A Unified Trace Meta-Model for Alignment and Synchronization of BPMN and UML Models

Aljia Bouzidi[1], Nahla Haddar[2] and Kais Haddar[2]
[1]ISIMM University of Monastir, Monastir, Tunisia
[2]University of Sfax, Sfax, Tunisia
E-mail: aljia.bouzidi@gmail.com, nahla_haddar@yahoo.fr, kais.haddar@yahoo.fr

Keywords: Traceability, synchronization alignment, use case diagram, class diagram, MVC, BPMN diagram, integration, transformation

Received: February 24, 2025

Organizations often face information system (IS) failures due to misalignment with business goals. Business process models (BPMs) play a crucial role in addressing this issue but are often developed independently of IS models (ISMs), resulting in non-interoperable systems. This paper proposes a traceability method to link BPMs and ISMs, bridging the gap between business and software domains.
We introduce a unified trace meta-model integrating BPMN elements with UML constructs (use cases and class diagrams) via traceability links. This meta-model is instantiated as the BPMNTraceISM diagram, ensuring seamless integration through bidirectional transformation models.
To validate our approach, we developed a graphical editor for BPMNTraceISM diagrams and implemented transformations using the ATLAS Transformation Language (ATL). A case study on a loan approval process demonstrates the method's effectiveness in aligning BPMN and UML elements, improving interoperability and model alignment across domains

Povzetek: Razvit je enoten sledilni meta-model, ki povezuje elemente BPMN in UML (diagram primerov uporabe, razredov po vzorcu MVC) za uskladitev poslovnih procesov in informacijskih sistemov, ki ga validirajo z grafičnim urejevalnikom in transformacijami ATL.

1 Introduction

In the software engineering field, Business Process Models (BPMs) playb an increasingly central role in the development and continued management of software systems. Therefore, it is crucial to have Information System Models (ISMs) that tackle BPMs. modelling. However, these models are mostly expressed using different modeling languages, and only a few information systems (IS) are developed with explicit consideration of the business processes they are supposed to support. This separation causes gaps between business and IS models. Thus, a methodology is needed to examine the gap between BPMs and ISMs, and keep them aligned even as they evolve. Traceability in software development proves its ability to associate overlapping artefacts of heterogeneous models (for example, business models, requirements, uses cases, design models), improve project results by helping designers and other stakeholders with common tasks such as analysis of change impacts, etc. Thereby creating an explicit traceability model that is not a standalone guideline, but it has significant benefits in terms of quality, automation, and consistency. Although creating it is not a trivial task, an explicit traceability model remains a reference for a consistent definition of typed traceability links between heterogeneous model concepts, helping to ensure their alignment and coevolution.

In our previous work presented in [1], we proposed a relevant explicit traceability model and defined it using the integration mechanism. Indeed, we propose a requirements engineering method that works at both the meta-model and model levels, establishing traceability between BPMs and ISMs to bridge the gap between business modeling and requirements elicitation.modeling. This method is deliberately influenced by the Object Management Group (OMG) specifications. Particular attention is given to UML use case models [2] as the most commonly used way to elicit software needs and BPMN [3] as the most widely used language to specify the business process model (BPMs). Indeed, in [1], we firstly defined a unified trace meta-model of the BPMN and the UML use case models in the form of an integrated single meta-model. It defines also traceability links between interrelated concepts to correlate overlapped concepts as new modeling concepts. This meta-model is then instantiated in the form of a new diagram that we called BPSUC (Business Process Supported Use Cases). This new diagram permits business teams and requirements design teams to work together within the same model, and allows specifying trace links graphically.

The practical benefits of the proposed method lie in its ability to bridge the gap between business process management (BPM) and software systems development. In the context of Business Process Models (BPMs) and Information System Models (ISMs), this method enables seamless integration and traceability across heterogeneous models, which is crucial for ensuring their alignment and coevolution. By establishing clear and accurate traceability links between BPMs, UML use case diagrams, and class diagrams, our method enhances communication and collaboration among business analysts, software engineers, and stakeholders, ensuring that software systems are developed with a clear understanding of the business processes they aim to support. Furthermore, the integration of these models, coupled with the explicit traceability model, provides several practical advantages, including improving change impact analysis, enhancing automation, and maintaining consistency throughout the system lifecycle. These benefits are particularly significant in dynamic environments where both business processes and software systems evolve frequently. Thus, the method not only improves the quality of the development process but also provides a robust framework for aligning business and software models, ensuring their cohesive adaptation to changing requirements and system developments.

This paper enriches and extends our work presented in [1]. The enrichment involves adding class diagram concepts structured according to the MVC pattern. Our intervention considers both the meta-model and the model levels. Hence, in the integrated trace meta-model proposed in [1], we add new modeling concepts to express trace links between the class diagram, use case diagram, and the BPMN concepts. Class diagram concepts that have no corresponding concepts are also included in the integrated trace meta-model. Proposed traceability concepts and class diagram concepts are instantiated in the BPSUC diagram. Accordingly, BPSUC now enables the design of class diagram elements and the proposed traceability concepts combined with their corresponding BPMN and use case diagram artefacts.

We validate our theoretical method by implementing a visual modeling tool to support the enriched integrated trace meta-model and the new diagram supplemented with class diagram elements.

The rest of this paper is organized as follows: Section 2 is dedicated to discussing related work. In Section 3, we give an overview of the method presented in [1]. Section 4 is devoted to explaining our contributions. Section 5 and section 6 are dedicated to demonstrating the feasibility of our proposal in practice and through a topical case study. In section 7, we evaluate and discuss our method. Finally, in section 8, we conclude the current work and we give some outlooks.

2 Related work

We classify related work into two groups based on the methodologies they have used to establish traceability between elements of heterogeneous models: (1) works that have proposed transformation models to define internal or implicit traceability models, and (2) approaches that defined external traceability models manually, basing on some mechanisms, such as model integration, model merge/ composition, UML profiles or matrices.

2.1 Traceability via transformation models

In the first category, existing implicit traceability models are commonly MDA-compliant approaches that define traceability through exogenous, endogenous, horizontal, or vertical transformation models. In these approaches, BPMN models are widely used to generate alternative models through different transformation model types. Among the various uses of BPMN models are: an exogenous-based transformation for the definition of mapping users/organizations' requirements with BPMN models [4]; a vertical transformation for the generation of artefacts between BPMN and user stories [5] and [6]; and the generation of UML models [7]; a horizontal and exogenous transformation for the generation of activity diagrams from BPMN [8] and [9]; and a vertical transformation of textual requirements into a BPMN model [10]. Some approaches define endogenous transformation between UML diagram elements to establish their traceability. For instance, the approach in [11] uses machine learning techniques to maintain traceability information between software models. Their focus is particularly on the requirements, analysis, and design models, which are specified by the UML language. To trace links between requirements documents and UML diagrams, several approaches use Natural Language Processing (NLP). For example, the approach in [12] uses a system requirement description expressed in natural language to extract the actors and the actions automatically.

The core benefit of defining implicit traceability is that it does not require supplemental effort because only one chain of transformation is sufficient to perform transformations in both directions. Moreover, it offers multiple trace links between generated artefacts. However, the identified trace links consider exclusively transformed artefacts. Moreover, the transformation chain is static and cannot be updated to obtain the required traces for such traceability scenarios.

2.2 Explicit traceability models

The second category includes approaches that define explicit traceability models separate from the source models. This category includes approaches that propose guidelines for creating traceability models. For instance, the author of [13] defines a method for guiding the establishment of traceability between the software requirements and the UML diagrams. This guideline has two main components: (i) a meta-model and (ii) a process step. The process step defines the detailed processes, the mapping of requirements to UML diagrams, and the types of requirements. Requirements can be classified according to their aspects. This classification can be carried out according to the type of requirement, which then requires the use of certain types of UML diagrams. However, this guideline focuses only on establishing traceability at the meta-model level. Moreover, the business field is not considered in this work. The authors of [14] propose a meta-model-based approach to create traceability links between different levels of the same system. Indeed, this approach focuses on defining traceability metamodel source code stored as an Abstract Syntax Tree (AST) and other possible artefacts such as requirements, test cases, etc. To show the identified trace links, authors develop an editor. Nevertheless, storing source code of a system as an AST can cause several problems such as the appearance of syntax errors in the source code, which leads to the loss of traceability links. There is other model-based research that aim to maintain traceability. For example, the research in [15] proposes a co-evolution of transformations based on the propagation of change. Its hypothesis is that knowledge of the evolution of meta-models can be disseminated by decisions aimed at driving the co-evolution of transformations. To address particular cases, the authors present composition-based techniques that help developers compose resolutions that meet their needs. For the same purpose, the approach in [11] refers to machine learning techniques to introduce an approach called TRAIL (TRAceability lInk cLassifier). The training of the classifier is based on a training dataset that contains histories of existing traceability links between pairs of artefacts to output the trace link (related or unrelated) of any future pair of artefacts (new or already existing). Some other approaches define traceability models for eliciting requirements of complex systems [16] and [17]. Likewise, the authors of [19] base their work on deep learning techniques and propose a neural network architecture based on word embedding and Recurrent Neural Network (RNN) algorithms to predict trace links automatically. The output of this model is a vector that contains the semantic data of the artefact. Then, the trained model compares the semantic vectors of a pair of artefacts and predicts if they are related or unrelated. However, considering all meta-models from many different abstraction levels in one unified single traceability model is not a trivial task and can result in very complex models.

In [18], the authors propose an approach to promote traceability and synchronization of computational models in an Enterprise Architecture (EA), using meta-models, model traceability, and synchronization structures. The authors represent the meta-models of the EA at all abstraction levels (strategic, tactical, and operational). These levels are denoted within the integrated meta-model by three packages. Each package incorporates the core concepts of the level it represents. They integrate the three meta-models by adding alignment points between them. In addition, they define a traceability framework and a synchronization framework to support the analysis of the impact of organizational changes.

There are also studies on specific languages. For example, the approach in [20] uses Natural Language Processing techniques to define a framework for managing traceability between software artefacts. To demonstrate their work in practice, the authors develop a tool that supports traceability links between software models, including requirements and UML class diagrams, and the source code written in the Java programming language.

2.3 Identified gaps in existing works

Overall, existing works that define explicit traceability models are mostly focused on the meta-model level only and ignore the model level. Moreover, existing explicit models establish traceability either between software models expressed in UML diagrams at the same or different abstraction levels or between business model artefacts. However, none of the existing approaches have achieved successful results in establishing or maintaining traceability between BPMN models, UML use case models, and the UML class diagram.

The disadvantages of the proposed approaches stem from rigid relationship types that fail to adapt to the changing needs and practices of organizations. Furthermore, most of the proposed approaches define or use very generic traceability meta-models, capable of generating highly abstract trace models. In practice, there is no prescription for how to add customized tracing information or how to adapt a generic traceability meta-model to express valuable and context-specific traces. Concerning the approaches that focus on concrete modeling languages, to our knowledge, there is no approach that proposes an explicit trace model or meta-model between BPMN and UML models, even though they are the most popular standards for modeling business processes and automated information systems.

3 Background of our previous traceability method

The method presented in this paper is an extension of our previous work [1]. In this previous work, we have explored the advantages of defining an integrated traceability model to establish traceability between the BPMN and the UML use case models and ensure their coevolution once a change has occurred. This method acts at both the meta-model and model level, and it includes three core steps:

(i) First, we have defined an integrated trace meta-model that is a specification of traceability between the existing artefacts, while keeping them unchanged and independent. This integrated trace meta-model contains all the BPMN and the UML use case meta-model artefacts (meta-classes and associations) unified with new meta-classes and associations for expressing traceability links at the meta-model level. The integrated trace meta-model favors simplicity and uniformity because source meta-models are kept and unified with their traceability information in one unified meta-model.

(ii) Next, we instantiated the integrated trace meta-model at the model level. We represent it as a new diagram called

Business Process Modeling Notation Traces Use Case (BPSUC). This diagram also incorporates the BPMN and the UML use case elements together with traceability links, and allows designing BPMN and use case diagram artefacts, jointly. Moreover, visualizations and queries on traced elements together are straightforward, because business analysts and software designers are now able to work together on one integrated model. BPSUC can also be used to analyse change impacts and validate them before propagating them to the source models.

(iii) Finally, we defined bidirectional transformation models between the BPSUC diagram and the source models (BPMN and the use case models) to ensure the coevolution of the origin models.

3.1 Integrated trace meta-model

In our previous work presented in [1], a unified trace meta-model is proposed based on a semantic mapping of pairs of BPMN and use case meta-model artefacts. The definition of this meta-model follows the following scenario: For each pair of overlapped BPMN and UML use case concepts, we add a new modeling concept that can be either a link, such as an association, a composition or an inheritance, or a new meta-class. Each trace link represented by a new meta-class is associated with the pair artefacts it specifies, generally, by an inheritance relationship.

Table 1 summarizes mappings between the BPMN model (first column) and the use case diagram concepts (second column), and the corresponding new meta-classes (third column) that are associated with them in the integrated meta-model (the full mapping and its explanation is available in [1]). To validate the proposed mapping further, we have conducted additional evaluations across a variety of BPMN and UML diagram scenarios.

This expansion includes not only the core BPMN and UML elements such as activities, actors, and use cases but also more complex diagrams, such as:

- BPMN Models: Event-driven processes, process variants, and sub-processes with different complexity levels, such as loan approval and inventory management systems.
- UML Diagrams: Class diagrams, including inheritance and association relationships, and more sophisticated use case diagrams representing different business functions, such as order fulfilment, customer support, and system maintenance.

These diverse scenarios have allowed us to assess how well the mapping between BPMN, UML use case, and class diagrams holds up in real-world business process and system modeling. By applying the proposed traceability method to these varied scenarios, we demonstrate the scalability and robustness of our meta-model. We have also provided examples where the mapping effectively handles the integration of different BPMN and UML model types, ensuring traceability between the business and software models.

Table 1: Mapping of BPMN, use case and the trace meta-model concepts

Use Case Concept	BPMN concept	Meta-model concept
Package	Empty Lane (a lane including other sub-lanes)	Organisation Unit Package
Actor	Non empty Lane (that does not contain other sub-lanes)	Organization Unit Actor
Use case	Fragment represented by a sequence of BPMN artefacts that is performed by the same role and manipulates the same item aware element (business object, input data, data store, data state) d	UCsF
Extends	Exclusive Gateway between two different fragments	Exclusive Gateway, Extends
Association	Fragment within the lowest nesting level of sub-lanes	Association
Includes	Redundant Fragment (that appears multiple Times in the BPMN model)	Fragment that appears multiple times, Includes
Extends	Inclusive Gateway between two different fragments	Inclusive Gateway, Extends
Extension Point	Condition of sequence Flow + the name of the fragment that represents to the extending use case	Extension Point

The integrated meta-model is depicted in Figure 1). In order to be able to read it, we have presented in this Figure only the core artefacts of the source meta-model (use case meta-model and BPMN met-model) and all the trace links (meta-classes and associations). Dark grey meta-classes represent new meta-classes; light grey meta-classes represent UML use case elements, white meta-classes represent BPMN elements, and black lines represent existing associations from the source meta-models The blue lines represent trace relationships, providing the foundational traceability between BPMN and UML elements, as further detailed in [1] and [22].

- *Organizational-Unit-Package*

In BPMN, a non-empty lane is a grouping element and therefore has the same meaning as a package in UML. Consequently, the Organizational-Unit-Package (OUPackage) is defined to trace the link between the pair BPMN non-empty lane and the use case package thereby defining an inheritance relationship between the new meta-class and this artefact pair.

– *Organizational-Unit-Actor*

In the proposed integrated trace meta-model of [1], we have defined a meta-class designated Organizational-Unit-Actor (OUActor). This new meta-class traces artefacts of the pair UML actors and BPMN empty lanes (i.e. lanes that do not have embedded lanes). That is, it unifies the properties of a lane and an actor, and combines them without changing their semantics there by defining the OUActor as specialization of the UML actor and the BPMN Lane pair. In this way, OUActor inherits properties of this pair of artefacts without updating their original semantics and structures. For example, in a loan approval process, the Loan Officer is represented in a BPMN diagram by a non-empty lane and in a UML use case diagram as an actor involved in the process. The OUActor meta-class links these representations, inheriting properties from both while preserving their original semantics. This ensures synchronization between BPMN and UML elements, offering a unified view of the Loan Officer's role across models.

– *Fragment*

A fragment is defined by [22] as "*a set of interrelated BPMN elements that has inputs and outputs, and which is executed by the same performer*". This artefact is specified in the unified trace meta-model as an instance of the meta-class Fragment (cf. Figure 1)). As a Fragment is just an activity that can contain other BPMN concepts such as tasks, events, gateways and sequence flows, we have aggregated a BPMN sub-process to a Fragment by creating an aggregation relationship called *fragments,* between the fragment and sub-process meta-classes in the integrated trace meta-model (cf. Figure 1)). Its cardinality is *1-** to point out that a sub-process should contain at least one fragment but it may incorporate more than one Fragment. In addition, we define a many-to-many reflexive association in the fragment to represent the fact that a fragment may be an aggregation of other fragments (cf. Figure 1)). Moreover, we create an association between the data object and the fragment (cf. Figure 1)) to associate each fragment with the objects it manipulates. The cardinality of this relationship is fixed to 1-* to indicate that each fragment manipulates at least one business object type, but it may manipulate more than one business object. Furthermore, we have defined an association called *organizationUA* between the meta-classes OU-Actor and fragment with a cardinality of 1-* to associate a fragment with its performer. For instance, tasks such as *Review Application*, *Assess Credit Score*, and *Approve Loan* in the BPMN Loan Approval Process are all performed by the Loan Officer. The Fragment aggregates these tasks into a cohesive group and connects them to the BPMN sub-process as well as the business objects (e.g., Loan Applications, Credit Scores) manipulated during the process. This provides clear traceability between tasks, business objects, and performers while maintaining logical consistency.

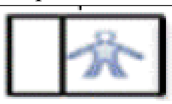
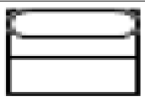
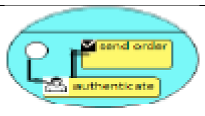
– *Use case supporting fragment*

In order to support business objectives, such a UML use case should be able to realize some business activities that are specified in the integrated trace meta-model by a Fragment. A separate specification of the use case and the fragment that is supposed to realise does not allow explicitly representing the semantic links between them. To do this, we have defined the integrated trace meta-model presented in [1], which introduces a new meta-class that we designate *Use Case supporting Fragment (UCsF)* This new meta-class is defined as a specialization of a UML use case in order to inherit all its properties without updating its initial meaning.

3.2 BPSUC diagram

To allow modeling the artefacts of the proposed integrated trace meta-model, we have instantiated it in the form of an integrated trace model, in our previous work [1]. We represent it as a new diagram that we have called BPMN Supporting Use Case model (BPSUC).

Table 2: Notation of the traceability artefacts

Meta-model concept	Graphical notation
OU-Actor	
OU- Package	
Use Case supporting Fragment (UCsF)	

For each concept, we have provided a graphical notation as follows: We have introduced new notations to the proposed new meta-classes UCsF, Organization Unit Package, and Organization Unit Actor. These notations are inspired by and extended from the icons of the pair of artefacts they represent. The inspiration ensures that experienced business and system designers are comfortable using the BPSUC diagram. Each concept originates form UML use case and BPMN models[1], and retains its original notation. In the BPSUC diagram, the Fragment is instantiated as a specific activity within the Loan Approval Process, linking BPMN tasks to the Loan Officer (OUActor) and business objects like the Loan Application. Additionally, the Organization Unit Package (OUPackage) meta-class is used to trace relationships between BPMN lanes and UML packages. For example, functional areas such as Loan Review, Credit Assessment, and Loan Approval in the BPMN diagram are mapped to corresponding UML packages, en-

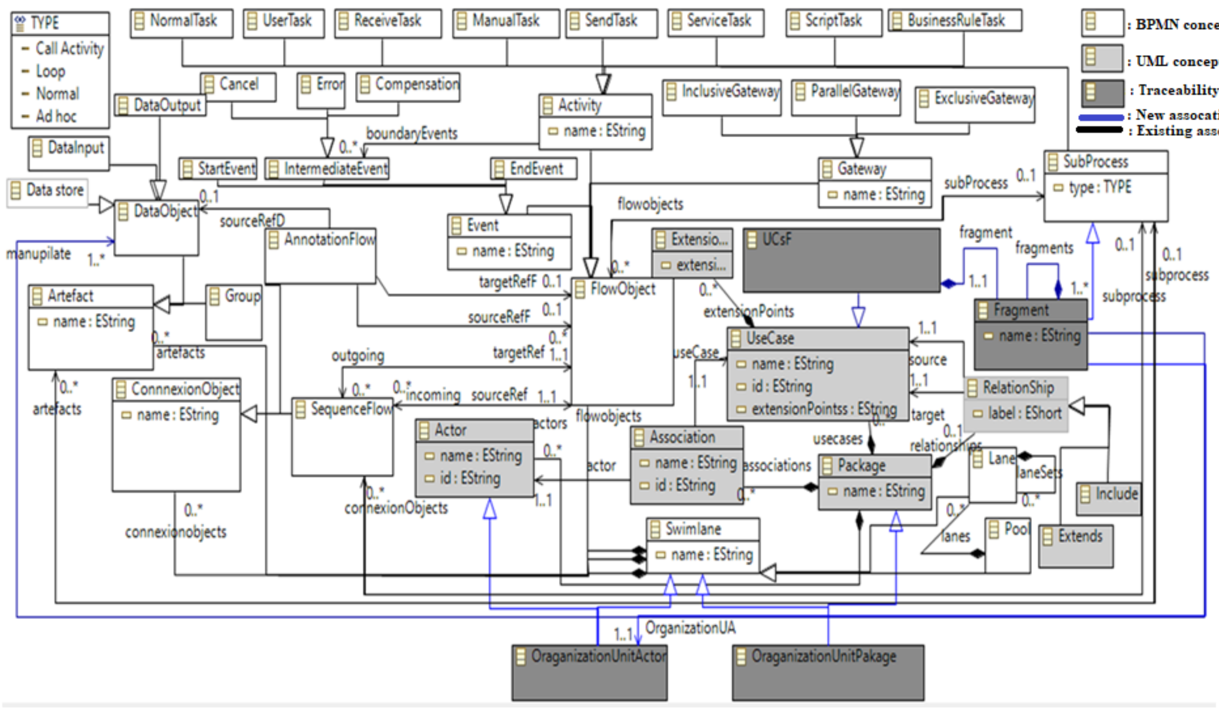


Figure 1: Traceability of BPMN and use case meta-model concepts

suring alignment and traceability between these functional areas and their UML counterparts.

Table 2 depicts the graphical notations of the new meta-classes Organisation Unit Actor, Organisation Unit Package and the UCsF.

4 Traceability method

The research work conducted in this paper is an extension and enhancement of our previous work presented in [1]. The extension consists of improving the integrated trace meta-model and the BPSUC diagram to include the artefacts of the UML class diagram structured according to the MVC design pattern. Our contribution aims not only to establish alignment but also to keep source models always aligned even if they evolve. The propagation of changes from our trace model to the source models is carried out through defined MDA model transformations. The former is explored to guarantee the coevolution of the BPMN and the UML models. Figure 2) demonstrates the background of our traceability method. In this section, we further explain how we extend and improve the integrated trace meta-model and the BPSUC diagram, as well as the rectifications made to them.

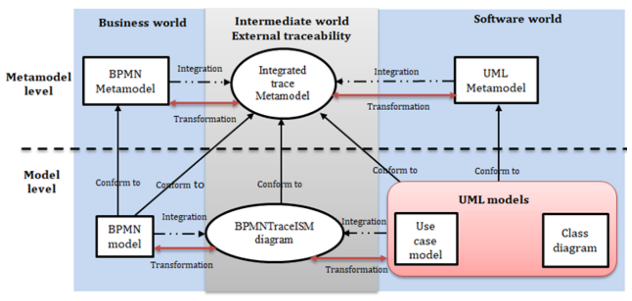


Figure 2: Background of the traceability method

4.1 Integrated trace meta-model improvement

Our first improvement of the integrated trace meta-model consists of defining an adequate strategy for defining its concepts and relationships between them. Indeed, we propose a methodology for defining the integrated trace meta-model which includes two main steps: (1) identifying overlapping concepts of BPMN and UML meta-models to define a relevant mapping between them, and (2) Defining an adequate methodology to link each pair of interrelated concepts, without changing their semantics. Thus, we propose to keep overlapping concepts and connect them either by a new concept, or a new relationship, which specifies the trace link between existing concepts at the meta-model level. Afterwards, we connect each pair of artefacts to the new concepts representing them through a generalization/specialization relationship. This relationship allows inheriting properties of both separated concepts as well as combining their usage without updating their initial semantics. Our second improvement consists of adding the class diagram meta-model artefacts to the previous version of the integrated trace meta-model.

By applying our meta-model construction, we need to identify adequate mappings between BPMN and UML class diagram concepts. In the literature mapping between BPMN and UML class diagram concepts is widely discussed. Among them, [23] defined model transformations from BPMN into UML class diagrams structured according to the MVC design pattern, and use case diagrams based on semantic mappings. For example, they propose mapping each BPMN empty lane (i.e. lane that does not include other lanes) into a class in the class diagram, and into an UML actor in the use case model. We reuse the defined semantic mappings in this approach to continue the definition of the integrated trace meta-model.

Table 3: Mapping of BPMN and class diagram meta-model concepts

BPMN concept	UML class diagram meta-model
Item Aware Element (Data object, Data store, Data input, Data output or Data state)	– Entity class – Association
Empty lane	– Entity class – View class – Control class – Association
Fragment	– View class – Control class
Exception event	– Exception class – Operation
Signal event	– Signal Class – Operation
Automated task *t* (business rule task, receive task, send task, user task, script task, service task) within a fragment	– Operation – Association
Item aware element type (single or collection) or Gateway or Loop task/ Rollback sequence flow	Cardinality of associations
Item aware element attached to an automated task *t* within a fragment *f*	Parameters of an operation
Conditional sequence Flow	Attribute

In Table 3, we summarize the semantic mapping between the BPMN meta-model artefacts and the class diagram meta-model artefacts from [23]. In this Table, the class diagram meta-model concepts are structured according to the MVC design pattern.

In contrast to the mapped concepts of the use case and the BPMN meta-model artefacts, the mapping between the interrelated concepts of UML class diagram and the BPMN meta-model is not tight, as shown in Table 3. Indeed, one UML concept may be represented by many BPMN concepts and vice versa. This is due mainly to the important degree of heterogeneity between the BPMN and the class diagram artefacts. Thus, our mapping is limited to defining associations instead of defining new traceability concepts, as we aim not to complicate our integrated trace meta-model, and therefore facilitate its readability while maintaining its consistency. The aforementioned trace meta-classes can also be reused to define BPMN- class diagram concept traceability.

The excerpt of the meta-model defined to trace the BPMN and the class diagram meta-models is presented in Figure.3. To ensure readability, Figure 3) depicts only the main artefacts of class diagram and BPMN meta-models, as well as the reused traceability concepts.

White meta-classes are BPMN concepts, orange meta-classes are UML class diagram meta-model concepts, khaki meta-classes denote UML class diagram concepts used for structuring the class diagram according to the MVC design pattern, while new concepts are specified by dark grey meta-classes. The blue associations represent the new trace links, while the black ones are the existing associations.

It is important to note that all the use case concepts, BPMN concepts, traceability links and existing associations defined in the previous extract of the integrated trace meta-model, which are not present in this extract remain valid.

In the excerptof Figure.3, each BPMN concept is associated with its corresponding concept in the class diagram meta-model. For example, we define a trace link called trace between the data object and the entity class to establish traceability between them.

The multiplicity of this association is 1..* to indicate that each item-aware element should represent exactly one entity class. Moreover, we define a trace link between the gateway and the property meta-classes as gateways can be indicators of association cardinalities. The multiplicity of this association is 0. On the other hand, UCsF is linked to the following meta-classes: class, ClassDIPackage, and association by composition . This means that a UCsF can include classes, associations, and packages. These associations mean that a UCsF is a use case that incorporates its supported class diagram elements, representing the supported fragment elements.

The cardinality of the composition association UCsF-ClassDIPackage is 3-* to indicate that an UCsF should incorporate at least three packages: View, Control, and Models, which represent the three parts of the MVC design pattern.

In addition, we define an association between OUActor and class to express that an actor in the integrated trace meta-model is represented as a class in the class diagram meta-model. Furthermore, in our integrated trace meta-model, a generalization/specialization relationship between the meta-classes OUPackage and ClassDIPackage is defined to point out that this trace meta-class inherits all properties of the *Package* meta-class.

4.2 BPSUC diagram improvement

In contrast to most existing approaches [11], [13], [14], [15], [16], [17], [18], [19], [20], [21], which focus only on the meta-model level, our traceability method includes both the meta-model and the model levels. Thus, the second step of our contribution is devoted to describing how the traceability of BPMN and UML artefacts is established at the model level. We have improved the proposed BPSUC diagram from [1], in which the BPSUC diagram features are limited to designing thoroughly the BPMN and the use case diagram artefacts, , combined with their traceability links, which already reflects its designation.

In this paper, we aim to enrich this diagram to incorporate class diagram elements combined with BPMN and use case diagram elements. The first thing we do is update the name of BPSUC to be in harmony with its newly supported features. The new designation we have chosen is *BPMNTraceISM* (**B**usiness **P**rocess **M**odel and **N**otation **Traces** **I**nformation **S**ystem **M**odels). *BPMNTraceISM* is an instantiation of the new version of the improved meta-model and forms a single unified model that combines the usage of UML elements including the use case diagram and class diagram elements, as well as the BPMN elements thoroughly. Thus, this diagram is now able to design elements and relationships of both UML use case and class diagrams, as well as BPMN models, concurrently. Moreover, it specifies the traceability information of the interrelated artefacts.

Each artefact in *BPMNTraceISM* diagram has its specific notation. Some of them retain the original notation (BPMN or UML notations), while the others have a new representation, which does not differ greatly from BPMN and UML notations.

4.2.1 BPMNTraceISM artefacts conserve their initial notations

The mappings on which we base the definition of the integrated trace meta-model comprises neither all the BPMN concepts nor all the UML concepts. This is due to the fact that, some BPMN artefacts do not have their corresponding UML artefacts, and vice versa. For example, the mapping does not define any UML concept representing a BPMN start event.

Even though, in a BPMNTraceISM diagram, it is possible to specify UML artefacts with no corresponding elements in BPMN. According to the mapping, many elements of UML are mapped to one BPMN element. Thus, the representation of these elements in UML diagrams requires grouping them. On the other hand, one UML element may be linked to many BPMN elements. For example, a data store in the BPMN diagram is transformed into (i) an association, (ii) an entity class, and (iii) an operation of a class, in the class diagram. In this situation, it is very difficult to represent the mapped elements in one unifying element. At the meta-model level, we have proposed associating each pair of these mapped concepts by an association instead of

Table 4: Graphical notations of overlapping elements of BPMNTraceISM diagram

BPMNTraceISM element	Graphic notation	BPMNTraceISM element	Graphic notation
Use case association	—	Signal event	⊚
Extends relationship	<<extends>> ←--------	Exclusive gateway	⊗
Includes relationship	<<includes>> --------→	Parallel gateway	⊕
Annotation flow	---	Inclusive gateway	◈
Start event	○	Data input	⇨ 🗋
End event	◯	Data output	➡ 🗋
Manual task	▭	Data store	⛁
Normal task	▭	Sequence flow	→
Error event	Ⓝ	Group	⬚
Cancel event	⊗	Entity class	<<Entity>>
Control class	<<Control>>	Generalization	⇧
Signal class	<<Signal>>	Aggregation association	◇
View class	<<Boundary>>	Composition association	◆
Exception class	<<Exception>>	Directed association	↑

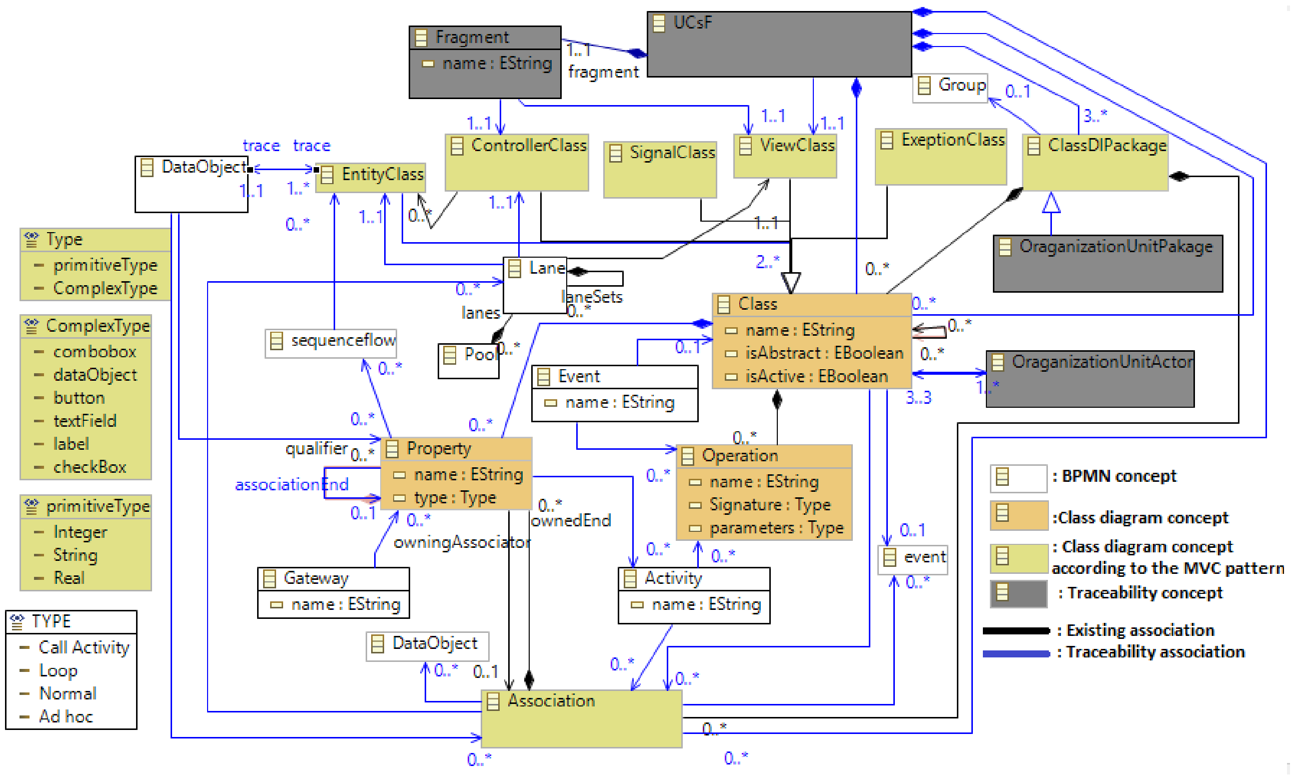


Figure 3: Traceability of the BPMN meta-model and the UML class diagram meta-model

defining new traceability meta-classes. At the model level, these artefacts are processed similarly to the non-mapped concepts, and retain their original notations in the BPMNTraceISM diagram. Table 4 outlines the graphical notations of the core artefacts of the BPMNTraceISM diagram that retain the initial notations. OUPackage and OUActor are new meta-classes defined by [1] to represent traceability links of BPMN and UML use case diagram elements. In the integrated trace meta-model, we did not reuse these meta-classes to define new associations. Thus, the instantiation of these meta-classes keeps the annotations provided in [1].

4.2.2 UCsF notation

In the previous version of the diagram *BPMNTraceISM* (in a BPSUC diagram), [1] states that a UCsF is a specialization of a use case and inherits its properties. Therefore, the graphical notation of UCsFs extends the graphical notation of a UML use case. Moreover, a UCsF has composition relationships to (i) a BPMN Fragment. To represent this trace link, graphically, [1] defines a compartment that incorporates the corresponding BPMN fragment.

In our integrated trace meta-model, we have defined composition relationships from a UCsF to some UML class diagram artefacts (cf. Figure 3)). Indeed, a UCsF should encapsulate classes, associations and packages, which correspond to its supported fragment. Accordingly, we propose to update the graphic notation of the UCsF. Thus, a UCsF should act as a complex symbol that describes concurrently BPMN elements and UML class diagram elements. In order to represent explicitly the different elements incorporated by a UCsF, the use case notation needs to be extended. Therefore, we adjust the UCsF notation by adding another compartment (cf. Figure 4) to encapsulate class diagram elements representing the components (classes, associations and packages) of the supported fragment. In order to avoid the complexity of this element, the designer can choose to hide or show each compartment. Figure 4) depicts the graphical notation of a UCsF, in which all compartments are hidden.

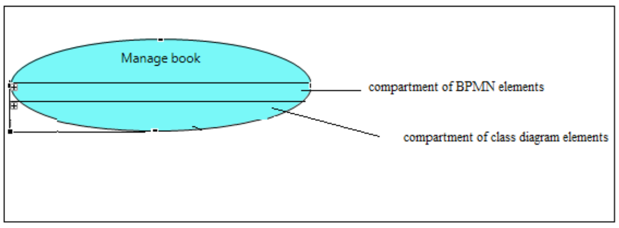

Manage book
compartment of BPMN elements
compartment of class diagram elements

Figure 4: UCsF notation

4.3 Change propagation improvement

Our traceability method aims to ensure the coevolution of the separated models when a change occurs either in the source models (BPMN model, use case model, and/or class diagram) or in the *BPMNTraceISM* diagram (cf. Figure 5)). To do this, we have improved the transformation model defined in [1] thereby including the class diagram concepts in the bi-directional transformation rules defined in [1] as two sets of transformation models (forward and backward transformation rules). They ensure the transformation between the BPMNTraceISM diagram, the BPMN, and the UML models that include a class diagram and a use case diagram using a semantic mapping between BPMN, *BPMN-TraceISM,* and UML elements derived from our integrated trace meta-model.

Each transformation model includes a set of well-defined transformation program or transformation rules (*Tab*) (*Tab* conforms to *MMt*) that transform source models (*Ma*) conforming to source meta-models (*MMa*) (noted Ma*/MMa*) to target models (*Mb*) confirming to target meta-models (*MMb*) (noted Mb*/MMb*) according to a mapping between the source and target model artefacts (noted $map_{(Ma,\, Mb)}$). Formally, we specify transformation models according to a function that we call *MtransF*

This function is formally written as follows:

$$MtransF\left(Ma/MMa,\ map_{(Ma,Mb)}\right) \overset{Tab/MMt}{\rightarrow} Mb/MMb \tag{1}$$

For example, consider the forward transformation rule R1 that transforms a UML package and a non-empty BPMN lane into a OU-Package in the BPMNTraceISM diagram. This transformation ensures that elements in the source models are correctly mapped to their counterparts in the BPMNTraceISM diagram, facilitating traceability across both business and software models.

The proposed bi-directional transformation models (backward and forward) ensure the coevolution of BPMN and UML models, as well as the coevolution of the source models (business model specified by a BPMN diagram and software models specified by a UML class diagram and an UML use case diagram) and the BPMNTraceISM diagram.

Formally, the forward and backward transformation model is specified as follows:

$$MtransF\left(Ma/MMa,\ map_{(Ma,Mb)}\right) \underset{\leftrightarrow}{forwardRules,\ backawardRules}\ (Mb/MMb) \tag{2}$$

The rest of this section will be devoted to providing more details on how we created the bidirectional transformation rules.

4.3.1 Forward transformation rules

We propose a forward transformation model (*Forward rules*) to produce automatically a BPMNTraceISM diagram ($M_{BPMNTraceISM}$) conforming to our integrated trace meta-model ($MM_{BPMNTraceISM}$) from the source models, namely a BPMN diagram (M_{BPMN}) conforming to the BPMN meta-model (MM_{BPMN}), a use case model (M_{UCM}), and a class diagram (M_{CD}) conforming to the UML meta-model (MM_{UML}). This transformation is carried out based on mappings between the new diagram, the BPMN and UML models.

The formal definition of our forward transformation rules is as follows:

$$\begin{gathered} MTransF\,((M_{BPMN}/MM_{BPMN}, M_{UCM}/MM_{UML}, \\ M_{UCD}/MM_{UML}, \\ (map_{(MUCM,MBPMNTraceISM)}, \\ map_{(UCD,MBPMNTraceISM)}, \\ map_{(MBPMN,MBPMNTraceISM)}) \\ \overset{forwardRules}{\rightarrow}\ M_{BPMNTraceISM}/MM_{BPMNTraceISM} \end{gathered} \tag{3}$$

There are two possible scenarios for producing the B*PMNTraceISM* elements based on the forward transformation rules:

The first scenario consists of applying a forward transformation rule (R_X) to derive trace modeling elements (*tre*) represented in the *BPMNTraceISM* diagram from a BPMN element (M_{BPMN}*!Element*) and a UML element (M_{UML}!Element). More precisely, a *OUActor, a OUPackage,* and a U*CsF* of the *BPMNTraceISM* diagram are generated from the BPMN and UML elements. Formally, these transformation rules are as follow:

$$\begin{gathered} MTransF_{tre}((M_{BPMN}!Element, M_{UML}!Element), \\ (map_{(MBPMN,MBPMNTraceISM)}, \\ map_{(MUML,MBPMNTraceISM)})) \\ \overset{R_X}{\rightarrow} M_{BPMNTraceISM}!tre \end{gathered} \tag{4}$$

For instance, we suppose that a forward transformation rule R_1 produces an OU-Package from a UML package and a BPMN non-empty lane. Formally, this rule can be written as follows:

$$\begin{gathered} MtransF_{OUPackage}((M_{BPMN}!lane, M_{UCM}!Package), \\ (map_{(MBPMN,MBPMNTraceISM)}, \\ map_{(MUCM,MBPMNTraceISM)})) \\ \overset{R_1}{\rightarrow} M_{BPMNTraceISM}!OUPackage \end{gathered} \tag{5}$$

The second scenario consists of generating unrelated elements (*ure*) from either, the UML models or the BPMN model only. Indeed, each concept in the BPMNTraceISM, that corresponds to a concept in the source models (BPMN model, a UML class diagram or a UML use case diagram) needs just its original model. In this case, the input of this transformation rule is either a BPMN model if this concept comes from BPMN or a UML model if its origin is the UML class diagram or the UML use case diagram. For example, the generation of a *manual task* in the BPMNTraceISM diagram requires the BPMN model only because a manual task does not have corresponding elements in the UML diagram.

This transformation rule is written as follows:

$$MTransF_{ure}(\ M_{BPMN}!manualTask,\ map_{(MBPMN,MBPMNTraceISM)} \overset{R(ManualTask)}{\rightarrow} M_{BPMNTraceISM}!manualTask \tag{6}$$

Let's illustrate how a change in the BPMN model (e.g., adding a new lane) is propagated into the BPMNTraceISM diagram. Assume that Rule R1 is applied to generate an OU-Package from the UML package and the BPMN lane. The transformation process involves the following steps:

1 The BPMN Lane and UML Package elements are identified.
2 Rule R1 is triggered, creating a corresponding OU-Package in the BPMNTraceISM diagram.
3 The OU-Package is updated within the BPMNTraceISM diagram, and the changes are synchronized with the BPMN and UML models.

4.3.2 Backward transformation rules

To have the opposite direction of the forward transformation rules, we have defined a backward transformation model. This means that the source elements of each forward transformation rule become the target elements of a backward transformation rule, and its target elements become the source elements of the backward transformation rule. Formally, backward transformation rules are written as follows:

$$\begin{gathered} MTransF(M_{BPMNTraceISM}/MM_{BPMNTraceISM}, \\ (map_{(MBPMNTraceISM,MUCM)}, \\ map_{(MBPMNTraceISM,MUCD)}, \\ map_{(MBPMNTraceISM,MBPMN)}) \\ \overset{backwardRules}{\rightarrow} \\ M_{BPMN}/MM_{BPMN}, M_{UCM}/MM_{UML}, \\ M_{CD}/MM_{UML}, M_{UCD}/MM_{UML} \end{gathered} \tag{7}$$

We use the same logic as in the forward transformation rules to define the reverse transformation rules. Therefore, each backward transformation rule of each pair of artefacts is defined according to the following formula:

MTransF$_{\text{tre}}$($M_{\text{BPMNTraceISM}}!tre$,

$$\begin{gathered} MTransF_{tre}\ (M_{BPMNTraceISM}!tre, \\ (map_{(MBPMNTraceISM,\ MBPMN\ )}, \\ map_{(MBPMNTraceISM,MUML\ )})) \\ \overset{Rx}{\rightarrow} (M_{BPMN}!Element, M_{UML}!Element) \end{gathered} \tag{8}$$

Non-overlapping artefact transformation rules are defined according to the formula below:

$$\begin{gathered} MTransF_{ure}(M_{BPMNTraceISM}!ure, \\ (map_{(MBPMNTraceISM,MBPMN)}, \\ map_{(MBPMNTraceISM,MUML)})) \\ \overset{Rx}{\rightarrow} (M_{BPMN}!Element, M_{UML}!Element) \end{gathered} \tag{9}$$

For example, when a change occurs in the BPMNTraceISM diagram, such as adding a new OU-Actor, the corresponding elements in the BPMN and UML models need to be updated. The backward transformation rule ensures that:

1 The OU-Actor is mapped to both the UML Actor in the use case diagram and a new BPMN lane in the BPMN model.
2 The changes are propagated back into the source models, maintaining alignment across the models.

4.3.3 Change propagation process

The bidirectional transformation rules allow propagating changes that occur in the source model into the target models. By applying these rules this approach enables the coevolution of the business and software models.

The change propagation process is carried out in two ways (cf. Figure 5): (1) by manually updating the source models (BPMN, UML, and UML use case diagram) , or (2) by designing the *BPMNTraceISM* diagram.

In the first case, software designers and business analysts separately and concurrently update the BPMN modeland consequently the use case diagram and/or the class diagram. For example, a software designer adds a new use case to the use case model, new classes responsible for realizing the new use case, and simultaneously, a business analyst changes the name of a lane in the BPMN model. A direct generation of the software models leads to the loss of changes made by the software designers. Additionally, to avoid unintentional updates, the impact of changes involved in a (business or UML) model needs to be analysed before propagating it to the target model. To tackle this problem, an intermediate step is required to make all updates made in the separate models. This step can be reached by executing our forward model (user task "Execute forward transformation rules"), which derives a *BPMNTraceISM* diagram from both UML and BPMN. Thus, all changes made on the BPMN and/or on the class and use case diagrams are considered in the derived *BPMNTraceISM* diagram.

In the second case, all updates made by business analysts and software designers are done in the unified trace model (*BPMNTraceISM* diagram) of being made in the BPMs and the ISMs. Using *BPMNTraceISM* overcomes the gap between the business analysts and software designers, and enables them to work together using the same model. Indeed, this diagram covers all business and software model elements and the traceability concepts of pairs of mapped artefacts. Any change involving a *BPMNTraceISM* element (*bp*) leads to the modification of the BPMN or/and UML model elements traced by *bp*. For example, the insertion of a new OU-Actor in the *BPMNTraceISM* diagram leads to the insertion of and a new UML actor in the UML use case diagram and a new BPMN lane in the BPMN model. *BPMNTraceISM* can act as a gateway allowing business analysts and software designers to work together to test,

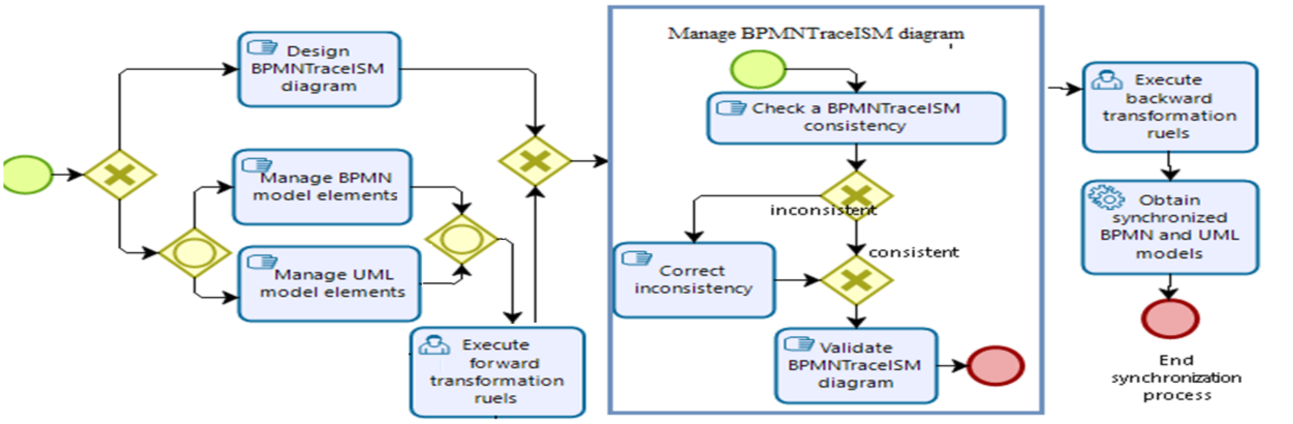

Design BPMNTraceISM diagram
Manage BPMN model elements
Manage UML model elements
Execute forward transformation ruels
Manage BPMNTraceISM diagram
Check a BPMNTraceISM consistency
inconsistent
consistent
Correct inconsistency
Validate BPMNTraceISM diagram
Execute backward transformation ruels
Obtain synchronized BPMN and UML models
End synchronization process

Figure 5: Synchronization process of BPMN and UML

analyse, and correct inconsistencies due to unwanted updates before propagating them to the source models (BPMN and UML models). In addition, this new diagram can be used to analyse and estimate the impact of changes made to business or system components or services. Until this step, although the *BPMNTraceISM* diagram is aware of the updates made by both business analysts and software designers, the source models are not aware neither the BPMNTraceISM diagram nor each other. Accordingly, propagating the modifications is an essential step to ensure the coevolution of the source models. We can do this easily by running our backward transformation model (user task ”execute backward transformation rules”). Once the backward transformation model is run, changes are propagated to BPMN and UML models and thus these models are aligned with each other.

5 Implementation

To use the proposed traceability approach, we implement a visual editor called Business Process model Traced with Information System Models (*BPTraceISM*).

Moreover, we develop a prototype called Business Process to Information System Models (BP2ISM) that provides significant practical support for the transformations involved in our traceability method. This prototype automates the suggested forward and backward transformation models between the business process and the ISMs on the one hand, and the BPMNTraceISM diagram on the other. These transformation models are automatically applied via transformation rules expressed in the ATL transformation language.

5.1 Visual editor implementation

To implement the *BPTraceISM* editor, we have used Eclipse EMF to implement the trace meta-model and Eclipse GMF to design the concrete syntax of the *BPMNTraceISM* diagram. Indeed, the modeling tool includes a graphical editor that conforms to the trace meta-model and enables concurrently seeing and managing trace relationships between the BPMN model, the use case, and the class diagram. *BPTraceISM* can be integrated within other modeling tools to enhance their modeling capabilities. To make our modeling tool available in any Eclipse environment without need to start an Eclipse runtime, we implement it as an Eclipse plug-in.

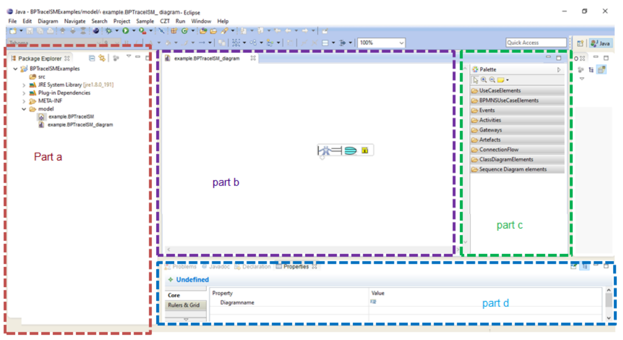


Figure 6: The environment of the *BPTraceISM* editor

The construction process of the *BPTraceISM* consists of two main phases: (1) the definition of the modeling tool, and (2) the definition of the plug-in that supports it. The first phase begins with the implementation of the trace meta-model using the ecore meta-modeling language. Then we build a toolbox for creating instances of the meta-model classes. In the second phase, we develop a feature that supports the modeling tool. Afterward, we construct an update site to ensure the portability of our plug-in and allow its installation via any Eclipse update manager.

BPTraceISM environment is composed of four main parts (cf. Figure 6): the project explorer containing an EMF project that includes *BPMNTraceISM* diagrams (part a), the modeling space (part b), the toolbox containing the graphical elements of a *BPMNTraceISM* diagram (part c), and the properties tab to edit the properties of an element selected in the modeling space (part d).

Figure 7) outlines a simple example of a BPMNTraceISM diagram created using the editor. The modeling space contains an OUActor called supplier associated with a UCsF called manage purchase order. In the business compartment of the UCsF *Manage purchase order*, we have a user task called *Accept purchase order*. In the class diagram compartment, we have four classes linked via undirected associations. Each class has a name and a stereotype. The boundary class *Manage purchase order* contains an operation called *acceptPurchaseOrder()*

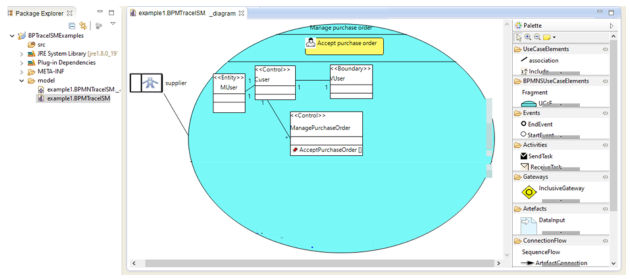


Figure 7: Example of a *BPMNTraceISM* diagram within the modeling tool

5.2 Prototype for the transformation models

BP2ISM is implemented within the Eclipse Modeling Framework (EMF) environment. It includes two components:

- *BPISM2BPMNTrISM*: It automates the forward transformation, which is the conversion of BPMN and UML models into BPMNTraceISM.
- *BPMNTrISM2BPISM*: It automates the backward transformations.

The transformation process requires tools, editors or plugins in order to specify source and target models. For this reason, tools are required to represent BPMN, UML and *BPMNTraceISM* diagrams, which serve as the source and target models of *BP2ISM* components. Because BPMN and UML are widely used standards, many plugins and tools have been created and certified to support them. We choose to employ internal plugins within EMF instead of existing plugins. As a result, we develop BPMN models with the Eclipse BPMN2 modeler plugin and UML use case and class diagrams with the UML designer plugin. Internal meta-models in these plugins closely adhere to OMG requirements. We incorporate these meta-models into the EMF environment for usage in the execution of our prototype components. In addition, we integrate the trace meta-model to visualize (backward transformation) or design (forward transformations) BPMTraceISM diagrams. We built the transformation rule sets in Atlas Transformation Language (ATL), which is provided as an internal EMF plugin.

BPISM2BPMNTrISM takes three files as input: (1) A file with the extension ".bpmn" that must conform to the BPMN2.0 meta-model, (2) two files with the extension ".uml" that must conform to the UML meta-model and contain the use case model and the class diagram. It generates as output a BPMNTraceISM diagram with the extension ".BPMNTraceISM".

BPMNTrISM2BPISM implements the backward transformations, i.e., the transformation rules from a *BPMNTraceISM* diagram into a BPMN and UML models. It takes as input a *BPMNTraceISM* diagram with the extension "*.BPMNTraceISM*". It generates as output three files: (1) A file with *".bpmn*" as extension. This file conforms to the BPMN2.0 meta-model, (2) two files with the extension *".uml*" that include the generated use case model and the class diagram.

6 Case study

We take a common business process model for online purchasing and selling to demonstrate the viability of our traceability method. The model is specified using BPMN2.0 (cf. Figure 8)) [23].

.

This business process begins when a customer selects a product to purchase and adds it to the basket, resulting in the creation of an online purchase order and the transmission of the order to the vendor. The customer has the option to cancel the purchase order before entering their personal information. Otherwise, they must fill in their personal information and submit an online purchase order to the stock management. When an online purchase order is received, the stock manager checks the warehouse for the availability of the ordered items to see if there are enough products to fulfil the order. If not, the restocking procedure is initiated to reorder raw materials and create the ordered products based on the supplier's catalogue. The restocking procedure can be performed as many times as necessary within the same business process instance. An extreme scenario occurs when raw materials are unavailable. If all items are available, sales validate the purchase order, generate an invoice, and begin collecting and packaging products for shipment. When sales receive payment and store the delivered order, the procedure is complete. Purchase order cancellation requests, however, can be made before the purchase order is verified. As a result, sales proceed with purchase order cancellation and a penalty charge to the buyer. In [23], the authors decompose the BPMN model of the case study into nine fragments (F1-F9) (cf. Figure 8)) based on their fragment definition (see [23] for further explanation). By applying their transformation rules, the approach from [23] allows generating the use case diagram and the class diagram from the case study BPMN model, which that is taken as the input model.

The *online purchasing and selling* BPMN model, use case model, and class diagram presented in [23] can be

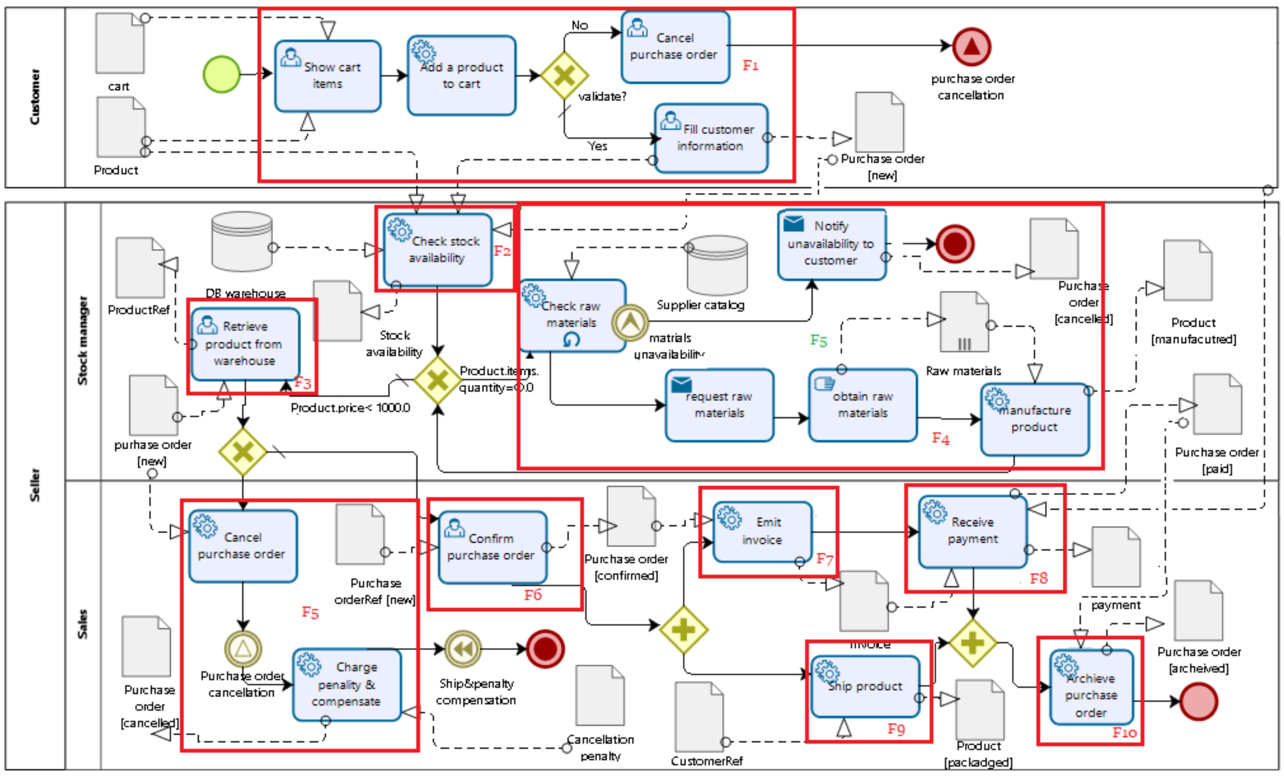


Figure 8: Online purchasing and selling in *BPMNTraceISM* diagram

combined and designed in a single unified model, namely the *BPMNTraceISM* diagram, by using the *BPTraceISM* editor. We would like to highlight that this diagram can be created manually by designers or automatically by running the BPISM2BPMNTrISM component. Figure 9) depicts the BPMNTraceISM diagram. Figure 9) shows how each fragment and its corresponding use case are merged and expressed as a UCsF. For example, we combine Fragment F1 with the use case "Manage preparing purchase order" to form the UCsF "Manage preparing purchase order". Each UCsF displays the traced BPMN elements and the corresponding class diagram elements. For each UCsF, the elements of the BPMN model are represented in the BPMN compartment, while the corresponding class diagram elements are represented in the class diagram compartment.

In Figure 9), these compartments are hidden in the UCsF *Cancel purchase order*, while the BPMN compartment is visible in all the other UCsFs.

In the UCsF *receive payment,* the BPMN compartment contains a service task called *receive payment,* a data object called *invoice,* and a data output called *purchase order [paid]* These elements are the BPMN elements of fragment *F8* Moreover, the class diagram compartment of UCsF *archive purchase order is* displayed and contains the class diagram elements, such as the *classes VarchivePurchaseOrder, CArchivePrchaseOrder, paid, archived, operations, attributes, etc* corresponding to fragment *F10*. Furthermore, an OUActor specifies each actor and the corresponding vacant lane. For example, the actor *Stock manager* and the empty lane *Stock manager* map to the OUActor *Stock manager*.

Assume that business analysts and system designers collaborate on the BPMNTraceISM diagram and update the business and system functionalities accordingly. Suppose they delete the UCsF *Manage preparing purchase order* and the OUActor *Customer* from the *BPMNTraceISM* diagram The UCsF *Manage preparing purchase order* is traced to the elements of fragment *F1*, the UML use case *Manage preparing purchase order,* as well as all class diagram elements derived from *F1* By deleting this UCsF, all its components are also removed from the *BPMNTraceISM* diagram Then, the change involved in the *BPMNTraceISM* diagram is propagated to the source models by executing the *BPMNTrISM2BPISM* tool. The output of this component is a BPMN model without the pool *Customer* or the fragment *F1*, a UML use case model that contains neither the use case *Manage preparing purchase order* nor the actor *Customer,* and a class diagram without elements corresponding to *F1*.

7 Evaluation results

7.1 Comparison with existing approaches

To evaluate the effectiveness of our traceability method, we compare it with existing traceability approaches based on

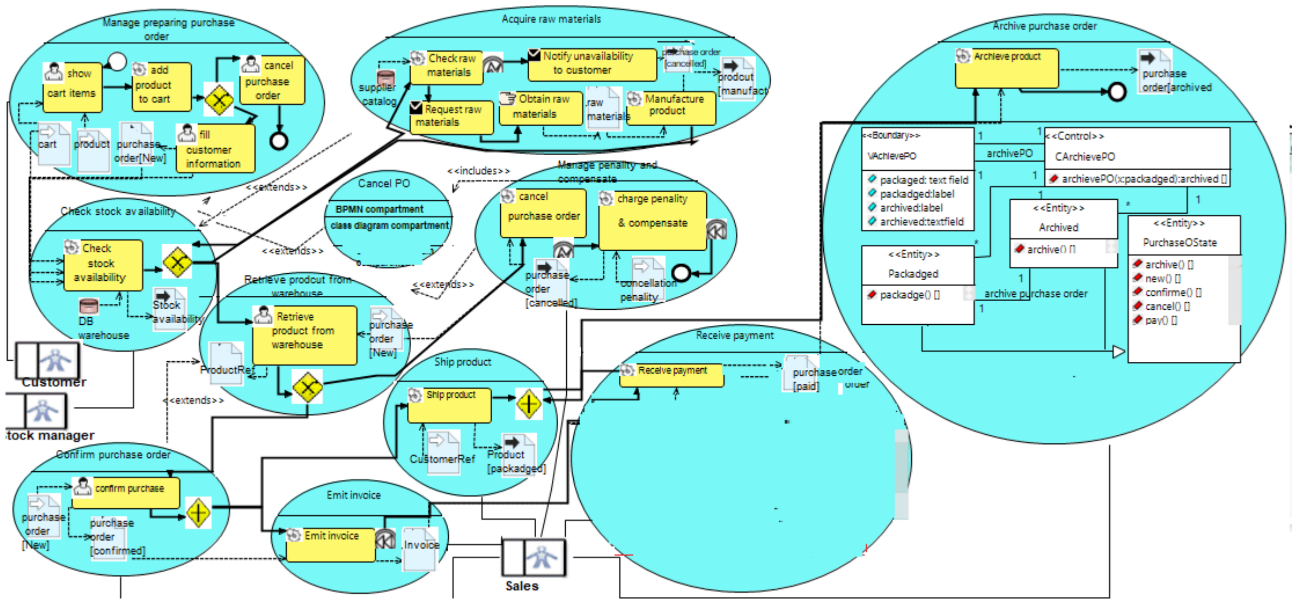


Figure 9: Online purchasing and selling in *BPMNTraceISM* diagram

defined evaluation criteria. These criteria include: (i) the proposal of a traceability approach at both the meta-model and model levels, (ii) explicit representation of relationship types between elements, (iii) graphical notation for trace links, and (iv) the consideration of both business and information system (IS) models.

Table 5 presents the results of this comparison, with rows listing the methods studied and columns representing the evaluation criteria. Cells are color-coded to show the extent to which each criterion is satisfied: dark grey indicates a criterion is not fully addressed, light grey represents partial satisfaction, and ”Y” stands for full satisfaction. Based on this comparison, we conclude that our approach is the only one that meets all evaluation criteria. Specifically, [17] and [20] are the only other works that consider both business and IS modeling, but they fall short in addressing the full range of traceability needs compared to our method. Furthermore, only a few approaches, such as those by [15], [18], and [20], take into account the functional and static views of IS models.

Our method is unique in that it does not require any extensions and works seamlessly with standard UML and BPMN tools, making it more adaptable and accessible. Additionally, our method provides rich software modeling-level artefacts, incorporating both static views (class diagrams) and functional views (use case models). The class diagrams are designed in accordance with the MVC pattern simplifying the prototyping process for developers.

When focusing on traceability, our method stands out because it provides traceability at both the meta-model and model levels, ensuring a unified view of BPMN and UML elements. In contrast, many approaches specify only traceability at the meta-model level, without providing a visual tool for combined model use. Additionally, our method introduces a graphical visualization for traceability links, enabling users to more easily trace and align elements across models. This graphical representation reduces analysis time, simplifies development, and minimizes the risk of misalignment.

Moreover, our method's assessment methodology is more comprehensive than most others in the field, which typically rely on simple case studies. We provide a fully implemented prototype of the transformation approach and an Eclipse plug-in for the traceability process, demonstrating the practicality and feasibility of our contributions through a relevant case study.

7.2 Shortcomings of our contribution

Despite the strengths of our traceability method, there are some limitations that need to be addressed. One notable drawback is that our evaluation was based on a single use case, which may not be sufficient to fully assess the accuracy and robustness of the method. To address this, we are conducting ongoing evaluations using more complex case studies, which will allow us to better validate the method's performance in different contexts. This extended evaluation will help ensure that the method is robust and adaptable across various scenarios, enhancing its overall credibility.

Additionally, our current transformation approach relies on forward and backward transformation rules, which require the recreation of all components, even if they have not been affected by changes. This process can lead to inefficiencies, especially when working with large or complex models. To overcome this issue, we plan to develop incremental transformations that will update only the components directly impacted by changes. This will improve efficiency and minimize unnecessary recalculations, ensur-

Table 5: Comparison of our contribution with approaches based on the external traceability practice

Approach	Construction model			Traceability approach					Assessment methodology
	Business field	software field		Meta level	Model level	Trace links	Graphic notation	BPM and ISM	
		functional	static						
[11]	N	P	P	Y	N	N	N	N	CS
[13]	N	CS		Y	N	N	N	N	CS
[14]	N	N	N	Y	N	N	N	N	tool
[15]	P	P	P	Y	N	N	N	N	
[16]	RM	CS		Y		N	N	N	CS
[17]	BPMN	N	N	Y	N	N	N	N	CS
[18]	p	p	p	Y	Y	N	N	Y	N
[19]	RM	N	N	Y	N	Y	N	N	N
[20]	N	N	CD	Y	N	N	N	N	N
[21]	BPMN	N	N	Y	Y	N	Y	N	T
Our contribution	BPMN2	UC	CCD	Y	Y	Y	Y	Y	CS &T

Legend: Y:Yes N: No CS: case study RM: requirement model T:tool/editor CS: Complex systems CCD:conception class diagram UC: uqe case diagram.

ing faster and more resource-efficient updates.

While we have explored the use of traceability information to keep BPMN and UML models aligned, the analysis process is still manual. As a result, we aim to improve this by investigating the development of heuristics that could automatically detect modifications in the source models and suggest necessary adjustments to the corresponding elements. These heuristics would dynamically support developers, making the process of maintaining alignment between the models more efficient and less error-prone.

To address the limitations mentioned above, we propose a comprehensive roadmap for future improvements. This will involve extending the evaluation process with more complex case studies, enhancing the transformation approach to support incremental updates, and automating diagram analysis. In addition, the implementation of heuristic-based tools will enable the automatic detection of changes across models, improving traceability and ensuring consistency without requiring manual intervention. These advancements will significantly strengthen the method's capabilities, making it more robust and easier to apply in practical scenarios.

8 Conclusion

The work conducted in this paper fits within the context of model-based development of ISMs, their alignment and their

coevolution with BPMs. Indeed, we have used integration and model transformation methodologies to define a

traceability method oriented towards the development of (meta) model-based solutions, purposely influenced by the Object Management Group (OMG) specifications. Particular attention is paid to the BPMN and UML use case and class diagram models. Our traceability method acts at the meta-model and the model levels. Hence, (1) we first defined a unified trace meta-model that includes all the BPMN and the UML elements (use case and class diagram) and traceability links between interrelated elements. (2) Then, we defined integrated model conforms to the proposed trace meta-model. We defined it as a new diagram named *BPMNTraceISM* (BPMN Traces Information System Models). This diagram serves many purposes: it promotes the collaborative between business and software designers and allows them to work together using one single unified model. The joint representation of both BPMs and ISMs elements enables users to drill down and easily trace any business artefact to its corresponding software artefacts. (3) Finally, we defined a set of bidirectional model transformation rules between the BPMN and UML models, as well as the BPMNTraceISM diagram.

The rules are useful when a change propagation-based co-evolution is required to synchronize models after changes. To prove the feasibility of our traceability method in practice, we developed a modeling tool in the form of a plugin that can be integrated into the Eclipse platform. This tool is named *BPTraceISM* (Business Process model Trace with Information System Models) and allows designing and handling *BPMNTraceISM* diagrams in accordance with the proposed integrated trace meta-model. Additionally, we specified the set of bidirectional transformation rules using the ATL language and we implemented them as components of the BPM2ISM prototype. Furthermore, we applied the proposed approaches to a typical case study.

In future research, we look forward to optimizing our editor to support traceability and synchronization between BPMN models and other UML diagrams.

References

[1] Bouzidi, A., Haddar, N., Haddar, K. (2019). *Traceability and Synchronization Between BPMN and UML Use Case Models*, Ingénierie des Systèmes d'Information, Vol. 24, No. 2, pp. 215-228. https://DOI.org/10.18280/isi.240214.

[2] OMG UML Specification, O. A. (2017). *OMG Unified Modeling Language (OMG UML), Superstructure, V2*, Object Management Group, Vol. 70.

[3] OMG BPMN Specification. *Business Process Model and Notation* Available at: http://www.bpmn.org/. Accessed: 2023-01-31.

[4] Driss, M., Aljehani, A., Boulila, W., Ghandorh, H., Al-Sarem, M. (2020). *Servicing your requirements: An FCA and RCA-driven approach for semantic web services composition*, IEEE Access, Vol. 8, pp. 59326-59339. 10.1109/ACCESS.2020.2982592.

[5] Ghiffari, K. A., Fariqi, H., Rahmatullah, M. D., Zulfikarsyah, M. R., Evendi, M. R. S., Fathoni, T. A., Raharjana, I. K. (2023). *BPMN2 user story: Web application for generating user stories from BPMN*, In AIP Conference Proceedings, AIP Publishing LLC, Vol. 2554, No. 1, pp. 040003. https://DOI.org/10.1063/5.0103685.

[6] Raharjana, I. K., Aprillya, V., Zaman, B., Justitia, A., Fauzi, S. S. M. (2021). *Enhancing software feature extraction results using sentiment analysis to aid requirements reuse*, Computers, Vol. 10, No. 3, pp. 36. https://DOI.org/10.3390/computers10030036.

[7] Khlif, W., Elleuch, N., Alotabi, E., Ben-Abdallah, H. (2018). *Designing BP-IS Aligned Models: An MDA-based Transformation Methodology*. 10.5220/0006704302580266.

[8] Kharmoum, N., Retal, S., Rhazali, Y., Ziti, S., Omary, F. (2021). *A Disciplined Method to Generate UML2 Communication Diagrams Automatically From the Business Value Model, In Advancements in Model-Driven Architecture in Software Engineering*, IGI Global, pp. 218-237. 10.4018/978-1-7998-3661-2.ch012.

[9] Rahmoune, Y., Chaoui, A. (2022). *Automatic Bridge Between BPMN Models and UML Activity Diagrams Based on Graph Transformation*, Computer Science, Vol. 23, No. 3. 10.7494/csci.2022.23.3.4356.

[10] Ivanchikj, A., Serbout, S., Pautasso, C. (2020). *From Text to Visual BPMN Process Models: Design and Evaluation*, In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp. 229-239. https://DOI.org/10.1145/3365438.3410990.

[11] Mills, C., Escobar-Avila, J., Haiduc, S. (2018). *Automatic Traceability Maintenance via Machine Learning Classification*, In 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 369-380. 10.1109/ICSME.2018.00045.

[12] Al-Hroob, A., Imam, A. T., Al-Heisa, R. (2018). *The Use of Artificial Neural Networks for Extracting Actions and Actors from Requirements Documents*, Information and Software Technology, Vol. 101, pp. 1-15. https://DOI.org/10.1016/j.infsof.2018.04.010.

[13] Min, H. S. (2016). *Traceability Guideline for Software Requirements and UML Design*, In International Journal of Software Engineering and Knowledge Engineering, Vol. 26, No. 1, pp. 87-113. https://DOI.org/10.1142/S0218194016500054.

[14] Eyl, M., Reichmann, C., Müller-Glaser, K. (2017). *Traceability in a Fine-Grained Software Configuration Management System*, In Software Quality: Complexity and Challenges of Software Engineering in Emerging Technologies, 9th International Conference, SWQD 2017, Vienna, Austria, January 17-20, 2017, Springer International Publishing, pp. 15-29. 10.1007/978-3-319-49421-0_2.

[15] Khelladi, D. E., Kretschmer, R., Egyed, A. (2018). *Change Propagation-Based and Composition-Based Co-Evolution of Transformations with Evolving Meta-Models*, In Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp. 404-414. https://DOI.org/10.1145/3239372.3239380.

[16] de Carvalho, E. A., Gomes, J. O., Jatobá, A., da Silva, M. F., de Carvalho, P. V. R. (2021). *Employing Resilience Engineering in Eliciting Software Requirements for Complex Systems: Experiments with the Functional Resonance Analysis Method (FRAM)*, Cognition, Technology and Work, Vol. 23, pp. 65-83. https://DOI.org/10.1007/s10111-019-00620-0.

[17] Lopez-Arredondo, L. P., Perez, C. B., Villavicencio-Navarro, J., Mercado, K. E., Encinas, M., Inzunza-Mejia, P. (2020). *Reengineering of the Software Development Process in a Technology Services Company*, Business Process Management Journal, Vol. 26, No. 2, pp. 655-674. https://DOI.org/10.1108/BPMJ-06-2018-0155.

[18] Moreira, J. R. P., Maciel, R. S. P. (2017). *Towards a Models Traceability and Synchronization Approach of an Enterprise Architecture*, In SEKE, pp. 24-29. 10.1109/CBI.2019.00028.

[19] Guo, J., Cheng, J., Cleland-Huang, J. (2017). *Semantically Enhanced Software Traceability Using Deep*

Learning Techniques, In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), pp. 3-14. 10.1109/ICSE.2017.9.

[20] Swathine, K., Sumathi, N., Nadu, T. (2017). *Study on Requirement Engineering and Traceability Techniques in Software Artefacts*, In International Journal of Innovative Research in Computer and Communication Engineering, Vol. 5, No. 1. 10.1109/ICSRS.2017.8272863.

[21] Pavalkis, S., Nemuraite, L., Milevičienė, E. (2011). *Towards Traceability Meta-Model for Business Process Modeling Notation*, In Conference on e-Business, e-Services and e-Society, Springer, Berlin, Heidelberg, pp. 177-188. 10.1007/978-3-642-27260-8_14.

[22] Bouzidi, A., Haddar, N., Abdallah, M. B., Haddar, K. (2018). *Alignment of Business Processes and Requirements Through Model Integration*, In 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA), pp. 1-8, IEEE. 10.1109/AICCSA.2018.8612870.

[23] Bouzidi, A., Haddar, N. Z., Ben-Abdallah, M., Haddar, K. (2020). *Toward the Alignment and Traceability Between Business Process and Software Models*, In ICEIS, Vol. 23. 10.5220/0009004607010708.