welch	71
On-line Algorithms for Allocating Periodic-time-critical Tasks on Multiprocessor Systems	S. Davari S.K. Dhall	83
A Semi-Distributed Load Balancing Model for Parallel Real-time Systems	K. Erciyeş Ö. Özkasap N. Aktaş	97
Optimal Algorithm for Real-Time Fault Tolerant Distributed Processing Using Checkpoints	Z.M. Wojcik B.E. Wojcik	111
Fully Deterministic Real-Time Protocol for a CSMA/CD Type Local Area Network	B.Tchouaffe J.Zalewski	123
Principles of a Formal Axiomatic Structure of the Informational	A.P. Železnikar	133
Reports and Announcements		159