Tool Integration at the Meta-Model Level within the FUJABA Tool Suite

Sven Burmester*, Holger Giese†, Jörg Niere, Matthias Tichy†, Jörg P. Wadsack‡, Robert Wagner§, Lothar Wendehals‡
Software Engineering Group
Department of Computer Science
University of Paderborn
Warburger Straße 100
33098 Paderborn, Germany
[burmi|hg|nierej|mtt|maroc|wagner|lowende]@upb.de

Albert Zündorf
Research Group Programming Methodologies
Department of Computer Science
University of Kassel
Wilhelmshöher Allee 73
34121 Kassel, Germany
albert.zuendorf@uni-kassel.de

Abstract

Current initiatives in the field of integrated development environment (IDE) and CASE tool integration such as Eclipse as well as the lately happened acquisitions of Rational and Together by major IDE vendors indicate that tool integration has become an important issue for the IT industry. However, as outlined in this paper the current integration platforms fall short to address the underlying problems of overlapping data models and their consistency when it comes to a tool integration. Within the FUJABA TOOL SUITE *in contrast a framework has been developed which enables an integration of tools at the meta-model level. We report the employed concepts for meta-model integration and consistency management in this paper and illustrate them by means of an example.*

Keywords

Tool interoperability, meta-model, model integration, framework, graph grammars, plug-in

1. Introduction

Nowadays tool integration gains importance in industry as well as in academia. This trend results from unifying efforts like UML and XML of the last years. Tool integration has three main problem areas.

First problem area is the integration of functionality. This means, that one tool would like to be able to use some functionality of another tool. This is usually easy to achieve via some Application Programming Interface (API). In addition, one tool may want to modify and enhance a certain functionality of some other tool. This requires mechanisms like the template method design pattern [3] or a listener concept for command executions. A tighter coupling of different functionality or even a replacement of existing functionality with an alternative implementation is not always supported.

Second problem area is the adaption of the (graphical) user interface of one tool by another. This includes the extension of menus, tool bars and dialog windows as well as the extension of the representation of certain data at the user interface. The former points are frequently supported, the last point is seldom.

Third problem area is the access and extension of data. The data of one tool may be utilised by another tool. A tool may even try to achieve a tight integration between the data of the used platform and its own data. Generally, the problem area of integrating the data of different tools may be classified as follows: (1) the same content in different formats is integrated; (2) an integrating part contains added information; (3) the integration has to deal with overlapping information.

To integrate tools that exchange the same content in different formats, typical solutions are standardised exchange formats (e.g. XMI) or format transformation without loss of information. A comparison about how and where the schemas are defined, i.e. organisation of the model data to be exchanged, is given in [9].

In case of added information a typical solution is to add extensions to the original format and hide them for the original tool. Another solution is to use a meta-model that is extended for the integration purpose. The resulting dependency between the models of the different tools is that the extension depends on the involved tool or format.

To integrate tools that exchange overlapping information the multiple meta-models have to be integrated and extended. To handle the data consistency during changes, an update mechanism has to be established. This usually requires bi-directional references between the data elements of the different tools in order to facilitate the propagation of data changes of one tool to the corresponding data elements of the other tool. Using bi-directional references easily creates mutual compile time dependencies. However, mutual compile time dependencies must be avoided in order to allow independent tool development and to provide platforms that

*Supported by the International Graduate School of Dynamic Intelligent Systems. University of Paderborn

†This work was developed in the course of the Special Research Initiative 614 - Self-optimizing Concepts and Structures in Mechanical Engineering - University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft (DFG).

‡This work is part of the FINITE project funded by the DFG, project-no. SCHA 745/2-1.

§This work has been supported by the DFG grant GA 456/7 ISILEIT as part of the SPP 1064.

shall be extensible by tools to be developed in the future.

To preserve consistency when exchanging overlapping artifacts is however not a new problem. The IPSEN approach [14] presented a framework to integrate tools through a common meta-model. However, this approach is not easily applicable for the integration of existing tools.

The CORUM approach [22] suggests the usage of a common information model which is used by all tools. For the integration of tools, which are not based on the CORUM approach and cannot use the CORUM API, the special input can be generated by means of transformation tools. In [11] the CORUM II approach is presented, integrating different reengineering tools operating on different levels. In order to exchange data all tools need to agree on a common syntactic format. Thus, these approaches are not applicable for the tight integration of software tools.

One prominent tool integration platform is Eclipse which is promoted by several mid-size and major IT enterprises. However the supported integration aspects are restricted to a framework which is build on a mechanism for discovering, integrating, and running plug-ins. Each plug-in uses so called extension points and the Eclipse API. Further two addons to Eclipse, the Generative Model Transformer (GMT) and the Eclipse Modeling Framework (EMF) are provided. The GMT enables the transformation of one model to another. The EMF, beside code generation facilities out of XMI descriptions, provides a basis for interoperability of EMF-based tools. This basis is the Ecore meta-model, i.e., a central meta-model for tool integration. Further to our best knowledge, most UML tools, like TogetherJ or Rational Rose, are also only extendable via APIs.

In this paper we present the approach chosen within the Fujaba Tool Suite which overcomes these problems by means of an extensible tool integration framework. Fujaba itself is an Open Source UML CASE tool project. It was started by the software engineering group at the University of Paderborn in fall 1997 and was designed as one monolithic application integrating several functionalities from different domains. In 2002 Fujaba has been redesigned and became the Fujaba Tool Suite with a plug-in architecture allowing developers to add functionality easily while retaining full control over their contributions.

At the early days, Fujaba had a special focus on code generation from UML diagrams resulting in a visual programming language. Today, at least four rather independent tool versions are under development at the Universities of Kassel and Paderborn for supporting (1) reengineering, (2) embedded systems, (3) the Fujaba Development Process, and (4) education of object-oriented concepts. According to our knowledge, quite a number of research groups have also chosen Fujaba as a platform for their own UML related research.

The next section describes the architecture of the Fujaba Tool Suite to support tool integration. The integration on the model level is presented in Section 3. In Section 4 the consistency management issues that result from integration on model level are sketched. Before concluding and discussing future work, Section 5 presents experiences with the Fujaba Tool Suite.

2. Architecture

The Fujaba Tool Suite provides a platform for the integration of third party modelling tools. It offers several means to add functionality and provides infrastructure support for building menus, loading images, drawing diagrams (including managing consistency between model and view), etc.

Plug-ins are the basic mechanism to add third party software to Fujaba. A plug-in has to provide information about its version, its dependencies on other plug-ins and its developers. Additionally, a plug-in can change menus and menu entries as well as pop-up menus and toolbars described in an XML based graphical user interface configuration file. Additionally, it is possible for plug-ins to notice the execution of menu actions of other plug-ins.

Fujaba's meta-model is based on the abstract syntax graph (ASG) concept. The ASG concept provides the building blocks for the integration of the Fujaba meta-model with the different meta-models provided by the third party developers. In order to support this integration Fujaba's ASG concept implements the meta-model integration pattern presented in Section 3.

A basic problem for extensible frameworks is to provide the possibility to integrate meta-models although the different meta-models might be in different parts (plug-ins) of the software system.

In monolithic applications a tight integration of different meta-models is no problem, since the classes just have to be connected via bi-directional associations. In an extensible framework this solution is obviously not appropriate since the different parts would not even compile separately.

Therefore another solution must be used which preserves the separation of core and plug-in as well as allows the tight integration of the meta-models. Technically spoken, it must be possible to connect both meta-models bi-directionally without explicitly adding references to the meta-models.

These requirements demand a smart and flexible solution for bi-directional integration. The basis of the architecture is the Fujaba meta-model. Every tool that participates in the Fujaba Tool Suite is a plug-in. Two plug-in interoperability variants exist. First, an existing tool (plug-in) with a meta-model should be extended by a new plug-in, i.e., the meta-model of the existing tool should be extended in the new plug-in. Second, two existing tools (plug-ins) should be used in a consistent manner, i.e., instances of two meta-models should be kept consistent.

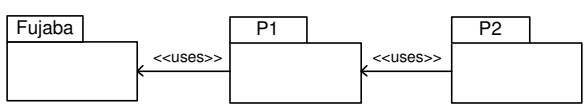

Figure 1: Architecture with meta-model extension

Figure 1 shows variant 1, the meta-model extension for an existing tool (plug-in). The plug-in P1 is an independent tool with its own meta-model. P2 is the plug-in that extends the meta-model of P1. In terms of compiling this means that P1 can be compiled alone whereas P2 cannot.

Second, two independent (existing) tools, plug-in P1 and plug-in P2, should be used together in a consistent manner. The meta-models of these two plug-ins are independent. The

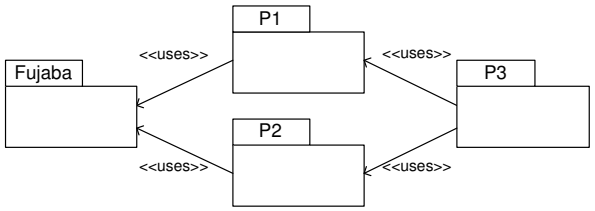

Figure 2: Architecture with meta-model integration

aim is that both tools P1 and P2 can be used whereas their meta-models are integrated. Figure 2 shows such a meta-model integration within the plug-in P3. P3 establishes the link between the two disjunct meta-models of P1 and P2. Hence, P1 and P2 are compilable separately. Of course P3 needs P1 and P2 to compile.

The next section describes the proposed meta-model integration pattern and its application in the FUJABA TOOL SUITE to fulfill the above specified requirement on meta-model extension and integration.

3. Integration on the Model Level

When connecting meta-models of two different tools or plug-ins, bi-directional associations are needed to navigate the elements of the meta-models. Figure 3 shows such a tight connection between two elements in different meta-models using an association. Each of both classes references the other class. Links between objects of those meta-model elements can then be navigated in both directions.

Figure 3: Tight connection between meta-models

Such a bi-directional association prevents a separate compilation of the two tools. Both meta-models depend on each other so that the tools cannot be deployed separately. For extending an existing tool where meta-model elements are distributed over different plug-ins, this approach is not sufficient.

Meta-Model Integration Pattern

In Figure 4 a pattern is presented to integrate different tools on the meta-model level. A Meta-Model B can be integrated into Meta-Model A without affecting the interface and the compilation of Meta-Model A, but still a bi-directional association between elements of the two meta-models is given.

The two classes MetaModelElement and MetaModelAssociation are provided by the tool that enables integration on the model level. All elements of meta-models have to subtype the class MetaModelElement. This class provides a qualified association to the class MetaModelAssociation. To model a bi-directional association between ConcreteElementA of Meta-Model A and ConcreteElementB of Meta-Model B as depicted in Figure 3, a subclass of MetaModelAssociation has to be created and an association between ConcreteElementB and ConcreteAssociation has to be established. This association has the same cardinalities as the intended association in

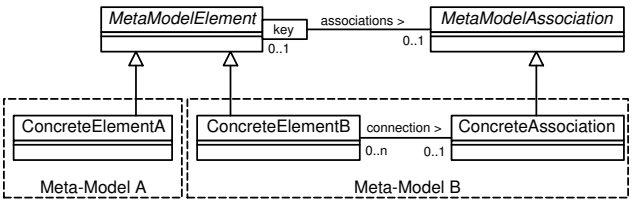

Figure 4: Meta-model integration pattern

Figure 3. All associations in this pattern are bi-directional. The name of the ConcreteAssociation class is used as the key for the qualified associations-Association.

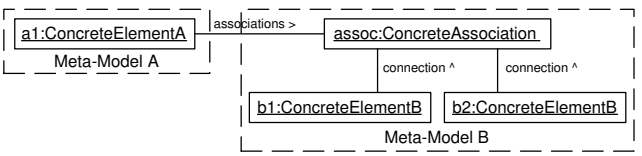

Figure 5: Objects linked via an association object

The association modelled by an additional class ConcreteAssociation can now be navigated in both directions. Figure 5 shows how objects of ConcreteElementA and ConcreteElementB are linked via an association object. It is obvious how to navigate from a ConcreteElementB object b1 via the object assoc of type ConcreteAssociation to an object a1 of type ConcreteElementA. In the other direction from a ConcreteElementA object a1 you can get the ConcreteAssociation object assoc by using the class name of the ConcreteAssociation as key and then navigate to an object b1 of type ConcreteElementB.

Sample Pattern Application

The following example for meta-model integration stems from our ISILEIT[1] project. ISILEIT explores the possibilities of modern specification languages concerning their usefulness for the specification of flexible and autonomous production control systems. The production control system depicted in Figure 6 consists of several working stations and robots connected by monorail tracks and switches. The switches are controlled components which allow forking and joining of the tracks. Special shuttles moving along the tracks transport materials and goods between the working station.

The specification of the controller software is a combination of SDL block diagrams [7], UML class diagrams, and UML behaviour diagrams like collaboration diagrams, activity diagrams and state charts [17] as an executable graphical language. We use SDL block diagrams to specify the overall static communication structure connecting processes by channels and signal routes. From this block diagram we derive an initial class diagram in which each process is represented by a respective class in the class diagram. In addition, each signal received by a process in the SDL block diagram is mapped to a method in the appropriate class in the UML class diagram. Finally, the class diagram is refined and for each process class a state chart is assigned. This briefly summarised modelling approach allows us both to

[1] ISILEIT is the German acronym for "Integrative Specification of Distributed Production Control Systems for the Flexible Automated Manufacturing".

Figure 6: Snapshot of a production system

specify the reactive behaviour and (with respect to the other diagrams) the modification of complex application specific object-structures. For a more detailed description of our modelling approach see [12, 15].

Since the UML meta-model and appropriate diagram editors were already integrated in our FUJABA TOOL SUITE, we just had to implement an editor and the underlying meta-model for SDL block diagrams. To achieve a consistent specification between the SDL block diagram (processes) and the UML class diagram (process classes), changes in the UML meta-model have to be reflected in the SDL meta-model and vice versa. Thus, a tight integration of both meta-models was required.

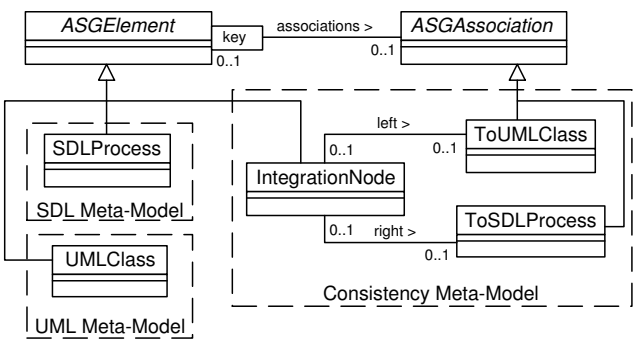

Figure 7: Integration of SDL and UML Meta-Models

Figure 7 shows how the meta-model integration pattern was applied twice to the UML and SDL meta-models. A connection between an SDLProcess and a UMLClass is established. Details about the IntegrationNode will be given in the next section.

4. Inter-Model Consistency

In an integrated development environment with one general meta-model, consistency is preserved by the abstract syntax of the meta-model itself and additional well-formedness rules. Therefore, the consistency can be preserved within the implementation of the development environment.

This does not hold for a tool integration platform. In a tool integration platform new tools are added and old tools are replaced by new ones. Usually, a new tool extends an existing meta-model of another tool or provides its own meta-model with overlapping data concerning other tools. This often leads to inconsistencies between the data of the integrated tools.

To overcome this problem, a third party tool developer has the possibility to develop the tool for one special case and to preserve the consistency in the tool itself, as seen in the previous section. The main drawback of this method is that the new tool cannot be used without the tool it relies on. For the example from Section 3 this would mean, that the SDL editor cannot be used without the UML editor. Another drawback is that a third party tool developer cannot foresee consistency rules and integrations for tools which will be developed in the future. Moreover, he cannot foresee in which way and in which combination with other tools his tool will be used. Therefore, we need a flexible mechanism to specify semantic relationships between syntactically unrelated meta-models.

For this purpose, we have implemented a flexible consistency management plug-in. The consistency management system allows the specification and the automated checking of consistency rules. The consistency rules and the consistency checking mechanism are based on triple graph grammar theory [18]. In the original work [13], triple graph grammars are used for the transformation between different meta-models using an integration model (cf. Figure 8).

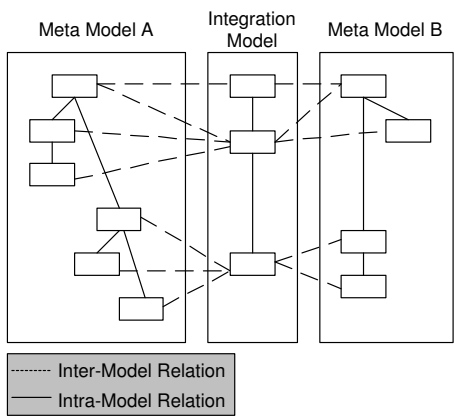

Figure 8: Triple Graph Grammar

Following this idea, we have adopted the triple graph grammars based on and using the integration pattern from the previous section. Assume that Meta-Model A is the SDL meta-model and Meta-Model B is the UML meta-model. Then we need bidirectional association from the Integration Model to both meta-models. These Inter-Model Relations are implemented using the meta-model integration pattern.

In Figure 9, an application of a consistency rule is depicted. In our example this means, that each time the user creates a new SDL process, cf. 1. create() in Figure 9, the mechanism will check the consistency rule and will search for the appropriate UML class in the class diagram. Since no such class can be found, the automatic repair action will create a new class, the appropriate integration node and new links between the SDL process and the UML class. This is depicted in Figure 9 where the :IntegrationNode to-

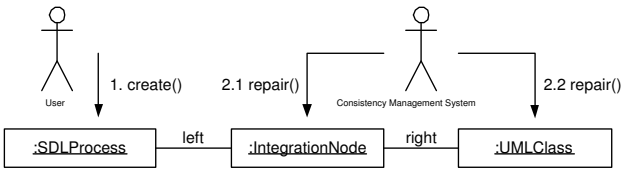

Figure 9: TGG Consistency Rule Example

gether with the left link and the :UMLClass with the right link are created. This way further consistency checks can be executed, for example checking for equal names or other constraints that should hold between the SDL process and the UML class. A more detailed description about our consistency management system can be found in [20].

5. Experiences

After the redesign of FUJABA, it is now possible to integrate third party software through plug-ins by using the meta-model integration pattern. The redesign process comprised reengineering and analysis of the monolithic architecture. The parts which build the application's core and the ones which should be rolled out into plug-ins needed to be identified. When extracting the components all associations between the core and the new created plug-ins needed to be transformed to instances of the meta-model integration pattern.

Reengineered Plug-Ins

Pattern Definition and Recognition. Two successfully extracted plug-ins are the pattern definition and pattern recognition plug-ins. The patterns are graphically defined as graph-rewrite-rules and related to FUJABA's UML meta-model for automatic code generation of pattern engines. The engines are used by the pattern recognition plug-in to detect pattern instances in source code. The source code is parsed into FUJABA's meta-model. Information about found instances within the ASG are stored as annotation objects linked to the ASG elements. A more detailed description of our pattern recognition approach can be found in [16].

SDL and Consistency Management. Besides the extracted pattern definition and recognition, and the already mentioned SDL and consistency management plug-ins a couple of new plug-ins have been developed since FUJABA offers this mechanism.

New developed Plug-Ins

Real-Time State charts. In order to support the design of embedded systems a real-time variant of state charts was created [2, 4]. Execution times, deadlines and time constraints are considered on the modelling level and induce transitions to react and fire within predictable time intervals. Events are generated through the call of special signal-methods wherefore an association to the class diagram is needed.

Service-Based Architectures. As part of an effort to create a process for the development of service-based architectures [19] two plug-ins have been created for deployment and component diagrams. In order to provide a seamless support it is necessary that the interfaces within the component diagrams are linked at the meta-model level with the classes modelling the interfaces in the class diagram. Similarly the components specified in the component diagrams must be linked to the components deployed on different nodes in the deployment diagram. We have successfully applied the proposed pattern to satisfy the above described requirements.

Memory Constrained Embedded Systems. In a collaborative research effort our proposed meta-model integration is used to support the development of memory constrained embedded systems. Here we have to integrate the meta-models of class diagrams and the real-time variant of state charts with the meta-models of UML 2.0 [1] component and deployment diagrams for memory constrained embedded systems. We are currently realising this support using our pattern for meta-model integration.

XProM. The XProM plug-in to FUJABA is a student project from University Braunschweig. The XProM plug-in extends the FUJABA TOOL SUITE with textual project documents supporting embedded editable UML diagrams and with direct support for the FUJABA development Process (FUP). In addition, XProM maintains a direct relationship between diagram elements and the sections of the project handbook, describing these diagram elements. This facilitates to keep diagrams and their descriptions consistent and up-to-date. To achieve this, the XProM plug-in makes heavy use of bi-directional links between the FUJABA core meta-model and the meta-model of its textual project documents.

Data Reengineering. Other plug-ins were constructed for data reengineering tasks. For data reverse engineering purposes a plug-in that enables access to data repositories via JDBC was added. This plug-in, which is part of the REDDMOM project, also comprises an extended entity relationship (EER) editor for data model representation and manipulation. Further it contains mapping rules from logical schemas (EER diagram) to conceptual schemas (UML class diagram) and Java code generation for transactional data access. Other small plug-ins in this context are an editor for schema refactoring operation definition and execution, and an editor for constructing and generating triple-graph-grammar transformations. Beside other publications, a good overview about the functionality of these plug-ins is given in [8]; the implementation is described in [21].

Web Application. Another plug-in which is part of REDDMOM is the web application plug-in. Given web page templates of the application the user can model the web application. Therefore the UML and web (XML) meta-models had to be combined. Thereby, the application logic can be processed and the results can directly be added to templates. Details of this plug-in can be found in [10]

Package Diagram. A plug-in that is under development is the addition of package diagrams to FUJABA. Package diagrams are used in several other plug-ins, e.g. XProM or REDDMOM. The functionality is restricted at the moment to nested package diagrams, i.e., a package diagram can contain another package diagram, a class diagram or a use case diagram. This will be extended to more than one package per diagram and visualisation of the nested structures for packages and classes.

6. Conclusions

The described concepts of the FUJABA TOOL SUITE for meta-model integration and consistency management provide a sound basis for the integration of plug-ins as demonstrated with the ISILEIT example. Thus, in contrast to other tool integration platforms such as Eclipse the integration between independently developed plug-ins can include the plug-in data models as well.

Our in the last section summarised experience further indicate that for many interesting solutions appropriate means for the tool integration at the meta-model level is advantageous as reuse can be improved. Without the presented concepts for integration at the meta-model level and consistency management in our example either a special UML or SDL plug-in had to be developed anew to realise the required consistent integration.

Besides the presented concepts for meta-model integration and consistency management, the ASG framework of the FUJABA Tool Suite further supports a generic model (diagram) exchange via the *Graph eXchange Language* GXL [5] or XMI. Therefore, class diagrams can be exchanged, for example, with TogetherJ via a special XMI dialect. Other XML (text) formats can also be easily realised within a generic transformation mechanism [6].

Acknowledgements

We thank all students, PhD students, and all people that have helped to build the current FUJABA TOOL SUITE which is available at http://www.fujaba.de.

7. References

[1] UML 2.0 Superstructure submission V2.0. Unpublished ad/03-01-02, Jan. 2003. Alcatel, Computer Associates, Enea Business Software, Ericsson, Fujitsu, Hewlett-Packard, I-Logix, International Business Machines, IONA, Kabira Technologies, Motorola, Oracle, Rational Software, SOFTEAM, Telelogic, Unisys, and WebGain.

[2] S. Burmester. Generierung von Java Real-Time Code für zeitbehaftete UML Modelle. Master's thesis, University of Paderborn, Department of Computer Science, Paderborn, Germany, September.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[4] H. Giese and S. Burmester. Real-Time Statechart Semantics. *TechReport tr-ri-03-239, University of Paderborn*, 2003.

[5] R. Holt, A. Winter, A. Schürr, and S. Sim. GXL: Towards a Standard Exchange Format. In *Working Conference on Reverse Engineering*, Brisbane, Australia, Nov. 2000. IEEE Computer Society Press.

[6] P. Hoven and M. Liebrecht. Entwurf und Implementierung einer Import/Export Funktionalität für die Entwicklungsumgebung Fujaba, Aug. 2002.

[7] International Telecommunication Union (ITU), Geneva. *ITU-T Recommendation Z.100: Specification and Description Language (SDL)*, 1994 + Addendum 1996.

[8] J. Jahnke, W. Schäfer, J. Wadsack, and A. Zündorf. Supporting Iterations in Exploratory Database Reengineering Processes. 45(2-3):99–136, Nov. 2002. (Special Issue on Software Maintenance and Reengineering).

[9] D. Jin, J. Cordy, and T. Dean. Where's the schema? A taxonomy of patterns for software exchange. In *Proc. of the* 10^{th} *International Workshop on Program Comprehension (IWPC), Paris, France*, June 2002.

[10] M. M. Kamneng. Entwurfsunterstützung Web-basierter Schnittstellen auf Basis der UML. Master's thesis, University of Paderborn, Department of Computer Science, Paderborn, Germany, Apr. 2003.

[11] R. Kazman, S. Woods, and S. J. Carrière. Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. In *Proc. of the Working Conference on Reverse Engineering (WCRE'98), Honolulu, Hawaii*, pages 154–163, october 1998.

[12] H. Köhler, U. Nickel, J. Niere, and A. Zündorf. Integrating UML Diagrams for Production Control Systems. In *Proc. of the* 22^{nd} *International Conference on Software Engineering (ICSE), Limerick, Irland*, pages 241–251. ACM Press, 2000.

[13] M. Lefering. Software Document Integration Using Graph Grammar Specifications. In *Proceedings of the 6th International Conference on Computing and Information, Journal of Computing and Information 1, 1, 1994, pp. 1222–1243*, 1994.

[14] M. Nagl, editor. *The IPSEN Approach*. LNCS 1170. 1996.

[15] U. Nickel, J. Niere, W. Schäfer, and A. Zündorf. Combining Statecharts and Collaboration Diagrams for the Development of Production Control Systems. In *Proc. of Object-Oriented Modeling of Embedded Realtime Systems workshop (OMER)*. Technical Report 1999-01 University of the German Armed Forces Munich, 1999.

[16] J. Niere, W. Schäfer, J. Wadsack, L. Wendehals, and J. Welsh. Towards Pattern-Based Design Recovery. In *Proc. of the* 24^{th} *International Conference on Software Engineering (ICSE), Orlando, Florida, USA*, pages 338–348, May 2002.

[17] Object Management Group. *OMG Unified Modeling Language Specification, Version 1.4*, September 2001. OMG document ad/01-09-67.

[18] A. Schürr. Specification of graph translators with triple graph grammars. In *Proceedings of the* 20^{th} *International Workshop on Graph-Theoretic Concepts in Computer Science*, Herrsching, Germany, June 1994. Spinger Verlag.

[19] M. Tichy. Durchgängige Unterstützung für Entwurf, Implementierung und Betrieb von Komponenten in offenen Softwarearchitekturen mittels UML. Master's thesis, University of Paderborn, Department of Mathematics and Computer Science, Paderborn, Germany, July 2002.

[20] R. Wagner. Realisierung eines diagrammübergreifenden Konsistenzmanagement-Systems für UML-Spezifikationen. Master's thesis, University of Paderborn, Department of Mathematics and Computer Science, Paderborn, Germany, Nov. 2001.

[21] F. Wolf. Entwicklung eines Generators für eine objektorientierte Zugriffsschicht auf einer relationalen Datenbank. Master's thesis, University of Paderborn, Department of Computer Science, Paderborn, Germany, Apr. 2001.

[22] S. Woods, L. O'Brian, T. Lin, K. Gallagher, and A. Quilici. An architecture for interoperable program understanding tools. In *Proc. of the* 6^{th} *International Workshop on Program Comprehension (IWPC), Ischia, Italy*, pages 54–63, July 1998.