Addressing Modularity for Heterogeneous Multi-model Systems using Model Federation

Fahad R. Golra Antoine Beugnard
Fabien Dagnat
IRISA / Télécom Bretagne, Institut Mines-Telecom
Brest, France
fahad.golra, antoine.beugnard, fabien.dagnat
@telecom-bretagne.eu

Sylvain Guerin Christophe Guychard
Openflexo
Brest, France
sylvain.guerin, christophe.guychard @openflexo.org

Abstract

Model-Driven Engineering (MDE) proposes to modularize complex software-intensive systems using multiple models where each module serves a specific concern. These concerns of a system might be diverse and the use of multiple heterogeneous models often becomes inevitable. These models adhere to different paradigms and use distinct formalisms, which makes it hard to ensure consistency among them. Moreover, these models might contain certain concepts (at times overlapping) that are reused for building cross-concern views/models. Maintaining models using separation of concerns in a heterogeneous modeling space becomes difficult. Traditional MDE suggests the use of model transformations to maintain the mappings between heterogeneous models. In this paper, we introduce a different approach based on model federation to map heterogeneous models. In contrast to traditional approaches where heterogeneous models are gathered in a single technological space, model federation keeps them in their own technological spaces. We provide a mechanism so that elements of these models are accessible for the development of cross-concern views/models from their respective technological spaces.

Categories and Subject Descriptors D.2.2 [*Design Tools and Techniques*]: Computer-aided software engineering (CASE); D.2.2 [*Design Tools and Techniques*]: Modules and interfaces; D.2.13 [*Reusable software*]: Reuse models

Keywords Model Driven Engineering, Model Federation, Modularity, Multi-model systems, Abstraction

1. Introduction

Models form the core of software development processes in the context of Model Driven Engineering (MDE). This increasingly important role of modeling has led to a plethora of domain specific modeling languages (DSLs/DSMLs) (Mernik et al. 2005). Each DSL captures the specific knowledge involving a certain domain to provide adapted features. This specialization contributes to improved efficiency and enhanced quality of systems. The design of complex software-intensive systems might require several DSLs to handle various concerns. Heterogeneity of models in the *modeling space* leads to multiple issues for synchronizing models, maintaining traceability and managing global consistency of the system. Complex software intensive systems usually need to integrate the viewpoints of various stakeholders and system experts (Tyree and Akerman 2005). These viewpoints of different stakeholders might be related to multiple models, each taking care of a specific concern. Adding those *cross concerns* to the equation makes modeling of these systems even more difficult. One way to handle this complexity is to improve the modularity of the modeling space that already seems modular because of multiple heterogeneous models.

Considering this situation, current approaches are typically oriented towards translation/model transformations to bring all DSLs under one operating paradigm (Eclipse 2016; Jézéquel et al. 2011), composition of (meta)models (France et al. 2007; Eker et al. 2003; Hardebolle and Boulanger 2008; Kong and Alexander 2003; Berg and Møller-Pedersen 2015) and unification through intermediate languages (Vernadat 2002). All these endeavors aim at providing a single language, (meta)model, modeling paradigm to cover all heterogeneous concerns that one can come across in the development of a complex software system. We argue that these DSLs should be kept in their own paradigms and modeling element redundancies should not be created by transforming them to a single paradigm through integration, composition or unification. A better approach would be to create bridges between these distinct paradigms and maintain a dynamic link so that these different models can be synchronized for global consistency, traced from one another and used for the development of cross-concern viewpoints/models.

It might seem that our view of modularity is the antithesis of *separation of concerns*, which advocates loose coupling between the modules. In reality, taking care of the cross-cutting concerns in multiple models of a modeling space ensures a better modularization, much like aspect oriented modeling (Masuhara and Kiczales 2003). These models can have considerable overlapping features that need to be synchronized for the global consistency. *Multi-Dimensional Separation of concerns (MDSOC)* defines modularity in multi-dimensional spaces of units, where dimensions represent different modularizations (Tarr et al. 1999). We consider two dimensions of modularity for our proposal. The *structural modularity* places all features, primarily serving for a common concern, in a single model (a module). The *conceptual modularity* looks for the projections of secondary concerns of these features in conceptual space and creates modules out of the overlapping or tightly coupled features from different models. A multi-model system is structurally well modularized because each model serves a specific concern. However, the overlapping features and tight semantic coupling between them in the context of heterogeneous models, call

MODULARITY Companion'16, March 14–17, 2016, Málaga, Spain
© 2016 ACM. 978-1-4503-4033-5/16/03...$15.00
http://dx.doi.org/10.1145/2892664.2892701

for conceptual modularity as well. For the system to work as a whole, well-defined interfaces should guarantee the contracts between these modules. By modularizing complex software-intensive systems in both these dimensions, we expect strong impacts on the reduction of complexity for modeling heterogeneous systems. We believe that our approach will favor reuse, synchronization, flexibility, maintainability and conceptual alignment of existing/legacy (meta)models in the modeling space.

The rest of this paper is organized as follows. First, we present our approach of model federation in Section 2. Then, in Section 3, we explain the proposed approach in the context of modularity for the design of multi-models systems that face the problem of heterogeneity. Then, Section 4 discusses the state of the art in multi-paradigm modeling in the context of modularity. Finally, we conclude this paper in Section 5.

2. Model Federation

Software Industry needs methods for seamless integration of specialized formalisms and tools that they use for the development of complex software systems. These specialized formalisms, tools and DSLs used in the development process add to the heterogeneity of modeling space. These models needs to be synchronized for the global consistency of the system. Synchronization of different models in the modeling space is handled through the co-evolution techniques (Hebig et al. 2013). However when these models belong to different paradigms, their co-evolution becomes difficult. Before going into the details of how we propose to deal with these issues, lets take a look at the concepts of model federation.

Model federation provides a mechanism for the integration of heterogeneous models. This integration of models may serve to develop new cross-concern viewpoints / models from several existing models or to synchronize the models that are used in the modeling space for designing a system. We propose to keep heterogeneous models in their own paradigms. Hence, we divide the modeling space in two distinct spaces *i.e.* the *technological space* and the *conceptual space*, as shown in the Figure 1. Technological spaces are defined environments that allow heterogeneous models to use their own paradigm, formalism and tooling within their private modeling space. For the design of a complex system, a conceptual space may be surrounded by many technological spaces. Each technological space contains the models that follow a specific modeling paradigm to serve specific concerns. A conceptual space is where a set of models can be federated to develop a new cross concern viewpoint/model, called a *virtual model*. Virtual models are made up of features, which are reused from the models of technological space. New features can also be developed for the virtual model, which do not belong to any of the existing technological spaces. Each of these features, called a *flexo concept*, serves as a building block for a virtual model. A virtual model is responsible for managing the life cycle of the flexo concepts that it contains. A virtual model can be serialized back into a new or existing technological space for the development of a complex system.

The virtual models in the conceptual space follow the formalisms defined by our methodology. We have developed a generic modeling language to define virtual models. Technological models follow their own paradigm or formalisms depending on their specific technological space. A virtual model can not access the features of a technological model, unless it knows what paradigm or formalism it is following. A connection is needed between these spaces for making the features of a technological model available in the conceptual space. A connection between a specific technological space and the conceptual space is realized using technological *connectors*. These connectors allow access for reading, writing and eventually synchronizing the information between virtual models and the technological models.

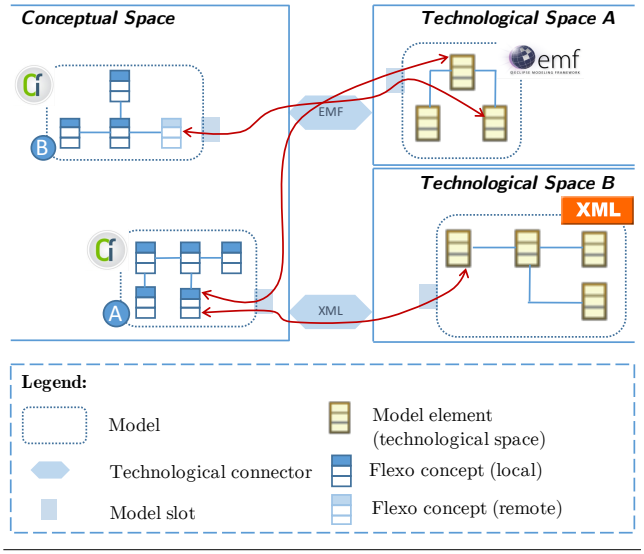

Figure 1. Modeling space in Openflexo

Once a technological connector is available for a specific technological space, all the models in that space and their respective elements become accessible from within the conceptual space. A virtual model is composed of the concepts precisely developed for it and the ones reused from other technological models. The concepts specifically developed for the virtual model are the local flexo concepts. The concepts from a technological model can be reused in two distinct ways. First, the designer can create a dynamic link between a local flexo concept and the technological model concept that it wants to use, as shown in *virtual model A* of Figure 1. Once created, this connection is maintained to synchronize any further changes. Second, the designer can also choose to create a local 'proxy' of that concept in the virtual model by translating it into a remote flexo concept. A remote flexo concept, as depicted in *virtual model B* of Figure 1, is a proxy of the technological model concept and thus introduces some redundancy. We suggest and prefer to use the first method of maintaining a dynamic link. Even in the second method, a link is maintained between the local proxy and the remote feature to ensure their synchronization.

The connection between the conceptual and technological spaces is bi-directional. A virtual model in the conceptual space can be updated when a corresponding technological model is evolved and conversely, a technological model can also be updated, once the corresponding virtual model is modified. The mechanism for updating the information in both directions is kept flexible. The designer can choose to get notified when a corresponding model is evolved or to get automatically synchronized, depending on the semantics defined in the technological connector.

A *model slot* is the modeling element that realizes a connection between a virtual model and a technological model. As already explained, technology connectors connect the conceptual space with a specific technological space. These connectors allow the creation of a dynamic link (using model slots) between a specific virtual model and a specific technological model of the connected technological space. If a virtual model is considered as a component, model slot serves as its interface. The model slot exposes a view on the structural and behavioral contents of the technological model to the virtual model. Once a virtual model is connected, it can read/modify the attributes using *roles* and execute the actions using *editionAction*. Roles and editionActions are to a flexo concept, what attributes and methods are to a class, except that roles and editionAc-

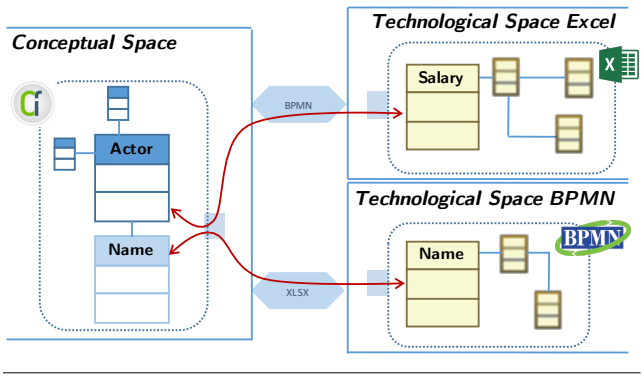

Figure 2. Mapping between concepts

tions are associated to attributes and methods of a model adhering to an entirely different paradigm. An example of this scenario is a flexo concept `Actor`, whose `name` is taken from a BPMN process model and his `salary` is taken from an xlsx[1] file, as shown in Figure 2. `Actor` is a local flexo concept. The `name` from the BPMN model is translated into a remote flexo concept, which makes it accessible to the virtual model. A dynamic link between the flexo concept (`name`) and the technological model concept (`name`) keeps them synchronized. `Actor` is also connected with the `salary` in the excel file using another dynamic link. In this example, `actor` in the virtual model can update the `salary` in the technological model using editionAction `setCellValue()`. Furthermore, any common piece of information can be synchronized between the BPMN process model and the excel file.

Tooling support for our methodology is provided through our open source project initiative, Openflexo (Openflexo 2016). This tool started as a business process modeling workbench, but by integrating our recent research endeavors it has evolved into a generic collaborative platform for multifaceted modeling. It provides support for model federation across multiple technological spaces. It uses the concept of model federation and introduces connectors for diagramming to support both graphical and textual models.

3. Modeling Space Modularity

A modeling space with heterogeneous models seems to be modular, because each model affects a distinct concern for the system under development. Our methodology places all those heterogeneous models in multiple technological spaces. Each technological space adheres to a specific paradigm, so a technological space may end up containing multiple models. A separate conceptual space allows the development of new models. Figure 3 illustrates four different perspectives from a scenario that we are going to use as a running example to explain the key notions of our methodology. In Figure 3a, a technological model for missile system and another for a radar system are placed in their respective technological spaces `TS-M` and `TS-R`. Placement of both these models in separate technological spaces illustrates that they use different paradigms. A virtual model for the development of a ship is under-development in the conceptual space, `CS`. To this point our modeling space is modular, all three models are serving their specific concerns and it does not really matter if they belong to different paradigms. We argue that this modularity of the modeling space is structural *i.e.* all related modeling elements are placed in a single model to take care of a particular concern. This specific concern may be quite different from other concerns, and thus a specialized modeling language, a DSL, might be required. So lets consider that the models of radar system and missile system are developed in two distinct DSLs. The user wants to develop a model for ship, which shall use the modeling language described by our language. Once developed, it is possible to serialize the ship model to any other technological space.

Development of some specialized views of the system or new models might need to access multiple modeling elements from different existing models belonging to distinct technological spaces. These new views/models are created as virtual models in the conceptual space and the technological models are considered as their resources. As resources, they allow the virtual models to extract information from them. Connecting information from varying resources allows the generation of new meaningful concepts. Modeling the concepts, whose building blocks are coming from multiple resources, needs to "cross-cut" the existing heterogeneous models. This seems to create additional couplings between the modular architecture of modeling space with models serving as distinct modules. In reality, these couplings between the modules already exist at a conceptual level, they are just reified in our approach. These existing semantic links are shown in Figure 3b.

Using multi-dimensional separation of concerns, we propose to use structural and conceptual dimensions for modularity. Figure 3a depicts a structural modularization of the modeling space. Structural modularization means that each modeling element is placed in a single structure, a model in our case, to achieve its primary concern. The other concerns of a modeling element can be projected to the conceptual dimension, which are used to identify meaningful semantic links between the modeling elements of different models. These semantic links might, for example, represent that two modeling elements from models following different paradigms are conceptually identical. In our scenario, the modeling element representing weight in `TS-M` and `TS-R` is identical, as shown in Figure 3b. Using such conceptual links between different modeling elements, our methodology can ensure a conceptual alignment of their respective models. Some semantic links between modeling elements of different paradigms might not make sense in their respective technological spaces. However combining those elements in a common conceptual space might yield new concepts. For example, the modeling element representing a *missile* in `TS-M` when linked to the modeling element representing *radar* in `TS-R` can help develop a model for a *battleship* that uses both missiles and radars. Figure 3b depicts this case, where the conceptual space is in need of these two concepts and a third concept of weight which is identical between the two technological models.

We like the definition of modularity as, "*a continuum describing the degree to which a system's components can be separated and recombined*" (Schilling 2000). It has two main considerations: 1) the partition of a system into smaller modules which serve specific concerns and, 2) to establish an interaction contract between them, so that they can serve as one whole. Figure 3c shows our scenario where our virtual model can be developed by reusing the concepts *missile*, *radar* and *weight* in `TS-M` and `TS-R`. This figure shows the conceptual projection of `TS-M` and `TS-R` on the `CS`. The features of the virtual model, depicted in grey, are the conceptual projections of the model elements that are still placed in their respective technological spaces. This conceptual projection means that the features of radar and missile models continue to serve their primary concern in their respective models. However their projection on the virtual model allows to use the secondary concerns of those features. It seems as if instead of modularizing the system, the proposed methodology is actually combining it. The actual architecture of the system after conceptual modularity is better represented by Figure 3d. Here, the original modules are kept intact and the development of the virtual model in the conceptual space is carried out by linking it to the cross concerns from other models.

[1] XML based spreadsheet file. We consider all files as models.

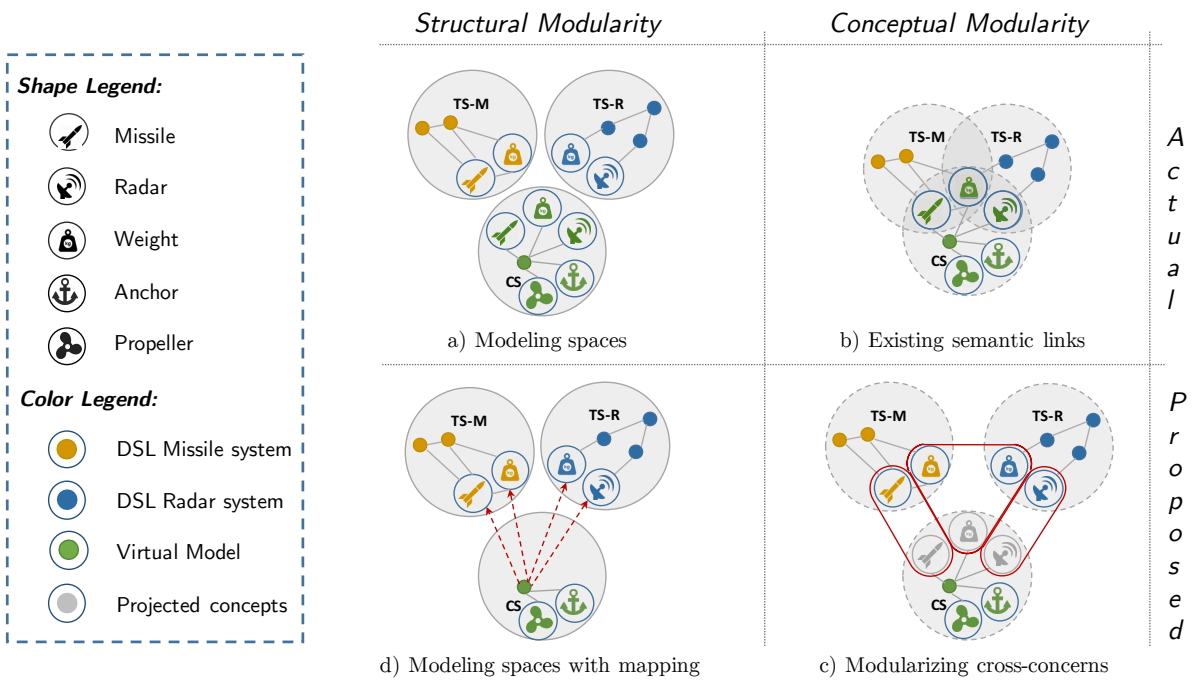

Figure 3. Mapping from conceptual space to technological space.

The proposed methodology provides mechanisms for establishing a dynamic link between the virtual model and the technological models. These mechanisms are offered by the use of technological connectors and model slots, as explained in the previous section. The overlapping nature of a cross-cutting concern forms an inherent coupling between models, and model federation offers a mean to conceptualize it in a single module, the *virtual model*.

Figure 4 describes the informal process to use model federation to create mappings between heterogeneous models. The development of a virtual model starts by developing local flexo concepts according to the specifications. This produces a virtual model has all local flexo concepts and needs the cross cutting concerns from other models for its completion, as shown in Figure 4a. Then the concepts needed to complete the virtual model are identified in the available technological spaces, as shown in Figure 4b. Finally, mapping are created between the flexo concepts and the concepts of technological models, based on either dynamic links (Figure 4-c1) or developing remote flexo concepts as proxies for the technological model elements(Figure 4-c2).

Software requirements evolve over time, resulting in model evolution for design and subsequent phases. It becomes hard to maintain multiple heterogeneous models in the modeling space when they are evolving over time. Evolution of a source model results in the evolution of all the target models. When traditional model driven approaches like transformations or model compositions are used, these target models need to be ”regenerated”. A typical drawback of model transformation and model composition techniques is that they do not allow independent modifications to individual parts of source/target models. The modeling elements of the output models in these techniques do not maintain any dynamic link with the corresponding elements of input models. Model federation does not adhere to the concepts of source and target models. It rather advocates the synchronization of multiple federated models. In case of evolution, the complete model does not need to be regenerated, only the concerned elements are updated. If a model in a specific technological space evolves over time, changes are synchronized to the virtual model and vice versa. The user remains in control whether to opt for automatic synchronization or to demand an alert-based configuration to remain in control. Openflexo platform takes care of maintaining the dynamic link between the heterogeneous models and their synchronization. Simultaneous changes to common parts of the models at both ends can cause conflicts. For the moment, it is the designers responsibility to resolve such conflicts. However, we are looking forward to develop some mechanism, inspired from version control and content management systems, that can assist the designer for resolving such conflicts.

4. Related Approaches

There are several approaches that address the integration challenges for domain viewpoints while trying to keep the modeling space as modular as possible. With new DSLs coming to prominence, the problem of model heterogeneity is being highlighted more than ever before. The most commonly used approach is the UML profiling mechanism that allows the customization of UML for any specific domain (Selic 2007). Even though this approach is rather easy to implement with case tool customizations using UML profiles, it has its drawbacks. This technique forces us to the world of UML and MOF conformity, which might not be an efficient solution for many heterogeneous models. For example, we have come across models for process engineering that do not adhere to MOF paradigm, and so its impossible to extend them using UML profiles. Furthermore the designer will face the usual problems linked to UML profile mechanisms (France et al. 2006). There are other approaches that deal with model heterogeneity by extending existing metamodels, like EMF facets (Eclipse 2016) and KerMeta (Jézéquel et al. 2011), but they pose the same problems of MOF conformity. EMF facets provide a non-intrusive extension mechanism to add additional attributes, references or operations on a model without modifying it or its metamodel *i.e.* these extensions can be loaded or unloaded on the fly (Eclipse 2016). On the other hand, KerMeta provides the functionality for weaving concerns to existing metamodels using aspect-oriented metamodeling

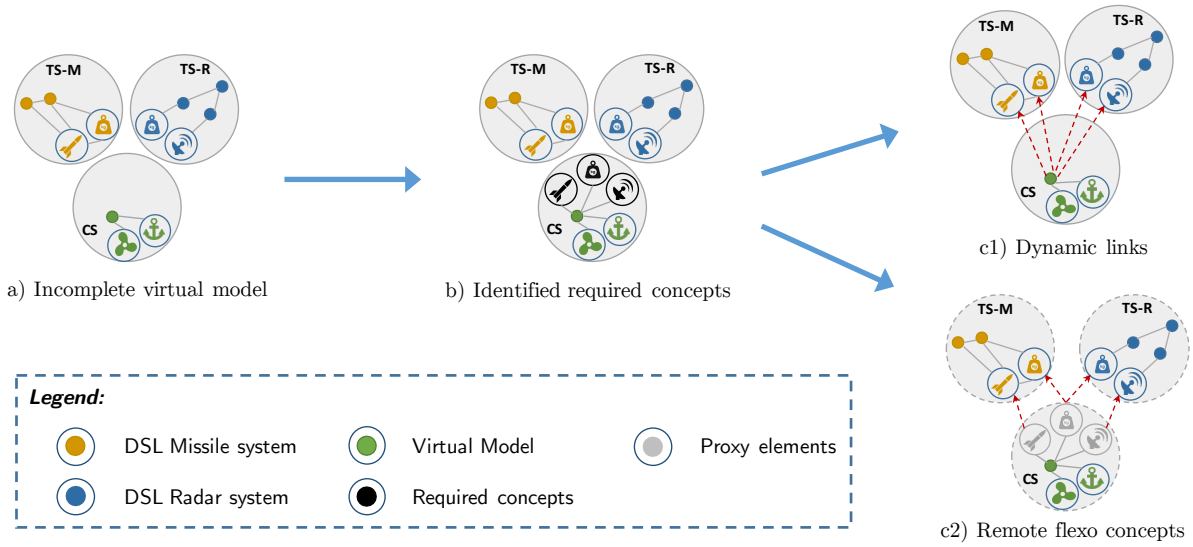

Figure 4. Model federation process for modularization

(AOM) (Jézéquel et al. 2011). Besides this, it also allows the definition of execution operations (semantics) for the elements of its meta-model at a "meta" level, thus providing the flexibility to manipulate any model which conforms to the metamodel. Generally, these profiling/metamodel extension mechanisms are quite restrictive to force conformity to their specific metamodels/MOF, which makes it difficult to handle multi-paradigm modeling languages that do not adhere to their specific paradigm. This issue is known to the OMG community and thus, MEF RFP solicits proposals for a mechanism of extending and integrating any metamodel without confining to UML (OMG 2011). We think that our proposal can be a good candidate to respond to this RFP.

There are other approaches that provide ways to combine several (meta)models into a single modeling space. Some of the techniques consider the composition of metamodels as a viable solution for addressing heterogeneity in multi-model systems (France et al. 2007). These techniques use several operator (equivalence, merge, etc.) to assemble the modeling elements of multiple metamodels in a single composite metamodel. However such approaches are not scalable, because they require modification of the composite metamodel, every time a new paradigm is added to the modeling space. Furthermore, a single composite metamodel is not an optimal way to modularize the modeling space even for *structural modularity*. On the contrary, our approach is scalable because of the modular structure and synchronization mechanisms. Some other techniques that use composition of models instead of metamodels, to deal with model heterogeneity are Ptolemny II (Eker et al. 2003), ModHel'X (Hardebolle and Boulanger 2008) and Rosetta (Kong and Alexander 2003). Ptolemy II and ModHel'X organize heterogeneous models in multiple hierarchical layers, where each layer corresponds to one modeling paradigm. Ptolemy II uses a white-box composition, whereas ModHel'X uses a black-box composition. Even though hierarchical composition of heterogeneous models seems to be a good modular proposition, it has its drawbacks. It results in redundancy of modeling elements when cross concern concepts are modeled in their own technological spaces and then again in the composite model. Managing modularity with added redundancy and without synchronization mechanisms for redundant concepts may not be a viable solution, specially if the technological models are actively used in their own technological spaces. Approaches that allow the composition of both models and metamodels together (Berg and Møller-Pedersen 2015), experience the same issues faced by other (meta)model composition approaches.

Approaches like megamodels for model management provide structures for managing the global relationships (decomposition, representation, belonging, conformance, transformation and semantics) between models, metamodels, modeling languages and the transformation definitions in a modeling space (Favre and NGuyen 2005). Megamodeling and model weaving techniques together, allow the definition of semantic relationships between (meta)models (Jouault et al. 2010). This approach can be used to synchronize heterogeneous models but provides no support for the definition of cross concern models from them. Similarly, methods for automatic co-evolution of metamodels and models deal with the issue of global consistency in a modeling space (Hebig et al. 2013), but they do not consider the possibility of defining new concepts by identifying the semantic links between the elements of heterogeneous models.

Another recent approach, EMF Views, uses existing heterogeneous models to extract new views from them (Bruneliere et al. 2015). Inspired from SQL, it offers three constructs (*select*, *project* and *join*) for selecting some elements from the models, projecting chosen properties from those elements and joining them together to form new virtual (meta)models. It also defines *weaving models* to describe the links between the elements of different models. This approach comes closest to our definition of conceptual modularity. The SQL based DSL of EMF Views creates light-weight proxies of the remote modeling elements, using existing transformation approaches. We also allow the creation of proxies in our virtual models, however our preferred approach is to link the *flexo concepts* directly to the concepts placed in their own technological spaces. Moreover we have developed a graphical editor that provides access to the concepts of multiple models in their own technical spaces. Hence, our tooling support (*Openflexo*) provides an intuitive way of mapping the concepts between conceptual and technological spaces.

5. Conclusions

This paper addresses the issues faced by software designers when heterogeneous models are used during the design phase of complex software-intensive systems. We proposed a modular approach

to handle such issues using model federation. The underlying concepts of model federation were explained, with a particular emphasis on modularity. Modularity of multi-concern systems is presented in two dimensional perspective *i.e.* structural modularity and conceptual modularity. Modeling elements are usually combined together to form a complete model that caters for a specific concern, which we call structural modularity. However, the same modeling elements might serve other secondary concerns, in respect to the model that they are placed in. Our approach proposes a mechanism to group these secondary concerns together to form a distinct model serving a particular concern, keeping modularity in mind. Model federation is particularly addressing modularity for heterogeneous modeling spaces, where multiple models come from different technological spaces. A limitation of our approach is the availability of different technological connectors. We have already developed the technological connectors for various paradigms (EMF, OWL, XML/XSD, MS Office document formats, *etc.*). Associated tooling is implemented as an open source project, Openflexo. This enables the users to develop new technological connectors for their specific needs. We are looking forward to develop a high level language that can replace Java code to *type* the interfaces of the modules for defining the contracts.

Acknowledgments

This research work is a result of a collaboration between PASS team and Openflexo under FORMOSE project. We are thankful to *Agence Nationale de la Recherche, ANR* for funding this project.

References

H. Berg and B. Møller-Pedersen. Towards non-intrusive composition of executable models. In *3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 1–11, Feb 2015.

H. Bruneliere, J. G. Perez, M. Wimmer, and J. Cabot. Emf views: A view mechanism for integrating heterogeneous models. In *Conceptual Modeling: Proceedings of the 34th International Conference, ER 2015, Stockholm, Sweden, October 19-22, 2015*, pages 317–325. Springer International Publishing, 2015. ISBN 978-3-319-25264-3. doi: 10.1007/978-3-319-25264-3_23. URL http://dx.doi.org/10.1007/978-3-319-25264-3_23.

Eclipse. EMF Facet. http://www.eclipse.org/facet/, 2016. Online; accessed: 2016-02-10.

J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - The Ptolemy approach. *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, 91(1):127–144, Jan 2003. ISSN 0018-9219. doi: 10.1109/JPROC.2002.805829. URL http://dx.doi.org/10.1109/JPROC.2002.805829.

J.-M. Favre and T. NGuyen. Towards a megamodel to model software evolution through transformations. *Electronic Notes in Theoretical Computer Science*, 127(3):59 – 74, 2005. ISSN 1571-0661. doi: http://dx.doi.org/10.1016/j.entcs.2004.08.034. URL http://www.sciencedirect.com/science/article/pii/S1571066105001398. Proceedings of the Workshop on Software Evolution through Transformations: Model-based vs. Implementation-level Solutions (SETra 2004).

R. France, S. Ghosh, T. Dinh-Trong, and A. Solberg. Model-driven development using UML 2.0: promises and pitfalls. *Computer*, 39(2):59–66, Feb 2006. ISSN 0018-9162. doi: 10.1109/MC.2006.65. URL http://dx.doi.org/10.1109/MC.2006.65.

R. France, F. Fleurey, R. Reddy, B. Baudry, and S. Ghosh. Providing support for model composition in metamodels. In *11th IEEE International Enterprise Distributed Object Computing Conference, 2007. EDOC 2007.*, pages 253–265, Oct 2007. doi: 10.1109/EDOC.2007.55. URL http://dx.doi.org/10.1109/EDOC.2007.55.

C. Hardebolle and F. Boulanger. ModHel'X: A Component-Oriented Approach to Multi-Formalism Modeling. In H. Giese, editor, *Models in Software Engineering*, volume 5002 of *LNCS*, pages 247–258. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-69069-6. doi: 10.1007/978-3-540-69073-3_26. URL http://dx.doi.org/10.1007/978-3-540-69073-3_26.

R. Hebig, H. Giese, F. Stallmann, and A. Seibel. On the Complex Nature of MDE Evolution. In A. Moreira, B. Schätz, J. Gray, A. Vallecillo, and P. Clarke, editors, *Model-Driven Engineering Languages and Systems*, volume 8107 of *LNCS*, pages 436–453. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-41532-6. doi: 10.1007/978-3-642-41533-3_27. URL http://dx.doi.org/10.1007/978-3-642-41533-3_27.

J.-M. Jézéquel, O. Barais, and F. Fleurey. Model Driven Language Engineering with Kermeta. In J. M. Fernandes, R. Lämmel, J. Visser, and J. Saraiva, editors, *Generative and Transformational Techniques in Software Engineering III: International Summer School, GTTSE 2009, Braga, Portugal, July 6-11, 2009. Revised Papers*, pages 201–221. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-18023-1. doi: 10.1007/978-3-642-18023-1_5. URL http://dx.doi.org/10.1007/978-3-642-18023-1_5.

F. Jouault, B. Vanhooff, H. Bruneliere, G. Doux, Y. Berbers, and J. Bezivin. Inter-DSL Coordination Support by Combining Megamodeling and Model Weaving. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2011–2018, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-639-7. doi: 10.1145/1774088.1774511. URL http://doi.acm.org/10.1145/1774088.1774511.

C. Kong and P. Alexander. The Rosetta meta-model framework. In *Proceedings of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, pages 133–140, April 2003. doi: 10.1109/ECBS.2003.1194792. URL http://dx.doi.org/10.1109/ECBS.2003.1194792.

H. Masuhara and G. Kiczales. Modeling Crosscutting in Aspect-Oriented Mechanisms. In L. Cardelli, editor, *ECOOP 2003 - Object-Oriented Programming*, volume 2743 of *LNCS*, pages 2–28. Springer, 2003. ISBN 978-3-540-40531-3. doi: 10.1007/978-3-540-45070-2_2. URL http://dx.doi.org/10.1007/978-3-540-45070-2_2.

M. Mernik, J. Heering, and A. M. Sloane. When and How to Develop Domain-specific Languages. *ACM Computer Survey*, 37(4):316–344, Dec. 2005. ISSN 0360-0300. doi: 10.1145/1118890.1118892. URL http://doi.acm.org/10.1145/1118890.1118892.

OMG. MEF RFP, Metamodel Extension Facility. http://www.omg.org/public_schedule, June 2011. Online; accessed: 2016-02-10.

Openflexo. Openflexo project. http://www.openflexo.org, 2016. Online; accessed: 2016-02-10.

M. A. Schilling. Toward a General Modular Systems Theory and Its Application to Interfirm Product Modularity. *The Academy of Management Review*, 25(2):312–334, 2000. ISSN 03637425. URL http://www.jstor.org/stable/259016.

B. Selic. A Systematic Approach to Domain-Specific Language Design Using UML. In *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC '07*, pages 2–9, May 2007. doi: 10.1109/ISORC.2007.10. URL http://dx.doi.org/10.1109/ISORC.2007.10.

P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton, Jr. N Degrees of Separation: Multi-dimensional Separation of Concerns. In *Proceedings of the 21st International Conference on Software Engineering*, pages 107–119, New York, NY, USA, 1999. ACM. ISBN 1-58113-074-0. doi: 10.1145/302405.302457. URL http://doi.acm.org/10.1145/302405.302457.

J. Tyree and A. Akerman. Architecture Decisions: Demystifying Architecture. *IEEE Software*, 22(2):19–27, 2005. ISSN 0740-7459. doi: 10.1109/MS.2005.27. URL http://doi.ieeecomputersociety.org/10.1109/MS.2005.27.

F. Vernadat. UEML: Towards a unified enterprise modelling language. *International Journal of Production Research*, 40(17):4309–4321, 2002. doi: 10.1080/00207540210159626. URL http://dx.doi.org/10.1080/00207540210159626.