Applied Computer Systems

doi: 10.1515/acss-2015-0011

2015/17

Problems and Solutions by Using the Integrated Domain Modeling Toolset

Kelly Verónica Fernández Céspedes, *Riga Technical University, Latvia*

Abstract – **To contribute to the analysis of tools that attempt to acquire a Computation Independent Model (CIM) from the domain system, the author explores the Integrated Domain Modeling toolset and explains how it automatically acquires a formal CIM from description of a business system in the form of textual business use cases. This paper recognizes the computation independent nature of a Topological Functioning Model and suggests it to be used as a CIM within a Model Driven Architecture. The author of this paper shares her experience of using the toolset and mentions several lessons learned during the usage process as well as her suggestions for improvements.**

Keywords – **Domain Modeling, Model Driven Architecture, Topological Functioning Model.**

I. Introduction

The domain knowledge about the business system and its environment exists sometimes in different documents in the form of natural language. In general, at the beginning of domain modeling, it is necessary to acquire domain knowledge, actual business system and its environment, i.e., there should be a simple but somewhat formal way to capture declarative knowledge (structure, concepts, relationships) as well as procedural knowledge (business processes).

Model Driven Architecture (MDA) is a software design approach for the development of software systems, which defines three layers of abstraction for a system analysis: Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM). Furthermore, the MDA is based on four-level architecture and the supporting standards: Meta-Object Facility (MOF), Unified Modeling Language (UML), and XML Metadata Interchange (XMI) [1].

As explained in [2], a Topological Functioning Model (TFM) is a modeling approach that uses a formal mathematical model to specify and analyze characteristics of a business system.

In the context of MDA, the TFM4MDA method is developed at Riga Technical University (RTU) and suggested in [3]. It allows system's TFM to be composed by the knowledge about the complex system that operates in the real world. This paper follows the TFM4MDA's suggestion of using the TFM as the CIM. Thus, it ensures acquiring a mathematically formal and transformable CIM.

Nowadays, there are a lot of domain modeling approaches, and some of them just specify the CIM at modeler's discretion. After that, it usually is extended by hand towards PIM enriching it with operational model elements; therefore, there is a semantic gap between a CIM and a PIM.

To cope with this issue, Šlihte in [4] states that the Integrated Domain Modeling (IDM) approach provides a formal means to define the CIM as well as to formally transform it to the PIM. In general, since supporting tools simplify the use of an approach, the IDM approach proposes a toolset whose functionalities and limitations will be explored in the following sections presented as lessons learned by using the Integrated Domain Modeling toolset.

This paper is structured as follows. Section II depicts a brief description of TFM. Section III describes the IDM approach as a domain modeling toolset. Section IV explains each of the components of the toolset and the experiences the authors have acquired when using them. The problems found and the possible solutions in the form of suggestions for improvements are explained in Section V. Finally, the authors present their conclusions in Section VI.

II. Topological Functioning Model in Brief

Theoretical foundations for the TFM were initially developed in 1969 by Janis Osis [5]. Since then it has been applied in numerous cases and extended for various problem domains. But since the introduction of OMG Model Driven Architecture, the TFM has also been successfully applied in the context of MDA. Some examples of its application are illustrated in [6], [7], [8], [9], [10] and [17].

The main advantage provided by the TFM is the possibility of analysis of topological properties, i.e., connectedness, closure, neighborhood and continuous mapping; as well as of the functional properties, i.e., cause-effect relation, cyclic structure, inputs and outputs, of the system to be modeled. The authors in [11] state that the domain model should be the cornerstone of software development – it is a design, documentation and a way of decreasing inconsistencies and over-budget costs.

As explained in [12], in the context of MDA, the TFM is mainly used to represent the computationally independent model of the system, i.e., it depicts the business domain model.

For a more elaborated explanation of TFM background and approach in general, the author refers to [13] and [14].

III. Integrated Domain Modeling Approach and Its Toolset

IDM toolset is positioned as one of the domain modeling tools and it is based on a solid basis of TFM and TFM4MDA approach. However, it is currently used as a prototype for academic purposes.

Moreover, the toolset complies with MDA standards; therefore, its implementation is based on Eclipse Modeling Framework [15], which is one of the frameworks for MDA. This can be highlighted as a plus because it ensures that the toolset is extendable and can be integrated with other modeling tools, becoming suitable to be considered part of the MDA life cycle.

IDM approach suggests using a common system analysis and ontology for capturing the domain knowledge and then transforms it into a corresponding domain model.

The users of the toolset are the knowledge engineers, who construct the ontology, and the system analysts that together with the business people construct business use cases.

Since the IDM approach is based on the TFM, the main goal of the toolset is to enable users to model the TFM, which ultimately is considered to be the CIM under the perspective of this approach.

Furthermore, this approach was thought as one of the improvements for implementing TFM4MDA steps for retrieving the TFM as the CIM.

TFM4MDA approach described in [3] proposes the following steps: 1) retrieving the system objects and functional features by analyzing the informal description of a system; 2) constructing a TFM topological space using the retrieved system objects and functional features; 3) constructing a TFM topological graph using its topological space; 4) verifying the functional requirements by mapping them to the corresponding functional features; 5) transforming the TFM to a UML (as a UML profile).

A. The Main Functionalities of the Integrated Domain Modeling

As explained in [4], an IDM toolset is the implementation of TFM4MDA and its main purpose is to enable users to automate the construction of the system domain as the TFM. To achieve it, the IDM approach supports the user in the following activities:

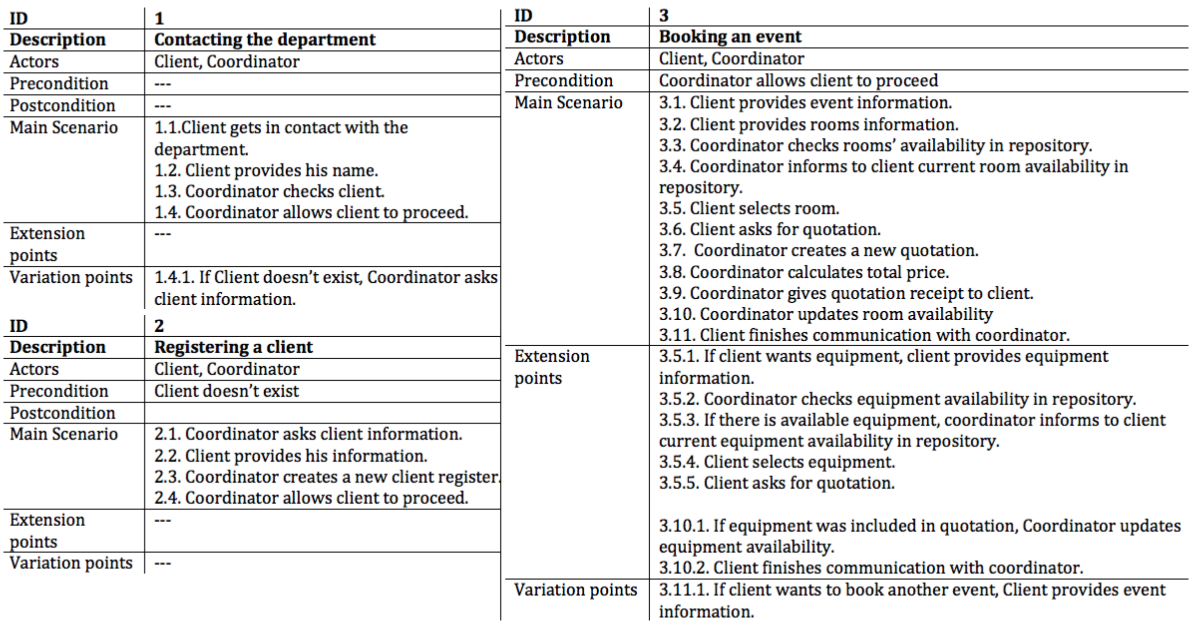
1. Constructing or reusing the existing domain ontology;
2. Developing use cases that describe the business processes of the domain.
3. Validating the use case model by using natural language processing and the domain ontology.
4. Automatically generating the CIM for this domain in the TFM form.
5. Allowing the user to further refine the TFM of the domain by adding the main functional cycle and logical relationships.
6. Validating the use case model against the corresponding meta-model.

Currently, the IDM toolset only supports the procedural knowledge and implements the steps from 2 till 6. Moreover, there is not automatic integration between ontology and the IDM toolset yet.

However, even though the IDM approach recommends ontology, it is not mandatory. This means that a user can just use the toolset without the support of ontology to generate the initial TFM and continue with CIM developments until a user is satisfied with the result.

The toolset is composed of: the Use Cases Editor, TFM Editor, TFM Diagram Tool and Use Cases to TFM Transformation.

This paper focuses on the first three listed tools for considering the Use Cases to TFM Transformation tool as part of the functionality provided by the Use Case Editor. Besides, this tool can only be launched just from the Use Case Editor. Furthermore, it basically fetches the functional features and topological relationships from business use cases, and then generates the TFM for a business domain.

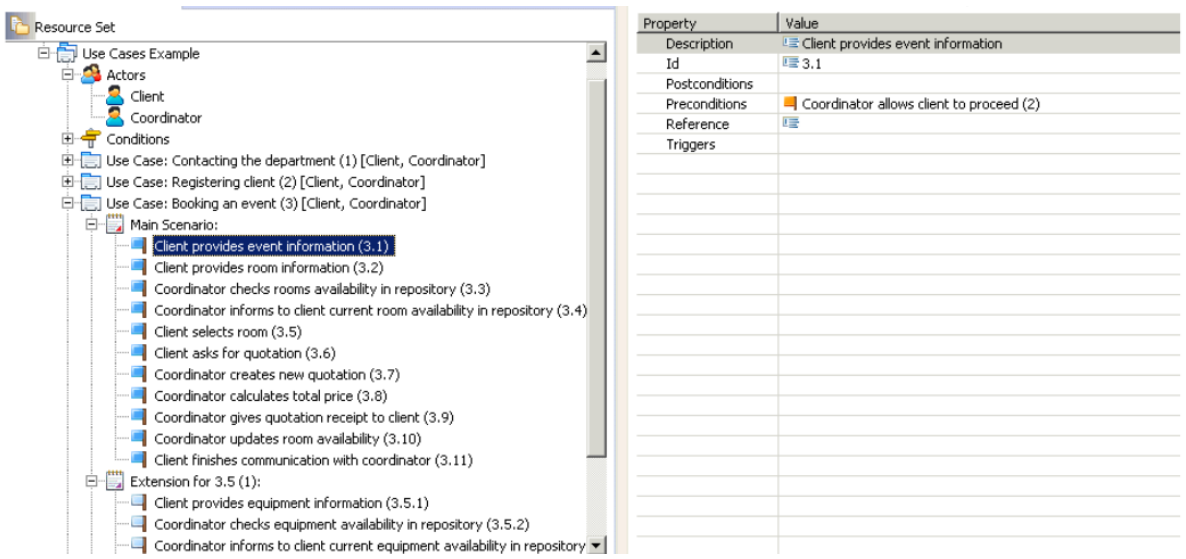


ID	1
Description	**Contacting the department**
Actors	Client, Coordinator
Precondition	---
Postcondition	---
Main Scenario	1.1.Client gets in contact with the department. 1.2. Client provides his name. 1.3. Coordinator checks client. 1.4. Coordinator allows client to proceed.
Extension points	---
Variation points	1.4.1. If Client doesn't exist, Coordinator asks client information.

ID	2
Description	**Registering a client**
Actors	Client, Coordinator
Precondition	Client doesn't exist
Postcondition	
Main Scenario	2.1. Coordinator asks client information. 2.2. Client provides his information. 2.3. Coordinator creates a new client register. 2.4. Coordinator allows client to proceed.
Extension points	---
Variation points	---

ID	3
Description	**Booking an event**
Actors	Client, Coordinator
Precondition	Coordinator allows client to proceed
Main Scenario	3.1. Client provides event information. 3.2. Client provides rooms information. 3.3. Coordinator checks rooms' availability in repository. 3.4. Coordinator informs to client current room availability in repository. 3.5. Client selects room. 3.6. Client asks for quotation. 3.7. Coordinator creates a new quotation. 3.8. Coordinator calculates total price. 3.9. Coordinator gives quotation receipt to client. 3.10. Coordinator updates room availability 3.11. Client finishes communication with coordinator.
Extension points	3.5.1. If client wants equipment, client provides equipment information. 3.5.2. Coordinator checks equipment availability in repository. 3.5.3. If there is available equipment, coordinator informs to client current equipment availability in repository. 3.5.4. Client selects equipment. 3.5.5. Client asks for quotation. 3.10.1. If equipment was included in quotation, Coordinator updates equipment availability. 3.10.2. Client finishes communication with coordinator.
Variation points	3.11.1. If client wants to book another event, Client provides event information.

Fig. 1. Specification of business use cases for room renting system for events.



Resource Set
- Use Cases Example
  - Actors
    - Client
    - Coordinator
  - Conditions
  - Use Case: Contacting the department (1) [Client, Coordinator]
  - Use Case: Registering client (2) [Client, Coordinator]
  - Use Case: Booking an event (3) [Client, Coordinator]
    - Main Scenario:
      - Client provides event information (3.1)
      - Client provides room information (3.2)
      - Coordinator checks rooms availability in repository (3.3)
      - Coordinator informs to client current room availability in repository (3.4)
      - Client selects room (3.5)
      - Client asks for quotation (3.6)
      - Coordinator creates new quotation (3.7)
      - Coordinator calculates total price (3.8)
      - Coordinator gives quotation receipt to client (3.9)
      - Coordinator updates room availability (3.10)
      - Client finishes communication with coordinator (3.11)
    - Extension for 3.5 (1):
      - Client provides equipment information (3.5.1)
      - Coordinator checks equipment availability in repository (3.5.2)
      - Coordinator informs to client current equipment availability in repository

Property	Value
Description	Client provides event information
Id	3.1
Postconditions	
Preconditions	Coordinator allows client to proceed (2)
Reference	
Triggers	

Fig. 2. General view of the Use Case Editor.

B. A Case Study

For better understanding of the functionalities and the evaluation of the toolset discussed in this paper, the author considers the following small example of room renting system for events shown in Fig. 1.

This example will be used throughout this paper: “To book an event, the client gets in contact with the company and provides his name. The coordinator verifies if the client has been previously registered. If the client register exists, the coordinator allows him to proceed. On the other hand, if the client is new, the coordinator asks for his information. Client provides his information and the coordinator creates a new client register. Then, the coordinator finally allows client to proceed. Just when the client is allowed to proceed, he provides the information of the event and rooms. After that, the coordinator checks the availability of rooms in the data repository. Then, he informs to the client the results. The client, in turn, selects the most appropriate room for him and asks for the quotation. Coordinator proceeds to create a new quotation, calculates the total price and gives a receipt to the client. Finally, the coordinator updates room availability and the client finishes communication with coordinator”.

From the informal description given, three business processes are identified for the system: Contacting the department, registering a client and booking an event.

IV. Integrated Domain Modeling Toolset

A. Use Case Editor

This tool is meant for creating and defining elements that compose the business use cases. Furthermore, it allows constructing the use case model that will be used for representing the procedural knowledge within the IDM approach.

It is important to clarify that this tool is not meant for a UML Use Case diagram, which is clearly standardized. Instead, this tool supports an adjusted use case template that describes business use cases.

From the IDM approach, business use cases are formed by sentences written in a natural language, which shows step by step how a process is executed, what the variations are and which actors are involved. For instance, for each of the business processes of the example, its respective business use case specification was created using the adjusted use case template proposed by the IDM (see Fig. 1).

However, the user needs to pay attention to certain considerations while creating the specification of the business use case. First, the user needs to write sentences in simple present tense and in the simplest and unambiguous way possible; although in realistic way, this might not be always possible. This is more advisable if ontology is not used.

For example, for the business use case specification “Booking an event” shown in Fig. 1, the precondition and Step 3.1 were combined in one sentence in the informal description as “Just when client is allowed to proceed, he provides the information of the event”. Clearly, the pronoun “he” refers to “client” but for the specification of the business use case using the Use Case Editor, the pronoun needs to be replaced by “Client”.

Furthermore, it is recommended that the user not define complex sentences. Instead, two use case steps should be written than one complex. For instance, in the specification of business use case “Registering a client” shown in Fig. 1: “Client provides his information and coordinator creates a new client register”. It should be written as two steps: “Client provides his information” and “coordinator creates a new client register” as in Steps 2.2 and 2.3, respectively.

As shown in Fig. 2, this tool allows users to define different elements for the root element of Use Cases, such as Actors, Conditions, Use cases and Main scenario with its steps. Moreover, it enables one to define Composite conditions and alternative scenarios (Subvariation and Extension) for main scenarios.

The attributes that can be defined for the root element of Use Cases are Domain, Scope and Ontology. However, the attribute Ontology just refers to the technical name of the ontology for the domain.

To start creating the model, the user needs to right click on the root node of Use Cases and select an element of any of the types shown in the menu. The menu items shown are: Actors, Conditions and Use case.

Under the Actors node, it is possible to create as many actor elements as the user wants. The Actors element just has Description as an attribute.

Under the Conditions node, it is possible to create elements of Simple and Composite Conditions that will serve as preconditions for the business use cases. Both types of conditions have Id and Operation as attributes, but only for Composite Conditions, the Conditions attribute also needs to be specified.

In order to create composite conditions, the user needs to consider that it is necessary to have previously defined two simple conditions. After that, the user needs to add those simple conditions inside the Conditions attribute of the composite condition and, finally, select one of the available pre-defined operator values (OR, XOR, AND).

In Use Case element, the user can set the list of Actors, Description and Id as attributes. Also, it is possible to set Main Scenario, Sub Variation and Extension.

Inside the Main Scenario, the user can start creating its steps, i.e., default event elements. Moreover, inside the Subvariation or Extension element, user can create alternative event elements.

The attributes for event elements are Description, Id, Postconditions and Preconditions. To set these attributes, the user shall use the property view.

The user shall save the model, and a file with extension .usecases will be created. User can open it with a text editor to see the XMI format it is written in.

After finishing the creation of the business use cases, the user can acquire the topological relations and its graphical representation automatically after opening the context menu on the use cases model and selecting “Transform to TFM” option. The Use Case to TFM transformation tool will be executed and then the TFM will be generated. After that, user can continue with the analysis in the TFM.

B. TFM Editor

This tool is meant for constructing and representing the topological space. After clicking on the option “Transform to TFM”, as mentioned earlier, the Use cases to TFM transformation tool transforms the use case model into TFM model automatically.

TFM shall consist of one or more cycles, at least two functional features and at least one topological relationship. It may contain logical relationships, as well.

As shown in Fig. 3, the tool allows the users to edit the generated TFM by adding different types of elements as Actor, Functional Feature, Topological Relationship, Cycles and Logical Relationship.

For creating an element, the user shall right click on the TFM root node and select one of types of elements mentioned before. To set the attributes for a specific element the user shall use the property view.

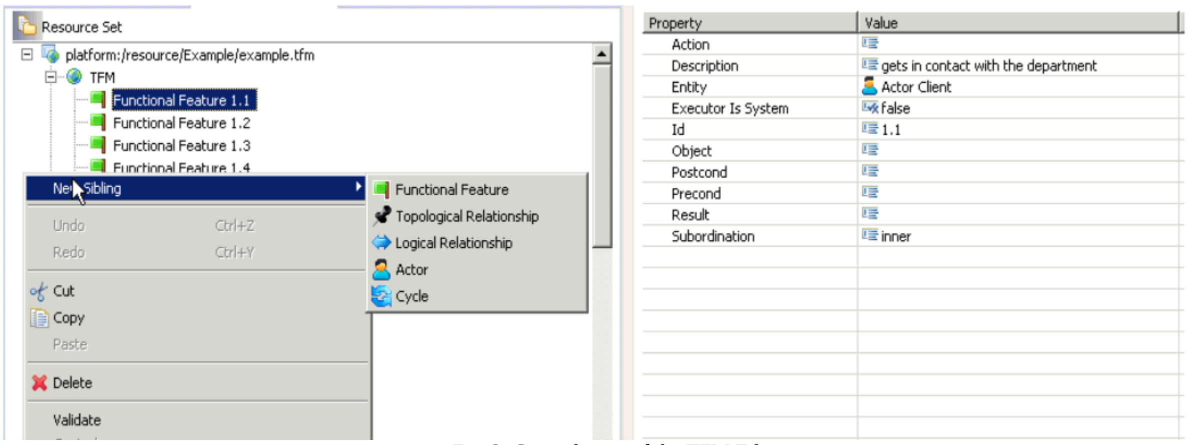
Functional Feature element has the following attributes: Id, Description, Action, Result, Object, Subordination (inner or external), Precondition, Postcondition and whether the Executor is the system.

The Topological Relationship element, which defines the cause-effect relationship between two functional features, has the following attributes: Id, Source and Target. Source and Target attributes are functional features previously defined.

The Logical Relationship element can be defined using at least two topological relationships. Moreover, it has the following attributes: Id, Operation (AND, OR, XOR) and the Related Elements, which are the topological relationships that define it.

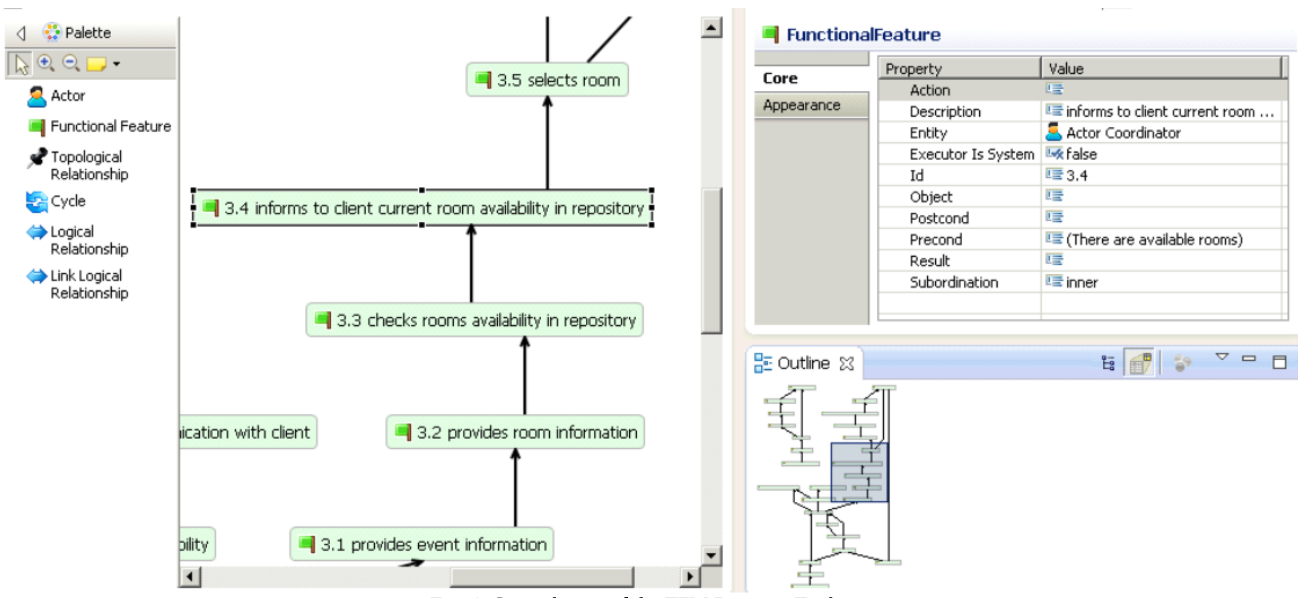
The Cycle element has Order, whether it is main, and Functional Features as attributes. Also, the Actor element just has Description as an attribute.

The user shall save the model, and a file with extension .tfm will be created. It can be opened with a text editor to see the XMI format it is written in.



Property	Value
Action	
Description	gets in contact with the department
Entity	Actor Client
Executor Is System	false
Id	1.1
Object	
Postcond	
Precond	
Result	
Subordination	inner

Fig. 3. General view of the TFM Editor.


Palette
Actor
Functional Feature
Topological Relationship
Cycle
Logical Relationship
Link Logical Relationship
3.5 selects room
3.4 informs to client current room availability in repository
3.3 checks rooms availability in repository
ication with client
3.2 provides room information
bility
3.1 provides event information
FunctionalFeature
Core
Appearance
Outline

Property	Value
Action	
Description	informs to client current room ...
Entity	Actor Coordinator
Executor Is System	false
Id	3.4
Object	
Postcond	
Precond	(There are available rooms)
Result	
Subordination	inner

Fig. 4. General view of the TFM Diagram Tool.

When transforming from use case model to TFM, the tool takes certain considerations. For instance, Step 1.4.1 from the business use case “Contacting the department” shown in Fig. 1: “If client doesn’t exist, coordinator asks client information”.

First, the tool composes the functional features. For each of them, the tool sets Id, the Id use case step to whom they refer.

As mentioned before, the TFM states that every functional feature consists of an object action, a result of this action, an object involved in this action, a set of preconditions of this action, an entity responsible for this action and subordination.

Thus, the tool identifies the object action from the Description of the step. The Description attribute is fetched automatically from the use case step. In the example, the description of Step 1.4.1 is “coordinator asks client information”.

After that, the tool should recognize the objects involved in the given action. From the example, the involved objects are: a client, coordinator and client information. However, the tool currently does not support this task.

The use case actors are considered as entities responsible for the functional feature. In the example, the entity responsible for this action is “coordinator”. It is important to notice that the tool also verifies if the specified actor is considered in the list of actors defined for the business use case. For the exposed case, the business use case “Contacting the department” has as actors: client and coordinator; hence, no errors will be displayed.

The tool views as Preconditions the preconditions of the use case step. From the example, the precondition is “client doesn’t exist”.

The default value for Subordination is “inner”, and it is the value that the transformation considers when transforming to the TFM since there is no way to establish this attribute from the Use Case Editor.

The setting of cause-effect relations between functional features represented within the same use case is straightforward. In this purpose, the tool follows the order of steps in the definition of the scenarios.

Every main scenario of use case is an ordered sequence of functional features. For instance, the tool will create cause-effect relations between Steps 1.1, 1.2, 1.3 and 1.4 for the business use case “Contacting the department” in Fig. 1.

Extensions and Subvariations help to detect branching in the TFM:

- Extension adds an effect to the functional feature represented by the step referenced by the extension. As shown in Fig. 1, there are just two extensions for the business use case “Booking an event” included in Steps 3.5 and 3.10.
- Subvariation, on the other hand, adds an effect to the functional feature represented by the previous step referenced by the subvariation. Similarly, the selected case study just has two variations included in Step 1.4 of business use case “Contacting the department” and Step 3.11 of the business use case “Registering a client”.

Also, no duplicate functional feature is created, i.e., functional features represented by the same tuple are considered the same functional feature, and two or more use cases can include the representation of the same functional feature.

From the example, the description in Step 3.11 and Step 3.10.2 is “Client finishes communication with coordinator”. Therefore, the tool will recognize the similarity and then, it will create just one functional feature whose description will be “Client finishes communication with coordinator”. After that, it will generate a cause-effect relation between the functional features of Step 3.10 with the newly generated functional feature. Similarly, the tool will create another cause-effect relation between the new one and the functional feature of Step 3.10.1.

User can continue the analysis from the just automatically generated TFM model produced by the transformation of the use case model. Therefore, there is no need to construct the

TFM from scratch and due to the fact that the initial model has automatically been generated from the Use Case Editor.

The author agrees that the generated TFM is intuitively illustrated and easily editable, so that any incompleteness, redundancy or inconsistency could be corrected. However, the user needs to pay attention to certain considerations, while using this tool. First, the IDM suggests several iterations back and forth between the use case model and the TFM until the system analyst can verify it is correct.

Moreover, the system analyst shall manually determine functional feature subordination after acquiring the TFM since the tool currently does not automate this feature. This can be particularly necessary when establishing subordination between functional features from different business use cases.

Finally, currently the object action, result of this action and object involved in this action are merged into the description attribute. Hence, the tool, when transforming, does not give them automatically.

C. TFM Diagram Tool

This tool is a graphical editor for constructing the TFM topological graph.

As mentioned earlier, after clicking on the option “Transform to TFM”, the Use Case to TFM transformation tool transforms the use case model into the initial TFM model. Then, the user shall click on “Initialize TFM diagram” to generate the diagram automatically.

Thereby, the diagram tool displays the generated TFM in a graph and provides a palette for the user to continue modeling TFM in the diagram form.

This tool allows the users to define the TFM in the diagram form and provides a palette of options for defining its elements. The elements that can be defined are actors, functional features, topological relationships, cycles and logical relationships.

To create elements, the user shall select the corresponding option from the palette or right click on the canvas and select the desired element.

All the attributes of the element can be set directly in the property view. To add a topological relationship from the palette, two functional features need to be in place first.

Moreover, to create a logical relationship, at least two topological relationships need to be already in the diagram. The link option for logical relationships allows binding the logical relationship to the topological relationship. Finally, to create a cycle, two or more functional features are needed.

As Fig. 4 shows, on the left, there is a palette with available options for the user to the model. In the center, there is a canvas for the diagram. On the right, there are the properyt and outline view.

The user shall save the diagram, and a file with extension .tfm_diagram will be created. It can be opened with a text editor to see the underneath XMI format it is written in.

User needs to pay attention to certain considerations, while using this tool. First, it is not possible to automatically acquire the main cycle, sub-cycles and logical operations. Thus, this has to be inserted manually by the user. The user, for the completeness of the main functioning cycle, shall set the cause-effect relation between functional features from different use cases since this type of relationship cannot be determined automatically.

Furthermore, when editing the diagram, the diagram tool also updates the changes into the TFM model of the TFM Editor so both artifacts are in sync. However, when there are errors or changes in cause-effect relationships in the TFM, the user needs to return to the Use Case Editor and make the corresponding modifications there as well. But since there is not any backward synchronization between the TFM Editor and the diagram tool to Use Case Editor implemented yet, the user might lose all those changes.

V. Suggestions for Improvements

When performing the usage of the IDM toolset, the author was able to successfully acquire a domain model in the form of TFM, i.e., to acquire a graphical representation of the business processes (procedural knowledge) automatically.

However, the author found some issues important to be resolved perhaps in further research and the possible solutions are presented here as suggestions for improvements in Table I.

Notice that the author considers the user practical usability as the main aspect that makes these suggestions important. The toolset components should be enough understandable so the end-user can work with the tool with low effort to learn the approach and produce valid and correct TFM diagrams.

A. Tasks in the Use Case Editor

Creating business use cases is quite simple. User just needs to create a new Use Case model in a new project and start writing each of the steps that will compose the Business Use Cases.

However, the following problems have been found while using this particular tool and will be explained further in the rest of this subsection:

- There is not integration with any tool that supports declarative knowledge;
- Lack of validations on business use cases when selecting “transform TFM option” directly;
- Configuration of reference and trigger properties that are not explicitly defined in the approach produce consequences in the automatically generated TFM model.

As mentioned earlier, it is important that the domain modeling tool acquires the domain knowledge that is composed by the declarative knowledge and procedural knowledge: knowledge model (dictionary of the domain) for a business domain (business processes of the domain).

In the IDM approach, ontology is used for defining the declarative knowledge and Business Use Cases are used for acquiring the procedural knowledge of the domain.

The author agrees about the fact that Business Use Cases provide a formal way to define the procedural knowledge showing step by step how a business process of the domain is executed, what the variations are and which actors are involved.

TABLE I

SUMMARIZATION OF SUGGESTIONS FOR IMPROVEMENT

Tool	Suggestions for improvement
Use Case Editor	• Integrate with a tool that supports "declarative knowledge". • Improve the handling of validations of Business Use Case. • Clarify the role of "reference" and "trigger" property when defining steps of the Business Use Case. • Provide numeration guidelines for elements in the Business Use Case.
TFM Editor	• Implement backward synchronization from TFM and diagram to Business Use Case. • Improve Business Use Case validation: some topological relationships are generated outside the main TFM node.
TFM Diagram Tool	• Provide suggestions of cycles for the generated TFM.

Nevertheless, currently the IDM approach does not provide any tool implementation or any integration with other tool that supports the declarative knowledge of the domain. Thus, the user needs to perform the use case validation against the declarative knowledge manually.

The IDM approach states that ontology is not mandatory and encourages the reuse of existing ontology tools that can help the system analyst in the validation of use cases to correspond with the domain ontology.

But still, the lack of support of declarative knowledge, i.e., integration with any ontology tool and the automatic use case validation with the declarative knowledge, can be pointed out as one of the most valuable improvements to be done. The author believes that constructing or reusing domain ontology is important for helping the domain analysis process and validating use cases against ambiguity and inconsistency problems.

As mentioned before, the validation is a necessary task to be performed manually by the system analyst together with the knowledge engineer. Consequently, since use case steps are written in a natural language, it cannot be guaranteed that the terms used within the use case definition are unambiguous and consistent.

Furthermore, it is necessary to improve the validations of use cases. While performing several tests with the Use Case Editor, the author confirms that it provides some basic validations that are launched just after selecting the "Validate" option in the editor's main menu. Nevertheless, any validation over the business use cases is executed if the transforming TFM option is selected directly. Apparently, the editor allows the user to generate the TFM even though there might be problems with the definition of some use cases.

The author believes that this can be considered a tool bug that can lead to inconsistency problems in user's work especially if the user spent a lot of time and effort in making changes over a generated TFM with errors that the user did not know it was incorrect in the first place. Moreover, this can be particularly hard to detect for users that are starting to learn the IDM approach.

As a way of dealing with this issue, the author suggests that the tool should not allow the user to generate the TFM if no validation has been performed first. Therefore, the "Transform to TFM" functionality should not be available if the user has not validated the model first by clicking on the "Validate" option.

Otherwise, the tool should always execute the validations of the use cases before the generation of TFM is performed. Then, if there are any mistakes, it should notify the user. If not, it should execute the transformation automatically.

Secondly, the tool allows the configuration, among other properties, of the Reference and Trigger property when defining steps of the Business Use Case.

As explained in [2], the TFM does not define those properties as part of the functional feature definition. Furthermore, in [4] it is not defined as part of the use case definition; therefore, it is not clear for the authors what their role is in the tool or in the generation of TFM. However, according to some tests performed with the tool, the setting of any of those properties produces an impact over the generation of the TFM diagram.

In case of reference attribute, it makes the use case step, for which the attribute has been defined, not be considered a functional feature in the generation of the TFM model. Thus, this use case step is ignored. Furthermore, it creates a cause-effect relation between the functional feature obtained from the use case step defined in the reference attribute and the functional feature obtained from the following use case step, i.e., the use case step located after the use case step with a defined reference attribute.

In case of trigger attribute set up by the user, it causes the creation of cause-effect relation between the functional feature defined in the trigger attribute and the functional feature generated from the use case step for which the trigger attribute has been established. This can be particularly beneficial for establishing the cause-effect relation between steps from different use cases, i.e., determining functional features subordination in the generation of TFM. Currently, the subordination of functional features is another necessary task to be performed manually by user.

The author guesses that those properties might be part of the leveraged functionality provided by the EMF when developing the tools. But revision is strongly recommended for avoiding user's misuse, especially since it is open for the users and their usage might produce unwanted side effects on the generated TFM. Thereby, unless the functionality is clearly established, the author suggests their omission.

Finally, as described before, the IDM approach proposes working with an adjusted use case template that describes the business use cases. But the adjusted template does not say anything related with the numeration guidelines for elements in the use case. Thus, even though the user uses the same numeration but in different steps for different use cases, the tool will consider this as an error and complain when it validates the model.

Hence, the author recommends establishing numeration or naming guidelines for use cases and its steps described in the main scenario, the extensions and variations. Otherwise, after finishing the use case model, the user will be alarmed by a long list of error messages indicating that the chosen number for a specific step has already been considered in another step. This can be particularly confusing for novice users and discourage its usage.

B. Tasks in the TFM Editor

The generation of TFM is quite straightforward. It is just needed to open the context menu on the use case model and select the "Transform to TFM" option. The editor automatically identifies the cause-effect relations between the functional features by considering the order, in which the functional features appear in the business use case description.

Once the generated TFM is in place, adding new topological elements becomes efficient. The tool offers the user to correct these initial system objects, functional features or cause-effect relations, and add new objects or functional features.

However, the following problems have been found while using this particular tool and will be explained further in the rest of this subsection:

- There is not backward synchronization from the TFM tool and diagram tool to Business Use Case tool.
- Lack of validations: some topological relationships are generated outside the main TFM node.

Because the IDM approach suggests an iterative development of the TFM model, the user is able to see the mapping between the changes in the TFM and the changes in the TFM diagram. Since they are in sync, changes in either place (topological space or topological graph) are automatically affected.

However, the same does not apply if changes in the use case source occur after changes in the auto-generated sources (TFM and TFM diagram). If this happens, it can cause incompatibility problems when regenerating the TFM from the recently updated use case source, i.e., changes in use cases can lead to backward incompatibility problems because changes in the TFM or TFM diagram are not automatically reflected in the use case source; thus, all changes in the TFM and TFM diagram, if there are any, will be lost.

As described in [16], the TFM tool supports several iterations back and forth between use case description and TFM generation until the system analyst can verify every functional requirement. Thus, changes are expected as part of the process of acquiring the TFM version that satisfies the user. However, all the changes in the TFM and TFM diagram will be lost if the user generates the TFM again from the use case editor because all those changes are not reflected in the Use Case Editor.

The author considers this to be another essential aspect to improve in the toolset. Every time, after changing the use case source and transforming it to the TFM, the user will need to remember all those changes in the auto-generated source and set them up again into the tool, causing extra job and time from the user perspective.

On the other hand, when using the TFM editor, the author noticed that some topological relationships were generated outside the main TFM node. The author thinks that all the elements automatically created from the use case source should be allocated inside the TFM node. Therefore, the author is not sure about the reason why some elements can be created automatically outside the main TFM node; thus, they consider this occurrence as a tool bug.

It is important to mention that novel users might not notice this error and can start immediately working on a generated TFM diagram with errors. The author can just guess that it might happen because the use case source had errors that were not initially detected when transforming to the TFM. Consequently, the tool generates the TFM from business use cases with errors and all the topological relationships that had errors were automatically allocated outside the main node. As suggested before, the author suggests giving a more emphasis on the validation from the Use Case Editor to avoid this kind of incompatibilities.

C. Tasks in the TFM Diagram Tool

The tool automatically constructs a TFM topological graph. Again, once the generated TFM diagram is in place, adding new topological elements becomes efficient.

The author has just found one suggestion that can contribute to improving the effectiveness of the tool when acquiring the TFM model.

As described in [2], in every topological model of system functioning there must be at least one directed closed loop (i.e., a directed closed path). Usually it is even expanded hierarchy of cycles. This property of the model enables analyzing similarities and differences of functioning systems.

The importance of cycles lies in providing a formal validation. The TFM allows it by a cycle analysis against the actual functioning system in real life (make sure that all functional features for a particular business process represented by the particular cycle are present). Thus, it is possible to trace through the functional features and see if the cycle makes sense.

The tool enables the user to manually point the main functional cycle and to add secondary cycles [16]. However, the author believes that if the tool provides some sort of mechanism of suggestions of cycles for the generated TFM, the analyst would validate at first glance the generated TFM. Therefore, the validations, additions or changes of the TFM might be easily pointed out.

VI. Conclusion

The results after using the toolset are as follows. As a prototype, its main goal for acquiring automatically the CIM as the TFM, as a formal domain model, through the business processes using business use cases with a model transformation has been accomplished successfully.

The toolset is intuitively illustrated and allows modifying models easily, so that the user can correct any incompleteness, redundancy or inconsistency in the generated TFM.

Due to its compatibility with MDA standards and MDA frameworks, the toolset projects itself as a candidate for integration with other modeling tools and has a big potential for becoming part of the MDA life cycle.

The synchronization between the TFM editor and TFM diagram tool ensures the TFM model keeps accurate while editing.

However, to become a fully usable toolset and to introduce it for wider audiences, there are some considerations that should be taken into account.

First, it is necessary to improve the validations and user notification messages, primary on the Use Case Editor. This can be particularly beneficial for users that are starting to work with the TFM4MDA approach. Currently, the Use Case Editor allows the user to transform business use cases into the TFM even though there might be some basic errors that should be corrected before the transformation. The tool should not allow the user to perform the transformation if any errors have been founded.

Secondly, it is necessary to provide the synchronization of changes from the Use Case Editor to the rest of the tools. Currently, the synchronization of changes exits solely between the TFM Editor and TFM Diagram Tool. Thus, after acquiring the TFM and making some changes in it, if the user is willing to make changes in the business use cases, all the changes made in the TFM Editor and TFM Diagram Tool will be lost when the user performs “Transform to TFM”.

Third, because of the limitation of the language processor used for the implementation of the toolset, users need to consider reformulating the steps by using simple tense and writing sentences as simple as possible.

Finally, even though, proving the cycles is not part of the initial transformation of the TFM, the author believes that it would be a great benefit for users to somehow see the potential cycles. This functionality can accelerate the process of editing the TFM because analysts could directly identify inconsistencies, if there were any.

References

[1] Gasevic, D., Djuric, D. and Devedzic, V., *Model Driven Architecture and Ontology Development*. Heidelberg: Springer, 2006.

[2] Asnina, E., and Osis, J., “Topological Modeling for Model-Driven Domain Analysis and Software Development: Architectures and Functions,” in *Model-Driven Domain Analysis and Software Development: Architectures and functions*, IGI Global, Hershey – New York, 2011, pp. 15–39.

[3] Osis, J., Asnina, E. and Grave, A., “Computation independent representation of the problem domain in MDA,” *J. Software Eng*, vol. 2, no. 1, pp. 19-46. [Online]. Available: http://www.e-informatyka.pl/ wiki/e-Informatica_-_V olume_2. [Accessed: March 2015].

[4] Šlihte, A., “Implementing a Topological Functioning Model Tool,” in *Scientific J. of Riga Technical University, 5. series, Computer Science*, vol. 43, Riga, 2010, pp. 68–75.

[5] Osis, J., “Topological Model of System Functioning,” in *Automatics and Computer Science*, *J. of Acad. of Sc*., no. 6, 1969, pp. 44–50.

[6] Osis, J., Asnina, E. and Grave, A., “Formal Problem Domain Modeling within MDA. Communications,” in *Computer and Information Science (CCIS)*, vol. 22, Software and Data Technologies, Springer-Verlag Berlin Heidelberg, 2008, pp. 387–398.

[7] Asnina, E. and Osis, J., “Computation Independent Models: Bridging Problem and Solution Domains,” in *Model-Driven Architecture and Modeling Theory-Driven Development: ENASE 2010*, J. Osis and O. Nikiforova (Eds.), 2ndMDA&MTDD Whs., SciTePress, Portugal, 2010, pp. 23–32.

[8] Osis, J., Asnina, E. and Grave, A., “MDA Oriented Computation Independent Modeling of the Problem Domain,” in *Proc. of the 2nd Int. Conf. on Evaluation of Novel Approaches to Software Engineering*, ENASE 2007, Barcelona, Spain, 2007, pp. 66–71.

[9] Osis, J., Asnina, E. and Grave, A., “Formal Computation Independent Model of the Problem Domain within the MDA,” in *Information Systems and Formal Models, Proc. of the 10th Int. Confe.*, ISIM’07, Silesian University, Opava, Czech Republic, pp. 47–54.

[10] Osis, J., Asnina, E. and Grave, A., “A Computation Independent Modeling within the MDA,” in *IEEE Int. Conf. on Software-Science*, Herzlia, Israel, no. E3021, pp. 22–34.

[11] Osis, J. and Asnina, E., “Derivation of Use Cases from the Topological Computation Independent Business Model,” in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*. IGI Global, Hershey - New York, 2011, pp. 65–89. http://dx.doi.org/10.4018/978-1-61692-874-2.ch004

[12] Asnina, E. and Osis, J., “Topological Functioning Model as a CIM-Business Model,” in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*. IGI Global, Hershey - New York, 2011, pp. 40–64. http://dx.doi.org/10.4018/978-1-61692-874-2.ch003

[13] Osis, J. and Asnina, E., “Is Modeling a Treatment for the Weakness of Software Engineering?” in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*. IGI Global, Hershey - New York, 2011, pp. 1–14. http://dx.doi.org/10.4018/978-1-61692-874-2.ch001

[14] Osis, J. and Asnina, E., “A Business Model to Make Software Development Less Intuitive,” *Proc. of the 2008 Int. Conf. on Innovation in Software Engineering*, Vienna, Austria. IEEE Computer Society CPS, Los Alamitos, USA, 2008, pp. 1240–1246.

[15] Eclipse Modeling Framework. [Online]. Available: http://eclipse.org/modeling/emf/. [Accessed: March 2015].

[16] Šlihte, A., “The Specific Text Analysis Tasks at the Beginning of MDA Life Cycle,” in *Data- bases and Information Systems Doctoral Consortium*, Latvia, Riga, July 5–7, 2010, pp. 11–22.

[17] Donins, U., Osis, J., Slihte, A., Asnina, E. and Gulbis, B., “Towards the Refinement of Topological Class Diagram as a Platform Independent Model” in J. Osis, O. Nikiforova (Eds.). *Model-Driven Architecture and Modeling-Driven Software Development: ENASE 2011*, 3rd Whs. MDA&MDSD, SciTePress, Portugal, 2011, pp. 79–88.

Kelly Verónica Fernández Céspedes currently is a Master Student at Riga Technical University. In 2005 she graduated from the Pontifical Catholic University of Peru, Lima, with Bachelor Degree in Computer Engineering.
The author is currently working as an Assistant Researcher at the Faculty of Computer Science and Information Technology, Institute of Applied Computer Systems, Riga Technical University. The research focus is Model Driven Architecture (MDA), more specifically applying Topological Functioning Model (TFM) as the Computation Independent Model (CIM) within MDA.
E-mail: kfernandez@pucp.pe