An Integrated Model-Driven Service Engineering Environment

João Paulo A. Almeida, Maria-Eugenia Iacob, Henk Jonkers, Marc Lankhorst, and Diederik van Leeuwen

Telematica Instituut, P.O. Box 589, 7500 AN Enschede, The Netherlands
{JoaoPaulo.Almeida, MariaEugenia.Iacob, Henk.Jonkers, Marc.Lankhorst, Diederik.vanLeeuwen}@telin.nl

Keywords: model-driven design, service engineering, tool interoperability

1 Introduction

The design and provisioning of ubiquitous services is a challenging task. The requirements of mobility, context-awareness, attentiveness and personalisation justify the development of novel methods, abstractions and infrastructures (e.g., [6, 7, 8, 17]). In addition, the complexity, diversity and fast-changing nature of enabling technology platforms require design approaches which shield designers and providers from platform-specific details allowing them to concentrate their efforts on the services they intend to create and offer.

These factors have led us to define a model-driven approach for ubiquitous service engineering, which is reported in [3] in the context of the A-MUSE project [11]. The approach comprises activities which range from early stages of service design to the run-time provisioning of services. In order to enable rapid service development, this approach is based on the reuse of application and platform services which are developed and maintained independently, with their own lifecycles and lifespan. The reuse of platform services is promoted by describing platform-independent and platform-specific aspects of services separately in different models and relating different models by model transformations (according to the Model-Driven Architecture (MDA) [21]). The reuse of application services is facilitated by: (i) modelling techniques that allow service composition to be captured at a high-level of abstraction; (ii) the use of repositories to allow the services that have been independently developed to be discovered and reused; and (iii) mechanisms for service mediation and composition based on semantic information captured in service ontologies and service descriptions.

An important characteristic of the A-MUSE model-driven approach is the use of a number of existing modelling languages and notations in a common design framework. In this framework, metamodelling [19] serves as a basis for modelling language definition, reuse and specialization; and model transformation [20] serves the purpose of automating design activities, supporting the verification of the

consistency between models, and the generation of analysis models, e.g., for simulation and analysis of non-functional aspects of designs [15].

The variety of the techniques, languages and infrastructure services used and the range of the activities performed require comprehensive tool and platform functionality. In this paper, we discuss how our model-driven approach can be supported by composing tool services in an environment for service engineering.

The paper is organised as follows: section 2 provides some background on our model-driven approach; section 3 illustrates how the approach is applied with specific modelling techniques and platforms; section 4 discusses tool interoperability challenges; finally, section 5 summarises our conclusions and indicates topics for further research.

2 Context and Problem Description

In this paper, we use the term ubiquitous services to designate *context-aware* and *mobile* services. *Context-awareness* refers to the capabilities of applications to provide relevant services to their users by sensing and exploring the users' context [7, 8]. *Mobility* refers to the ability to access services anywhere.

The development of ubiquitous services involves the re-use of a number of existing enabling technologies and platforms, which include:

1. resource-constrained mobile device platforms (e.g., Symbian, UIQ, Series 60, Pocket PC), which allow users to access services anywhere;
2. sensor technology (e.g., GPS devices), which capture context information;
3. telecommunication platforms and standards (e.g., Parlay, Parlay/X, OMA, GPRS, UMTS), which provide mobile device connectivity, facilities for sending and receiving (multimedia) messages (MMS, SMS), establishing calls, billing, capturing context information, etc.; and,
4. middleware platforms (e.g., Web Services, CORBA, J2EE), which support the distributed interaction of the various components used to realize a service.

The complexity and heterogeneity of these technologies require service designers to be knowledgeable in these techniques. In addition, these technologies are often subject to change or evolution, which increases the cost of maintaining service realizations. Our model-driven approach addresses these issues by (i) providing a suitable abstraction for defining context-aware services at a high-level of abstraction; (ii) separating platform-independent and platform-specific aspects of designs (with the notion of abstract platforms [2]); and (iii) capturing expert's reusable design knowledge in model transformation specifications. For an extensive presentation of the methodological support we refer to [3].

We distinguish the following stakeholders in A-MUSE's model-driven approach: the *expert architects*, who define (domain specific) modelling languages, abstract platforms and transformations; the *service designers*, who create or compose specific services; the *service users*, who are the end-users of services; and, the *service providers*, who offer services to the end-user. These groups of stakeholders cover three main phases of the service engineering process: the *preparation phase,* the *service creation phase* and *the execution phase* [3].

From the perspective of the expert architects, the service engineering environment must be set-up, preferably using available platforms and tools, and tool customization and integration mechanisms. From the perspective of the service designer, the environment should intuitively support the rapid engineering of services using model transformations and analysis and simulation support. From the perspective of the service provider, the environment should support low-cost maintenance and operation of service realizations. Finally, service users expect the environment to deliver services with acceptable functional and non-functional properties.

3 Service Engineering Environment Usage Scenario

In this section we illustrate how the model-driven approach is applied with specific modelling techniques and platforms. This allows us to present specific activities that are performed, which sets the requirements for the service engineering environment.

Let us consider that the expert architect defines the following levels of models [4] which are populated by the service designer during service creation (see Fig. 1):

1. The *service specification level*. This level of models describes the behaviour of the context-aware service from an integrated perspective, without distinguishing application parts that provide the service. This level consists of the following models:

- a model expressed in a domain-specific language (A-MUSE DSL) that specifies the behaviour of the service being designed and its relation to context information values. Context information value types are defined in a context ontology;
- a UML class diagram that defines globally the information model of the service;
- a UML global component diagram that models the structural organisation of the service in terms of abstract components;
- an architectural overview model of the service, expressed in the ArchiMate language [16]. This is an integration model that manages the cross-model relationships between the former models. This integration model can be derived from them through abstraction transformations.

2. The *platform-independent service design level*. At this level of models the service designer will derive, using refinement transformations (defined by the expert architect), three types of detailed models that describe the behaviour, information and structure of the context-aware service from a distributed perspective. The DSL model is refined into a detailed behavioural model expressed in ISDL [14], the global class diagram is refined into a detailed information model also expressed as a class diagram, and the global component diagram is refined into a detailed structure model, expressed as component/deployment diagrams. At this level, the service designer describes the service being designed as a composition of: (i) service-specific components;

(ii) the A-MUSE abstract platform (as defined by expert architects); and (iii) services offered by third parties.

3. The *platform-specific service design level*. This level of models describes the realization of the service for a particular set of technology platforms. The flexibility of the relation between the platform-independent and the platform-specific service design levels allows different platforms to be used. A service designer applies model transformations to platform-independent models to obtain models at this level. For example, the transformation denoted by T_2 in Fig. 1 generates target models for the platforms denoted by Π_{B2}. These models are BPEL specifications [13] which orchestrate (using a BPEL engine and SOAP a number of web services (e.g. context or action services provided by Parlay-X [25]) for which WSDL interfaces are provided as well as code for mobile devices in Java (following Java MIDP restrictions). Transformation T_1 and T_2 are defined by the expert architect in the preparation phase.

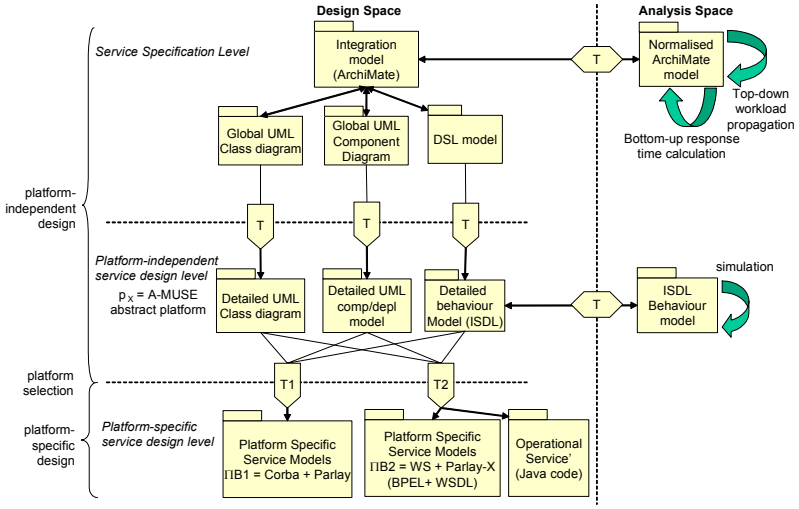

Fig. 1. Levels of models: from service specification to platform-specific realization

At each level, models can be analysed or simulated [15]. Once the service designer has completed the creation process, the *service provider* can proceed by registering the newly created service in a run-time, publicly accessible repository, and by monitoring the usage of this service. Here the service can be discovered and eventually executed, possibly, on a wearable device (e.g. a Pocket PC) by the *service user*. Services can be subject of activities, such as: publication, usage management, monitoring, search, discovery, selection, negotiation, withdrawal etc.

4 Integration Solutions

A single software tool that provides complete support to the design and provisioning of ubiquitous services does not exist. However, many tools exist that

fill in some of the desired functions; e.g., there are separate model editors, transformation tools, monitoring tools, simulation tools, etc. We propose to reuse some of these tools and integrate them in the service engineering environment. In order to achieve this, we must overcome differences in concepts, standards and techniques that these tools use, e.g., for model representation and storage. In other words, we will have to solve several aspects of the 'tool interoperability' problem.

Technically, the integration of tools can be characterised by the following aspects (including those defined in [24, 26]): (i) *data integration* addresses the issue of sharing data between tools and the storage of diagrams, models, views and viewpoints; (ii) *control integration* addresses the issue of communication and coordination between tools (and the integration framework, if existing); (iii) *presentation integration* concerns the user interaction with the integrated set of tools, and facilitates user interaction through intuitive and uniform interfaces; and (iv) *process integration* concerns the coordination of the various design activities supported by different tools.

In the considered application domain, data integration concerns the integration of *model* data. Model data interoperability issues may occur at the syntactic or the semantic level, and at different meta-levels. We distinguish the following issues: (i) *syntactic data interoperability issues*: model data is stored in different formats, e.g., XMI [19] or in a proprietary binary tool format; (ii) *model-level interoperability issues*: models are expressed in different modelling languages, e.g., UML and ISDL; and (iii) *metamodel-level interoperability issues*: tools use different metamodelling formalisms, e.g., Meta-Object Facility (MOF) [19] and Eclipse Modelling Framework (EMF) [10]. A combination of these issues may also occur, e.g., a model may be expressed in different modelling languages, based on different metamodelling formalisms.

The alternative tool integration solutions that we propose in this section differ in the way in which they deal with the data integration issue, and in the degree of presentation integration. In some of the solutions, tool service orchestration is included in order to automate process integration. Data integration is achieved in different ways, e.g., by using common formats for data exchange and transformations.

4.1 Ad Hoc Integration

We consider ad hoc tool integration as the baseline for the comparison of our solutions. This is current practice in most situations where tool integration is required. Data integration is supported by custom-built point-to-point adapters connecting the various repositories. Experts and service designers must manually coordinate the usage of the various tools, effectively performing control, presentation and process integration manually. Despite the rudimentary integration, this architecture enables direct reuse of different tools and their front-ends. These front-ends are not integrated, and experts and service designers must learn how to use different environments and must share the same tools using different functionalities.

This solution is exemplified with following tools: OptimalJ [22] is used to support UML modelling, metamodelling and model transformation. Protégé [23] is

used to support OWL and SWRL modelling for context information and context rules. Eclipse [9] is used to support metamodelling and to support the following plug-ins: a plug-in for the modelling of context-aware services in A-MUSE's DSL, a plug-in for modelling various aspects of context-aware services in the ArchiMate language [16] and a plug-in to support services composition. Grizzle [12] is used to edit platform-independent service designs in ISDL [14], which are created by model transformations implemented in OptimalJ's transformation engine. Sizzle [14] is used to simulate these platform-independent service designs. A services execution platform is connected to the design environment through a run-time repository, where services can be registered and discovered by the service composition mechanism.

An example adapter in this tool environment converts OWL models stored in Protégé's RDF/XML repository and imports them into Eclipse's EMF repository. Another example of adapter transforms A-MUSE DSL models in EMF to A-MUSE DSL models in MOF. This is necessary for transformation in OptimalJ.

4.2 Tool Bus Integration

The ad hoc solution requires custom-built point-to-point adapters and supports rudimentary data integration only. An architecture that improves on this is the tiered tool integration architecture depicted in Fig. 2. In this solution, back-end tool services (bottom of Fig. 2) are "virtualized" through the tool services bus, which offers a common interchange format for models. Adapters to these tool services should be built to bridge the differences in model format. The tool bus provides a notification service which is used to decouple the various tool services.

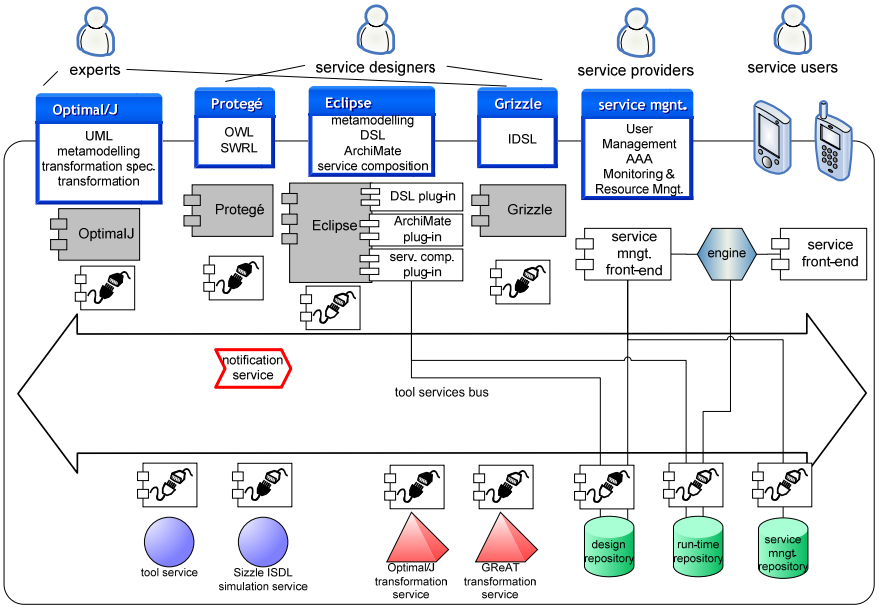

Fig. 2. Tool bus integration

This approach is similar to the one that is followed in the tool bus of the MODELWARE project [18]; however, in our approach, the bus is used to support both service design and service provisioning activities, and is thus relevant at service design-time and run-time. Because of the virtualization of repository services, it is possible to use a single repository for design models, simplifying the issues of model synchronization (control integration). In Fig. 2, lines show which front-end components use each back-end repository tool services. For the sake of clarity, we omit relations between front-end components and other tool services.

Front-ends in this solution are not integrated, i.e., the solution does not provide presentation integration. End-users access tool services through existing front-ends, such as Eclipse or Protégé, or directly use modelling tools such as Grizzle. Also for the front-end, adapters are required so that these tools can use repository services offered by the bus instead of their own repositories.

4.3 Tool Bus Integration with Meta-level Transformations

A drawback of the integration solutions described above is that a dedicated adapter has to be built for every tool. This limits the flexibility, because adding a new tool requires development effort. Fig. 3 shows an alternative architecture, which does not impose the use of a single interchange format. In this architecture, no adapters are used. Instead, the bus uses model transformations to convert between different model formats. These are transformations at the meta-level, as opposed to the transformations on models that are used in the regular model-driven service development process. Orchestration functionality takes care that the right transformations are carried out at the right time; combining repository and transformation services in order to bridge differences in model formats. For example, an EMF/MOF transformation service allows EMF models to be stored in the MOF repository, from which OptimalJ transformations can be applied.

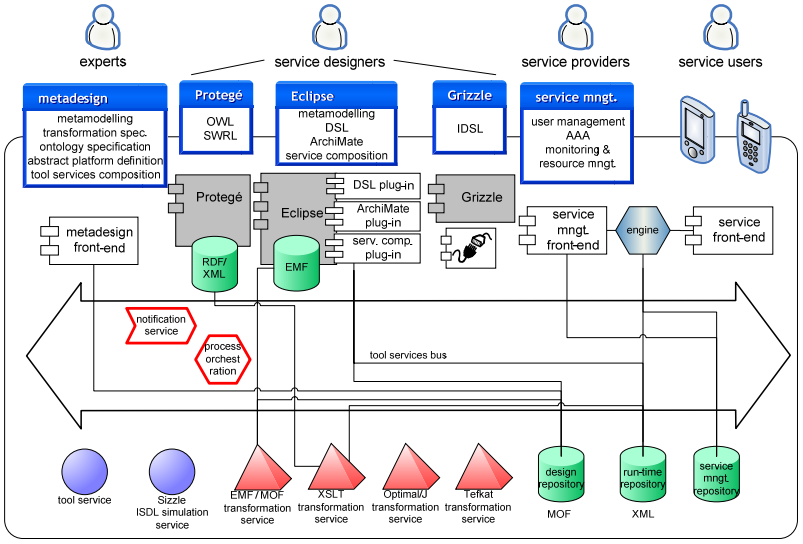

Fig. 3. Tool bus integration with meta-level transformations

4.4 Tool Bus Integration with Integrated Front-Ends

The tool bus solutions as presented above provide, to different degrees, solutions for data integration, control integration and process integration. However, they do not provide presentation integration (for service designers). In order to achieve this, the solution in Fig. 4 contains front-end components that are custom-built to support presentation integration. Each stakeholder is offered its own integrated environment through which interaction with the service engineering environment is possible. Back-end tool services exemplified here include Sizzle ISDL simulation, GReAT [1] and OptimalJ transformation services, and repository services. A generic tool service is depicted to denote that this architecture is open for inclusion of other tool services. In addition, orchestration functionality in the tool bus supports process integration. Fig. 4 depicts integrated front-ends in combination with a tool bus solution using adapters. Integrated front-ends can also be used in combination with the tool bus solution which uses meta-level transformations.

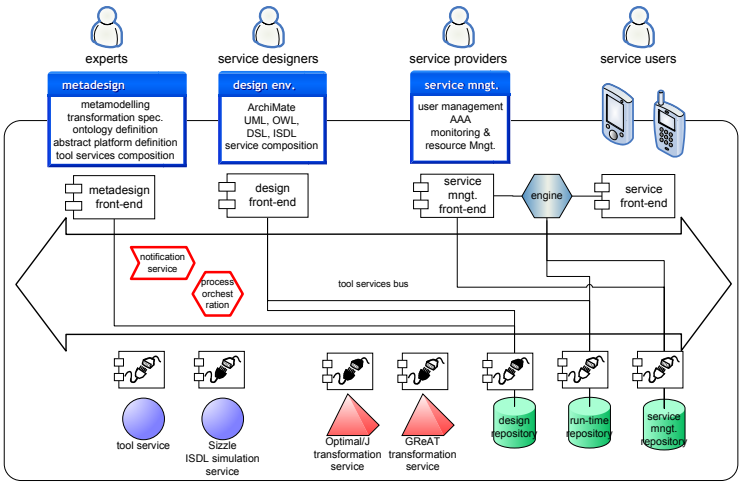

Fig. 4. Tool bus integration with integrated front-ends

Table 1 summarizes the alternative tool interoperability solutions, in terms of the aspects of integration addressed and the level of flexibility of the solution.

Table 1. Summary and comparison of integration solutions discussed in this paper

	Data integration	Control integration	Presentation integration	Process integration	Level of flexibility
Ad hoc	adapters	no	no	no	very low
Tool bus	common format	yes	no	no	low
Tool bus with meta-level transformations	transformation	yes	no	yes	high
Tool bus with integrated front-ends	common format	yes	yes	yes	low
Bus meta-level transf. integrated front-ends	transformation	yes	yes	yes	high

5 Conclusions and Outlook

In this paper we have proposed an integrated environment for service engineering. This environment can support a number of existing modelling languages and notations which populate a common design framework. We have discussed how existing (model-driven) tools can be integrated in this environment and outlined the main choices with respect to tool interoperability. We have discussed four integration solutions. These solutions address data integration in different ways, and offer different degrees of flexibility, control, presentation and process integration.

We regard ad hoc integration as a short-term solution for the implementation of the A-MUSE service engineering environment. As an intermediate step, we intend to align our tool development efforts with those of MODELWARE, exposing tool services for (context-aware mobile) services engineering in the ModelBus. However, our long-term goal is the implementation of the tool bus with meta-level transformations and integrated front-ends. Since the construction of integrated front-ends requires significant customization effort, this raises the need for flexible meta-tools which simplify the activities of the expert architect. This includes not only support for metamodelling and transformation specification but also support for the definition of different notations and means of visualization for analysis and simulation results.

Further investigation is necessary with respect to repository services. Since we intend to use model repositories to support service design and provisioning, model repositories are relevant at design-time and run-time. In particular, repositories are used for the registration and discovery of services developed independently by third-parties. Main challenges for this use of model repositories refer to meeting time performance, scalability and access control requirements. The use of a federation of distributed model repositories to handle time-sensitive context data should also be investigated. This could lead to a seamless integration of design-time and run-time data/model representation techniques.

We also envision the use of transformation services at run-time, in particular for handling with the diversity of end-user terminal devices. In this case, platform-specific user interfaces may be generated at run-time depending on screen resolution, available input devices and user preferences.

Acknowledgements

This work is part of the Freeband A-MUSE project (http://a-muse.freeband.nl), which is sponsored by the Dutch government under contract BSIK 03025.

References

1. Agrawal, A., Karsai, G., Ledeczi, A.: An end-to-end domain-driven software development framework. In: Proc. 18th Annual ACM SIGPLAN Conference on Object-

Oriented Programming, Systems, Languages and Applications (OOPSLA'03). ACM Press (2003) 8–15
2. Almeida, J.P.A., Dijkman, R. van Sinderen, M., Ferreira Pires, L.: On the Notion of Abstract Platform in MDA Development. In: Proc. 8th IEEE Int'l Conf. on Enterprise Distributed Object Computing (EDOC 2004). IEEE CS Press (2004) 253–263
3. Almeida, J.P.A., Iacob, M.E., Iacob, S.: Methodological Framework for Freeband Services Development, Freeband A-MUSE/D2.3a. Telematica Instituut, The Netherlands (2004)
4. Almeida, J.P.A., Jonkers, H., Iacob, M.E., Quartel, D.: Platform-Independent Modelling of Service Infrastructure Components: Towards the A-MUSE Abstract Platform, Freeband A-MUSE/D1.6. Telematica Instituut, The Netherlands (2005)
5. Blanc, X.: ModelBus: A MODELWARE White Paper (2005); http://www.modelware-ist.org/public_documents/ModelBusWhitePaper_MDDI.pdf
6. Chen, H. Finin, T., Joshi, A.: An Ontology for Context-Aware Pervasive Computing Environments. Knowledge Engineering Review, Special Issue on Ontologies for Distributed Systems, Vol. 18, No. 3. Cambridge University Press (2003) 197–207
7. Dey, A. K., Salber, D., and Abowd, G. D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction, 16(2-4) (2001) 97–166
8. Dockhorn Costa, P. Ferreira Pires, L., van Sinderen, M.: Designing a Configurable Services Platform for Mobile Context-Aware Applications, Int. Journal of Pervasive Computing and Communications (JPCC), Vol. 1, No. 1. Troubador Publishing (2005)
9. Eclipse; http://www.eclipse.org/
10. Eclipse Modeling Framework (EMF); http://www.eclipse.org/emf/
11. Freeband A-MUSE project; http://a-muse.freeband.nl
12. Grizzle; http://isdl.ctit.utwente.nl/tools/grizzle
13. IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems: Business Process Execution Language for Web Services (BPEL4WS), Version 1.1 (2003)
14. Interaction Systems Design Language (ISDL); http://isdl.ctit.utwente.nl
15. Jonkers, H.; Iacob, M.; Lankhorst, M., Strating, P.: Integration and Analysis of Functional and Non-Functional Aspects in Model-Driven E-Service Development. In: Proc. 9th IEEE EDOC Conference, IEEE CS Press, (2005) 229–238
16. Jonkers, H., Lankhorst, M., van Buuren, R. et al.: Concepts for Modelling Enterprise Architectures. In: Int. J. of Cooperative Information Systems, Vol. 13, No. 3, (2004) 257–287
17. McFadden, T., Henricksen, K., Indulska, J., Mascaro, P.: Applying a Disciplined Approach to the Development of a Context-Aware Communication Application. In: 3rd Annual IEEE Conf. on Pervasive Computing and Communications (Percom 2005), IEEE CS Press (2005)
18. MODELWARE IST: ModelBus Architecture Specification. MODELWARE D3.1 (2005); http://www.eclipse.org/mddi/docs.html
19. OMG: Meta Object Facility (MOF) Specification, V1.4, formal/02-04-03 (2002)
20. OMG: MOF 2.0 Query / Views / Transformations RFP, ad/2002-04-10 (2002)
21. OMG: MDA-Guide, Version 1.0.1, omg/03-06-01 (2003)
22. OptimalJ; http://www.compuware.com/products/optimalj/
23. Protégé; http://protege.stanford.edu/
24. Schefstrom, D., van den Broek, G.: Tool Integration: Environments and Frameworks. John Wiley & Sons, Inc., New York (1993)
25. The Parlay Group: Parlay X Web Services Specification, Version 2.0; http://www.parlay.org/en/specifications/
26. Thomas, I. and Nejmeh, B.A.: Definitions of Tool Integration for Environments. IEEE Software, Vol. 9, No. 2 (1992) 29–35