Model Federation in toolchains

Joël Champeau, Vincent Leilde, Papa Issa Diallo

Model Federation in toolchains

Joël Champeau [1], Vincent Leildé [2], Papa Issa Diallo [1]

[1] ENSTA Bretagne STIC/IDM LabSTICC

2 Rue François Verny

Brest, FRANCE

joel.champeau@ensta-bretagne.fr , diallopa@ensta-bretagne.fr

[2] OPEN Flexo

Brest, FRANCE

vincent.leilde@openflexo.org

Abstract

In this paper we introduce the toolchain topic as a federation of models based on an abstraction of differents tool definitions. We consider the toolchain in the context of embedded systems, in particular the co-design which implies a co-engineering approach with many tools. Our main goal is to define a tool integration model to carry out an abstraction of several data formats and for a domain model as a reference vocabulary. This model gathers the concepts for managing the development process artifacts and the roles attributed to these artifacts over the process. We have experimented this approach during the europeen ARTEMIS iFEST project over the OSLC layer (Open Services for Lifecycle Collaboration).

Keywords: tool integration, model federation, roles, OSLC.

1 Introduction and challenges of the integration of tools

Embedded systems are becoming increasingly complex in the last recent years and the higher level of expected requirements, both in terms of robustness, scalability and design costs, also increases. This change requires designers to adapt their practices to use specialized software tools for the different phases of the system development cycle, from requirements elicitation to product deployment. These tools are by nature independents and allow to develop different kinds of artifacts during the system development, such as models in several formats, documentations and source

code in several languages. The main objective of this situation is the separation of concerns and the use of dedicated tool for each concern. During the development process, it is necessary to enable the collaboration and the exchange of process artifacts. To do so tools supporting the process artifacts should be connected to each other to constitute a toolchain. The embedded systems and specially the co-design domain is highly representative of this situation due to an association of the software and hardware design. In this context, the heterogeneity of languages and models is an evidence and requires specific attention to build toolchains to cover the whole process development.

Currently, the toolchains exist in two forms, the point-to-point chains, and integration frameworks. The point-to-point one is an ad-hoc connection between two tools, thereby efficient but these toolchains are not scalable if we want to integrate a new tool interacting with the other tools. The second form of integration is more durable, it integrates a set of tools based on common rules, such as a common format and data type definitions. In this context the tools can be attached or detached more easily on this toolchain. However defining such kind of framework remains complex and mainly based on a trade off on the common definition. A generic definition allows the integration of many tools, in contrario the specialization, maximizing efficiency while reducing the number of connectable tools. In the modeling context, the modelBus [30] is one of the most significant approaches based on transformations and models to create a toolchain.

In the case of modeling approaches, the common definition can be achieved from a common metamodel. Then models conform to this metamodel can be generated or created from model transformations. The scope of this common meta-model is the major issue with this kind of approaches. The challenge is to have a relevant common definition that must be abstract enough to cover a large scope but also contain details enough to take into account specialized concepts. Most of the time, this relevant trade-off is difficult to achieve if we want to limit the size of the metamodel. Another challenge is the management of the evolution of this common metamodel that constrains to update the transformations and the resulting models over time.

So abstract the tool inputs/outputs by a model and implement the tool connections by transformations, this is not sufficient to obtain a powerful toolchain framework. Mainly this kind of approaches promotes solutions where each tool element is created, or instantiated, for each model element and afterwards transformed, and so recreated, for another model. Most often after applying a transformation, no semantics link is maintained between the tool elements. In this case the semantics link is the information that a specific transformation was performed on tool elements. So this semantics information, that the new tool element was created from another tool element, is lost. By extension, several transformations are applied during the process development, the links between all the tool elements are lost and the evolution of the primary tool element is disconnected from the other tool elements. At the opposite, the last tool element is agnostic of its past history in the process development which prevents any impact analysis, for example.

The current modeling approaches to abstract over the tool format are too fragile and too close of the input/output format and don’t provide any support to manage the semantics evolution of a tool element during all the process development.

To improve the tool integration, we promote a flexible approach, independent and non intrusive of the tool format (or model or metamodel) as well as independent of the tool integration platform. This approach ensures semantics consistency between

tool elements all along the process development. Also this kind of approach can support semantics evolution of the tool element, impact analysis and support of collaborative works. Our approach is similar to a virtualization of the tool integration platform. We produce an abstraction based on the artifact concept, to represent the tool elements, and semantic roles are associated to this artifact, to symbolize all the types of the artifact evolution.

The article is organized as follows: the section 2 presents the state of the art of the tool integration frameworks. Secondly the section 3 presents our approach and the core concepts. The section 4 is related our experiments included in the iFEST project, and based on OSLC, dedicated to a tool integration framework for the embedded systems. Finally we conclude on our approach and experiments before some further works.

2 Context and state of the art

Tool integration is an active research area since the begining of 90's [27] and several characterizations were attempted. In the paper [4], a clear separation is achieved on 2 axes, one *Conceptual* which try to define the integration, itself, and the other *Mechanical* which focuses on the technological space of the integration space. For [26], the tool integration is based on five dimensions control, data, presentation, process and platform. [5] characterizes the integration levels of interoperability among the syntactic level which represents the degree of agreement between the tools on a data structure, and semantics that addresses the meaning of the data exchanged. Many frameworks have been established and have never converged, often a toolchain is a coexistence of several frameworks and several individual tools.

As presented in the introduction two integration patterns exist, point to point, which is an ad-hoc connection between two tools, and common frameworks can be based on standardized architectures [9], formats [7], or infrastructure to facilitate the inter-tools [14] collaborations. Some aspects are most important for us like the communication layer between tools, the sharing of data on both levels syntactic and semantic, but also the adaptation of the tool chain and its products to the context and specially to the engineering domain targeted.

Through the years, communication between tools was established on several technologies and more recently the service approach and specially web services are applied with standardized communication protocols (WSOA), favoring a weak coupling between the tools. More recently, an open initiative was launched and called OSLC for Open Services for Lifecycle Collaboration. This community provides a framework based on a REST architecture [6] and a common format. This emerging standard has also been chosen by many research projects and including IFEST [13] in where we were involved.

The tool integration is increasingly based on the models, taking advantage of a meta level which provides the defintiion of the models. The models, conform to the metamodel, become the data exchanged between tools[15]. There are different use of these metamodels in the integration of tools, Fujaba [10] approach provided a generic integration solution based on different patterns, VMTS [8] targets only design tools based models, MOFLON [1] GeneralStore [22] and Semantic integration [16] is

based on a generic integration but do not take into account the context of integration. Wotif [17] Jeti [19], and ModelBus [24], are also based on web services. Besides using metamodels, some approaches manipulate ontologies as more favorable to the reasoning. In this case, the ontology is considered as a domain model and defines the model of this area. ModelCVS [18] uses ontologies to define the semantics of the data and seeks to establish a link between the tool metamodels.

Such tools can be combined in different contexts, the semantics of the produced data changes over the context of the tool usage and objectives of the toolchain. The role concept provides a dynamic adaptation by allowing to objects to have different needs depending on the roles that are attached [21] [3]. [25] provides an overview of these different defintion and properties, while [23] provides an example of meta-modeling approach based on roles for tool integration.

3 Semantics integration in toolchains

The context of our toolchain is applied on embedded system domain and specially focused on the co-design. This domain involves a co-engineering development process for the software and the hardware part. In this domain, the tools are mainly specific to dedicated tasks like code parallelization, mapping on hardware, dedicated modeling, etc. This heterogeneity is also valid for the input/output formats with several programming and modeling languages. Regarding this situation, many times one domain concept, such as Parallel Entity, is realized in different languages and formats like a behavior function in Haskell, a C structure with a set of functions or a UML class with a MARTE stereotype.

In this context to improve our toolchain, we focus our approach on defining an engineering domain model, the co-design domain, to have a reference and a usage context for the tools. But also mainly, we define a mechanism to manage at the same level, the metatypes of the tool elements and our engineering domain types. Our goal is to create a federated model for our toolchain, based on our tool integration model to aggregate information at several level of abstraction.

First, we present our domain model in the goal to manage it in the tool integration model, detailed in the second part of this section.

3.1 A domain model for the semantics integration

Most of the time, tool integration consists of exchanging not directly interpretable data between tools, thus transformations are required. As such in order to keep the semantics consistency in the process, the semantics of tools as well as the mappings between the tools must be defined. We take as an example two tools used in the context of design and implementation.

A UML modeler produces UML models representing various concerns of the system under study. In our case study described latter, some of these models have to be processed by the Bluebee compiler/mapper for heterogeneous hardware architectures. This compiler requires two kinds of inputs either algorithmic or architectural ones, in two different formats respectively C and XML. In the UML model a StructuredComponent, a specialized metatype, can be interpreted as a functional bloc as well as a hardware component, so transformation rules towards

Bluebee require specific information to generate either C or XML. And if we extend the development process towards the verification and validation phases, the semantics of the StructuredComponent would change again as a test case definition, for example. So a context definition is required to ensure the semantics consistency in the tool integration framework.

In this context, the most classical way is to base the tool integration on a common metamodel which provides the shared semantics according to the addressed domain. The tool models can be independent of the common metamodel but model transformations are required to produce models to and from this metamodel.

In the one hand this common metamodel gathers all the concepts of each integrated tool, but in this case the metamodel must evolve when new tools are integrated and thus might be hard to manage. In the other hand if this metamodel is fixed to a set of concepts, and thus properties can be lost if new tools with new concepts are inserted into the toolchain.

To bring both flexibility and efficiency, another approach is necessary for not defining a common metamodel from which models are conformed to. But rather an independent domain model is created to qualify the tool models. This domain model captures a semantics reference among all the domain tools and applies it on every new inserted tool. In any case, this domain model is never instantiated since it is strictly used as metadata to qualify the model elements exchanged between tools.

In conformity with ISO/CEI 15940 we define an engineering domain as a set of tools and process activities. In the domain of embedded systems engineering, there is unfortunately no consensus on a standard to represent the domain with the purpose of tool integration. Therefore we propose a new model called D&I (Design and Implementation) adapted to tool integration and aligned with current standards like the MARTE UML profile. Our model consists of three viewpoints namely application, architecture and mapping. Application viewpoint defines functional aspects of the system under study, while the Architecture viewpoint describes the topology of the physical platform in which the application will be allocated thanks to the rules given by the Mapping viewpoint.

The main objective of this model is to define a contextual reference regarding the tool usage in the engineering domain, design and implementation. The content of this domain model can be improved certainly but our purpose is to explicit the management of this model for an integration scenario between tools rather to define yet another new domain model. The experiment performed in our integration scenario shows that the mechanism used to manage the mapping between the tool model and the engineering domain is not specialized to a domain and also independent of the content of the engineering domain model.

3.2 A tool integration model to explicit semantics consistency

In order to define a general mechanism to manage this domain model and the tool models, we based our work on a modeling approach to create an abstraction to ensure semantics consistency in the framework. The purpose of this model is to support a general mechanism to manage the evolution of the tool elements through the

development process from a semantics point of view. Our artifact and role model is the support of the semantics management.

3.2.1 Artifact and role model

Each tool manipulates models composed of elements conform to the tool metamodel, for instance an UML tool manipulates UML classes. In this example the metamodel is the tool metamodel. For a C compiler, we must create a dedicated metamodel which is not necessarily the C language metamodel. This metamodel is more an abstraction of the structure of a C program dedicated to hardware synthesis or hardware mapping. Because, this kind of compiler takes into account C programs with some coding restrictions and annotations.

Our main goal is to avoid point-to-point model transformations between tools or between a tool and the engineering domain model. So we have created an abstraction only dedicated to the integration framework to be independent of the tool models and to provide a support for the semantics evolution in the tools integration space. Our approach is similar to a virtualization of this tools integration space with an abstraction based on an artifact concept. The artifact concept is an abstract representation of tool models or tool modeling elements. This artifact owns only semantic properties favoring the tool integration like tool intern properties or lifecycle properties.

While the artifact is the representation of modeling elements inside the tool chain, we associate the role concept to the artifact as a property of the artifact. The role concept was used to modeling knowledge[28] and also to provide flexibility in object oriented programming[25]. The role concept provides three kinds of properties to another concept, or an object, that lacks semantic rigidity regarding its primary type, secondly a role depends on relationships between objects and finally an object may play different roles simultaneously. We reuse the role concept according to the properties, extracted from the Steimann's paper[25]:

- Of course an artifact is created to represent a dedicated tool element but in the tool integration space and during the process development, the semantics of this artifact is evolving in flexible way to take into account several transformations. The flexible way means that a role can be added or removed dynamically during the process development.
- A role is modeling the relationships between an artifact and a tool element without applying a particular transformation. The role models a tool specific property for a given artifact.
- At a specific step in the process development, an artifact can play several roles simultaneously according to several tools. Each role models the result of applying a transformation on the original tool element. A role can be also the support of the engineering domain type independently of any tools.

This modeling approach based on artifact and roles is orthogonal to the tool elements and completely dedicated to the tool integration space.

Figure 1 illustrates concepts and relations of the integration model. As described previously an artifact plays a set of roles, which are related to the engineering domain

(like D&I:Architecture role) or relying to a tool (Rhapsody UML:Class role). To increase the decoupling between the tool integration space and the tool elements, an artifact accesses to a modeling element through the roleproxy allowing different representations of model elements in a same tool for the same artifact.

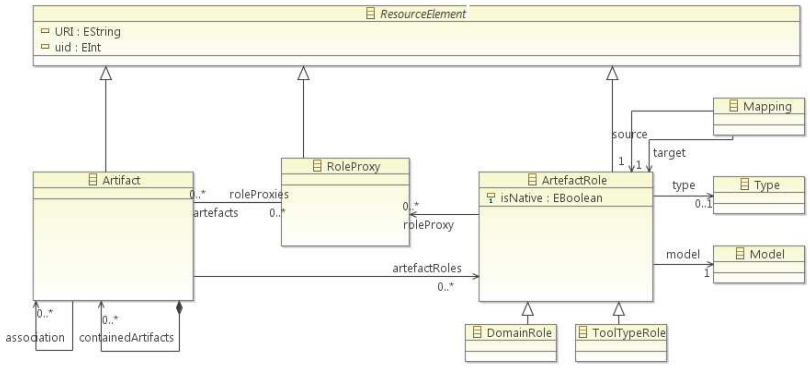

Figure 1: Entities and relations of the integration model

3.2.2 Artifact and role properties

Adding roles to artifacts may be assimilated to annotation mechanisms or in the UML context as stereotypes. In contrast with stereotypes, the purpose of the roles is to provide a generic mechanism to manage the types of the artifact in the tool integration space independently of the tool formats. Also, this approach is more flexible due to the dynamic adding and removing of the roles during the process development. Indeed the roles being dynamic, an artifact fits always to the current tooling and all along the process development. New roles, related to new tooling, can be also added dynamically in the tool integration space without any impact on the previous role definitions and the previous life of the tool integration space.

Whether roles are linked to an engineering domain like D&I and related to a tool/language, they make explicit the semantic interpretation of tool elements, in the context of the engineering domain. This might be useful to improve the semantics relationships between several tool elements and the engineering domain. For example, characterizing the model of computation combined with a UML component helps to improve the underlying transformations to another tool. Once these roles are shared between different tools, it becomes possible to create execution semantic links between tool concepts. For example, a UML StructuredComponent can be referenced by an artifact with the role ApplicationComponent of the D&I domain and so interpreted as a set of functions by the Bluebee compiler, but also another UML StructuredComponent can be referenced by an artifact with the role ArchitectureComponent of the D&I domain and so interpreted as a computing ressource (like a processor) in the scope of the Bluebee compiler.

4 Experiments

We study the feasability of our approach with a tool integration scenario made in the context of the Europeen Artemis iFEST project which aims to define a tool integration framework for embedded system tools. More precisely tools involved are part of Design and Implementation process phase. In our scenario models representing the system are defined using a UML modeler (Rhapsody) in the respect of MoPCoM methodology. MoPCoM defines three levels of abstraction addressing particular concerns, including the functional specification of the system, the representation of the platform and the allocation of the functional elements onto the platform are described. At one point of the process the UML/MoPCoM model representing the system is transformed to Bluebee comprehensible code in order to generate the system for the target architecture. Bluebee requires annotated C code as input in order to describe the mapping onto hardware. An XML file describes features of the target architecture. The transformations are done with MDWorkbench, a model to model transformation tool[20].

This scenario is interesting because one must take care of exchanged model's semantics to keep their consistency over transformations. Indeed MoPCoM allows to describe functional as well as hardware elements thanks to a same UML element (Structured Component), although Bluebee mades a distinction with hardware elements which are required to be in XML code and functional's ones required to be in C.

Besides iFEST extends OSLC, Open Services for Lifecycle Collaboration[29], with specifications for embedded systems. OSLC is a community providing specifications for tool integration based on REST web services and RDF data formalism. In addition iFEST proposes a set of tool adapters facilitating tool integration by making the bridge between tools and OSLC. An adapter is a client/server providing services to OSLC connected tools. It exposes OSLC ressources wrapping or referencing real model/tool elements. In our approach with artifact and roles, each tool(MoPCoM, Bluebee, MDWorkbench...) is linked to OSLC via an adapter. The adapter maps artifacts(in a form of OSLC ressource) to a tool model element(for instance a UML class). Artifacts and Roles are OSLC ressources managed by the Model Federation Adapter(MFA), adapter for the Model Federation Tool. The MFA provides basic services to other OSLC adapter in order to manage Artifacts and Roles(CRUD, mappings) as well as more higher level services such as finding artifacts playing a set of roles.

The various services are sequenced by the orchestrator tool. In the whole tool chain given by the figure 2, Bluebee tool is connected to OSLC via the Automation Adapter Server providing a set of common services to automation tools(this same automation adapter server is used for Forsyde tool as well as DiVine tool).

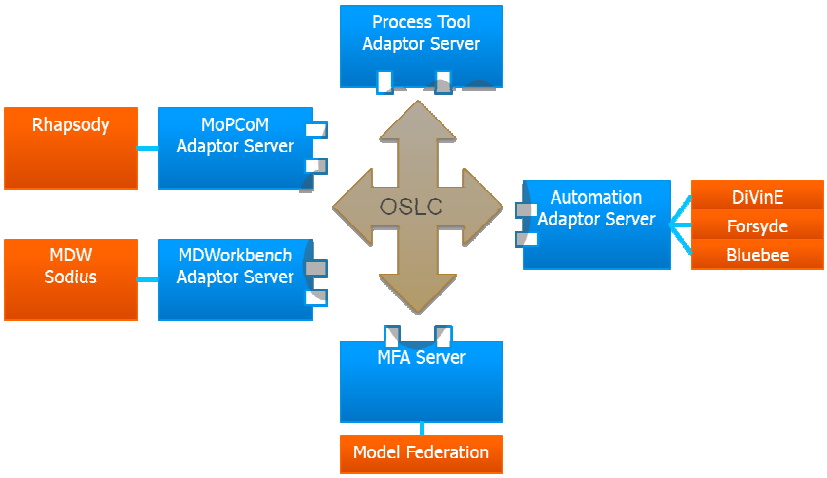

Figure 2: The experimental toolchain

The lifecycle of the artifact is the following :

1. When a model element is produced by a tool, an artifact is also created in the model federation tool, and plays a rôle associated to the tool. Besides the adapter of the tool handle a rôle proxy which make a bridge between artifact and model element.
2. If there is a mapping defined between the tool rôle and an engineering domain rôle, then engineering domain rôle is attached dynamically to the artifact. Thus the artifact plays two roles, one relative to the tool and one relative to the engineering domain.
3. The artifact is used by an another tool(a transformation tool for instance), to produce a new model. In this case two scenarios are possible, either no artifacts are produced for this new model, and thus this artifact just plays a new rôle, or an artifact is produced and so it might plays two roles, one from the tool and one from the engineergin domain. This scenario can be extended to several engineering domains, in this case artifact plays roles from different engineering domains.

The sequence diagram illustrates how integration occurs in our scenario :

- Step 1: Tools declare their mappings to the MFA by referencing their own set of roles, and their semantics(*mapping(RoleX, RoleY)* regarding an engineering domain *(RTES, short for Real Time Embedded Systems)*). For instance MoPCoM concept of FunctionalBehavior and Bluebee concept of CFunction, are both likened to an RTES concept of StructuredComponent in the RTES domain model
- Step 2: Orchestrator gets a FunctionalBehavior model from MoPCoM Adaptor (funcBehavior(updateTargets)). Consequently an Artifact is created (MopcomFuncBehavior) for this model (createArtifact()). Its semantics is given by Semantics Roles dynamically attached to the artifact according to the predefined mapping
- Step 3: A transformation service is called through a MDWorkbench service (transfo(mopcomMM, bluebeeMM, MopcomFuncBehavior Artifact)). This service requires three parameters: the source metamodel (mopcomMM), the

target metamodel (bluebeeMM) as well as the Artifact representing the source model. Bluebee C code is generated (transformMopcom2bb()) and represented by a new Artifact (bluebeeCFunction Artifact). Roles (Bluebee CFunction and RTES StrucComponent) are dynamically attached to this Artifact.

- Step 4: Bluebee tool generates the system with a predefined target architecture and this Artifact (executeBluebee(bluebeeCFunction Artifact)).

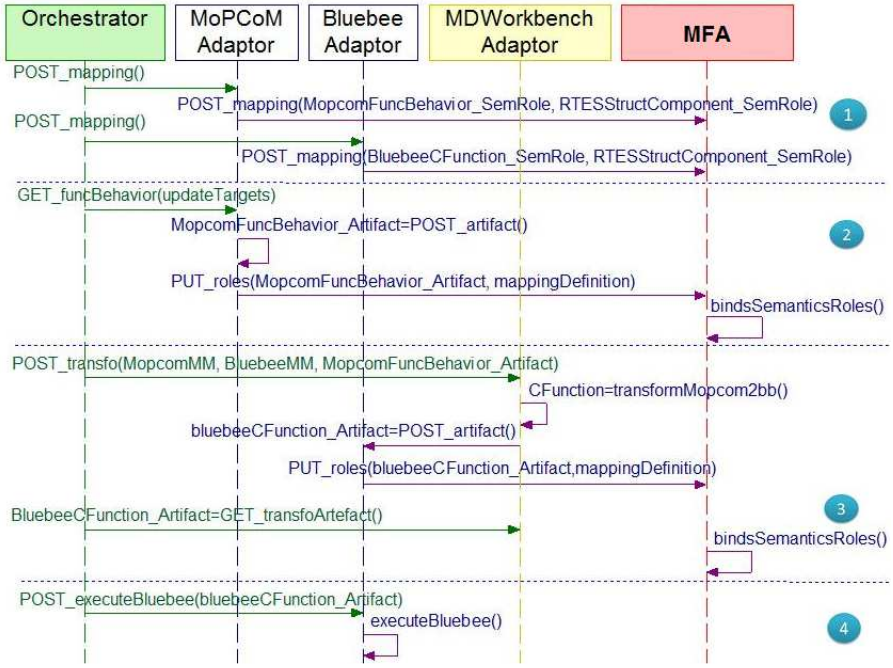

Figure 3: sequence diagram of the experiment

5 Conclusion

In the field of embedded systems there is still no satisfactory solutions for tools integration. To facilitate this integration, the IFEST project proposed an innovative framework based in particular on OSLC. Mainly oriented lifecycle, OSLC is not dedicated to the concepts of embedded systems. Our approach provides primarily a conceptual model independent of technologies/standards. This model provides the capacity to build high level services on top on all the federated models like impact analysis or semantic consistency checkings.

Our model allows both to represent the elements of the tools in the tool integration space (artifact) and associates their semantics via the roles. A tool chain is prototyped and semantics of handled data depends on the co-design engineering domain. This model can be extended to any engineering domain for defining and managing the context of the use of the toolchain.

6 Acknowledgments

The authors would like to thanks the support of the european ARTEMIS iFEST project and all the partners that were contributed to this work directly and indirectly.

7 References

[1] C. Amelunxen, F. Klar, A. Königs, T. Rötschke, and A. Schürr. Metamodel-based tool integration with moflon. In Proceedings of the 30th international conference on Software engineering, ICSE '08, pages 807-810, New York, NY, USA, 2008. ACM.

[2] D. Aulagnier, A. Koudri, S. Lecomte, P. Soulard, J. Champeau, J. Vidal, G. Perrouin, and P. Leray. SoC/SoPC development using MDD and MARTE profile. In Model Driven Engineering for Distributed Real-time Embedded Systems. ISTE, 2009.

[3] D. Bäumer, D. Riehle, W. Siberski, and M. Wulf. Role Object, pages 15-32. Addison-Wesley, Massachusetts, 2000.

[4] A. W. Brown, P. H. Feiler, and K. C. Wallnau. Past and future models of CASE integration. In [1992] Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering, pages 36-45. IEEE Comput. Soc. Press, 1992.

[5] A. W. Brown and J. A. McDermid. Learning from ipse's mistakes. IEEE Softw., 9:23-28, March 1992

[6] R. T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. Phd thesis, University of California, 2000.

[7] R. G. Flatscher. Metamodeling in eia/cdif|meta-metamodel and metamodels. ACM Trans. Model. Comput. Simul., 12:322-342, October 2002.

[8] T. L. Gergely Mezei, Sandor Juhasz. Integrating model transformation systems and asynchronous cluster tools. In 7th International Symposium of Hungarian Researchers on Computational Intelligence, 2006.

[9] O. M. Group. The common object request broker:
Architecture and specifcation. Technical report, Object Management Group, Oct. 1999.

[10] S. Henkler, J. Meyer, W. Schäfer, M. von Detten, and U. Nickel. Legacy component integration by the fujaba real-time tool suite. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2, ICSE '10, pages 267-270, New York, NY, USA, 2010. ACM

[11] Iso/iec 15940:2006 software engineering environment services. Technical report, International Organization for Standardization. Information technology, 2006.

[12] Edward A. Lee and Alberto L. Sangiovanni-Vincentelli, Component-based design for the future DATE 2011

[13] iFEST Project. iFEST - industrial Framework for Embedded Systems Tools. ARTEMIS-2009-1-100203, 2010.
[14] Jazz. http://jazz.net/.
[15] E. Kapsammer and T. Reiter. Model-based tool integration- state of the art and future perspectives 1.
[16] G. Karsai and J. Gray. Component generation technology for semantic tool integration. 2000 IEEE Aerospace Conference Proceedings Cat No00TH8484, pages 491-499, 2000.
[17] G. Karsai, A. Ledeczi, S. Neema, and J. Sztipanovits. The model-integrated computing toolsuite: Metaprogrammable tools for embedded control system design. In Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE, pages 50 -55, oct. 2006.
[18] G. Kramler, G. Kappel, T. Reiter, E. Kapsammer, W. Retschitzegger, and W. Schwinger. Towards a semantic infrastructure supporting model-based tool integration. In Proceedings of the 2006 international workshop on Global integrated model management, GaMMa '06, pages 43-46, New York, NY, USA, 2006. ACM.
[19] T. Margaria, R. Nagel, and B. Ste en. jETI: A Tool for Remote Tool Integration Tools and Algorithms for the Construction and Analysis of Systems. Volume 3440 of Lecture Notes in Computer Science, chapter 38, pages 557-562. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2005.
[20] MDWorkbench. MDWorkbench. http://www.mdworkbench.com, 2012
[21] T. Reenskaug, P. Wold, and O. A. Lehne. Working with objects: the Ooram software engineering method. Manning Publications, Greenwich, CT, 1996.
[22] C. Reichmann, M. Kiihl, P. Graf, and K. Muller-Glaser. Generalstore - a case-tool integration platform enabling model level coupling of heterogeneous designs for embedded electronic systems. In Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the, pages 225 - 232, may 2004.
[23] M. Seifert, C. Wende, and U. Amann. Anticipating Unanticipated Tool Interoperability using Role Models. pages 52-60, 2010.
[24] P. Sriplakich, X. Blanc, and M.-P. Gervais. Collaborative software engineering on large-scale models: requirements and experience in modelbus. In R. L. Wainwright and H. Haddad, editors, SAC, pages 674-681. ACM, 2008.
[25] F. Steimann. On the representation of roles in object-oriented and conceptual modelling. Data Knowledge Engineering, 35(1):83-106, 2000.
[26] A. I. Wasserman. Tool integration in software engineering environments. In SEE, pages 137-149, 1989.
[27] M. N. Wicks and R. G. Dewar. A new research agenda for tool integration. Journal of Systems and Software, 80(9):1569-1585, 2007.
[28] N. Guarino. Concepts, attributes and arbitrary relations. Data and Knowledge Engineering 8, 1992, 249-261.
[29] OSLC, Open Services for Lifecycle Collaboration http://open-services.net/
[30] Hein, C., Ritter, T., Wagner, M.: Model-Driven tool integration with ModelBus. Workshop Future Trends of Model-Driven Development. (2009)