Project: FORMOSE

Formal Requirements Modeling for Critical Complex Systems, Method and Toolkit

Formose Requirements Modeler (FORMOD)

Tool documentation version 1.0

Institut Mines-Télécom, Openflexo

Authors:
Fahad R. Golra
Fabien Dagnat

Development Team:
Sylvain Guerin
Christophe Guychard

February 6, 2017

Contents

1 Overview

1.1 What is FORMOD?

Formose Requirements Modeler (FORMOD) is a requirements elicitation and modeling tool. As a start, it allows gathering the requirements from different software project artifacts like project proposals, feasibility reports, standards, etc. It uses KAOS approach for the development of goal models, corresponding to the identified high-level requirements. It also helps in refining these requirements to a coarse grain level, where they become unambiguous and verifiable.

FORMOD methodology aims to model software project requirements of critical and complex systems in a formal way. It uses model federation at its core, which helps in creating and maintaining dynamic links between several models of different paradigms *e.g.* requirements specification (textual), goal models, project artifacts, system design, *etc.*

1.2 Who can use FORMOD?

FORMOD as a requirements elicitation tool can be used by the requirements engineers to gather early requirements. Formose methodology, working at the core of this tool, allows for refining the requirements. This helps the requirements engineer in achieving requirements completion. Apart from requirements elicitation, this tool also allows to model the requirements. Current version of FORMOD is using KAOS goal modeling diagrams. This aspect of the tool can help system analysts to analyze the domain models in relation to different modeling diagrams offered by KAOS.

1.3 How does it work?

FORMOD tool is based on the FORMOSE requirements engineering methodology. This methodology connects the artifacts/models of different paradigms used in the process, so as to keep them synchronized all along the project development lifecycle. Synchronization of models is exploited to develop traceability links that can be operationalized. Operationalization of traceability links is ensured through the definition of behavior for mappings between the model elements. These mappings with behavior are defined using model federation approach. Once such mappings are developed between different models used in the requirements engineering process, new instances of artifacts can be generated and the old instances can be kept up to date.

1.4 Licenses

FORMOD is a requirements elicitation and modeling tool, developed under the research project, FORMOSE. It is distributed under FLOSS licenses considered as "free" by the Free Software Foundation[1]. This documents of the tool is distributed under GNU Free Documentation License[2].

[1] `http://www.fsf.org/licensing/licenses/`
[2] `http://www.gnu.org/licenses/fdl.html`

2 Installation procedure

FORMOD is available for install on Linux, Windows and Mac OS systems. Two mechanisms are offered to install the software on these machines: 1) as a packaged application and 2) as an eclipse project. This software is distributed under Free Licenses, so only the first mechanism is available for the general public.

! → The second mechanism of installation is only for the project partners and development team members. In order to access the implementation details, the project partners should contact Openflexo team partners.

2.1 Installation - Packaged distribution

1. Open any browser and access the download page for the packaged version of FORMOD through the following url.

URL : *https://downloads.openflexo.org/Formose/* [3]

2. The download page should list different versions of the software. Click the link corresponding to the latest release.
3. The new page should again list different versions of the package. Use the latest package version in the list.
4. The next page will give you the choice amongst three different platforms: Linux, Max and Windows. Click on the appropriate platform for access the corresponding package.
5. Click on the installation package to start its download.
6. Once the download is complete, you can double click on the image file to mount it[4]. It should open a window showing the contents of the image *i.e.* the software package, as shown in Figure 1.

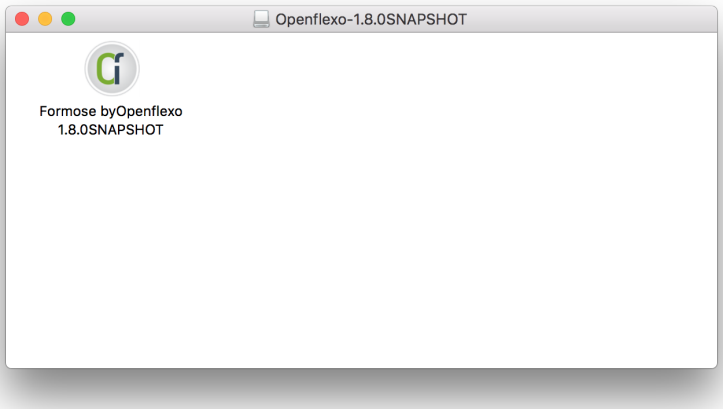

Figure 1: Mounted image of installation package

7. Copy the application launcher to your desired location (Software in Mac OS X are typically installed in `/Applications`)

[3] Download url of FORMOD is subject to change, please refer to FORMOSE wiki for further updates.
[4] This step is considering Mac OS installation process. Corresponding actions need to be taken for Linux and Windows Systems

Figure 2: Warning - Unidentified Developer

8. Trying to open the application launcher directly might trigger a warning, as shown in Figure 2. In this case, right click on the application launcher and click `Open`. This will pop-up a new dialog box, as shown in Figure 3. Click `Open` again to launch the application.

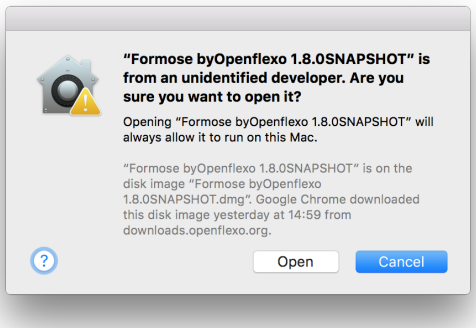

Figure 3: Dialog box with possibility to launch

3 Tool Layout

FORMOD tool interface has different zones to deal with different parts of the project. Once a new project is created, the tool loads the provided document file into its own interface, as shown in Figure 4.

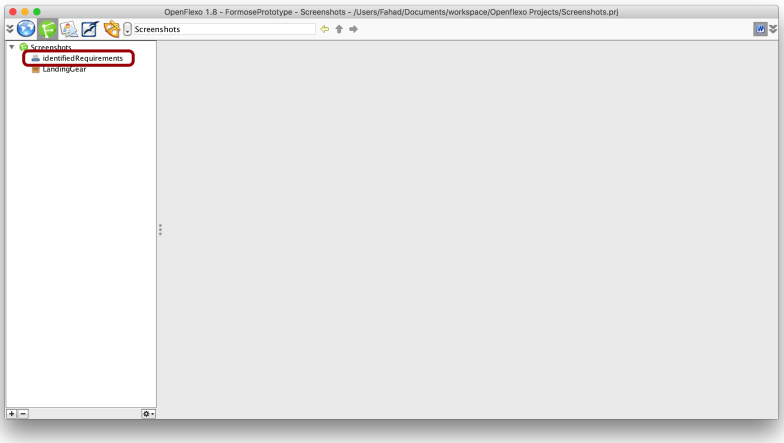

Figure 4: Tool layout - Loading new project

The left pane of the tool shows two packages under the project name: *the identified requirements* and the *goal diagram* for the project. Double clicking the identified requirements opens up the tool with further different zones, as shown in Figure 5. These zones are explained one by one, in the rest of this section.

3.1 Project Explorer Zone

This is the top left zone of the tool interface. It presents the structural hierarchy of the project, currently opened by the tool. At the top of this hierarchy is the name of the project. Under its name, we find two items, if only one document file is loaded in the project. The first one is the "identified requirements" view of the project. Opening this view displays other zones of the interface that help in requirements elicitation. The second item in the hierarchy is the goal diagram for the project.

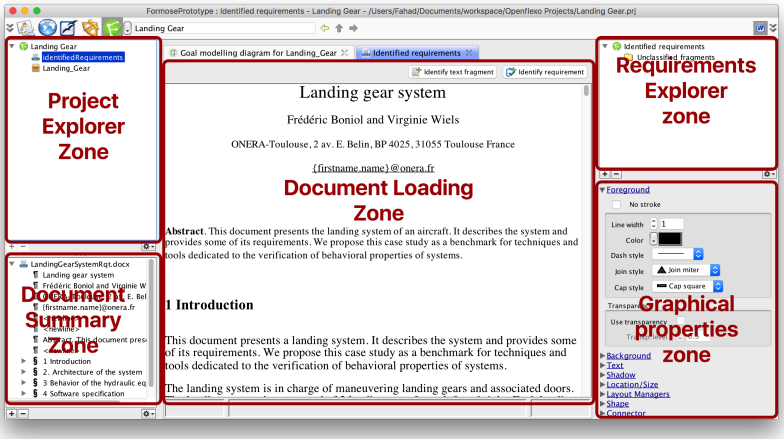

Figure 5: Mounted image of installation package

Opening this item displays the view of goal modeling. Goal modeling view presents different zones for graphical modeling of the requirements.

3.2 Document Loading Zone

This is the middle zone of the tool interface. It loads the document that the project is currently using for requirements elicitation. In case of a word document, the contents of the documents are loaded into this zone. These contents are selectable in order to mark a portion of text as requirements or a fragment of interest. On the right top corner of this zone, we find two buttons: *Identify text fragment* and *Identify requirement*. Identify text fragment button is used to select a fragment of text that might contribute to building a requirement. Identify requirement button is used to select a portion of text or already identified *fragment of text* as a requirement.

3.3 Requirements Explorer Zone

The right top zone of the interface is the requirements explorer zone. Every time a portion from the document is marked as an *identified fragment*, it lists under the identified fragments item of this explorer. Likewise, whenever a requirement is defined from the identified fragments or direction from the document, it lists itself under the *identified requirements* item.

3.4 Document Summary Zone

This zone is located in the lower left side of the interface. It allows to get an overview of the complete document. It also serves as a hyper-linked table of contents for the document, that allows to easily access specific portions of the document.

3.5 Drawing Zone

When the goal diagram item from the project explorer zone is opened, it opens the goal modeling view for the project. In this view, the document loading zone is replaced by the Drawing zone, as shown in Figure 6 The drawing zone allows the development of goal models for the project. The top left corner of the drawing zone offers some controls for aligning the graphical objects placed in the drawing zone.

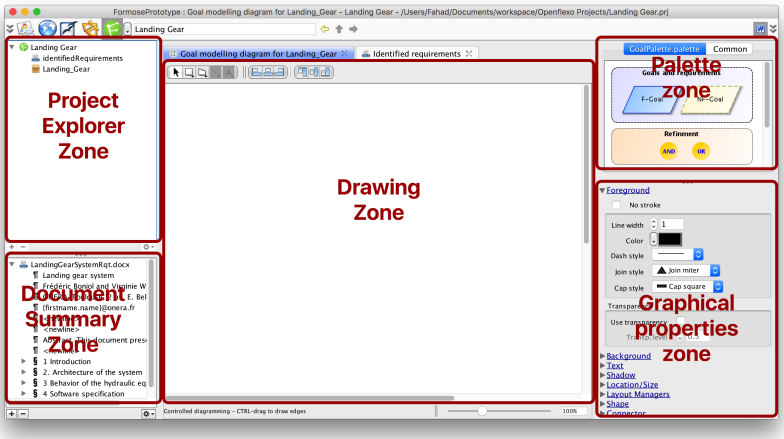

Figure 6: Mounted image of installation package

3.6 Palette Zone

When in the goal modeling tab, the requirements explorer zone is replaced by the palette zone. This zone has two different tabs: i) the goal diagram palette and ii) the common palette. The goal diagram palette offers the common concepts used for the development of goal models. These graphical items can be dragged and dropped into the drawing zone for the development of goal models. The common palette offers different common shapes, which do not have any associated concepts in the goal modeling diagrams. However they can be used to annotate, modify a goal modeling diagram with additional graphical items.

3.7 Graphical Properties Zone

This zone is located in the lower right part of the interface. It allows to manipulate the graphical properties of the graphical items in the drawing zone. This way, we can change the color, font, shadows, shapes, etc.

4 FORMOSE Process

FORMOD tool implements Formose methodology for the requirements engineering. This methodology concerns the requirements elicitation phase. The activities involved in the process are performed according to the control flow described by Figure 7. All these activities are explained as under.

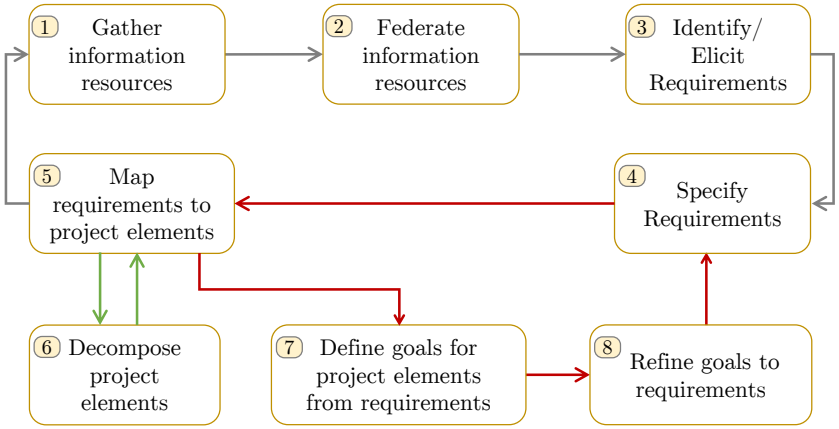

Figure 7: Formose Process

1. *Gather information resources:* This activity involves the identification of possible information resources that can be used to elicit requirements. They can vary from early information resources like feasibility reports, standards and minutes of the meetings to late informations sources like deployment models, test plans, *etc.*

2. *Federate information resources:* Once the information resources are collected they need to be linked with the requirements. This is handled through the model federation approach where requirement models are developed as virtual models in the conceptual space and other models (information resources) are placed in their respective technological spaces.

3. *Identify/Elicit requirements:* Once the information resources are linked with the requirements model, we can identify the requirements from those information resources. For the moment, we have used the technological space for MS Word documents to identify multiple fragments for possible requirements. Different documents are used for this purpose like project proposals, interview transcripts, feasibility reports, etc.

4. *Specify requirements:* Once the requirements are identified from multiple information resources, they are specified in the requirements model. In case the requirements are already specified, they are linked with the information resources for synchronization.

5. *Map requirements to project elements:* In this activity, the specified requirements are mapped to the project elements. Multiple requirements can be mapped to a single project element.

6. *Decompose project elements:* Multiple requirements mapped to a project element describe the expected functionality from that element. This allows the decomposition of the element into sub el-

ements, such that each sub element takes care of a subset of the requirements mapped to its parent element.

7. *Define goals for project elements from requirements:* All the requirements associated with a project element are considered as goals in this activity. A goal model is developed for each of these goals.
8. *Refine goals to requirements:* All the goal models are refined to get requirements at the leaves of the goal models. These requirements can be specified in activity 4.

This is an iterative flow of activities where we see two sub-cycles in Figure 7; one corresponds to the decomposition of project elements (shown with green arrows) and the other to the refinement of requirements (shown with red arrows). A single activity, *Map requirements to project elements*, serves as the entry point for both these cycles. These cycles are often alternating, which describes that the system decomposition occurs in parallel to the refinement of requirements.

5 Getting started by example

Let us see the FORMOD tool usage through a simple example. We demonstrate the use of this tool using the Landing Gear Example.

1. Open the FORMOD tool by double clicking on its icon. This shall open a startup screen which asks whether you need to develop a new project or load an existing one. Choose a new project in this screen.

! → In case you opened up the tool through Eclipse, make sure you select the Formose view from the startup screen, as shown through the red box in Figure 8.

Figure 8: Startup Screen

2. Enter the name and location of the project file and click `save`. This name would be used to create a physical directory on the file system, where all the files related to the project would be saved.

3. After enter the project file name, a wizard will ask you for the project name and requirements file. You can name you project as you wish. Browse to the word document that you want to use for the elicitation of requirements. We are using the Landing Gear Example file. Once done, click `Finish`.

4. Now you should have the main interface of the tool in front of you. At the left side, you see the project explorer, where you will see two items under your project file name: 1) identified requirements item, and 2) your "project name" item. Double clicking the identifiedRequirements element will open up further panels of the tool, as shown in Figure 9. This is the Requirements elicitation view of the tool. This will allow you to elicit requirements from the document file that you provided when creating the project.

5. In the requirements elicitation view, you can select portions of text from the document body that might be interesting for the development of a requirement. In our document, we select the fourth

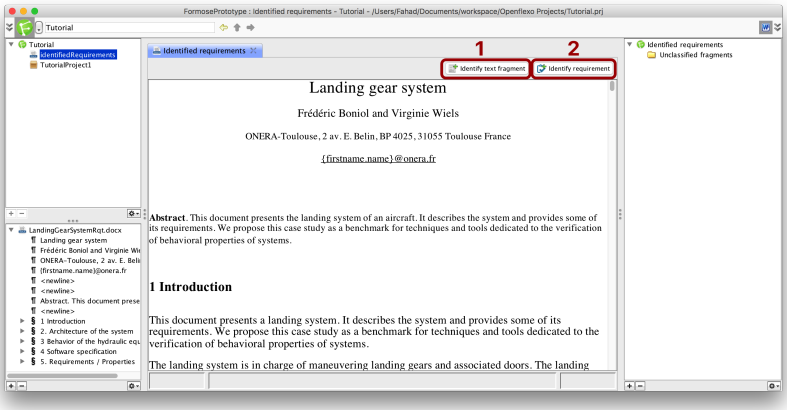

Figure 9: Requirements elicitation view

paragraph of the introduction section, which shows the landing and retraction sequences. Once selected, we click the button marked 1 in Figure 9. Now we choose to make this selected fragment a requirement, so we select the text of this fragment and click the button marked 2 in Figure 9. This will open a dialog box for the creation of a requirement, as shown in Figure 10. Enter the name (landing retracting sequence) and the description of the requirements and click `Finish`.

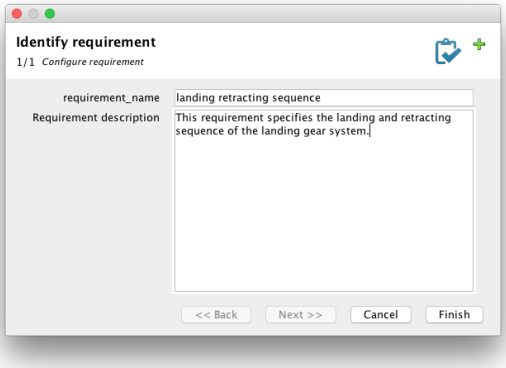

Figure 10: Requirements dialog box

6. This lists the requirement *landing retracting sequence* in the *Requirements Explorer Zone*. Now we shall add another portion of text to the same requirement. To do this, make sure that you have selected this requirement in the *Requirements Explorer Zone*. Now we can navigate to section 2.1 of the document through the *Document Summary Zone* and then select the first paragraph. Click `Identify requirement` button. This shall list both the selected fragments under the chosen requirement.

7. Keep selected other portions of text that might interest you to identify requirements. And then identify some requirements from those selected text fragments.

8. Once we have selected some coarse-grain requirements, its time to refine those requirements to get some fine-grain requirements. We shall model each of these high level requirements as goals in a goal

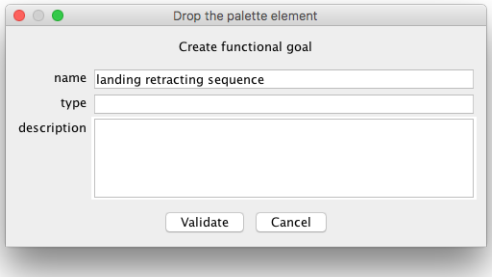

Figure 11: Functional Goal dialog box

modeling diagram[5][6]. In order to switch to the goal modeling perspective, we need to open the goal modeling package in the *Project Explorer Zone*.

9. In order to refine the requirements, we take our requirement, *landing retracting sequence* as a functional goal and refine it. From the *Palette Zone* of the interface, we select the goal modeling tab. We can drag and drop functional goals into the *Drawing Zone* to develop goal models from this palette. So we drag and drop a function goal into the drawing zone, which opens a dialog box for creating functional goals, as shown in Figure 11. Enter the name of the requirement, *landing retracting sequence* and click `Validate`[7]. You can resize the graphic to make it clear.

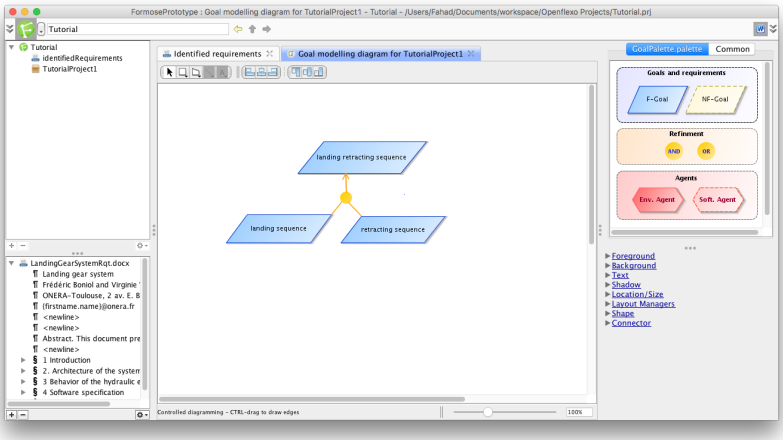

Figure 12: Goal Model

10. You can drag and drop further goals to refine this requirement into *landing sequence* and *retracting sequence* goals. These two goals need to be linked to the initial requirement through the refinement links and the *AND* connector. In order to do this, hover over the child goal and grab the arrow sign and drag it to the parent goal. This will create the connector and a link between the child goal and

[5] For the moment, we are using KAOS goal models, but the future versions would be incorporating SysMLKAOS modeling diagrams as well

[6] In the current version, we are jumping to the goal refinement directly. In future versions, we shall link requirements to project elements as well. This step of linking to project elements might come before goal modeling

[7] Next version of the tool shall provide a list of requirements to attach to a functional goal, e.g. from a drop down box and will populate other fields automatically

the connector. Now hover over the connector and drag the shown arrow to the parent goal. This will connect the child goal to the parent goal with a refinement link. To add the second child goal, hover over the child goal and drag the arrow to the connector. This shall result in an AND refinement of the parent goal into two child goals, as shown in Figure 12.

11. In case, two refinements of a goal are using *OR* connector, continue to refine the first child goal first. For the second child goal, connect it directly to the parent goal, instead of connecting to the connector. This will create a new connector. Then connect that connector with the parent goal.

6 Behind the Curtain

Requirements engineering serves as an interface between a system and its environment. It focuses on gathering the information (domain models) from the environment by collecting the requirements for the system to be developed. So, the artifacts and processes of requirements engineering are tightly related to both the system under development and its environment. In the context of requirements engineering, sharing information between these processes and artifacts is handled through different activities, ranging from manual (*e.g.* feasibility study, requirements elicitation techniques, *etc.*) to semi-automatic (*e.g.* requirements validation, requirements management, *etc.*). These artifacts and other information resources used or produced by these activities often share cross-cutting concerns. These cross-cutting concerns are captured through traceability links between them, which allow to traverse from initial phase artifacts to later ones (*forward traceability*) or from later phase artifacts to initial ones (*backward traceability*). Our approach is to use model federation to connect these artifacts and information resources.

6.1 Model Federation

Model federation is an approach that provides the means to integrate multiple models conforming to different paradigms. Different techniques like model merging, profiling or extension mechanisms provided by UML, model weaving, or other model composition approaches allow integrating models from the same paradigm to get a different views for each stakeholder. However, it becomes difficult, when models are conforming to different paradigms. Model federation allows the integration of heterogeneous models to develop new cross-concern viewpoints/models or to synchronize the models used for designing a system. Contrary to other techniques that transform, merge or compose the models in a common paradigm, it keeps models in their respective paradigms to avoid redundancies.

The implementation of model federation by Openflexo uses multiple modeling spaces *i.e. conceptual*, *technological* and *design space*, as shown in Figure 13. The different modeling spaces of our approach serve together for the development of a complex system. The *conceptual space* is where new models or views are developed by federating the concepts from already existing models. These federated models are called *virtual models*. As virtual models reuse the concepts from various models conforming to different paradigms, the conceptual space is surrounded by multiple technological spaces. Each of these *technological spaces* is a collection of models adhering to a common paradigm which gives a common ground for their interpretation. Usually, the technological models are already developed to serve some specific concerns. They might even belong to some different system. Finally, a *design space* is a specific kind of technological space that serves for diagrammatic representations of the virtual models, using the same interaction mechanism.

Model federation is realized through virtual models to develop new concepts. We have developed a generic modeling language to define virtual models in the conceptual space. They are developed using features that can either be specifically defined for virtual models or by reusing elements of models from any of the existing technological spaces. Each of these features, serving as a building block for a virtual model, is called a *flexo concept*. A virtual model is responsible for managing the lifecycle

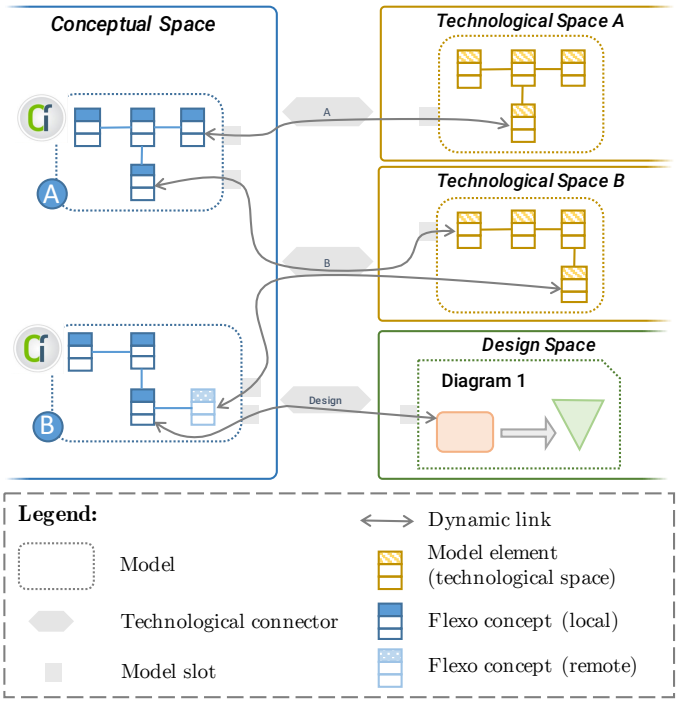

Figure 13: Mapping between concepts in modeling spaces

of all the flexo concepts that it contains. While virtual models follow the formalisms defined by our methodology, technological models follow their own paradigm depending on their specific technological space. A virtual model can not access the elements of a technological model, unless it can interpret the formalisms used in its technical space. This is done using a connection between the technological space and the conceptual space, realized as *technological connectors*. These connectors allow access for reading, writing and synchronizing the information between the virtual models and the technological models. Once developed, a virtual model can be serialized back into a new or existing technological space for further development.

A virtual model is composed of flexo concepts. Some of these flexo concepts are defined specifically for the development of a virtual model, we call them *local flexo concepts*. Virtual models can also use certain elements from connected technological models. One way is to create a dynamic link between these local flexo concepts and the modeling elements of technological models. `Virtual model A` shown in Figure 13 uses two dynamic links; one to the model in `technological space A` and the other to the one in `technological space B`. The second way of using model elements from technological spaces is to translate them into flexo concepts. A *remote flexo concept* of a virtual model is a translated modeling element from a linked technological space, serving as a 'local proxy'. `Virtual model B` shown in Figure 13 contains a remote flexo concept, which is a local 'proxy' of a model element from `technological space B`. In both these techniques, the link between the flexo concept and the technological model is bi-directional and is maintained to synchronize any changes. A virtual model in the conceptual space can be updated when a corresponding technological model evolves and conversely, a technological model can also be updated, once the corresponding virtual model is

modified. As the information is shared between multiple models, it might be accessed and modified by multiple stakeholders at the same time. We do not enforce any specific consistency model in this case, we rather give this flexibility to the designers to implement any consistency model that suits their specific application. We provide a notification system that notifies the designer/stakeholder whenever a corresponding model is updated. One can choose for automatic synchronization of models, but then human intervention would be needed for resolving conflicts. Design space is a specific technological space provided by our methodology for diagrammatic representations of the virtual models. A technological connector between conceptual space and design space allows linking the flexo concepts to their graphic representations. These bi-directional links are maintained, so a virtual model can be edited either from design space or the conceptual space.

The technological connectors allow models in conceptual space to access the models of technological spaces, however the actual dynamic link between these models is realized using *model slots*. Our approach of model federation can be explained through the analogy of components based design, where models serve as components, model slots as component interfaces and technological connectors as connectors. Technological connectors allow the creation of a dynamic link between a virtual model and a technological model using model slots. A model slot defines an access point both for a virtual model and a technological model to link them together. It exposes a view on the structural and behavioral contents of the technological model to the virtual model and vice-versa. Once linked to a technological model, a virtual model can read/write to its attributes using *roles* and execute the actions using *edition actions*. Roles and editionActions are to a flexo concept, what attributes and methods are to a class, except that roles and editionActions are associated to attributes and methods of a model adhering to an entirely different paradigm.

Taking the example of the requirements specification model for an embedded system, as explained in the introduction section, we can connect it to cost analysis spreadsheet[8] to prioritize requirements according to a cost-value approach. In order to realize a cost-value requirements priority model, we need to develop it as a virtual model in the conceptual space. The conceptual space needs to be connected to two technological spaces *i.e.* MS Excel space for cost analysis spreadsheet and MS Word space for requirements specification document[9]. Both these technological connectors are already available through our tooling support. Using these connectors, we can develop our virtual model with two flexo concepts *i.e.* `Requirement` and `Priority`. `Requirement` can be implemented as a remote flexo concept, thus making a 'proxy' for each requirements in the virtual model that is dynamically linked to the requirements of the specification document. `Priority` concept can directly be linked to the cost analysis document to assign a priority value to each requirement. Model slots for these links need to be developed by the modeler to connect these concepts. Even though it is not a good idea to update cost analysis document from this model, for the purpose of explanation we assume that the designer has the possibility to update/modify the cost of each requirement in the cost analysis document from the virtual model, using the edition action setCellValue(). Furthermore, any other common

[8] .xlsx files are XML based spreadsheets. We consider all artifacts in software development as models.

[9] Different tools like IBM Rational RequisitePro, CaseComplete, Visure requirements *etc.* share requirements based on MS Word format.

piece of information can be synchronized among cost-value requirements priority model, requirements specification document and cost analysis document. The cost-value requirements priority model can be developed using a graphical editor of design space, hence editing/synchronizing the flexo concepts in the virtual model.