Adapting Heterogeneous ADLs for Software Architecture Reconstruction Tools

Dung Tien Le

Thai German Graduate School of Engineering,
King Mongkut's University of Technology North Bangkok
Bangkok - Thailand
Email: `le.t-sse2013@tggs-bangkok.org`

Ana Nicolaescu, Horst Lichter

Software Construction Research Group
RWTH-Aachen University
Aachen - Germany
Email: `ana.nicolaescu@swc.rwth-aachen.de`, `horst.lichter@swc.rwth-aachen.de`

***Abstract*—Architecture reconstruction tools were proposed to enable the extraction of descriptive architecture models based on prescriptive input models. A limitation of these tools is that they employ specific meta-models to which the input prescriptive models must adhere. These are often incompatible with the languages or notations that architects use in practice, leading to substantial effort to overcome terminology differences, to transform possibly already existing prescriptive models in tool-compatible ones and interpreting the results. To alleviate this problem we propose to leverage model engineering techniques in order to enable heterogeneous prescriptive and descriptive models as input and output artifacts of reconstruction tools. We exemplify our proposal by extending the Architecture Analysis and Monitoring Infrastructure (ARAMIS) - an approach developed within our previous work for the reconstruction and evolution of software architectures with a strong focus on the behavior view.**

Keywords*–*Software Architecture; Architecture Reconstruction; Model-To-Model Transformation; Architecture Description Language; Unified Modeling Language.

I. Introduction

It is generally acknowledged that the architecture greatly affects the quality of a given software and that its description is crucial to support understanding, decision making, etc. For example, Bass et al. stated that the software architecture is essential because of three main reasons: it is the basis for communication among stakeholders, it encompasses the important, early design decisions and it is a transferable abstraction of a system [1]. Because of its importance, over the years numerous attempts and even standards [2] have been proposed to support the description of the software architecture. A plethora of methods, tools and languages covering a very wide spectrum of formality were proposed and used to serve this purpose. Architects often use informal descriptions in the form of text, boxes and lines diagrams and alike but also employ more formal languages like the Architecture Description Languages (ADLs) or Unified Modeling Language (UML) when more formality is needed and/or required. Nowadays, there are more than 100 published ADLs available for use [3]. The use of UML to describe architectures has also increased, especially after the introduction of UML Profiles in UML 2.0 [4]–[7]. When considering the wide pallet of choices and the uncertainty regarding their suitability for use in a given context, it can seem natural to consider unifying these in one single highly-expressive architectural language. However, in a recent journal publication [8], Malavolta et al. stated that such an universal language is unlikely to become popular. Instead, each architectural language will be created based on specific stakeholders requirements.

Due to the numerous possibilities to describe architectures, their purpose and the various involved stakeholders, it is common that even in the same project or company, the software architecture is described differently using various tools and languages. Typically, most of the effort to document architectures is invested in the early phases of the software development process and the result thereof is the so-called ”prescriptive architecture”. Although descriptions are in later phases very useful to support the system's further development, these usually go out of date soon because of the relatively high effort that should otherwise be invested to keep them consistent to the actual architecture [9].

To approach this problem, several architecture reconstruction (AR) techniques were proposed. These aim to identify ”the descriptive architecture” which is the actual description that reflects the system's reality. In order to use these approaches, usually the architects must specify the prescriptive architecture in advance. The descriptive architecture model is then derived by correcting the prescriptive one with information extracted from the real system. However, for defining the prescriptive architecture, the architects are bound to use the meta-model of the employed AR tools [10]. These meta-models are usually stiff and cannot be extended. For example, even though the architects have initially used UML Profiles or a given ADL to describe a prescriptive architecture, if the tool that they currently want to employ only defines layers, then the architects must re-describe the architecture using only this concept. As our previous work has shown [10], this situation can lead to misunderstandings and in the end, prohibit the wide adoption of the considered AR tool. While the meta-models of other AR tools are extendible, there might still be gaps between what the architects are familiar with and the new meta-models. Furthermore, effort must be invested in order to understand and extend a given AR meta-model.

In our opinion, there is a need for reconstruction tools that address this **heterogeneity problem**. The architects should be able to model the prescriptive architecture using their familiar languages or tools. Then, by employing such an AR tool, a descriptive architecture model should be retrieved that adheres to the same meta-model as the prescriptive one. In this paper, we present an approach to extend the ARAMIS Workbench - developed during our previous work to evaluate the communication between the architecture units composing software systems - with the possibility to allow the input and output of heterogeneous prescriptive and descriptive architecture descriptions respectively.

This paper is structured as follows: in Section II, we present the ARAMIS concepts that are the foundations of the ARAMIS Workbench. Section III presents our solution to enable different types of architecture descriptions within ARAMIS. Section IV discusses the related work and Section V concludes the paper.

II. ARAMIS

The Architecture Analysis and Monitoring Infrastructure (ARAMIS) is "*a tool-supported framework for run-time monitoring, communication integrity validation, evaluation and visualization of the behavior view of software architectures*" [11], [12]. ARAMIS allows the architects to validate the communication between the hierarchies of architecture units that constitute a given system. In order to do so, ARAMIS maps extracted low-level run-time traces on architecture units and validates the mapped communication according to the rules given in the prescriptive architecture. The ARAMIS meta-model (ARAMIS-MM) [12] to which the prescriptive architecture should adhere to, although developed for flexibility is still specific. The ARAMIS Workbench offers technical mechanisms for the mapping and validation of the communication and the visualization of the result using various interactive views.

One of the major limitations of this concept is that both the prescriptive and descriptive architecture models must adhere to the ARAMIS-MM. The visualizations are also ARAMIS-specific. This leads to situations in which architects must first (1) re-describe their prescriptive, e.g., component-based diagram using the ARAMIS Architecture Modeller and then (2) interpret the result as displayed in an ARAMIS-specific visualization that has no traceability links with their prescriptive architecture model from step (1).

In order to loosen this limitation and increase the acceptability of ARAMIS, we currently work on enhancing ARAMIS so that it allows flexible input and output architecture descriptions. In such a scenario the architect would merely upload, e.g., a component diagram and receive as output the same diagram, augmented with run-time information (e.g., frequency with which one component accesses another one) and information regarding occurred architecture violations.

III. Goals and Solution Concept

Our main goals that we pursue with our approach are:

- enable the architects to reuse their prescriptive architecture models even though these might not necessarily conform to ARAMIS-MM.
- enable the generation of outputs that conform to the same meta-model as the input. Preferably, the output should be obtained by simply augmenting the prescriptive input model, in order to boost understanding by leveraging recognition effects.

In order to solve the heterogeneity problem mentioned in the introduction, we developed a solution concept to fill in the gap between the popular architectural languages - that are being used by the architects - and ARAMIS. The core of the concept is to transform an existing architecture description (AD) of a software system into an AD that conforms to the ARAMIS meta-model and subsequently to reverse the transformation to present the output.

Model-to-model (M2M) transformation is the process of producing one or more output (target) models based on one or more input (source) models. Based on the modeling languages used for the input and output, we can differentiate between two types of transformation: *exogenous* - the input and output languages are different, *endogenous* - the input and output languages are the same [13]. To enable the transformation, a so-called transformation definition consisting of transformation rules must be created. The transformation rules are specified at meta-model level and prescribe how one or more elements from the output model must be produced based on one or more elements from the input model. Upon performing the actual transformation, the application of these rules leads to the emergence of transformation links between the elements of the input and output models. If the transformation rules are bidirectional, then the transformation is also named bidirectional, otherwise it is called unidirectional.

A M2M transformation suitable to solve the problem described before must be (1) exogenous - because the input models are probably not ARAMIS-specific - and (2) unidirectional. Given that the ARAMIS-MM is very general (see Figure 1), we assume that the probability that more

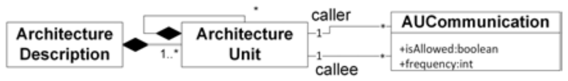

Figure 1. Excerpt from the ARAMIS-MM

elements from the input meta-model (e.g., box, component) must be transformed to the same ARAMIS-MM element (e.g., architecture unit) is relatively high. In such a scenario, defining bidirectional transformation rules can be complex. Instead, in order to enable the architects to analyze the result on their own architecture description, we propose to store the concrete links resulted during the transformation and reuse them after the ARAMIS validation results are available in order to map these on the input model. This leads to the same effect as the bidirectional transformation.

Another important aspect deals with the nature of the input and output models. The output model expresses the architecture from a behavior point of view. More explicitly, the communication of two architecture units is assigned a frequency and is possibly marked as a violation. *If the input model offers a structural overview of the architecture, then there are probably no dedicated meta-model elements to express these behavioral aspects.* There are at least two options to address this problem. We can either *reuse general purpose elements with a loose semantic from the input meta-model* (e.g., in UML we can append the results using UML *comments*) or, alternatively, we can *extend the input meta-model with additional suitable elements*, (e.g., a new property called "frequency" can be added to an already existing element called "DirectedLine").

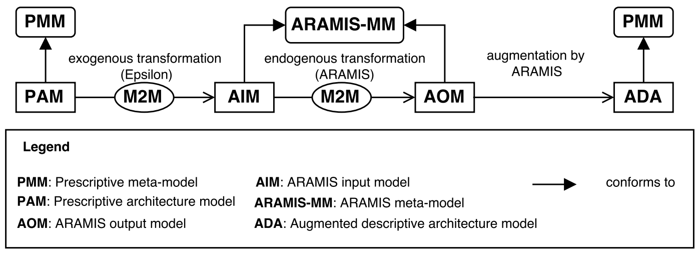
PMM
ARAMIS-MM
PMM
exogenous transformation (Epsilon)
endogenous transformation (ARAMIS)
augmentation by ARAMIS
PAM
M2M
AIM
M2M
AOM
ADA
Legend
PMM: Prescriptive meta-model
PAM: Prescriptive architecture model
AOM: ARAMIS output model
AIM: ARAMIS input model
ARAMIS-MM: ARAMIS meta-model
ADA: Augmented descriptive architecture model
conforms to

Figure 2. The model transformation chain in ARAMIS.

In order to enable the exogenous, unidirectional transformations, we implemented a solution that uses Epsilon

[14], a fully integrated environment for model engineering that, among other features, supports meta-model design, creating inter-model links, generating model editors, and M2M transformations. Also, because of its active community and provided documentation with comprehensive examples, its learning curve is reduced. The model transformation chain that resulted when we extended ARAMIS with our Epsilon-based solution is represented in Figure 2.

To use Epsilon, we first converted the ARAMIS-MM [11] into an equivalent Ecore model representation. When a prescriptive architecture model with a new, previously not analyzed meta-model must be considered, this meta-model must first also be documented in an Ecore model and then the transformation rules between it and the ARAMIS-MM can be defined. By applying the transformation rules, a set of transformation links emerges.

Assuming that each model element can be uniquely identified and differentiated (e.g., by its ID), we can keep track of every transformation with the exact source and target model elements. The result of the transformation, i.e., the ARAMIS input model, will further undergo a subsequent endogenous transformation performed by the ARAMIS Workbench which will then create the ARAMIS output model. This endogenous transformation creates an important issue: the ARAMIS output model might contain elements that were not present in the input model and thus are not linked to the prescriptive model (e.g., unforeseen communication between architecture units). This issue is a sign of a mismatch between the prescriptive and descriptive architecture. The architects can use this result to further investigate the considered software system.

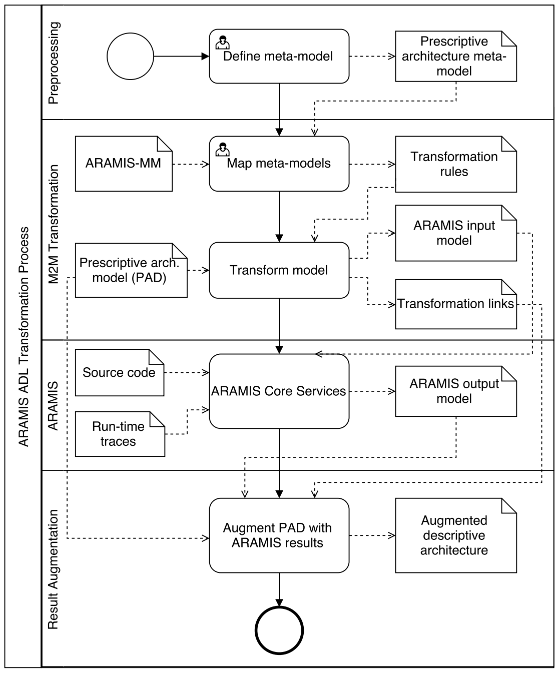

Figure 3. The ARAMIS ADL transformation process

The process encompassing all activities necessary to employ ARAMIS using a new ADL is depicted in Figure 3. This process consists of four major steps: 1. Preprocessing, 2. Model to Model Transformation, 3. ARAMIS Processing, 4. Result augmentation.

In the following, we exemplify the steps 1,2 and 4 using an example based on a simple boxes-and-lines ADL. Step 3 is not further detailed in this paper, since it was covered by our previous work [11].

A. Preprocessing

To exemplify our approach, we have implemented an example using a *boxes-and-lines* ADL. Figure 4 shows its meta-model (BL-MM). As mentioned above, the preprocessing step prepares the prescriptive meta-model (in this case BL-MM) for the next steps, by creating a corresponding Ecore meta-model.

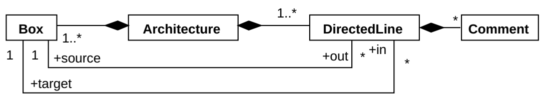

Figure 4. Meta-model of a simple boxes-and-lines ADL

In this case, an extension of the BL-MM that permits the addition of behavior-related information is not necessary, because we can use for this purpose the *Comment* BL-MM element. The validation results from ARAMIS,i.e., the frequency of the calls and their validity, will be augmented in the initial model using *Comment* elements.

B. Model transformation

Based on the ARAMIS-MM (see Figure 1) and the BL-MM, we define the *transformation rules*. In our example, we want to create transformation rules for (1) mapping *Box* in BL-MM on *ArchitectureUnit* in the ARAMIS-MM and (2) mapping *DirectedLine* of BL-MM on *AUCommunication* of ARAMIS-MM. Figure 5 presents a simple boxes-and-lines model (on the left hand side) and the ARAMIS model elements that are the result of the M2M transformation. Facade and Controller are transformed to the *AUFacade* and *AUController* respectively. The call from Facade to Controller is transformed to an AUCommunication *Facade2Controller* that has *AUFacade* as caller, *AUController* as callee, an initial frequency *unknown* and a *true* isAllowed attribute. These correspondences are then saved as transformation links.

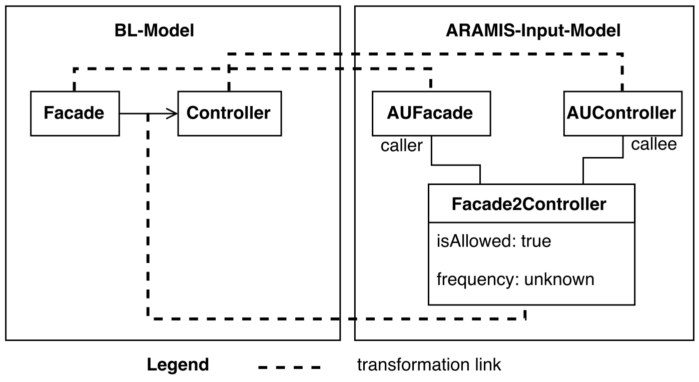

Figure 5. Example of a model transformation

C. Augmenting the ARAMIS results

The ARAMIS output model is presented on the left side of Figure 6. First, we can see that, after running ARAMIS, *Facade2Controller* now has the updated *frequency* value of *100*. Second, there is a new element that did not exist in the input model: *Controller2Facade*. This element appears because ARAMIS detected that AUController has also accessed the

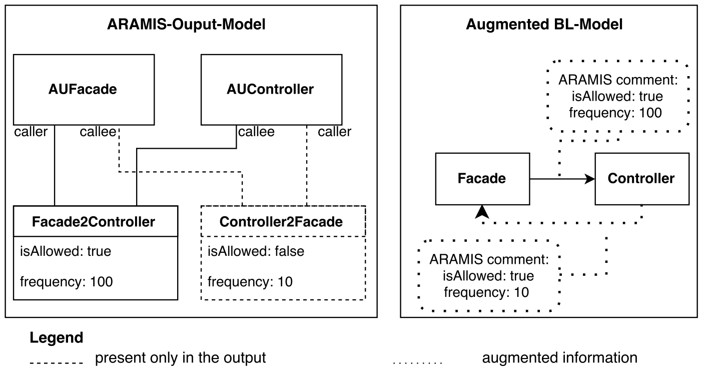

Figure 6. Example: ARAMIS augmented result

AUFacade during run-time. Based on this result, the prescriptive architecture model is augmented. Based on the previously generated transformation links, we know that our M2M transformation transformed the prescriptive model element *Facade* into the ARAMIS *AUFacade*; transformed *Controller* into *AUController*; and transformed the directed line between *Facade* and *Controller* into *Facade2Controller*. We can now use this information to augment the prescriptive model. For this we create in the input model a new Comment element that we attach to the DirectedLine from Facade to Controller as shown in the left side of Figure 6. This comment contains the *isAllowed* and *frequency* attributes that characterize the communication between *Facade* and *Controller*. Furthermore, since there is no transformation link for the ARAMIS Controller2Facade, a new DirectedLine is added to the initial model for the detected communication from Controller to Facade. To this, we also attach a corresponding comment with information regarding its frequency and permission.

IV. Related work

Most of the architecture reconstruction tools have rigid architecture description meta-models. For example, Sonargraph-Architect [15] allows users to define the architecture of the software systems using layers, layer groups, vertical slices, vertical slices groups and subsystems. The architects cannot use other types of ADL.

Malavolta et al. [16] proposed the **DUALLY** framework that supports architectural and tools interoperability. By using its intermediate ADL meta-model for architectural language, it provides ADL interoperability, but no support for architecture reconstruction or validation is available.

The meta-model of the SoftArch reconstruction tool includes 3 architecture concepts: *components*, *associations* and *annotations*. The users can then create customized figures for the various elements, to simulate the use of various meta-models [17].

V. Conclusion and future work

In this paper, we presented an approach for enabling heterogeneous input and output architecture descriptions for the ARAMIS Workbench. We have implemented an extension for ARAMIS to leverage a M2M transformation using the Epsilon framework. Our solution aims to close the gap between the ADLs that the architects are familiar with and ARAMIS. To reduce the amount of time/complexity for further model transformations, we are offering pre-defined transformation rules for the most popular cases, such as boxes-and-lines and UML component diagrams. In the future we plan to evaluate our solution within an extensive case-study on a real-world system.

An open question related to our work is how to reduce even more the effort needed to be invested by the architects when using ARAMIS. For example, if the input boxes and lines diagram is simply a drawing, we currently expect that the architect ”translates” the diagram to an Ecore model. A complete solution would employ image recognition techniques to directly transform the model. Given that different techniques might be necessary depending on the type and form of input model, this represents an important limitation of our approach.

References

[1] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[2] “ISO/IEC/IEEE 42010,” http://www.iso-architecture.org/42010 [accessed: 2015.10.01].

[3] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang, “The up-to-date list of currently existing architectural languages,” http://www.di.univaq.it/malavolta/al/ [accessed: 2015.10.01].

[4] J. Pardillo and C. Cachero, “Domain-specific language modelling with UML profiles by decoupling abstract and concrete syntaxes,” Journal of Systems and Software, vol. 83, no. 12, Dec. 2010, pp. 2591–2606.

[5] M. H. Kacem, A. H. Kacem, M. Jmaiel, and K. Drira, “Describing dynamic software architectures using an extended uml model,” in Proceedings of the 2006 ACM Symposium on Applied Computing, ser. SAC ’06. ACM, 2006, pp. 1245–1249.

[6] P. Selonen and J. Xu, “Validating uml models against architectural profiles,” in Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2003, pp. 58–67.

[7] N. Medvidovic, D. S. Rosenblum, D. F. Redmiles, and J. E. Robbins, “Modeling software architectures in the unified modeling language,” ACM TOSEM, vol. 11, no. 1, Jan. 2002, pp. 2–57.

[8] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang, “What industry needs from architectural languages: A survey,” IEEE Trans. Softw. Eng., vol. 39, no. 6, Jun. 2013, pp. 869–891.

[9] S. Ducasse and D. Pollet, “Software architecture reconstruction: A process-oriented taxonomy,” IEEE Trans. Softw. Eng., vol. 35, no. 4, Jul. 2009, pp. 573–591.

[10] A. Dragomir, M. F. Harun, and H. Lichter, “On bridging the gap between practice and vision for software architecture reconstruction and evolution: A toolbox perspective,” in Proceedings of the WICSA 2014 Companion Volume. ACM, 2014, pp. 10:1–10:4.

[11] A. Dragomir, H. Lichter, J. Dohmen, and H. Chen, “Run-time monitoring-based evaluation and communication integrity validation of software architectures,” in Proceedings of the 2014 21st Asia-Pacific Software Engineering Conference - Volume 01, 2014, pp. 191–198.

[12] A. Nicolaescu, H. Lichter, A. Göringer, P. Alexander, and D. Le, “The aramis workbench for monitoring, analysis and visualization of architectures based on run-time interactions,” in Proceedings of the 2015 European Conference on Software Architecture Workshops, ser. ECSAW ’15. ACM, 2015, pp. 57:1–57:7.

[13] M. Brambilla, J. Cabot, and M. Wimmer, Model-Driven Software Engineering in Practice, 1st ed. Morgan & Claypool Publishers, 2012.

[14] “Epsilon,” http://www.eclipse.org/epsilon/ [accessed: 2015.10.01].

[15] “Sonargraph Architect,” https://www.hello2morrow.com/products/sonargraph/architect [accessed: 2015.10.01].

[16] I. Malavolta, H. Muccini, P. Pelliccione, and D. Tamburri, “Providing architectural languages and tools interoperability through model transformation technologies,” IEEE Trans. Softw. Eng., vol. 36, no. 1, Jan. 2010, pp. 119–140.

[17] J. Grundy and J. Hosking, “Softarch: Tool support for integrated software architecture development,” International Journal of Software Engineering and Knowledge Engineering, vol. 13, 2003, pp. 125–152.