https://doi.org/10.31449/inf.v48i4.5845 Informatica **48** (2024) 663–684 **663**

Towards an Ontological-based CIM Modeling Framework for IoT Applications

Mohamed Bettaz[1], Mourad Maouche[2]
[1]Czech Technical University in Prague, Prague, Czech Republic
[2]Freelance Researcher, Constantine, Algeria
E-mail: bettamoh@cvut.cz, maouche805@gmail.com

Keywords: CIM modeling, requirement engineering, UFO ontology, IoT applications

Received: March 5, 2024

Few works addressed contributions of ontologies to Computation Independent Modeling (CIM) of Internet of Things' (IoT) applications. This work targets CIM artefacts developed using a combination of a goal-oriented requirements (KAOS) and a service-oriented (SoAML) modeling frameworks. This paper proposes an ontological-based framework intended to help CIM modelers in their preliminary analysis of IoT applications. We adopt the ontology reuse approach, an approach often used by the ontology engineering community, where specific ontology fragments are selected, adapted and/or refined, and merged. We use OntoUML to describe our fragments. The OpenPonk tool is used to edit and verify the syntax and the semantics of these fragments' models. The results of our contribution are summarized as follows. Improving the semantics carried by the metamodels of KAOS and SoaML modeling languages, through our proposed conceptualization grounded by the Unified Foundational Ontology (UFO), a sound ontological framework; setting a link between our proposed KAOS and SoaML ontology fragments; designing a (partial) IoT domain ontology to be integrated into our proposed CIM. An illustrative example, showing how to instantiate selected ontology fragments, demonstrates the applicability of our results to IoT applications.

Povzetek: Prispevek predlaga ontološko zasnovan okvir za modeliranje CIM (Computational Independent Modeling) IoT aplikacij. Uporablja pristop ponovne uporabe ontologij za izboljšanje semantike metamodelov KAOS in SoaML.

1 Introduction

Nowadays requirements' engineering, a discipline covering the analysis and the specification of systems under development, strongly relies on the Model Driven Engineering (MDE) methodology [1]. System analysis is concerned with the understanding and description of domains under examination and their business aspects (problem domain), whilst system specification is essentially concerned with the statement and the analysis of the prescribed requirements for the system under development (solution domain). Specific models, intended to capture domains (domain modeling view), business issues (business modeling view), and system requirements (requirement modeling view), put together constitute the so-called CIM [2]. A lot of research works conducted by the software engineering community have addressed and covered several aspects related to this kind of models. Some of them have been devoted to the analysis of IoT applications, services, and systems: for instance the authors of [38] proposed a dedicated UML profile intended to put together relevant computing and business modeling concepts, and the authors of [15, 9] suggested either an extension, or a combination of existing general purpose modeling languages.

Usually, modeling languages are supported by metamodels (models of models) intended to describe their abstract syntax and (structural) semantics. Models are supposed to be conform to the metamodel associated with the modeling language used to build them. CIM models described in appropriate modeling languages are intended to provide necessary descriptive (domain, business, and requirements) knowledge [2] that may be transformed into systems' implementation through a series of well-defined steps. Very often, these languages are thought and designed in terms of specific (combinations of) paradigms (goal-oriented, agent-oriented, task-oriented, and service-oriented paradigms for instance). Some of these paradigms may also share common kinds of modeling constructs. Moreover, the ontology engineering community also has conducted several research works investigating the potential contributions of ontologies to the Requirement Engineering [12] in general, and to the CIM modeling in particular [2]. Roughly speaking, "ontologies are formal explicit representations of shared conceptualization of parts of reality". They provide powerful descriptive means making them able to formally capture the essence of real-world phenomena relevant to various domains. We may mention in the following some of their potential links to MDE and also uses in the context of CIM modeling: Firstly, the theoretical work in [2] contributed to clarify the differences as well as the

inter-play between metamodeling, and ontological-based approaches to modeling languages. It states that, while ontologies are intended to capture only descriptive aspects of relevant parts of reality, CIM models are intended to capture both descriptive (domain, business) and prescriptive (requirements) aspects of systems under development. Secondly, the availability of worked-out ontologies covering not only the IoT domain itself (sensors, actuators, measurements, etc.), but also specific application domains where IoT technology may potentially be applied (health, transportation, business, and others), allows us to (re)use them (i.e., such ontologies) in IoT applications' CIM modeling (domain and business views). Thirdly, the literature reports various works showing the usage of ontologies in the context of MDE modeling languages. The authors in [20] raised the benefits of the use of ontological models as theoretical tools for analyzing and improving conceptual modeling languages, stating that "providing real-world semantics for modeling constructs" of conceptual modeling languages constitutes one of these expected benefits. Such conceptualizations of modeling languages may help to detect and make explicit hidden concepts carried by modeling languages, to derive solid metamodels for new developed modeling languages, or to support model transformations' activities thanks to a set of well-known ontology engineering tasks (integration, mapping, and merging of ontologies). Basic knowledge on these tasks is given in Section 3. The wide and ever-growing deployment of both ontologies, covering aspects of the real life and the computing discipline on one side, and IoT applications and services on the other side, motivates our interest into this subject. To the best of our knowledge, few works have been specifically devoted to the use of ontologies to capture the semantics of CIM models associated with IoT applications under development. The reported works mainly addressed the use of ontologies for Ambient Assisted Living (AAL) applications' development [13, 11], a particular application domain where IoT technology is applied. The objective of this paper is to propose an ontological-based framework intended to help CIM modelers in their preliminary analysis of IoT applications under development. This is achieved through the construction of a global ontological model regrouping and relating IoT domain key concepts, application domain concepts, and concepts associated with key constructs offered by the used CIM modeling languages. Instances of this global ontological model can be analyzed, simulated, and validated using appropriate tools. Ontologies' merging is the employed mechanism to build the intended global ontological model [35]. More concretely we propose to:

(a) Employ KAOS (Knowledge Acquisition in Automated Specification of Software Systems) for the goal-oriented aspects [46], and SoaML (Service-oriented architecture Modeling Language) for the business view [14]. These choices are mainly motivated by the availability of many works reporting on ontological views of, respectively the goal-oriented and service-oriented paradigms.

(b) Build preliminary ontological models for KAOS and SoaML respectively, thus providing real-world semantics for their respective key modeling constructs. The ontological models associated with these modeling languages are built by exploiting the knowledge carried explicitly by their metamodels, and also the informal knowledge carried in their textual documentations [24].

(c) Employ ontological models to represent and describe CIM domain views associated with IoT applications. This is mainly motivated by the availability of several ontological models representing the IoT domain (sensors, actuators, etc.), and a lot of ontological models representing various application domains.

(d) Construct an ontological model intended to represent a global conceptual view regrouping and relating in a "pragmatic" way the three CIM views of IoT applications. This is achieved by merging the ontological models mentioned in (b) and (c). The merging operation is done in such a way that the ontological models to be merged, seen as constituent fragments, are kept unchanged in the resulting merged ontological model.

It is worth mentioning that our ontological models rely on the UFO infrastructure, and are expressed in OntoUML, a UFO ontology-based modeling language. Basic knowledge on OntoUML and its supporting UFO foundation is included in Section 3, while the motivation behind the decision of such choices is exposed in Section 4.

The rest of the paper is organized as follows. Related works are presented in Section 2. A preliminary knowledge on ontologies, OntoUML, UFO infrastructure, Goal Oriented Requirement Ontology, and a set of relevant modeling languages is summarized in Section 3. The method used to carry out this work as well as the motivations behind the choice decision is explained in Section 4. Section 5 exposes and details our proposed UFO-based ontology fragments for the KAOS framework. Section 6 presents our proposed reinforcement of the business dimension for the KAOS framework. A preliminary SoaML ontology is described in Section 7. Section 8 presents and explains the approach used to merge our KAOS and SoaML ontologies. In Section 9, we show how to transform OntoUML models into appropriate gentle UFO (gUFO) classes aimed at their computerization. Section 10 presents an illustrative example demonstrating the applicability of our results to IoT applications. Concluding remarks, limitations, and future works are outlined in Section 11.

2 Related works

The use of ontologies in the software engineering discipline has been investigated for over a decade addressing its different fields with various motivations and intentions. Ontologies for CIM modeling of systems in general, and of IoT-based systems specifically have also been subject of

research interests and works. Table 1 encloses the results of related works. It summarizes key concepts used in these works and shows that our proposal is not addressed in them.

2.1 Ontologies for CIM modeling

The main contribution of [13] consists in a "smooth" combination of various existing ontology fragments covering the goal concept using a variant of an i-star (also denoted i*) [16] based ontology, and other AAL related domains such as IoT (sensors, measures, etc.) and domotic. Their objective is to propose a goal-oriented ontology-based methodology for AAL systems motivated by the need of building AAL knowledge sources allowing the access to stored or inferred relevant knowledge. Our contribution, whilst similar to the work in [13], differs in three aspects; (i) it includes a new modeling dimension, namely the business view, hence covering all aspects of CIM modeling; (ii) we adopt KAOS rather than i-star. Both (goal-oriented requirement engineering frameworks) present benefits and drawbacks [45]. The KAOS ability to distinguish between agent roles is one of the motivations behind our choice of KAOS; (iii) our proposed ontology is built on UFO, a solid foundational top-level ontological framework. The work in [11] consists in building an ontology, called Goal Service Ontology (GSO), as a support for designing a framework for dynamic service discovery, composition, and use. Although GSO is not dedicated to IoT applications, it has been applied to the modeling of AAL applications. GSO, as our ontology, is built on refining UFO-C ontology fragments for the goal and tasks (an operation-like concept). Contrarily to our contribution whose objective is to build an ontology to formalize CIM models using a combination of KAOS and SoaML, GSO is intended to back the metamodel of a new designed DSL language. Lastly, our conceptualization of the service paradigm relies mainly on refinements of pieces of UFO-A [17] and UFO-B [21] ontology fragments rather than on the UFO-S [30] ontology fragment as in GSO. The work in [43] consists in adopting an ontology approach for the modeling of SysML/KAOS domains. Roughly speaking, a metamodel for domain modeling is built on some OWL (the web ontology language) model elements, together with new added model elements. In our work, domains are modeled as pieces of refined UFO-A categories smoothly combined with our proposed KAOS and SoaML ontologies. Go4SoA [10] is a proposal combining the goal with the Service-oriented Architecture (SoA) paradigm. The approach consists in extending and enriching the SoaML metamodel with the goal concept. Regarding our work, the combination is done in an ontological way, by bridging KAOS and SoaML ontologies. [3] treats on the "semantic interoperability across IoT domains in cross-domain applications".

2.2 Ontologies for KAOS and SoaML

The literature reported ontology proposal(s) for KAOS [27], and for SoaML [30, 37, 29]. In [27], the authors used on the one hand a standard KAOS metamodel as a reference to build their KAOS ontology, and on the other hand the Unified Enterprise Modeling Language (UEML) as an ontological modeling language. UEML is usually intended for enterprises and information systems' modeling. Our KAOS ontology, not only refers to a standard KAOS metamodel, but also it is based on proposed refinements of modeling elements drawn from the UFO-C (Agent and Goal) ontology fragments [19, 23], and also on proposed refinements of the Goal-oriented Requirement Ontology (GORO) [31, 23]. Regarding SoaML, [30] and [37, 29] are among the recent works aiming at introducing ontology concepts for SoA and SoaML. The authors of [30] have analyzed and evaluated the use of UFO-S, an UFO sub-ontology for services, in various approaches including the SoaML one; those of [37] have proposed a set of ontologies covering the general aspects of the so-called Service Engineering (service-oriented architecture, software service ontology, etc.) including ontologies for SoaML. Their work relies on the Open Group Service Ontology, and also on the ISO/IEC SoA Reference Architecture. [29] provides a comparison between the UFO-S service ontology and other similar service ontologies including the Open Group Service Ontology.

Our SoaML ontology is built using refinements and adaptations of some modeling elements of UFO-A and UFO-B ontology fragments. The suggested adaptations mainly rely on relevant knowledge drawn from the OMG SoaML specification document [33] as well as from [37, 29].

3 Preliminary knowledge

In order to make our paper (as much as possible) self-contained, this section will summarize basic knowledge on ontologies, GORO, and various relevant modeling languages (KAOS, OWL and RDF/RDFS, and SoaML).

3.1 Ontologies

Ontologies have been widely used in various disciplines and real-world domains. The authors in [26] state that "in computer science, ontology is a formal representation of the knowledge by a set of concepts within a domain and the relationships between those concepts. Ontologies are used to conceptualize domains and to reason about domain properties". Ontologies are often characterized by their level of generality. General classes are specified in top-level ontologies, whilst more specific classes are described in lower-level ontologies [18]. Top-level ontologies are intended to describe very general concepts. Some of these ontologies are qualified as reference ontologies in the sense that they are dedicated to a specific domain and recognized as a reference by its relevant communities. UFO

Table 1: Summary table

Ref.	Proposal	CIM Requirement	CIM Business	CIM Domain	Ontology / Metamodel	Application	Remark
[38]	IotReq	KAOS-like goal	SoaML services	SoaML and UML	UML profile	IoT app.	no use of ontology
[11]	GSO	goal, agent, task	services (client, provider)	UML class models	UFO grounded	includes IoT app.	ontology for new DSL
[10]	Go4SoA	goals	BPMN	UML class models	extended metamodel with goal	bus. SoA oriented app.	no use of ontology
[44]	SysML/ KAOS domain model.	KAOS/ SysML method	…	ontol. based domain models	domain knowl. as an ontology	CPS; IoT-based systems	not fully integr. in an ontology
[13]	An ontol. based methodol. for AAL	Requirem. as GRL goals	…	mix. frag. from various dom.	GoAAL an ontology for AAL app.	AAL systems	specific
[27] [32] [45]	KAOS model. methodol.	goals for requirem. model.	…	domain model. with UML	metamodels & ontology	general purpose; mainly CPS app.	missing bus. aspect model.
[29] [30] [33] [37]	SoaML model. methodol.	from bus. proc. modeling.	…	UML class diag. & profile	UML profile	bus. oriented app.	missing goal approach for requirements

is a formalized top-level reference ontology [17]. It has been successfully used to conceptualize a lot of specific domains, and particularly the domain of modeling languages [20, 22]. UFO includes four ontology modules: UFO-A, intended for the conceptualization of Endurants (commonly called Entities) [17], UFO-B for the Perdurants (commonly known as Events) [21], UFO-C for the Social and Intentional Objects [19], and UFO-S for Services [30]. OntoUML is an UFO-based UML profile, formally conceptualized by the UFO reference ontology [40]. It provides a set of class stereotypes such as Category, Kind, Subkind, Relator, Events and their related "association relationships" stereotypes (cf. [41] for more details on OntoUML specifications). OntoUML, mainly used in the ontology engineering field, has retained the attention of the software engineering community [36]. In particular, [6, 5] refer to a newly OntoUML-inspired language that is intended to augment the object-oriented formal language Object-Z with OntoUML-like features. Ontology integration, merging, and mapping are among the interesting topics discussed by the ontology engineering communities; these topics have been investigated for more than two decades. Recent research surveys [35, 39] discussed the confusion about these ontology operations and their misuse.

Furthermore, ontologies may be seen as implementations of knowledge based systems. Thus they may be populated (instantiated) and queried as databases.

3.2 RDF and RDFS

OWL is an extension of Resource Description Framework (RDF) for building ontologies. RDF is a framework for describing resources organized as data graph models. An RDF statement (rdf:<statement>) is a triple (subject, predicate, object), where subjects and objects are graph nodes and predicates are graph edges. Internationalized Resource Identifiers (IRIs) are used to identify nodes and edges of an RDF graph according to user defined namespaces. An IRI described by the symbol ":" preceding a string means that a default namespace is used. Among RDF syntax notations we may mention those that define the elements of triples (rdf:subject, rdf:predicate, rdf:object) and RDF properties (for example rdf:type employed to indicate that a given named resource is an instance of a class). RDF Schemas (RDFSs) is a vocabulary extension of RDF that allows the construction of taxonomies of classes and properties. Among RDFS syntax notations we may mention those that define a class of classes (rdfs:Class), those that declare a subject as a subclass of a class (rdfs:subClassOf), and those that declare the domain and range of a subject property (rdfs:domain, rdfs:range). SPARQL, a "SQL-like" query language, is widely used for querying RDF graphs.

3.3 GORO in a nutshell

GORO is an ontological framework intended to conceptualize a well-established requirement engineering approach

called Goal Oriented Requirement Engineering (GORE) [31, 4]. Several notions and relationships related to the requirement engineering discipline are identified and semantically defined. However, GORO is used in this work only as a reference facilitating the understanding of the basic key GORE notions. One of the objectives of GORO is to enable the interoperability between different GORE modeling languages [4]. It is worth mentioning that the KAOS modeling approach supports the GORE approach.

3.4 The KAOS modeling framework

KAOS is a well-known requirements' engineering framework, supported by a set of modeling constructs, that primarily relies on the goal paradigm to specify system requirements. It has been used in the industry by software as well as system engineering communities either as a single modeling language [25, 42] or combined with other modeling languages [34]. More concretely, a KAOS specification consists of four inter-related models: a goal model, a responsibility model, an operation model, and an object model. The goal model is intended to describe the intentions of actors (stakeholders and software). It is organized as a hierarchy of goals representing a set of strategic (high-level and middle-level goals) and operational (leaf-level goals) intentions as well as their relationships. KAOS identifies two sorts of operational goals, named requirements and expectations. The distinction between these two sorts of operational goals is illustrated in Section 5. The responsibility model describes the way the elicited operational goals are assigned to specific actors, namely environment agents (stakeholders) and software agents, for achievement purpose. The operational model concerns the operationalization of the elicited leaf goals in terms of the operations performed by the assigned agents . The object model is intended to model elicited agents, and elicited application domain entities to be monitored (through sensors) and/or controlled (through actuators) [15]. More details on key KAOS modeling constructs are given in Section 5. In this work, we propose a preliminary UFO-based ontology intended to conceptualize a subset of the KAOS key modeling constructs, that are based on the KAOS metamodel reported in [32, 44]

3.5 The SoaML modeling language

SoaML is a modeling language supporting the Object Management Group (OMG) Services' Reference Model [33]. SoaML has been used in industry and enterprises by software engineering and business communities. It mainly focuses on the service paradigm where business and software services are treated in a uniform way. SoaML provides business experts with means allowing them to describe business issues, and it provides software engineers with means allowing them to model computing issues (software). Its key modeling constructs, provided by a specific UML profile, capture relevant SoA entities such as service, service contract, service interface, service architecture, agent, participant, capabilities, operations, and others. More details on these modeling entities are given in Section 7. A UML metamodel is associated with this profile. It is worthwhile to mention SoaML4IoT, an extension of the SoaML profile, intended for IoT applications [9]. It refines some of the (native) SoaML model elements and introduces new ones. In this work, we focus on SoaML and propose a preliminary UFO-based ontology intended to conceptualize a subset of the SoaML key modeling constructs, based on the SoaML metamodel reported in [33].

4 Method

The objective of this work is to propose a framework aiming at conceptualizing IoT CIM models in an ontological way. For this purpose, we followed two steps.

1. State the starting point
   Two alternatives are to be considered: CIM aspects are expressed either directly by using available appropriate modeling languages or in a modeling language-neutral. The advantage of the first alternative is that the literature reported a number of modeling languages that have already explored, identified and captured (through metamodels) a large set of key concepts and an almost "shared" vocabulary related to the whole CIM aspects. The issue is now to select, among the available languages, the ones that not only cover the different aspects of CIM models but also have been successfully for modeling IoT requirements.

2. Build the targeted ontology (fragments)
   The ontology engineering community suggests two alternative methods to build ontologies. Either constructing ontologies from scratch starting from well-defined requirements (for instance the so-called Competency Questions), or by reusing available ontology fragments. Ontology integration and ontology merging are the two well-known methods adopted in the ontology reuse approach. Ontology integration consists in reusing selected and evaluated ontology fragments, extending them, customizing and adapting them to the targeted context. Ontology merging consists in merging fragments that capture the same "reality" or share some common concepts. In our work we used both integration and merging methods in an ad-hoc way.

4.1 Selection of appropriate CIM modeling languages

Usually, CIM modelers select and combine specific languages to cover the three CIM aspects (i.e., domain, business, and requirements aspects). In this work, ontologies, due to their high descriptiveness power, are used in conjunction with suitable languages to improve the modeling

of CIMs. We retain KAOS for the modeling of the requirements as well as for the modeling of IoT and specific application domains, and SoaML for the modeling of the business aspect. This work proposes to build, and merge ontologies associated with each of these two languages. Two main reasons motivate the choice of KAOS. Firstly, it has shown its effectiveness in the modeling of the requirements of real-life and academic projects. It has been also used, in combination with the SysML language [15] for the development of (engineered) systems in general and AAL systems in particular; these kinds of systems are characterized by a strong use of sensors and actuators. Secondly, we can reuse and, by the way, build on available relevant ontology fragments ([19], [31], [4], [27], [23]). Three reasons motivate the choice of SoaML. Firstly, due to its support for SOA, SoaML provides means to model business issues as well as software issues, thus making it not only able to support CIM business views but also to offer a smooth bridge to software design views. Secondly, a previous work [11] and a more recent one [10] emphasized the use of the service paradigm in an ontological-based approach for respectively the AAL systems, and IoT systems. Thirdly, as for KAOS, the availability of ontologies intended to conceptualize the service paradigm ([30]] as well as SoaML ([37], [28]) allows us to work-out our preliminary ontology for SoaML without starting from scratch.

4.2 Building of our ontology fragments

For this purpose we followed two steps.

1. Construction of preliminary KAOS and SoaML ontologies
   Nowadays, new ontologies are mainly built by reusing (parts of) available ontologies. The ontology integration methods requires to select, evaluate and customize available ontology fragments matching with the reality to conceptualize. On one side, for the purpose of our KAOS ontology, we selected well-evaluated fragments provided either by UFO or GORO. It is worthwhile to note that the GORO ontology allowed us to get a better understanding of the concepts inherent to the goal-oriented requirement engineering. On another side, our proposed SoaML ontology fragments are mainly inspired from a set of Service and SoA ontology proposals reported in the literature ([29], [30], [37]). These works allowed a better understanding of key concepts of SoA and SoaML.
   Finally, the adaptation and customization of the selected fragments to KAOS and SoaML is mainly governed by an appropriate mapping of their concrete key constructs in conformance with their associated published metamodels. For convenience and uniformity purposes, the proposed ontologies are expressed using a unique ontology modeling language, namely OntoUML. This modeling language has been chosen because it is backed up by a theoretical reference ontology that puts solid foundations for the goal, agent, and service paradigms.
2. Merging our worked-out preliminary ontologies
   As a first attempt, we chose a pragmatic approach to conduct the merging of our proposed preliminary ontologies. A mapping operation stating a direct or indirect correspondence between similar constructs is required before performing the merging. We start by identifying potential similarities between shared modeling constructs, then we either merge both equivalent concepts into one concept in case of a direct mapping, or we build an intermediate ontology serving as a bridging ontology in case of an indirect mapping.

5 Ontology fragments for KAOS

5.1 Choice of UFO and OntoUML

To the best of our knowledge, the sole attempts to build an ontology-based model for KAOS is in [27], using the Unified Enterprise Modeling Language (UEML). This language, intended to unify enterprise modeling and information system modeling, is backed-up by the Bunge-Wand-Weber (BWW) model and Bunge's ontology. Among its use cases, we can mention its use to describe the metamodel of KAOS [27]. The authors of this work noticed that UEML is difficult to use, and exposed some limitations of the UEML language itsellf as well as its ontological framework (at the date of the publication of their work). In our work, we chose to use OntoUML (an ontological-based version of UML) mainly for the following reasons: OntoUML is backed up by UFO, a formal top-level foundational ontology presenting good relevant ontological properties such as completeness; availability of reliable middle-level UFO ontology fragments conceptualizing in a very abstract way concepts like goal, agent, event, and others; OntoUML is easy to learn and use because of its proximity with UML (widely used in various communities); OntoUML offers a palette of stereotyped classes and relationships able to cover and face numerous and various modeling situations. We used the OpenPonk tool [8] to edit and verify the syntax and the semantics of each of our models. The semantics is checked against well-defined anti-patterns.

5.2 Approach

The scope of this work encompasses the four KAOS models constituting the KAOS framework (Agent, Goal, Operation, and Object). The concepts of Obstacle/Conflicts relating to Goal modeling are not addressed, because not pertinent to the results of this (part of) work. Our proposal builds on and refines specific UFO ontological elements to conceptualize concepts relevant to the KAOS models, according to the KAOS universe of discourse. The KAOS metamodel is used as a primary source to capture this universe of discourse, augmenting by the way the degree of validity of our ontology fragment. GORO ontology

has also served as a reliable source for this work, however GORO is mainly used for comparing similar goal-oriented requirement frameworks (KAOS, i-star, and others). Thus, GORO is built in such a way that it abstracts features specific to each framework. Various UFO ontology fragments give a solid theoretical conceptualization of the goal and agent concepts, and their inter-play as well. Relevant UFO classes and relationships of these fragments are selected, reused, and then refined in such a way to be the most closest as possible to the KAOS constructs. The following exposes, for each model of the KAOS framework, the aspects that will be emphasized by our ontology fragments.

- **KAOS Agent Model**
  The KAOS agent fragment has to conceptualize the following aspects:
  (a) KAOS agent, its different sorts as well as the specificity and the meaning of each of its sorts (conformance with the metamodel).
  (b) The eventual real world entities that may be behind each sort of agent. These aspects are out-of scope of the KAOS metamodel. We added them because, in this work, we are interested in the IoT applications.
  (c) The relationships (and their nature) that capture the links between the entities mentioned in (b) and their corresponding agents elicited in (a).

  Figure 1 depicts the ontology fragment that captures the (a) and (c) aspects, while Figure 2 and Figure 3 depict, respectively, the ontology fragments that capture the aspect (b).

- **KAOS Goal Model**
  The KAOS Goal ontology fragments have to conceptualize the following aspects:
  (a) KAOS goal, and its different forms as well as the specificity and the meaning of each of its forms (according to the metamodel).
  (b) The way KAOS goals are classified, structured and organized.

  Figure 4 together with Figure 5 depict the ontology fragment that captures the aspects mentioned above .

- **KAOS Operation and Object Model**
  The KAOS Operation and Object ontology fragments have to conceptualize the following aspects:
  (a) The KAOS operation and their different sorts.
  (b) The operation parameters.
  (c) The pre-state and post-state attached to the operations.
  (d) The Object model state on which operations apply.

The fragment related to these aspects is not described in this paper. All the above mentioned figures are detailed in the following sub-sections.

5.3 KAOS agent ontology

The KAOS metamodel identifies and reveals in an explicit way two sorts of agents according to the role they play in the context of a KAOS based requirements' model, namely an environment agent and a requirement agent. KAOS environment agents are intended to enforce the "satisfaction" of the so-called expectations, whilst KAOS requirement agents are intended to enforce the satisfaction of the so-called "requirements". Expectations and requirements are specific KAOS goals explicitly defined in the KAOS metamodel. KAOS environment agents are exemplified by software agents (artificial active entities) that are concepts related to the computing context, and KAOS environment agents are exemplified by either humans /corporations (physical active entities) that are concepts related to the business and technical contexts. While the previous sentence is just an informal statement written in KAOS documents, our ontology makes it explicit. Figure 1 shows four sorts of OntoUML stereotyped classes (Category, Kind, Relator, and Role). The stereotype Category is generally used for representing abstract classes. We use them to glue our ontology to relevant UFO top- and middle-level categories and then specialize them into for instance Kind, or other stereotyped rigid sortal entities. KAOS agents are characterized by two features. On one hand they are agents (active entities), on the other hand they are intentional agents belonging to the KAOS framework. Figure 1 shows that the KAOS Agent concept is represented by a class (stereotype Kind) that specializes the abstract class Category Agent. This way we ensure that our ontology fragment (Figure 1) faithfully captures the intentional agent concept. Because these agents belong to the KAOS framework, we need to conceptualize the fact that the KAOS framework reveals two sorts of agents. Here we have the choice to capture this aspect by specializing the Kind Agent either into two subkind agents or into two role agents. However the nature of these two sorts of agents seems to be more close to Perdurants rather than Endurants (according to the UFO terminology). That is why we retain the stereotype Role rather than the stereotype Subkind to deal with this aspect. Moreover, Figure 1 captures also the informal statement related to the real-world entities that may exemplify these two roles. For this purpose, we introduce two relators. The first one, Requirement Agent Reification, conceptualizes the idea that software agents, through their specialized role (RoleMixin) KAOS IoT Software Agent, exemplify KAOS Requirement Agent (RoleMixin). The second relator, Environment Agent Reification, conceptualizes the idea that Stakeholder, through their specialized role (RoleMixin) IoT Application Stakeholder, exemplify KAOS Environment Agent.

5.4 Stakeholder ontology fragment

The following fragment, which is outside of the scope of the KAOS metamodel, is useful and relies on the following UFO knowledge: UFO ontology defines a category, named Substantial, which is specialized into the Object and

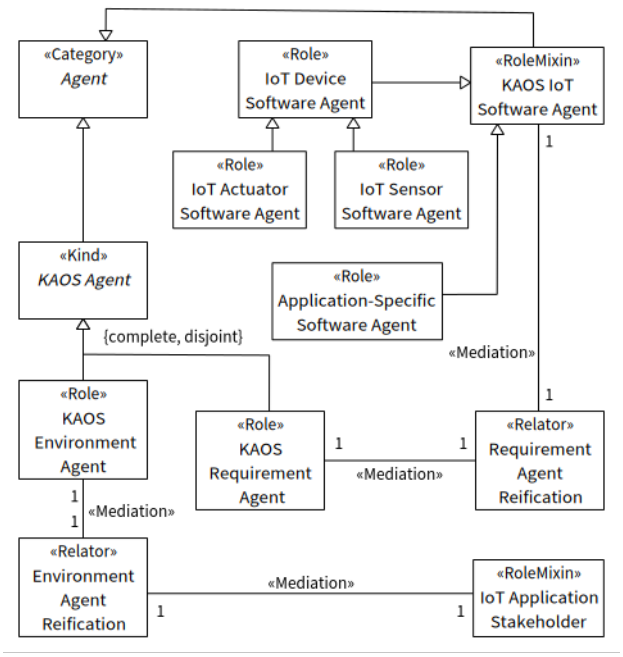

Figure 1: KAOS agent ontology

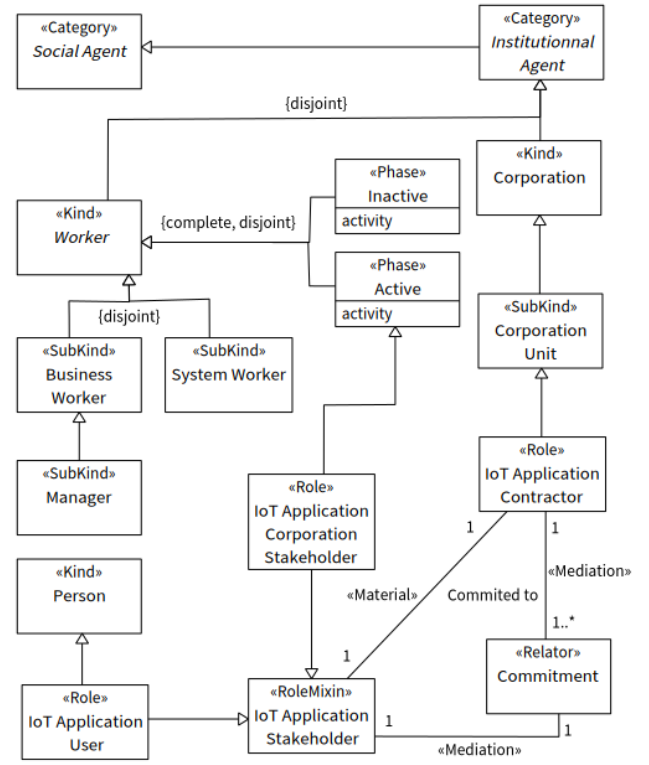

Figure 2: Stakeholder ontology

the Agent categories. Agents conceptualize active entities, whilst objects conceptualize passive and unintentional entities. Agents are of different sorts: physical (e.g., machine, human being), artificial (e.g., software), and social agents (e.g., society associations, institutions, stakeholders). Social Object, a refinement of UFO Object, is intended to conceptualize social objects like value, money, etc. In Figure 2, we propose a conceptualization of stakeholders in the context of IoT applications. Category Institutional Agent, a specialization of UFO Social Agent, is intended to abstract all sorts of institutional agents. Figure 2 shows two sorts of concrete institutional agents. The first one represents workers, conceptualized by a Kind class, acting as IoT project stakeholders, and the second one represents corporations, conceptualized by also by a Kind class, acting as initiator and/or developer of IoT applications. Corporations (Kind class) are modeled as a group of corporation units (SubKind class). Workers may be eventually engaged or not in IoT projects. Two disjoint Phase classes (Active, Inactive) allow to conceptualize this situation. Moreover we distinguish three sorts of workers: Business and System, modeled as Subkind of Kind Worker, and Manager modeled as Subkind of Business. The fragment includes also, a Role class, intended to conceptualize users of IoT applications (Role IoT Application User specializing Kind Person). IoT Application Stakeholder is intended to model the fact that the IoT application stakeholder notion includes application users as well as active workers. Because IoT Application Stakeholder is aggregating two Roles (customer, worker), RoleMixin is the more appropriate OntoUML stereotype class to represent it. Finally, the Relator Commitment conceptualizes the fact that a commitment links the IoT application stakeholders with the with the corporation units that are contractors with the project, i.e., the corporation unit that owns the project and also the corporation units such as the Ambulance Supplier Center, the Security Center, and the Home CareGiving Center. The IoT Application Corporation Stakeholder class represents the active corporation workers participating into the IoT application project.

5.5 IoT software agent ontology fragment

Figure 3 depicts a software agent ontology fragment. In the context of IoT applications, we identify two sorts of IoT software agents: those which are embedded into IoT devices, and those which are hosted by computing devices like servers and others. The first sort is modeled as a Subkind class (IoT Device Software Agent), and the second one as a Role class (Application-Specific Software Agent). The Subkind IoT Device Software Agent is specialized into two Role classes (IoT Sensor Software Agent and IoT Actuator Software Agent).Three Relators (S-Embedding, A-Embedding, and Hosting) conceptualize the embedding as well as the hosting of software agents according to their respective digital devices. Digital Device, is itself specialized into two subkinds: Subkind IoT Device and Subkind Computing Device.

5.6 KAOS goal model ontology

The proposed KAOS goal model ontology is composed of two ontology fragments. The first one, depicted in Fig-

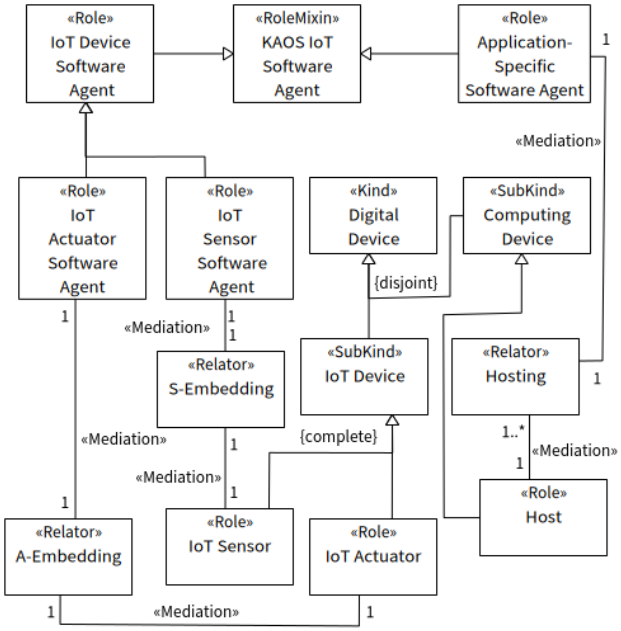

Figure 3: IoT Software agent ontology

ure 4, models the KAOS goal concept as well as the hierarchy structure of the KAOS goal model. The second one, depicted in Figure 5, conceptualizes the various sorts of KAOS goals as well as the relevant links to their corresponding KAOS agents.

5.6.1 Goal and hierarchical model

Figure 4 presents an ontology fragment including three parts: the first part represents the KAOS goal concept, the second part the hierarchical structure of the KAOS goal model, and the third part the concept of the KAOS abstract goal (decomposable goal).

- *Goal Concept*
  A KAOS goal is seen as a RoleMixin of the Category Requirement Engineeging Goal Specializing the Category Goal. Figure 4 shows, for instance, the RoleMixin i^* Goal representing the concept of goal in the i^* framework. The Category Requirement Engineering Goal is introduced to factorize abstract shared features that may exist between different frameworks that deal with the goal oriented requirement approach.

- *Goal Hierarchical Model*
  The RoleMixin Retained KAOS Goal in Figure 4, a specialization of the RoleMixin KAOS Goal, conceptualizes the fact that an elicited KAOS goal is retained to be inserted into the hierarchy of a goal model. The hierarchy concept is modeled as a Kind (Goal Hierarchy), a refinement of an abstract Category, named Structure, intended to represent all sorts of structures (not represented in the figure). The Relator Incorporation, a relationship between a retained KAOS (RoleMixin Retained KAOS Goal) and a KAOS goal hierarchy (RoleMixin KAOS Goal Hierarchy) that conceptualizes the fact that a retained goal is inserted into a goal hierarchy. According to its position in the goal hierarchy, a retained goal may play either the role of an abstract goal or the role of a concrete goal. Figure 4 includes a Kind (Tree Node) to represent nodes of the Tree structure. Tree Node is used as a secondary class allowing the modeling of abstract goals incorporated into a non-leaf node of the hierarchy, and the modeling of a concrete goal incorporated into a leaf node of the goal hierarchy.

- *Abstract Goal*
  Contrarily to a KAOS concrete goal, an abstract one may be refined into sub-goals (decomposable). The KAOS metamodel emphasizes two sorts of goal refinement: AND refinement and OR refinement. The AND refinement specifies that a refined goal is achieved if and only if all sub-goals are achieved. The OR refinement specifies that a refined goal is achieved if and only if one of its sub-goals is achieved. For this purpose, Figure 4 includes two roles (Role AND Decomposable and Role OR Decomposable) refining the RoleMixin Abstract Goal. Finally two relators (Relator OR Parent and Relator AND Parent) conceptualize the relationships between an abstract goal (as a parent goal) and its sub-goals (as child goals). To this end, Figure 4 includes two roles (Role AND Child and Role OR Child) which are refinements of the RoleMixin Retained KAOS Goal.

Figure 5 depicts an ontology fragment intended to represent three sorts of KAOS concrete goals defined in the KAOS metamodel, as well as their relationships with their corresponding KAOS agents or with the KAOS object model. KAOS concrete goal sorts are often named realisable or operational goals in the KAOS literature.

The RoleMixin (KAOS) Concrete Goal is specialized by three roles (Role Expectation, Role Requirement, and Role Domain Property) intended to represent the three sorts mentioned above. Expectations are taken in charge by KAOS Environment Agents, whilst Requirements are to be made operational (realizable) by KAOS Requirement Agents. For this purpose, Figure 5 includes two relators (Relator Assignment and Relator Responsibility) for the conceptualization of the (above mentioned) relationships between Role Expectation and Role KAOS Environment Agent on one hand, and between Role Requirement and Role KAOS Requirement Agent on the other hand. The Role Domain Property is specialized by the Role Domain Hypothesis and the Role Domain Invariant. Domain hypothesis represents hypothesis on the state of the object model that must hold. They are supposed to be enforced by the environment of the application domain. For this purpose the Relator Hypothesis Enforcement conceptualizes the relationship between the Role Implicit Enforcer and the Role Domain Hypothesis. The Kind Environment is modeled as a (shared) part of

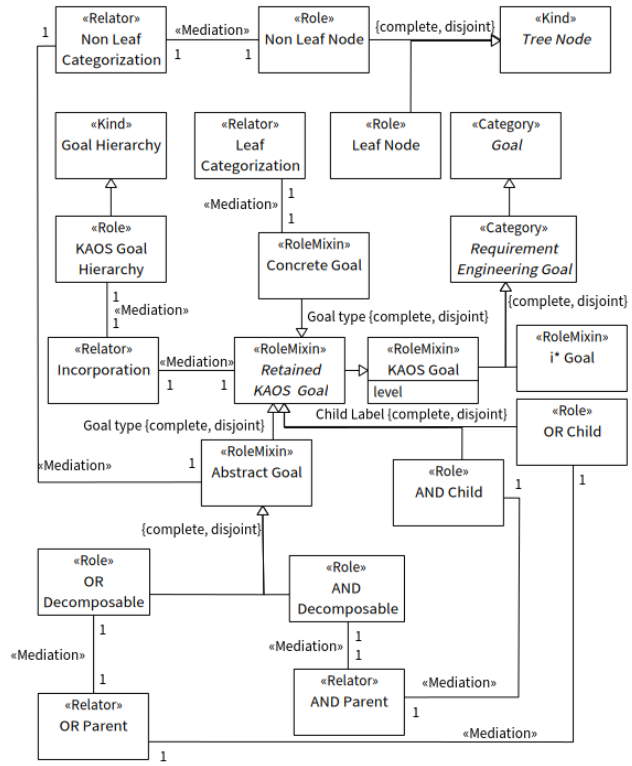

Figure 4: KAOS goal hierarchy ontology

the Kind Application Domain. The domain invariants represent invariants that must always hold in any state of the application domain. The application domain must always fulfil them. For this purpose, the Relator Fulfilment conceptualizes this relationship between the Role As A State and the Role Domain Invariant. Finally, Figure 5 includes the Relator Obligation intended to represent the situation where one or many sub-goals of an abstract goal are domain properties. In this case these properties must hold in order to achieve such an abstract goal.

5.6.2 Responsibility model

Figure 1 shows two Relators intended to conceptualize the reification of the Role KAOSEnvironmentAgent and the Role KAOSRequirement Agent into respectively the stakeholder RoleMixin IoT Application Stakeholder (defined in Figure 2), and into the software agent RoleMixin KAOSIOTSoftwareAgent. This RoleMixin is a specialization of the Category Agent (defined in Figure 1).

5.7 Object and operation models

5.7.1 Object model

In the context of IoT applications, a KAOS object model represents relevant entities (passive objects) and agents belonging to a specific IoT domain and to an IoT application domain. These two domains, represented by their respective Kind (specialization of the Category Object), are modeled as components of the Kind KAOS Object Model.

5.7.2 Operation model

The operation model focuses on the satisfaction of the so-called operational goals (requirements and expectations), and more precisely on the means (operations) and ways to achieve them. KAOS operations are fully described by their trigger, their pre-conditions, their inputs, their outputs, and their eventual generation of events. All these operation components should be modeled. OntoUML provides a set of useful stereotypes intended for the conceptualization of situations (state-like concept), events (action and non-action events), and specific associations. We also reuse the UFO concepts of Atomic and Complex Action.

6 KAOS business aspect reinforcement

6.1 Preamble

Although business goals (high-level KAOS goals) are made explicit in the goal model, they are mainly used for the derivation of operational goals. KAOS models do not carry explicit information on business processes attached to

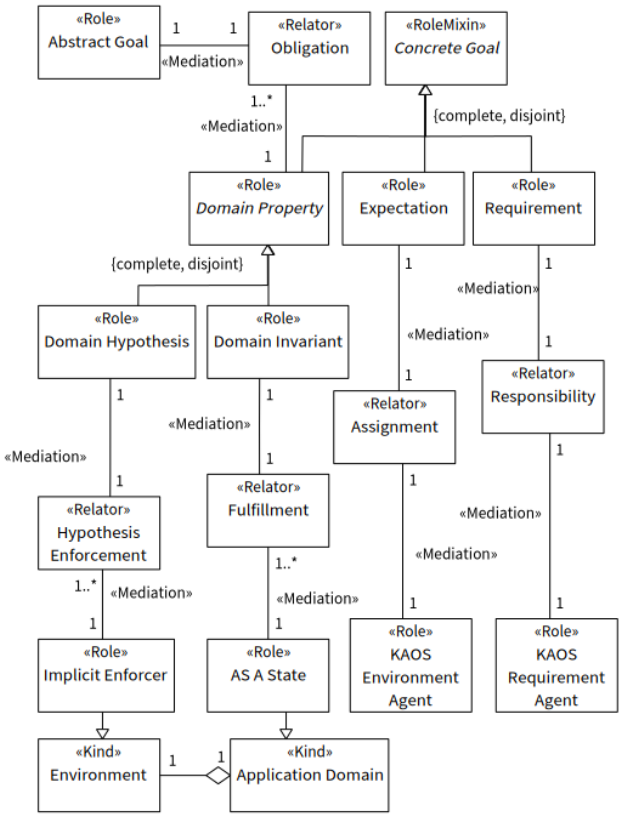

Figure 5: Concrete goal sorts

high-level business goals. The (native) KAOS framework did not give much consideration to the business dimension which is an important pillar for CIM modeling. Many works related to goal-oriented business modeling have been reported. Among these works, we mention GO4SoA [10], where the concept of business goal is incorporated to the SOA, a business architecture reputed centred around the concept of service. GO4SoA specifies SoA applications based on business goals. In this approach, as reported in [10], the goals become part of the semantic services' descriptions.

6.2 Proposal

We adopt another alternative which consists in reinforcing the business dimension of the KAOS modeling framework. This is achieved by augmenting our proposed KAOS ontology with a fragment intended to incorporate the capability modeling construct. We exploit the knowledge carried on one hand by the KAOS goal (top-level goals, abstract goals, concrete goals, and agents), and by the KAOS operations on the other hand. The main idea consists in identifying an additional KAOS-based construct "semantically" equivalent to the SoaML capability construct. This (added) construct, built from a set of specific (native) KAOS constructs is then used to set up an (indirect) mapping between KAOS and SoaML concepts. Firstly, top-level goals (abstract goals positioned just at the first level under the tree root), extracted from the KAOS goal model and representing strategic business goals, are considered as first class elements. They are called Top Business Goals. The last descendant goals (concrete goals) reachable from a given top-level goal constitute the set of operational goals to be achieved in order to satisfy the given top-goal. Secondly, we introduce the notion of KAOS Agent Capability. This notion is intended to represent the set of all KAOS atomic operations that an agent is capable to perform in order to achieve each of the concrete goals under its responsibility (one KAOS operation for each concrete goal). Thirdly, we extend and generalize the notion of KAOS Agent Capability representing the set of capabilities required from a group of KAOS agents to achieve a top-level KAOS goal: it is called Group-Agent Capability.

6.3 Augmented KAOS modeling

Figure 6 depicts a piece of ontology intended to conceptualize the new introduced concepts (Role Top Business Goal, Mode KAOS Agent Capability, and Mode KAOS Group Agent Capability), and their relationships with native KAOS concepts (Agent, Goal, and Operation). The diagram should be read as follows.

- The Role Top Business Goal is modeled as a specialization of the RoleMixin Abstract Goal (defined in Figure 4). The association named Leads to is intended to model that from a top business goal (Role Top Business Goal) a set of concrete goals (Role Concrete Goal) are reachable.

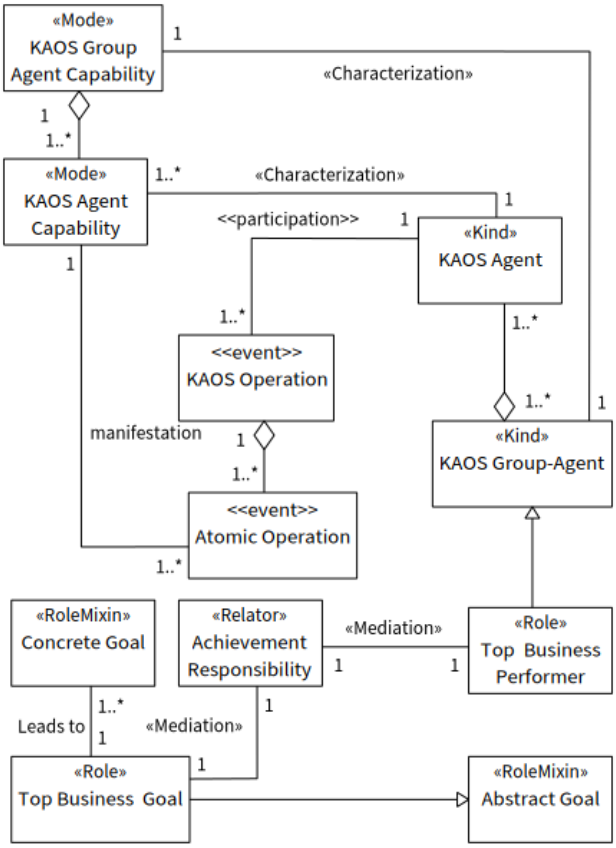

Figure 6: KAOS capabilities ontology

- The Relator Achievement Responsibility is intended to relate a top business goal (Role Top Business Goal) with its corresponding group of KAOS agents (Role Top Business Performer) which are together responsible for the top business goal achievement. These groups, named KAOS Group-Agent, are modeled as a composition of KAOS agents (Kind KAOS Agent).
- A KAOS Agent is responsible for a set of KAOS Operations. We define a KAOS Agent Capability as a coherent subset of the set of atomic operations composing a given KAOS operation; the choice of these subsets and their consistency is the responsibility of the relevant requirements' engineer. In this way, a KAOS Agent owns a set of capabilities. From the OntoUML perspective, KAOS Atomic Operations (considered as events) are a manifestation of the Capabilities of KAOS Agent.
- KAOS Agent Capability is modeled as a Mode class depicting a feature of a KAOS Agent. A stereotyped association (<<Characterization>>) links the Kind KAOS Agent to the Mode KAOS Agent Capability.
- A KAOS Group Agent Capability is characterized by

a feature intended to represent operational skills that it can potentially perform. This feature is modeled as a Mode class. Concretely it is modeled as a composition of the capabilities associated with all KAOS agents composing the group.

– The Kind KAOS Agent can participate into several KAOS operations. This is modeled by the association stereotyped by <<participation>>. KAOS operations are composed of a set of atomic operations (Atomic Operation).

7 Our preliminary SoaML ontology

7.1 Approach

The scope of this (part of) work encompasses the core SoaML business concepts and constructs. We are interested only in the constructs related to the modeling of the business aspect (CIM model). More precisely our SoaML ontology framework includes entirely the models of service, capability, participant, port, service interface, service architecture, and service contract. Our proposal builds on and refines specific UFO ontological elements to conceptualize concepts relevant to the SoaML business models, according to the SoaML universe of discourse. The SoaML metamodel as well as published works aiming at developing ontologies for the Service-oriented Architecture (SOA) [37, 28] are used as primary sources to capture this universe of discourse, augmenting by the way the degree of validity of our ontology fragment.

7.2 SoaML capability

A capability represents the ability of a SoaML entity to produce an outcome (business value) through a service. A SoaML service is a mechanism allowing to access exposed capabilities through an interface. Figure 7 depicts a piece of ontology conceptualizing the notion of SoaML capability and its relationship with the notion of a SoaML service. The diagram includes the Kind Service, a specialization of the Category Business Object, which is itself a refinement the Category Social Object. The Role Enabled Service, a specialization of the Kind Service, conceptualizes the fact that when services are created, a stereotyped association (<<Characterization>>) links the Category Business Object to the Mode Value, expressing the fact that business objects intrinsically carry values. The diagram shows two Relators (Production and Exposure): the Relator Production links the Mode SoaML Capability to the Mode Value, expressing the fact that a capability produces a value. The Relator Exposure links the Role Enabled Service, a specialization of the Kind Service, to the Mode SoaML Capability.The Relator Exposure conceptualizes the fact that when services are enabled, their capabilities become exposed to the environment.

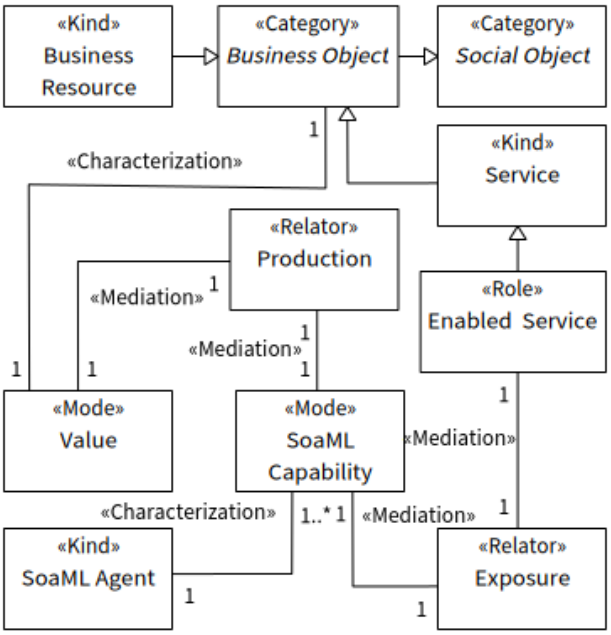

Figure 7: SoaML capability and service ontology

7.3 SoaML agent

According to the SoaML specification document [33], participants are constructs intended to represent domain entities able to provide and/or consume services through ports. In the business domain, participants may be business workers and stakeholders, corporation units, and artefacts like business architectures. In the computing domain participants may be software agents, and artefacts like software components and architectures. Two sorts of domain entities are identified in the SoaML metamodel: intentional entities (agents) and unintentional ones. These domain entities are abstracted by the concept of participants. (SoaML) agents are seen as a special sort of participants. (SoaML) ports are a kind of interaction points "anchored" to participants. SoaML identifies two sorts of ports, named respectively Request Port and Service Port: the first sort is used by service consumers to submit their requests to service providers, whilst the second one is used by service providers to offer their services. Figure 8 depicts a piece of ontology defining the concept of SoaML agent. The diagram in Figure 8 should be read as follows. SoaML Agent is conceptualized as a Kind that specializes the Category Agent. It is characterized by two features: the Mode Business Capability and the Mode Communication Capability. The figure also includes the Kind KAOS Agent, specializing the Category Artificial Agent and characterized by the Mode Communication Capability such as the SoaML Agent.

Figure 9 depicts a piece of ontology conceptualizing the participant, the ports' constructs as well as their interrelationship.

The depicted diagram should be read as follows. The

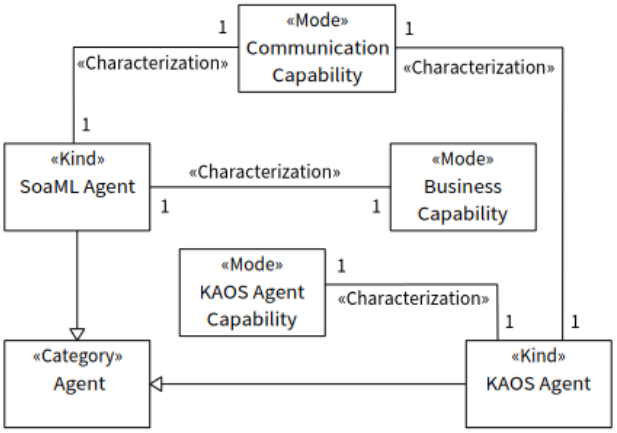

Figure 8: SoaML capability

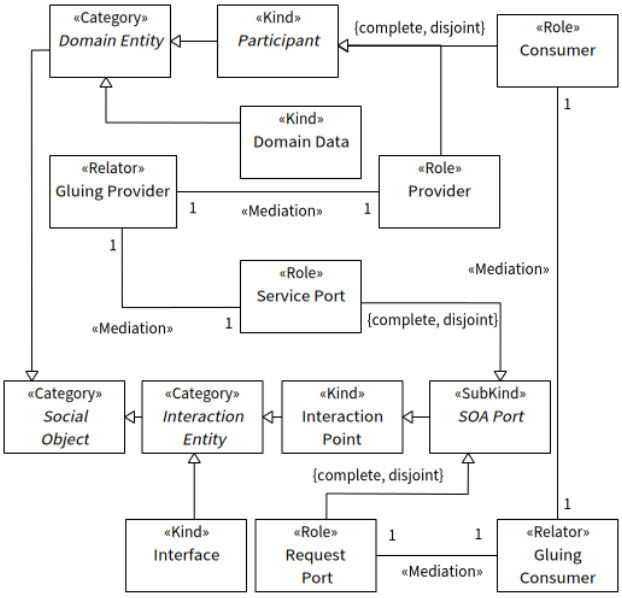

Figure 9: SoaML participant-port ontology

Category Social Object is specialized by two Categories. The first one named Domain Entity is intended to conceptualize the business entities belonging to domains. The second one named Interaction Entity is intended to model all sorts of interaction entities. The Kind Participant, specializing the Category Domain Entity, conceptualizes the concept of SoaML participant. The Role Provider and the Role Consumer, specializing the Kind Participant, conceptualize the fact that SoaML participants can play either the role of service provider, the role of service consumer, or both. We introduce a new Category, named Interaction Entity, intended to conceptualize various kinds of abstract interaction items such as interaction points, interaction protocols, interfaces, and connectors. The Kind Interaction Point, specializing the Category Interaction Entity, represents one sort of interaction entities. The Subkind SoA Port, a specialization of the Kind Interaction Point, conceptualizes the SoA

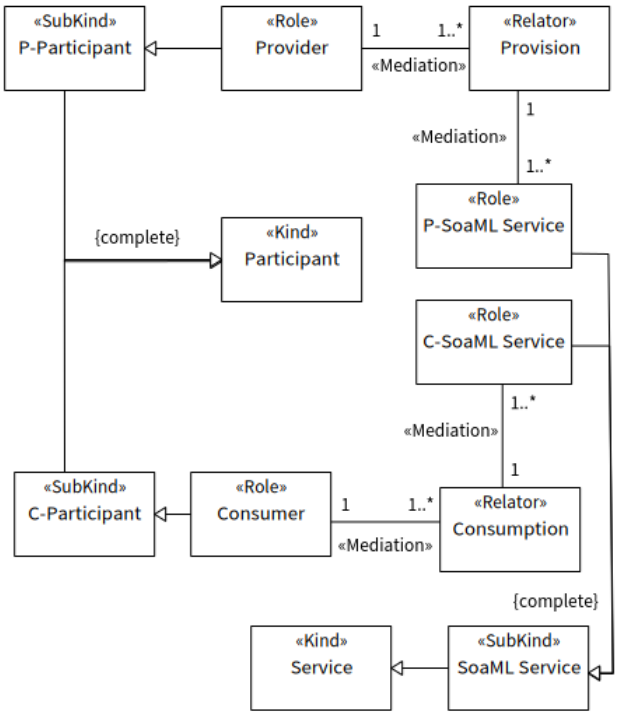

Figure 10: SoaML participant-service ontology

notion of port. There are two sorts of ports: the Request Port that represents the interaction point through which the service is requested, and the Service Port that represents the interaction point through which the service is offered. The SoA Port is specialized into the Role Service Port and the Role Request Port, thus emphasizing its specific roles, namely Request and Service. Two Relators, named Gluing Provider and Gluing Consumer, conceptualize the "glue" relationships respectively between Role Provider and Role Service ports, and between Role Consumer and Role Request ports.

7.4 SoaML participant and service

Figure 10 is intended to conceptualize the relationship between the participant and the service concepts. The Kind Participant is specialized by two Subkinds, namely P-Participant and C-Participant. They are dummy classes introduced to make the intended model more readable and semantically correct. P-Participant is specialized by the Role Provider, while C-Participant is refined into the Role Consumer.

The SoaML Service is modeled as a SubKind of the Kind Service. It is specialized by two Roles (P-SoaML Service, and C-SoaML Service). The Relators Provision and Consumption conceptualize the fact that SoaML services are offered or consumed by SoaML participants.

7.5 SoaML interfaces

As reported in [34], "service interfaces are used to describe provided and required operations to complete services' functionality". SoaML provides three sorts of interfaces:

- Simple Interfaces that give access to very basic services requiring a "one-way" interaction (request/response pattern), where a service consumer calls the (unique) operation offered by service provider.
- Service Interfaces that give access to services requiring bidirectional interactions (conversation pattern) between a service provider and a service consumer. A service interface types a service port belonging to the provider. From the perspective of a service provider, three elements completely define Service Interfaces: the interface it realizes and those it uses in order to offer and accomplish its provided service, the description of the (required) conversation between the service provider and its consumer, and an enclosed part specifying the specific role played by each of the connected participants. This last element is not addressed in the current work.

Figure 11 depicts a diagram intended to conceptualize the SoaML interface notion. The diagram includes also useful and required surrounding modeling elements. The modeling elements depicted in this figure are briefly described in the following.

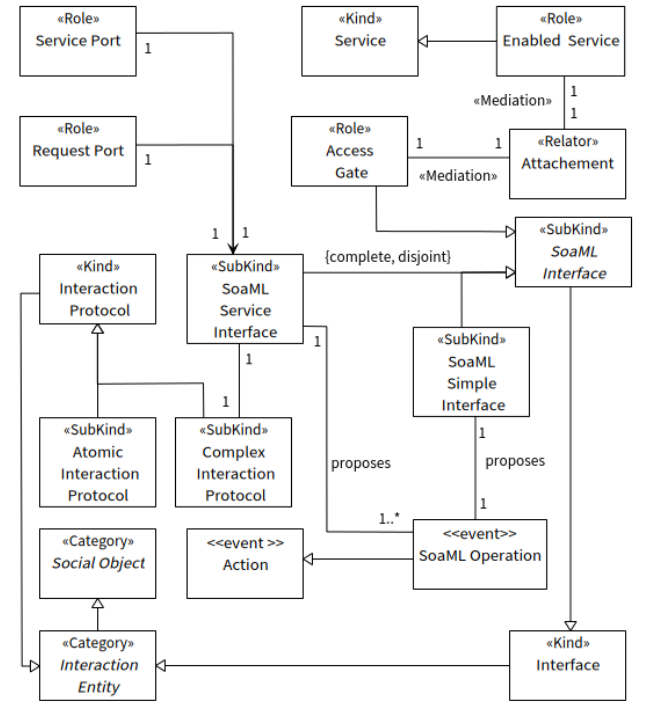

Figure 11: SoaML interfaces ontology

- Surrounding Modeling Elements
  We define the Category Interaction Entity as a specialization of the Category Social Object. Two Kinds, specializing the Category Interaction Entity, conceptualize respectively the general concepts of Interface and Interaction Protocol. On the one hand, we define SoaML Interface as a special sort of Interface, and on the other hand, two sorts of Interaction Protocol, namely Atomic Interaction Protocol intended to represent a simple one-way interaction protocol and Complex Interaction Protocol for bidirectional interactions. Conversations (Complex Interaction protocols) are modeled as ordered sequences of atomic interactions.
- Interfacing
  SoaML advocates the separation between the interface giving access to a service (visible part) and the service implementation (hidden part). The Role Access Gate, a specialization of the SubKind SoaML Interface, represents the visible part of the SoaML service. The Role Enabled Service, a specialization of the Kind Service, represents the service hidden part.
- SoaML Interfaces
  The subkinds SoaML Simple Interface and SoaML Service Interface, specializing the Kind SoaML Interface, conceptualize the SoaML simple interface and the SoaML service interfaces. The association named proposes expresses the fact that a simple interface offers a unique SoaML operation modeled as <<event>> Action. an event Action is intended to represent operations, tasks, and processes [21]).
- SoaML Service Interface
  The SoaML Specification Document [33] specifies two typing constraints: "the Service Interface is the type of a "Service" port on a provider and the type of a "Request" port on the consumer".

7.6 Services' architecture

This sub-section focuses on an important SoaML construct, namely Services' Architecture. It represents a high-level SoaML business construct that describes how participants work together for a purpose by providing and using services expressed as service contracts [33]. They are mainly used to define and specify compound services resulting from a composition of services provided and consumed by a community of (two or more) participants. A Services' architecture consists of a diagram linking services (instances) to be composed, with both their corresponding providers (instance) and consumers (instance). It can be derived from appropriate instantiations of the piece of ontology depicted in Figure 10. Because OntoUML does not provide means to instantiate instances (UFO individuals), we cannot express service architecture with OntoUML diagrams. However, we can visualize a services' architecture as a set of related

pairs, the first element of such pairs representing an instantiation of the Relator Provision and its second element representing an instantiation the Relator Consumption.

8 Merging KAOS and SoaML ontologies

The merging operation between two source ontologies is generally done through their potential shared (or overlapped) native concepts (if any), and/or through the elaboration of appropriate ontological bridges (pieces of intermediate ontologies). In this work we use both approaches in a complementary way. The first approach requires a prior identification and deep analysis of such shared native concepts, whilst the second requires a prior identification of suitable anchoring points, in both source ontologies, where the bridge may be attached. The ontology mapping operation is considered as a preliminary step along a process of merging or integration of ontologies. It consists in the precise characterization of the semantic correspondence (semantic similarity) relating potentially two or more "similar" concepts. Such correspondences between concepts may be simple (one to one), or complex (via a transforming operation or a logical expression relating similar concepts). In some situations, an appropriate intermediate piece of ontology is used as a bridge between similar concepts of different source ontologies. The analysis of both KAOS and SoaML modeling languages reveals two sorts of situations: (1) both languages share in an explicit way similar concepts (KAOS domain object versus SoaML participant; KAOS agent versus SoaML agent; KAOS operation versus SoaML operation); (2) one of these languages provides a concept in an explicit way, while a similar concept is hidden or implicit in the other language (KAOS goal versus SoAML business objective; KAOS capability versus SoaML capability).

– Explicit Similar Concepts

  (a) Domain Object versus Participant
  Both concepts refer to intentional (active) and non-intentional (passive) entities. However there are two slight differences: Participants provide /consume services, whilst the concept of service is not an inherent feature of domain objects; participants communicate through message exchanges, whilst the communication aspect between domain objects is left unspecified. Although these two concepts are not semantically equivalent, we can say that they are sibling concepts, because of their overlapping correspondence. One way to put these two sibling concepts together is to root them to a common parent concept. In our ontology, the Object category fills this need.

  (b) KAOS Agent versus SoaML Agent
  A priori, they are similar in the sense that both are intentional entities, but KAOS agents can be refined to distinguish two specific agents, which is not the case for SoaML agents. Although these two concepts are not semantically equivalent, we can say that they are sibling concepts because of their overlapping correspondence. This situation is treated in the same way as in the previous case.

  (c) KAOS Operation versus SoaML Operation
  At first sight these concepts seem to be very similar. However there are two differences between them: in contrast to KAOS operations, which generally are complex actions intended to satisfy requirements, SoaML operations are atomic actions, offered through specific interfaces, and intended to provide services; SoaML operations can either be called (request/response) or invoked through message exchanges, while KAOS operations are triggered by events. It is worth noting that message sending and receiving are themselves sorts of non-action events.

– Implicit Similar Constructs

  (a) SoaML Capability versus KAOS Capability
  SoaML provides explicit constructs intended to represent both the capability and service concept, the service concept being built on the capability concept. The capability concept, though mentioned in the reported KAOS literature, is not perceived as an explicit KAOS construct. In this situation, our approach consists in extending KAOS in such a way to make explicit the notion of KAOS capability, and to define it in such a way to make possible a mapping to the SoaML similar concept. Furthermore, the proposed extension ensures, through this mapping, a "natural connection" with the notion of service.

  (b) KAOS Goal versus SoaML Business Objective
  KAOS deals in an explicit way with goals distinguishing between business (top level) and operational goals (leaf level) thanks to their relative positions in the so-called KAOS goal model. Regarding SoaML, it does not support an explicit goal construct. However, business goals and processes are supposed to be (previously) identified in an explicit way using an appropriate business modeling language, and then SoaML models (capabilities, service contracts, and services' architecture) may be derived from these identified business goals and processes. Business goals are considered only as implicit input knowledge exploited by SoaML modelers. Thus we may conclude that KAOS goals have no similar SoaML construct.

9 Instantiating OntoUML models

9.1 Transformation approach

The OntoUML models describing our proposed ontology, once designed, edited and their syntax and semantics verified, are ready to be instantiated in concrete OWL classes and populated with concrete individuals (lowest instance level) related to specific IoT applications, thus leading to a "query-able" Knowledge/Data graph. More precisely, OntoUML models can be transformed (manually or automatically) into appropriate gentle UFO (gUFO) classes. gUFO is an extension of OWL supporting UFO theoretical framework as well as the set of OntoUML stereotypes. To this end, we use the following approach. First, we transform (a selected subset of) our proposed ontology fragments into their corresponding gUFO descriptions according to UFO semantic rules. Second, we instantiate these gUFO descriptions into specialized gUFO classes and concrete individuals corresponding to the needs expressed by stakeholders (cf. Section 10).

The transformation is performed as follows. Our OntoUML (stereotyped) classes (reflecting a reality such as KAOS, and generic IoT concepts independent from a specific IoT application) are directly transformed into corresponding gUFO classes, i.e., gUFO:Category, gUFO:Kind, gUFO:SubKind, gUFO:Role, gUFO:RoleMixin, gUFO:Relator, and others. This is performed, of course, according to the class hierarchies and relationships belonging to our OntoUML models. For this purpose we mainly use rdf and rdfs sentences such as
:C rdf:type gufo:X.
:C rdfs:subClassOf gufo:D.
:C rdfs: superClassOf gufo:E.
The first sentence means that C, a stereotyped class belonging to our model, is a class instance of the gUFO class X, the second sentence means that C is a subclass of D, and the third sentence means that D is a superclass of E. These descriptions are useful to correctly describe the hierarchy of the stereotyped classes belonging to our OntoUML ontology fragments. Other specific rdf sentences are used to express mediations linking OntoUML classes and Relators.

9.2 Samples of queries on our populated ontology fragments

This subsection presents samples of generic queries that may be issued by relevant users to the ontology fragments populated with instances related to IoT applications under specification. The intended users of these ontology fragments are the stakeholders engaged in the CIM development of IoT applications. These samples are categorized as follows.

Goal Hierarchy

1.1 What are the top goals? 1.2 What are the concrete goals associated with a top goal? 1.3 What are the children goals of a parent goal? 1.4 Which kind of composable goal is a parent goal?

Goal-Agent-Stakeholder-Operation

2.1 List the requirement agents. 2.2 Which stakeholder is responsible for a goal? 2.3 What are the domain properties enforced by an environment agent? 2.4 What are the domain properties required by an abstract goal? 2.5 Which expectation is as- signed to an environment agent? 2.6 Which requirement agent is responsible for a requirement? 2.7 List the expectations. 2.8 List the domain invariants 2.9 Which stakeholder is reified by an environment agent? 2.10 Which software agent is reified by a requirement agent? 2.11 List the input parameters of an operation. 2.12 List the output parameters of an operation. 2.13 Which agent participates to an operation? 2.14 What are the capabilities of an agent?

IoT

3.1 List the IoT sensors 3.2 List the IoT actuators 3.3 List the software agents 3.4 Which software agent is embedded into a IoT device? 3.5 Which software agent is hosted in a computing device ? 4.1 List the participants. 4.2 List the agents. 4.3 What is the business capability of a participant? 4.4 What is the communication capability of a participant? 4.5 List the services. 4.6 What are the capabilities exposed by a service? 4.7 What is the value produced by a capability? 4.8 What is the interface of a service? 4.9 What is the provider participant of a service interface? 4.10 What are the ports of a service interface? 4.11 What is the operation provided by a simple interface?

10 An illustrative example

This section has two parts. A problem statement for our illustrative IoT application, along with the needs elicited, is first outlined, then followed by a gUFO instantiation of selected OntoUML models represented in Figures 1, 2, 3 , 4, and 5.

10.1 Problem statement and needs' elicitation

10.1.1 Problem statement

The chosen example scenario, inspired from the one described in [11], is adapted and enriched in order to fulfil our illustrative needs. A platform, equipped with various smart sensors and actuators located inside the patient home, is intended to monitor and control a set of patient vital parameters, of home-related parameters (temperature, humidity), and of security-related parameters (face recognition). The objective of this platform is to maintain patient health condition and to provide patients with medical care in case of emergency situations.

10.1.2 Elicitation

(a) Stakeholders
Corporation owning and managing the platform: Manager, Remote Patient Followers, Health CareGiving Centers, Ambulance Supplying Centers, Security Centers, Patients.

(b) Domain Entities

+ IoT devices: smart sensors (medical, temperature, humidity, face recognition camera, windows' status monitor), smart actuators (dehumidifier, air conditioner, window opener/closer).
+ Health - vital parameters: heartbeat, blood pressure, oxygen rate, and so on.
+ Home Infrastructure Hypothesis: automatic (opening/closing) windows, Internet/IoT infrastructure, Health Platform installed.

(c) Goal Model

+ Functional Goals
  – Root goal: Manage Patient.
  – First-level goals (AND): Keep Patient Healthy, Achieve Patient Authentication.
  – Keep Patient Healthy (AND): Patient Condition Monitoring, Corrective CounterMeasures.
  – Patient Condition Monitoring (AND): Medical Parameters Monitoring, Home Parameters Monitoring.
  – Medical Parameters Monitoring (AND): Heartbeat Measurement, Blood Pressure Measurement, Oxygen Rate Measurement.
  – Home Parameters Measurement(AND): Temperature Measurement, Humidity Rate Measurement.
  – Corrective CounterMeasures (OR): Medical Care CounterMeasures, Home CounterMeasures.
  – Home CounterMeasures (AND): Control Temperature, Control Humidity. Control Temperature (OR): Device based Air Conditioning, Natural Air Conditioning.
  – Control Humidity (OR): Device Based De-Humidification, Natural De-Humidification.
  – Medical Care CounterMeasures (OR): Emergency Evacuation, At Home CareGiving.
  – Achieve Patient Authentication (AND): Patient Registration, Patient Authentication Checking.
  – Patient Registration (AND): Platform Prompting, Patient Information Filling.
  – Patient Authentication Checking(AND): Face Capturing, Face Checking.
+ Domain Properties
  – Domain Invariants: Admitted ranges of vital and home parameters values.
  – Domain Expectations: Specific Home Appliances installed and working, IoT Internet available.
+ Listing and categorizing of the potential pertinent agents
  – Users and Workers: Patient (IoT Application User), HomePlatform Product Responsible (manager), At-Home CareGiver (business worker), Remote Patient Follower (business worker), Security Checker (business worker).
  – Corporation Units: Health CareGiving centers, Ambulance supplying centers, Security centers.
  – Software
    · Application Specific Software: Patient Platform Manager.
    · IoT Software: A specific Software Agent for each smart sensor and actuator engaged in the platform.

10.2 Transformation into gUFO and instantiations

In this section we apply the two-steps' approach stated in Section 9.

10.2.1 Transformation into gUFO

We cover almost all sorts of stereotyped classes used in our OntoUML models. The following shows the transformation of a set of significant classes belonging to some selected figures.

+ **KAOS Agent Ontology** (cf. Figure 1) **Category and Kind** :Agent rdf:type gufo:Category.
:KAOSAgent rdf:type gufo:Kind;
rdfs:subClassOf :Agent.
Role and RoleMixin
:KAOSEnvironmentAgent rdf:type gufo:Role;
rdfs:subClassOf :KAOSAgent.
:KAOSIoTSoftwareAgent rdf:type gufo:RoleMixin;
rdfs:subClassOf :Agent;
rdfs:superClassOf :IoTDeviceSoftwareAgent;
rdfs:superClassOf
:ApplicationSpecificSoftwareAgent.

+ **IoT Software Ontology** (cf. Figure 3)
RoleMixin and Role

:KAOSIoTSoftwareAgent rdf:type gufo:RoleMixin;
rdfs:subClassOf :Agent;
rdfs:superClassOf :IoTDeviceSoftwareAgent;
rdfs:superClassOf
:ApplicationSpecificSoftwareAgent.
:IoTSensorSoftwareAgent rdf:type gufo:Role;
rdfs:subClassOf :IoTDeviceSoftwareAgent.
:IoTActuatorSoftwareAgent rdf:type gufo:Role;
rdfs:subClassOf :IoTDeviceSoftwareAgent.

+ **KAOS Goal Hierarchy** (cf. Figure 4)
Category and RoleMixin
:Goal rdf:type gufo:Category.
:RequirementEngineeringGoal rdf:type
gufo:Category;
rdfs:subClassOf :Goal.
:KAOSGoal rdf:type gufo:RoleMixin;
rdfs:subClassOf :RequirementEngineeringGoal.

+ **Role and RoleMixin**
:RetainedKAOSGoal rdf:type gufo:RoleMixin;
rdfs:subClassOf :KAOSGoal;
:rdfs:superClassOf :ConcreteGoal;
rdfs:superClassOf :AbstractGoal.
:AbstractGoal rdf:type gufo:RoleMixin;
rdfs:subClassOf :RetainedKAOSGoal;
rdfs:superClassOf :ORDecomposable;
rdfs:superClassOf :ANDDecomposable.
:ORDecomposable rdf:type gufo:Role;
rdfs:subClassOf :AbstractGoal.
:ANDDecomposable rdf:type gufo:Role;
rdfs:subClassOf :AbstractGoal.

+ **Concrete Goal Sorts Ontology** (cf. Figure 5)
Role and RoleMixin
:ConcreteGoal rdf:type gufo:RoleMixin;
rdfs:subClassOf :RetainedKAOSGoal;
rdfs:superClassOf :Requirement;
rdfs:superClassOf :Expectation;
rdfs: superClassOf :DomainProperty.
:Requirement rdf:type gufo:Role;
rdfs:subClassOf :ConcreteGoal.
:Expectation rdf:type gufo:Role;
rdfs:subClassOf :ConcreteGoal.
:DomainProperty rdf:type gufo:Role;
rdfs:subClassOf :ConcreteGoal.
:DomainHypothesis rdf:type gufo:Role;
rdfs:subClassOf :DomainProperty.
Relator and Mediation
:Assignment rdf:type gufo:Relator.
:AssignmentInvolves rdf:type owl:ObjectProperty;
rdfs:subPropertyOf gufo:mediate;
rdf:domain :Assignment;
rdf:range :Expectation;
rdf:range :KAOSEnvironmentAgent.

10.2.2 Instantiations of concrete individuals

The following shows samples of concrete individual instantiations. These are either directly instantiated from stereotyped classes of our OntoUML models or in an indirect way through the specialization of stereotyped classes of our OntoUML models.

(A) Direct concrete individuals instantiations

+ **Abstract goals** (samples)
Patient (:ag0), Keep Patient Healthy (:ag1), Achieve Patient Authentication (:ag2), Patient Condition Monitoring (:ag3).
:ag0 rdf:type :AbstractGoal, …

+ **Concrete goals** (samples)
Blood Pressure Measurement (crg2), Oxygen Rate Measurement (crg3)
Temperature Measurement (crg4) are instances of :Requirement Role
:crg2 rdf:type :Requirement
Patient Information Filling(ceg13) is an instance of :Expectation Role
:ceg13 rdf:type :Expectation

+ **IoT Devices** (samples)
dhd: rdf:type IoTActuator; (dehumidifier device)
:wd rdf:type IoTActuator; (window actuator device)
:tempd rdf:type :IoTSensor; (temperature device)
:hbd rdf:type :IoTSensor. (heartbeat device)

+ **Environment Agent**
Only two instances can be instantiated from this Role class
:asanappuser rdf:type :KAOSEnvironmentAgent;
:asacorpworker rdf:type :KAOSEnvironmentAgent.

(B) Indirect concrete individual instantiations

+ **Patients**: First we specialize the Role class IoT Application User (cf. Figure 2) into the class Role Patient, then we instantiate concrete individuals of the class Patient
:Patient rdf:type gufo:Role
rdfs:subClassOf IoTApplicationUser
:patient1 rdf :Patient

+ **Stakeholder**: First we specialize the Role class IoT Application Corporation Stakeholder (cf. Figure 2) into four Role classes :AtHomeCareGiver, RemotePatientFollower, SecurityChecker, and Home-PlatformProductResponsible.

Then we instantiate concrete individuals of these four Role Classes:
:atg rdf:type :AtHomeCareGiver;
:rpf rdf:type :RemotePatientFollower;
:sc rdf:type :SecurityChecker;
:hppr rdf:type :HomePlatformProductResponsible.

11 Concluding remarks, discussion, and future research directions

11.1 Results

This work presents an ontological-based framework intended to help CIM modelers in their preliminary analysis of IoT applications under development. The use of ontologies is mainly motivated by their high aptitude to address descriptive aspects inherent to CIM modeling, and the profusion of available domains' ontologies making them reusable in various IoT applications. The proposed framework relies on a combination of KAOS (a good candidate for the requirements and domain modeling) and SoaML (for the business service modeling). The outcomes of this work are fourfold. Preliminary ontologies for KAOS and SoaML, addressing their respective key concepts, as well as basic IoT elements, are built; a pragmatic merging of these two ontologies is proposed; an implementation of our ontology fragments in gUFO classes aimed at their computerization is given; an illustrative example demonstrating the applicability of our results to IoT applications is presented. It is worth mentioning, that OpenPonk [8] was used to edit and verify the syntax and the semantics of our models before transforming them into gUFO. The semantics is checked against well-defined anti-patterns.

11.2 Discussion

In this section, we first focus the discussion around the comparison of our work with (only) those mentioned in Table 1, i.e., the works that are the closest to our work; then we compare our proposed KAOS and SoaML ontologies with those reported in the literature.

11.2.1 Comparison with works reported in Table 1

According to Table 1, [38] is (to the best of our knowledge) the sole work combining KAOS and SoaML concepts aimed at addressing IoT requirements. What primarily distinguishes our work from[38] is our adoption of ontological metamodeling rather than "traditional" UML metamodeling. The advantage of this alternative approach reported in [24] is based on the mapping of elements of modeling languages to appropriate ontological concepts; it lies mainly in the improvement of the semantic perspective thanks to the use of well-defined and sound (OntoUML) stereotypes. Our ontology fragments are built according to top-level UFO goal and agent fragments on one side, and KAOS and SoaML metamodels on the other side. Additionally, the resulting OntoUML models, as conceptual ontologies, are transformed into either operational ontologies (gUFO/OWL) or formal specifications (Alloy formal specification language) for further analysis and simulation purposes. [43] enriched KAOS/SysML (a combination of KAOS and SysML) with an ontological approach. However, unlike our entirely ontological approach, the concept of ontology used in [43] covers only the domain aspect. Furthermore, our proposal, aimed at IoT applications, not only explicitly addresses KAOS software agents specific to the IoT context, but also addresses the application domain (KAOS Object Model) from an IoT perspective. [11] provided a UFO ontological approach bringing together the concepts of Goal, Agent, Task, and Service. Their objective differs from ours: first, their proposed ontology serves as a "framework" for developing metamodels for new domain-specific modeling languages, while ours reinforces the semantics of metamodels of an existing, well-established modeling language (KAOS and SoaML); secondly, they consider the SoA principle, where services are traditionally discovered and searched from the outside of the "calling" application, while on our side we focus on the SoaML modeling language rather than the mentioned SoA principle. The novelty consists in an ontological bridging between our proposed KAOS ontology fragments and SoaML fragments: bringing "semantically" closer KAOS agents and SoaML participants, KAOS Goal and SoaML Service through their "shared" capability concept which is implicitly mentioned in the KAOS literature and explicitly defined in the SoaML metamodel. This way facilitates a shift from a KAOS goal model to a SoaML architecture. Finally, [13] proposes a goal oriented methodology intended for AAL requirements (GoAAL), that is mixing ontology fragments from diverse sources (such as IoT domain and health domain). The work adopts a goal ontology fragment based on a variant of i-star, rather than KAOS, for goals. Although the concepts of task and operation are retained, the service concept is not considered in [13].

11.2.2 KAOS and SoaML ontologies

The literature reported ontology proposal(s) for KAOS [27], and for SoaML [30, 37, 29]. In [27], the authors used on one hand a standard KAOS metamodel as a reference to build their KAOS ontology, and on the other hand the Unified Enterprise Modeling Language (UEML) as an ontological modeling language. UEML is usually intended for enterprises and information systems' modeling. Our KAOS ontology not only refers to a standard KAOS metamodel, but is also based on proposed refinements of modeling elements drawn from UFO-C (Agent and Goal) ontology fragments [19, 23], and also on proposed refinements of the Goal-oriented Requirement Ontology (GORO) [31, 23]. Regarding SoaML, [30] and [37, 29] are among the recent works aiming at introducing ontology concepts for SoA and SoaML. The authors of [30] have analyzed and evaluated

the use of UFO-S, a UFO sub-ontology for services, in various approaches including the SoaML one; those of [37] have proposed a set of ontologies covering the general aspects of the so-called Service Engineering (service-oriented architecture, software service ontology, etc.) including ontologies for SoaML. Their work relies on the Open Group Service Ontology, and also on the ISO/IEC SoA Reference Architecture. [29] provides a comparison between the UFO-S service ontology and other similar service ontologies including the Open Group Service Ontology. Our SoaML ontology is built using refinements and adaptations of some modeling elements of UFO-A and UFO-B ontology fragments. The suggested adaptations mainly rely on relevant knowledge drawn from the OMG SoaML specification document [33] as well as from [37, 29].

11.3 Limitations and future research directions

This work consists in proposing an ontological conceptualization of IoT CIMs initially modeled using a combination of KAOS and SoaML, with KAOS addressing both requirements and domain aspects, while SoaML addressing the business aspect through the service and service architecture concepts. A set of preliminary related ontology fragments illustrated by samples of instantiations constitutes our main results. At this first and current stage of our work, we identified some limitations that deserve to be addressed in future contributions. A first limitation concerns the use of structural models in general and OntoUML in particular to address business process modeling. We envisage for a future work to investigate a type of model more adapted to process modeling such as BPMN (Business Process Model and Notation) or a combination of BPMN and DEMO (Design & Engineering Methodology for Organisations). A second limitation comes from our (re)-use of the Agent and Goal UFO-C fragments, while for the time being gUFO does not support UFO-C, which prevents their potential instantiation using gUFO. In order to make our fragments more reliable, an extension of gUFO constitutes a future research direction. In our proposed fragments, we introduced some aspects related to the IoT world, particularly IoT Software Agents and IoT devices. We also suggested that, in the context of IoT, the KAOS Object Model encloses available public ontologies related to the IoT domain as well as the application domains. However, we adopted in our approach a generic and general purpose SoaML service concept. A future development of our work in this direction using a combination of BPMN and DEMO seems to be a promising future work.
As other future works, we plan to:

- transform OntoUML based ontologies into Alloy formal specifications for verification and simulation purposes [7].
- Enrich the KAOS ontology by adding the conceptualization of the obstacle and conflict concept.

Acknowledgement

The authors thank the anonymous reviewers for their valuable comments that helped improve this version of the paper.

References

[1] S. Assar. Model driven requirements engineering mapping the field and beyond. In *Model Driven Requirement Engineering Workshop MoDRE*, 2014. https://dx.doi.org/10.1109/MoDRE.2014.6890820.

[2] U. Aßmann, S. Zschaler, and G. Wagner. Ontologies, meta-models and the model-driven paradigm. In *Ontologies for Software Engineering and Software Technology*. Springer Berlin Heidelberg, 2010. https://dx.doi.org/10.1007/3-540-34518-3_9.

[3] S. Benkhaled, M. Hemam, M. Djezzar, and M. Maimour. An ontology – based contextual approach for cross-domain applications in Internet of Things. *Informatica An International Journal of Computing and Informatics*, 2022. https://doi.org/10.31449/inf.v46i5.3627.

[4] C. H. Bernabe, V. E. S. Souza, R. de Almeida Falbo, R. S. S. Guizzardi, and C. Silva. GORO 2.0: Evolving an ontology for goal-oriented requirements engineering. In *Advances in Conceptual Modeling ER*, 2019. https://dx.doi.org/10.1007/978-3-030-34146-6_15.

[5] M. Bettaz. Implementing OntoUML Models with OntoObject- Z Specifications: A Proof of Concept Relying on a Partial Ontology for VLANs. In *14th International Conference on Simulation and Modeling Methodologies, Technologies and Applications, SIMULTECH 2024*. SciTePRESS - Science and Technology Publications, Lda, 2024. https://dx.doi.org/10.5220/0012854500003758.

[6] M. Bettaz and M. Maouche. Towards a New Ontology-based Descriptive Language: OntoObject-Z. In *International Conference on Contemporary Computing and Informatics (IC3I)*. IEEE, 2023. https://dx.doi.org/10.1109/IC3I59117.2023.10397921.

[7] B. F. B. Braga, J. P. A. Almeida, G. Guizzardi, and A. B. Benevides. Transforming OntoUML into Alloy: towards conceptual model validation using a lightweight formal method. *Innovations in Systems and Software Engineering*, 2010. https://dx.doi.org/10.1007/s11334-009-0120-5.

[8] CCMI. OpenPonk platform. https://ccmi.fit.cvut.cz/tools/openponk/, 2023.

CCMI Research Group, Faculty of Information Technology, Czech Technical University in Prague.

[9] B. Costa, P. F. Pires, and F. C. Delicato. Modeling SOA-based IoT Applications with SoaML4IoT. In *World Forum on Internet of Things (WF-IoT)*, 2019. https://dx.doi.org/10.1109/WF-IoT.2019.8767218.

[10] I. C. Costa and J. M. P. de Oliveira. GO4SOA: Goal-oriented modeling for soa. In *International Conference on Web Information Systems and Technologies*, 2016. https://dx.doi.org/10.5220/0005800902470254.

[11] L. O. B. da Silva Santos, G. Guizzardi, and R. S. S. Guizzardi. GSO: Designing a well-founded service ontology to support dynamic service discovery and composition. In *Enterprise Distributed Object Computing (EDOC)*, 2009. https://dx.doi.org/10.1109/EDOCW.2009.5332016.

[12] D. Dermeval, J. Vilela, I. I. Bittencourt, J. Castro, S. Isotani, P. Brito, and A. Silva. Applications of ontologies in requirements engineering: a systematic review of the literature. *Requirements Engineering*, 2016. https://dx.doi.org/10.1007/s00766-015-0222-6.

[13] C. Diamantini, A. Freddi, S. Longhi, D. Potena, and E. Storti. A goal-oriented, ontology-based methodology to support the design of AAL environments. *Expert Systems With Applications*, 2016. https://dx.doi.org/10.1016/j.eswa.2016.07.032.

[14] B. Elvesæter, C. Carrez, P. Mohagheghi, A.-J. Berre, S. G. Johnsen, and A. Solberg. Model-driven service engineering with soaml. In *Service Engineering Book*. Springer, 2011. https://dx.doi.org/10.1007/978-3-7091-0415-6_2.

[15] S. J. T. Fotso, M. Frappier, R. Laleau, A. Mammar, and M. Leuschel. Formalisation of SysML/KAOS Goal assignments with b system component decompositions. In *Integrated Formal Methods (IFM)*, 2018. https://dx.doi.org/10.1007/978-3-7091-0415-6_210.1007/978-3-319-98938-9_22.

[16] X. Franch, L. López, C. Cares, and D. Colomer. The i* framework for goal-oriented modeling. In *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*. Springer, 2016. https://dx.doi.org/10.1007/978-3-319-39417-6_22.

[17] G. Giancarlo, B. B. Alessander, F. Claudenir, P. Daniele, A. J. Paulo, and P. S. Tiagoa. UFO: Unified foundational ontology. *Applied Ontology*, 2022. https://dx.doi.org/10.3233/AO-210256.

[18] N. Guarino. Formal ontologies and information systems. In *Formal Ontology in Information Systems (FOIS)*. IOS Press, 1998.

[19] G. Guizzardi, R. de Almeida Falbo, and R. Guizzardi. Grounding software domain ontologies in the unified foundational ontology (UFO): The case of the ODE software process ontology. In *Conferencia Iberoamericana de Software Engineering*, 2008.

[20] G. Guizzardi and G. Wagner. Using the unified foundational ontology (UFO) as a foundation for general conceptual modeling languages. In *Theory and Applications of Ontology: Computer Applications*. Springer, 2010. https://dx.doi.org/10.1007/978-90-481-8847-5_8.

[21] R. S. Guizzardi and G. Guizzardi. Applying the UFO ontology to design an agent-oriented engineering language. In *Conceptual Modeling ER*, 2013. https://doi.org/10.1007/978-3-642-15576-5_16.

[22] R. S. S. Guizzardi and G. Guizzardi. Applying the UFO ontology to design an agent-oriented engineering language. In *Advances in Databases and Information Systems (ADBIS)*, 2014. https://doi.org/10.1007/978-3-642-15576-5_16.

[23] R. S. S. Guizzardi, G. Guizzardi, A. Perini, and J. Mylopoulos. Towards an ontological account of agent-oriented goals. In *Software Engineering for Large-scale Multi-Agent Systems (SELMAS)*, 2006. https://dx.doi.org/10.1007/978-3-540-73131-3_9.

[24] K. Hinkelmann, E. Laurenzi, A. Martin, and B. Thönssen. Ontology-based metamodeling. In *Business Information Systems and Technology 4.0*. Springer, 2018. https://dx.doi.org/10.1007/978-3-319-74322-6_12.

[25] L. Kadakolmath and U. D. Ramu. Goal-oriented modeling of an urban subway control system using KAOS. *Indonesian Journal of Computer Science (IJCS)*, 2023. https://doi.org/10.33022/ijcs.v12i3.3239.

[26] D. Man. Ontologies in computer science. *DIDACTICA MATHEMATICA*, 31(1):43–46, 2013.

[27] R. Matulevicius, P. Heymans, and A. L. Opdahl. Ontological analysis of KAOS using separation of reference. In *Contemporary Issues in Database Design and Information Systems Development*. IGI Global, 2007. https://doi.org/10.4018/978-1-59904-289-3.ch002.

[28] J. C. Nardi, J. P. A. Almeida, P. H. A. da Silva, and G. Guizzardi. An ontology-based diagnosis of mainstream service modeling languages. In *International Enterprise Distributed Object Computing*

Conference (EDOC), 2019. https://doi.org/10.1109/EDOC.2019.00023.

[29] J. C. Nardi, R. de Almeida Falbo, J. P. A. Almeida, G. Guizzardi, L. F. Pires, M. J. van Sinderen, and N. Guarino. Towards a commitment-based reference ontology for services. In *Enterprise Distributed Object Computing (EDOC)*, 2013. https://doi.org/10.1109/EDOC.2013.28.

[30] J. C. Nardi, R. de Almeida Falbo, J. P. A. Almeida, G. Guizzardi, L. F. Pires, M. J. van Sinderena, N. Guarino, and C. M. Fonseca. A commitment-based reference ontology for services. *Information Systems*, 2015. https://doi.org/10.1016/j.is.2015.01.012.

[31] NEMO. Goal oriented requirements ontology (GORO). https://dev.nemo.inf.ufes.br/seon/GORO.html. Research Group.

[32] J. C. Nwokeji, T. Clark, and B. S. Barn. Towards a comprehensive meta-model for KAOS. In *Model-Driven Requirements Engineering (MoDRE)*, 2013. https://doi.org/10.1109/MoDRE.2013.6597261.

[33] OMG. Service oriented architecture modeling language (soaml) specification, v 1.0.1. https://www.omg.org/spec/SoaML/1.0.1/PDF. Object Management Group.

[34] M. A. Orellana, J. R. Silva, and E. L. Pellini. A model-based and goal-oriented approach for the conceptual design of smart grid services. *Machines*, 2021. https://doi.org/10.3390/machines9120370.

[35] I. Osman, S. B. Yahia, and G. Diallo. Ontology integration: Approaches and challenging issues. *Information Fusion*, 2021. https://doi.org/10.1016/j.inffus.2021.01.007.

[36] R. Pergl, T. P. Sales, and Z. Rybola. Towards OntoUML for software engineering: From domain ontology to implementation model. In *Model and Data Engineering (MEDI)*, 2013. https://doi.org/10.1007/978-3-642-41366-7_21.

[37] Y. Purnomo, R. Doss, N. B. Suhardi, and N. B. Kurniawan. Consolidating service engineering ontologies building service ontology from SOA modeling language (SoaML). *International Journal of Computer and Information Engineering*, 2018. https://doi.org/10.1109/ICITSI.2018.8695936.

[38] G. Reggio. A UML-based proposal for IoT system requirements specification. In *International Workshop on Modelling in Software Engineering (MiSE)*, 2018. https://doi.org/10.1145/3193954.3193956.

[39] C. Reginato, J. Salamon, and M. P. Barcellos. Ontology integration approaches: A systematic mapping. In *CEUR Worshops*. CEUR-WS.org, 2018.

[40] Z. Rybola and R. Pergl. Towards ontouml for software engineering: Transformation of rigid sortal types into relational databases. In *Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2016. https://dx.doi.org/10.2298/CSIS170109035R.

[41] F. M. Suchanek. OntoUML specification. https://ontouml.readthedocs.io/en/latest/, 2018.

[42] M. Tabatabaie, F. A. C. Polack, and R. F. Paige. KAOS-B A goal-oriented process model for EIS. In *International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (ICEIS)*, 2010. http://dx.doi.org/10.5220/0003016000400049.

[43] S. Tueno, R. Laleau, A. Mammar, and M. Frappier. Towards using ontologies for domain modeling within the SysML/KAOS approach. In *International Requirements Engineering Conference Workshops (REW)*, 2017. http://dx.doi.org/10.1109/REW.2017.22.

[44] A. van Lamsweerde. The KAOS meta-model: Ten years after. Technical report, Universite Catholique de Louvain, 1993.

[45] V. Werneck, A. de Padua Oliveira, and J. C. S. do Prado Leite. Comparing GORE frameworks: i-star and KAOS. In *Workshop on Requirement Engineering (WER)*, 2009.

[46] F. Zickert. Evaluation of the goal-oriented requirements engineering method kaos. In *Americas Conference on |Information Systems (AMCIS)*, 2010.