Comparison of six ways to extend the scope of Cheddar to AADL v2 with Osate

Mickaël Kerboeuf, Alain Plantec, Frank Singhoff
LISyC - University of Brest
France
{*kerboeuf,plantec,singhoff*}*@univ-brest.fr*

Arnaud Schach, Pierre Dissaux
Ellidiss Technologies
France
{*Arnaud.Schach, Pierre.Dissaux*}*@ellidiss.com*

***Abstract*—*Cheddar* is a framework dedicated to the specification of real-time schedulers, and to their analysis by simulation. It is developed in Ada. Some parts of its modular architecture are generated by *Platypus*, a software engineering tool based on the STEP standards. Cheddar owns a dedicated specification language. It can also process AADL v1 specifications.**

In order to extend the scope of Cheddar to AADL v2 specifications, we introduced a translation component called *Dairy*. It aims at creating valid Cheddar data from AADL v2 specifications. The frontend of Dairy comes from Osate v2. Hence, the backend of Dairy must produce Cheddar data from instances of the AADL metamodel that has been implemented into Osate.

Both of Cheddar and Osate are legacy systems built with different frameworks, different standards and different languages. Hence, the design of Dairy poses the problem of their *integration*. We postulate that an implemented metamodel should neither be rewritten nor be duplicated in order to keep unchanged its legacy equipment. Then, integration should better rely on *data interoperability* standards.

In this paper, we illustrate this idea by investigating six different designs of Dairy to perform the integration of Cheddar and Osate. We compare them with each other according to reusability, code generation, and transformation of metamodels.

***Keywords*-EMF; Express; Metamodels; Interoperability**

I. Introduction

Model driven software development and model transformation are promissing prospects to integrate large software systems [1], [2]. It is especially the case when some, or even all the components are existing legacy systems built and/or used with different technologies, and different standards. However, the increasing number of MDD paradigms with dedicated variants and specific implementations is not conductive to the emergence of the best solution.

In this article, we investigate an integration case study where two existing legacy systems have to be connected by a specific external component called *Dairy*. The first system, namely Osate, implements a large metamodel of AADL with EMF. From this model, among the many available functionalities, AADL v2 specifications can be parsed, edited, processed, and serialized in AAXL. The second system, namely Cheddar [3], implements a metamodel of specific data with Ada. This metamodel is *generated* from a metamodel implemented with the Express language into Platypus [4], a software engineering tool which is itself implemented into Squeak, a free Smalltalk environment.

Hence, from a technological point of view, Osate and Cheddar are obviously unconnected. Both of them are legacy systems built with different frameworks, different standards and different languages. However they both capitalize on metamodels and functionalities that it would be expedient to reuse. Thus, the design of Dairy poses the problem of their *integration*. To address this problem, we can either rewrite one system within the technical framework of the other one, or we can focus on *data interoperability* standards like XML to translate and exchange data. We postulate the second solution is preferable and we illustrate this idea with a comparison of six different ways to connect Osate and Cheddar. Each way is assessed according to different measures such as the need for metamodels' redefinings, the need for tools' updates, the reuse of existing tools, and the generation of new tools.

In the next section, the legacy systems, namely Osate, Cheddar and Platypus, are presented. The following section details Dairy and the different foreseen ways to achieve the translation of AADL v2 specifications to valid Cheddar data. Then we present some related works and to conclude this paper, we present future works.

II. Integration of Cheddar and Osate

In this section, we first present the two sides of the connection we plan to implement: Osate on the one hand, and Cheddar with its Platypus' side on the other hand. Then, we state more formally the problem we address with this case study.

A. Osate

Osate [5] is a set of Eclipse analysis plug-ins for the system representations developed using AADL. It is based on the Eclipse Modeling Framework, and it includes a large metamodel of AADL. Its frontend provides a toolset to parse and print AADL specifications, and to parse and print XMI-serialized metadata of the AADL metamodel (namely AAXL files). The parsers provide instances of the metamodel that a large set of analysis tools can process. The latest stable release of Osate (v1.5.7) can handle AADL v1

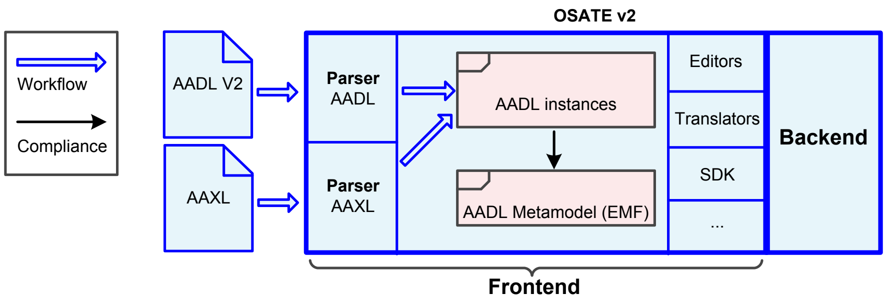

Figure 1. White-box view of Osate v2. The frontend encompass an EMF metamodel with witch instances coming from AADL v2 specifications comply.

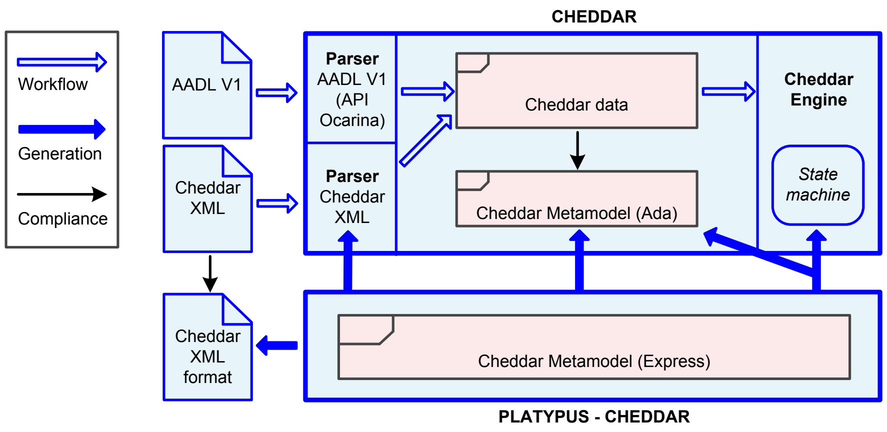

Figure 2. White-box view of Cheddar and connexions with Platypus. Cheddar encompass an Ada metamodel with witch instances coming from AADL v1 specifications or Cheddar XML files comply. The Ada metamodel is generated from an Express metamodel implemented within Platypus.

specifications. The next release of Osate (v2) can handle AADL v2 specifications.

As depicted in figure 1, Osate also provides a platform for system integration to develop tools that operate on AADL models. Its modular architecture and its integration into Eclipse make it easier to reuse the parsers and the related metamodel.

B. Cheddar and Platypus

Cheddar is a tool designed for the performance analysis of real-time applications. With Cheddar, a real-time application is typically modeled as a set of tasks, processors, schedulers and buffers. Cheddar comes with a set of standard real-time schedulers and analysis tools. They are implemented in Ada. It is also possible to extend Cheddar with specific scheduler and specific analysis tools. For that purpose, Cheddar provides a dedicated programming language. Cheddar specifications can be serialized (resp. deserialized) to (resp. from) a specific XML format called *XML Cheddar*. Cheddar also handles AADL v1 specifications thanks to the Ocarina [6] API it integrates. Ocarina is a tool suite written in Ada to process AADL models.

The architecture of Cheddar is composed of several units. Among them, some are *generated* by a specialized version of *Platypus*, a software engineering tool fully integrated inside Squeak, a free Smalltalk system. Platypus allows metamodel specification, integrity and transformation rules definition. Platypus benefits from the ISO 10303 standard, namely the STEP standard for metamodels specification and implementation [7]. STEP defines an object oriented modeling language called Express that can be used as a modeling language as well as a metamodeling language.

The specialized version of Platypus dedicated to Cheddar is called *Platypus-Cheddar*. In this version of Platypus, the metamodel of Cheddar specifications has been implemented. As depicted in figure 2, Platypus-Cheddar has been used to generate the Cheddar metamodel in Ada with which the data coming from Cheddar XML or AADL specifications comply. Moreover, Cheddar-Platypus can be reused to *extend* Cheddar with new schedulers and/or new analysis tools. This feature makes it possible to greatly improve the efficency of large scheduling simulations. Indeed, such kinds of specifications have not to be *interpreted* anymore. They can be *executed* as *built-in* functionnalities of Cheddar [8].

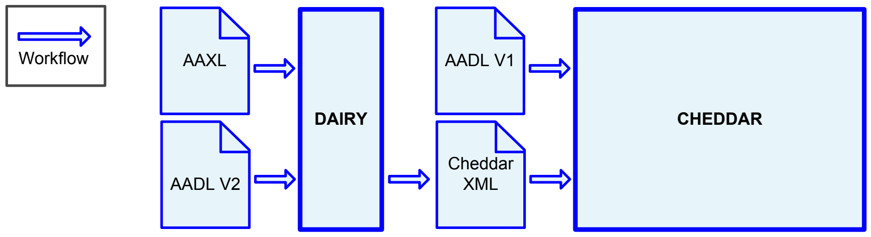

Figure 3. Joint of Dairy and Cheddar by means of Cheddar XML: extending the scope of Cheddar to AADL v2 is performed *externally* thanks to Dairy.

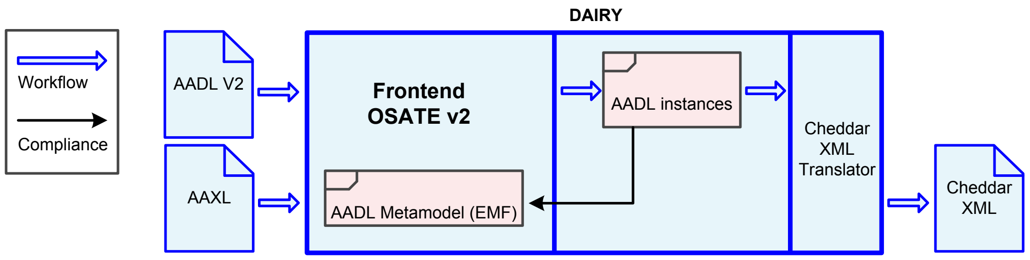

Figure 4. White-box view of Dairy. The frontend of Osate provides the EMF metamodel with witch instances coming from AADL v2 specifications comply. The Cheddar XML Translator produces Cheddar XML files from these instances.

Figure 2 suggests that the XML parser unit of Cheddar is generated by Platypus-Cheddar, as also the specification of XML Cheddar. This feature is not yet implemented. However, it comes straight from the metamodel by means of translation rules.

C. Problem statement

The problem we address here is the *integration* of *legacy* systems. Osate and Cheddar are widely used and should not be deeply impacted by their integration. If two implementations of their metamodels were available within a common technical framework (*e.g.* EMF), then their integration could rely on efficient model-driven transformation techniques. Insofar as the only implemented metamodel available for Cheddar is technically incompatible with the metamodel of AADL implemented in Osate, we have two solutions. The first one consists in rewriting Cheddar with EMF. The second one consists in translating and then exchanging data between the tools. To choose the most suitable solution, we propose the following postulate: *an* implemented *metamodel is a fundamental artefact that should neither be rewritten nor be duplicated*. Indeed, it capitalizes on domain semantics and it often results from a long study. It is also subject to modifications that should be relayed in as many as existing implementations. Hence in this case, integration should rely on *data interoperability* standards like XML or STEP.

For that reason, we decided to build an *external* tool component called *Dairy* to translate a valid AADL specification into a valid Cheddar XML file. Its use is depicted on figure 3. It handles AAXL or AADL v2 specifications and it produces valid XML Cheddar. AADL v1 specifications can be directly handled by Cheddar. Thus, Dairy must process data complying with one metamodel (AADL v2), and it must also produce data complying with another metamodel (Cheddar data). Rewriting one metamodel within the framework of the other one would enable to write or generate translation rules, but it would also imply to synchronize two implementations of the same metamodel. If we keep the metamodels separate, then the connection can be performed at a data level by using interoperability standards.

III. Alternative designs of Dairy

In this section, we detail the architecture of Dairy, and we discuss the design of the Cheddar XML translator it involves.

A. Global design of Dairy

As depicted in figure 4, the frontend of Dairy comes from Osate v2. The metamodel of AADL is notoriously sizeable. It is therefore very interesting to reuse it within the frontend of Osate. It is also very interesting to benefit from its parsers and editors. Moreover, it makes Dairy *de facto* the natural interface to Cheddar for regular Osate users. The backend of Dairy is a Cheddar XML translator. It must implement translation rules from instances of the EMF AADL metamodel to Cheddar XML.

B. Cheddar XML translator

Taking advantage of Osate facilities implies to handle EMF artefacts and tools. However, we do not dispose of any EMF metamodel of Cheddar. Hence, the translation cannot be straightforward. We enumerated six ways to produce

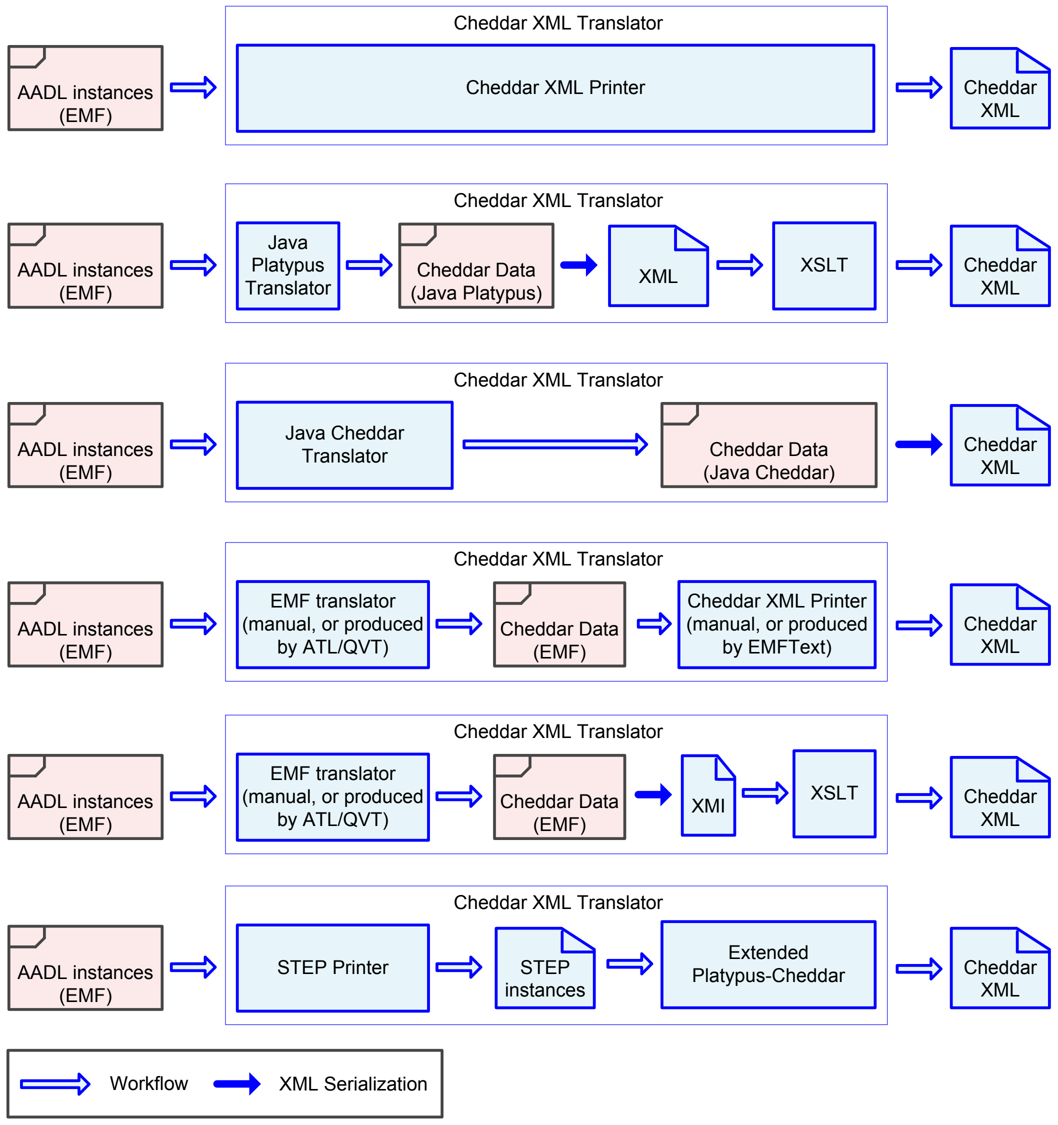

Figure 5. Six different designs of the AADL to Cheddar XML translator.

Tool Chain	Redefinition of metamodel	Update of Platypus-Cheddar	Reused tools	Generated tools	Impact of meta-model update
Cheddar XML Printer	no	no	none	none	high
Java Platypus and XSLT	no	no	XSLT	XML serializer	middle
Java Cheddar translator	no	yes	none	XML serializer	low
EMF translator and XML printer	yes	no	none	translator and/or printer	low
EMF translator and XMI translator	yes	no	XSLT	translator and XMI serializer	low
STEP Printer	no	yes	Platypus-Cheddar	none	low

Table I
COMPARISON OF THE 6 WAYS TO PRODUCE CHEDDAR XML

Cheddar XML from AADL instances (figure 5). We explain and comment each of them in the following subsections. Their main features are summed up in table I.

1) Manual Cheddar XML Printer: The first way consists in browsing the AADL instances to generate manually strings of Cheddar XML. This approach is rather simple but it has two important drawbacks: it does not enable to capitalize on the translation rules, and it does not lean on a metamodel for Cheddar. Hence, it is not easy to guaranty the soundness of the translation. Moreover, any modification of both AADL and Cheddar metamodels implies modifications that cannot be easily targeted. However, this approach is a good way to quickly develop prototypes.

2) Java generation by Platypus and XSLT: Platypus-Cheddar contains a metamodel of Cheddar data from which parts of Cheddar can be (re-)generated. From this metamodel, it is also possible to generate automatically java classes for each of its entities. In other words, it is possible to translate the metamodel from Express to Java. This translation follows the same way as for the translation from Express to Ada (see figure 2). Thus, this second way to create Cheddar XML with Dairy consists in first browsing the AADL instances to create Java instances of the Java classes generated by Platypus. Finally, the standard XML serialization of Java enables to define translation rules to Cheddar XML. These rules can be stated with XSLT, a language that can be used to transform XML documents into other documents (like XML documents of other kinds). This approach enables to capitalize on the translation rules.

3) Specific Cheddar-Java generation by Platypus: The Java classes generated by Platypus are *generic*, *i.e.* they cannot encompass specificities of the Express metamodel from which they result. The third way we consider here to produce Cheddar XML consists in modifying Platypus-Cheddar so that it could generate Cheddar specific Java classes. The main extra feature these specific classes would have would be the direct serialization into Cheddar XML. Hence, this approach is rather like the previous one, except there is no need to use XSLT to produce Cheddar XML. On the other hand, Platypus-Cheddar has to be modified, but this update is not supposed to imply any modification of Cheddar.

4) EMF translator and XML printer: The forth way consists in transforming the AADL instances into instances of a Cheddar metamodel in EMF. This transformation can be performed manually by browsing the AADL instances, or automatically using transformation languages like ATL [9] or QVT [10]. Once the Cheddar instances have been created, they can be translated into Cheddar XML. Again, this translation can be performed manually by browsing the Cheddar instances, or automatically using tools like EMFText [11]. This approach is very consistent from a technological point of view, and it is highly likely to be automated. However, this approach has a major drawback: it implies to completely redefine the Cheddar metamodel into EMF. It would not only be tedious, it would require to synchronize two versions of the same metamodel (with EMF and with Express).

5) EMF translator and XMI translator: The fifth way is a variant of the previous one. Here, instead of generating Cheddar XML from the Cheddar instances, we use the standard XMI serialization of these instances. Then, as in the second approach using XSLT, we generate Cheddar XML.

6) STEP printer: The sixth way consists in browsing the instances to translate them into STEP instances of the Express Cheddar metamodel. These instances can then be processed to generate Cheddar XML. Platypus-Cheddar could *almost* perform this processing. It involves the related Express metamodel, and it can *print* Cheddar XML as directly as it can *parse* it. However, it has not been planed that Platypus-Cheddar could produce Cheddar XML from STEP instances. Anyway, this feature comes straight from the metamodel. Thus, this approach enables to reuse Platypus-Cheddar, but it implies its update.

C. Comparison

To decide which approach should be kept, in accordance with the postulate we stated, we first eliminated those who require the rewriting of one metamodel. Then, we eliminated approaches 4 and 5. The first two solutions are interesting because they do not require any modification of Platypus, the environment that implements the target metamodel; and they enable to quickly develop prototypes. However, they are very sensitive to the metamodels' updates. We therefore prefer the third or the last solutions. They minimize the impact of the metamodels' updates, at the cost of updating the environment that implements the target metamodel.

This first selection has to be refined using other criteria such as tools reusability and complexity.

IV. Related works

The works that are related to our approach mainly concern model transformations and interoperability standards. But model transformations come sometimes with an implicit prerequisite for a common logical framework. For example, the initial model and the targeted model have to be both available within EMF, Express, or any other modelling facility. Hence, Kompose [12] is a generic model composition tool that could be very useful to connect the AADL metamodel to the Cheddar metamodel if we had an EMF version of the Cheddar metamodel. Within a common logical framework, we are highly likely to find very optimized solutions to our problem. But we stated as a fundamental property that metamodel should neither be rewritten nor duplicated. Thus interoperability standards are more suitable because they clearly exclude any hypothesis of a common logical framework. The last way we suggested to produce Cheddar XML is based on STEP, which is a major standard to exchange data. Another major standard has been suggested in the second and fifth propositions, namely XML proposed by the W3C consortium. Both XML and STEP provide interoperability by *data exchange*. Other standards like CORBA proposed by the OMG provide interoperability by *service calls*.

The interoperability with Cheddar is very important insofar as several other tools are related to it. For instance Cheddar has been jointly used with other modelling languages such as MARTE/UML [13] or PPOOA [14]. Thales RT has developed a set of Eclipse plug-ins which allows scheduling analysis with Cheddar on MARTE/UML models edited with IBM Rational Software Architect [15]. Similarly, PPOOA is a modelling framework based on UML with a customized profile for pipe-line architecture which has been proposed by the University of Madrid. In [14], the University of Madrid has described how PPOOA/UML can be used with Cheddar. These projects will eventually face the problem we address in this paper when it appears Cheddar metamodel has to be updated. Indeed, neither of them relies on the original Express Cheddar metamodel. Other solutions have also been developed outside the Eclipse world such as performing performance analysis with Cheddar on textual AADL files generated by the Stood modeling tool edited by Ellidiss [16].

From a more restricted technical point of view, Dairy implements a translator from AADL v2 specifications to Cheddar XML. This functionnality has been newly developed for Ocarina [6].

V. Conclusion

We analyzed six different ways to translate instances of an EMF metamodel to an XML file whose format depends on a metamodel defined in a completely different environment with a completely different language. The main point was to reuse existing metamodels and existing tools while minimizing manual production. We suggest the most suitable approach should not rely on the duplication of metamodels. This principle reduces the impact on related legacy systems, and it does not require to synchronize different implementations of a same metamodel.

We plan to develop a first prototype with the first approach. At the same time, we will study in depth the impact of approaches 3 and 6 on Platypus-Cheddar. We will adopt the least costly to develop a full version of Dairy.

When Dairy is finalized, the main way to use AADL with it is to model a system that will be validated thanks to Cheddar. For instance the ADELE graphical editor of Topcased can be used that way. Another way could be to use AADL as a pivot language in order to apply Cheddar validation abilities on to another modeling language. Additional transformations can thus be foreseen in order to connect Cheddar to other upstream Eclipse modeling tools. For instance, the MARTE/UML to AAXL transformation implemented with ATL, that is currently being developed in the context of the *Lambda project* could provide a way to perform performance analysis on UML real-time models with Cheddar through Dairy.

References

[1] R. France and B. Rumpe, “Model-driven Development of Complex Software: A Research Roadmap,” in *FOSE ’07*. IEEE Computer Society, 2007.

[2] D. Schmidt, “Model Driven Engineering,” *IEEE Computer*, vol. 39, no. 2, pp. 25–31, Feb. 2006.

[3] F. Singhoff, J. Legrand, L. Nana Tchamda, and L. Marcé, “Cheddar: a flexible real time scheduling framework,” *ACM Ada Letters journal, 24(4):1-8*, Nov. 2004.

[4] Platypus, “Technical Summary and download,” http://cassoulet.univ-brest.fr/mme.

[5] Carnegie Mellon Software Engineering Institute (SEI), “Open Source AADL Tool Environment (OSATE),” http://www.aadl.info/aadl/currentsite/tool/osate-down.html.

[6] J. Hugues, B. Zalila, L. Pautet, and F. Kordon, “From the prototype to the final embedded system using the Ocarina AADL tool suite,” *ACM Transactions on Embedded Computing Systems*, jully 2008.

[7] *ISO 10303-1: Part 1: Overview and fundamental principles.*, ISO 10303-1, 1994.

[8] F. Singhoff and A. Plantec, “Towards user-level extensibility of an ada library : an experiment with cheddar,” in *Proceedings of RST Ada-Europe. LNCS springer-Verlag.*, Jun. 2007.

[9] F. Jouault and I. Kurtev, “Transforming models with atl,” in *MoDELS Satellite Events*, 2005, pp. 128–138.

[10] B. Appukuttan, T. Clark, S. Reddy, L. Tratt, and R. Venkatesh, “A model driven approach to model transformations,” 2003.

[11] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende, “Derivation and refinement of textual syntax for models,” in *ECMDA-FA*, 2009, pp. 114–129.

[12] F. Fleurey, B. Baudry, R. France, and S. Ghosh, “A generic approach for automatic model composition,” pp. 7–15, 2008.

[13] OMG, *A UML Profile for MARTE, Beta 1*. OMG Document Number: ptc/07-08-04, august 2007.

[14] J. L. Fernandez and G. Marmol, “An Effective Collaboration of a Modeling Tool and a Simulation and Evaluation Framework.” 18 th Annual International Symposium, INCOSE 2008., 2008.

[15] E. Maes, “Validation de systèmes temps-réel et embarqué à partir d’un modèle MARTE.” Thales RT, Journée Ada-France 2007, Brest, december 2007.

[16] P. Dissaux and F. Singhoff, “Stood and cheddar: Aadl as a pivot language for analysing performances of real time architectures,” in *4th European Congress Embedded Real-Time Software*, Jan. 2008.