

Using Metamodels in Service Interoperability

Andreas Winter Jürgen Ebert

Institute for Software Technology

University of Koblenz

D-56070 Koblenz

Universitätsstraße 1

www.uni-koblenz.de/~winter/

[\(winter|ebert\)@uni-koblenz.de](mailto:(winter|ebert)@uni-koblenz.de)

Abstract

Interoperability in service oriented environments is heavily influenced by the view that the cooperating services have on their data.

Using the term service for the abstract contract concluded between a service requester and a service provider, three different data schemas have to be identified, namely the requester's schema, the provider's schema and the reference schema introduced by the service specification.

Metamodeling and schema transformation approaches from the area of model driven architecture can be used to define these schemas and their mappings as well as the appropriate transformations that have to be applied to the data.

This paper explains an approach towards metamodel-based service interoperability along an extended example of providing visualization services for the Bauhaus re-architecting tool.

1 Motivation and Related Work

Interoperability is the challenge of enabling software tools from different suppliers to share their functionality. Coupling different, but interoperable tools allows to form integrated collaborative tool suites without re-inventing or re-implementing already existing software components.

The issue of *tool integration* has already been addressed in the ECMA Reference Model [9], where data integration, control integration, presentation integration, process integration, and framework integration have been recognized as different and important aspects. Process integration and framework integration

focus on providing reasonable infrastructure for developing, integrating and using interoperable components and presentation integration emphasizes similarity of user interfaces for these components. The direct interaction between interoperable components is addressed by data and control integration.

Interacting components are usually based on especially *tailored data structures*, which allow the most efficient execution of provided services. Thus, coupling those components requires to synchronize the data between participating software systems [29, 26]. This includes the direct data exchange by standardized data interchange languages as well as common access to (virtually) joined data repositories by appropriate interfaces.

Service Oriented Architecture [8] is the current abstract approach to build software systems formed by components that provide certain functionality and components that use this functionality. These components are *loosely coupled* at runtime via network technologies. In service oriented architectures *services* subsume coherent functionality. They supply their functionality by published interfaces encapsulating data and control information.

In service oriented architectures, data synchronization between collaborating services is viewed as an automatic process at run time. Service interfaces act as *contracts* between service provider and requester. Existing systems usually have not been developed in a service oriented manner, but all more or less open systems offer import and export facilities according to their own internal data structures. For making those tools collaborate, they also have to agree on common service contracts. Each reasonable service has to be identified and the data required for providing the services has to be specified. To avoid coupling tools and components

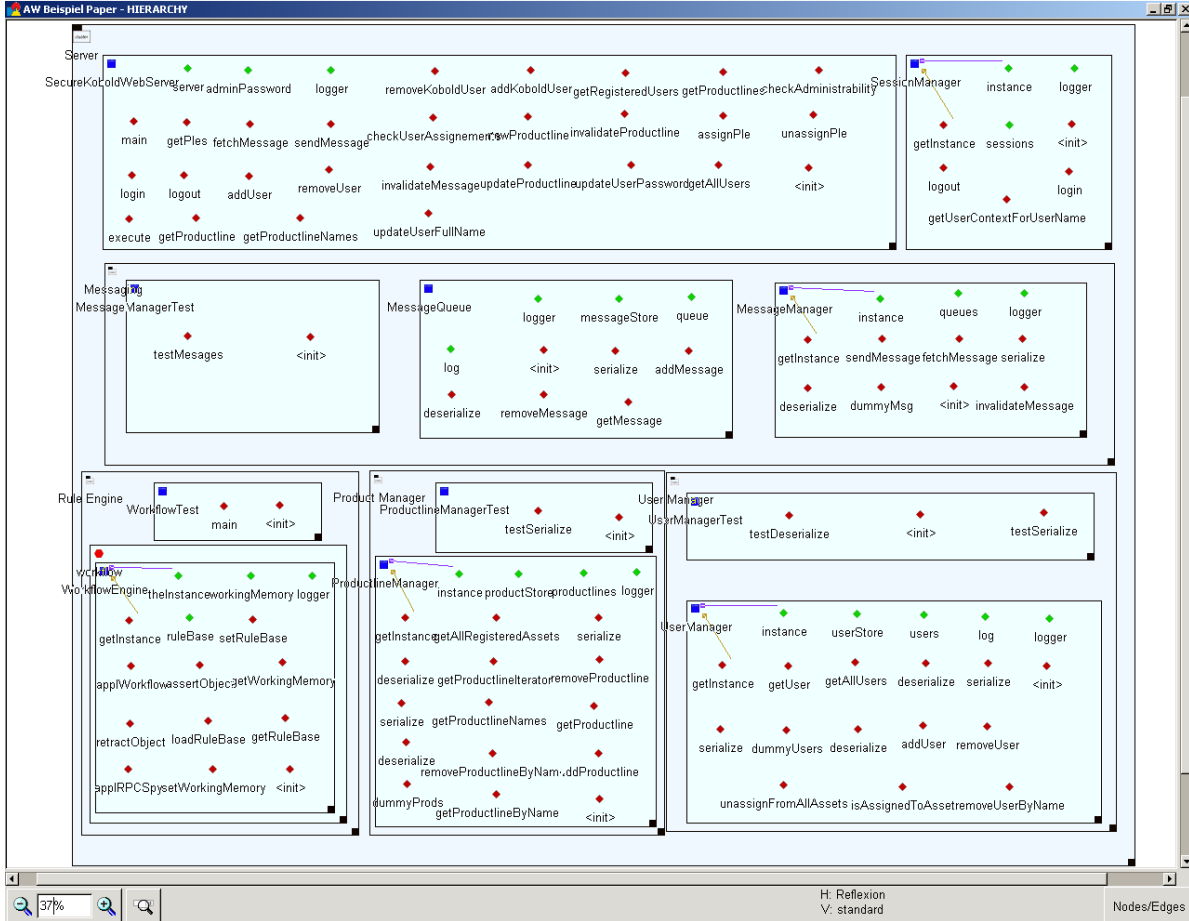


Figure 1: Decomposition View in Bauhaus

based on ad-hoc data structures and converters, a more general specification of the service and its interfaces is required.

This paper focuses on the use of *metamodels* as mediators to synchronize data between interoperable tools. Metamodels are used to define data shared between collaborating tools and found the base for transforming data between tool specific structures. Making these metamodels explicit and providing appropriate filters to convert data matching the metamodel to the individually used data structures of service providing tools, enables existing tools to be embedded in service oriented architectures. Furthermore, these metamodels serve as data structures for implementing new services.

A similar approach, based on grammars, is presented in [13]. Grammars specify document structures of interoperable tools. Pattern based association grammars act as an intermediary to synchronize these structures. That paper also addresses a standardized way to share individually structured data by intermediate structures. While their approach originates in corre-

lating annotated text based grammars, the approach presented here is oriented towards reference modeling in a model driven environment.

An ontology based approach towards tool interoperability is presented in [15]. This approach provides a general infrastructure to relate concepts of collaborating tools. The use of metamodels in service interoperability, presented here, completes this approach by showing how to realize data mappings.

Service Interoperability refers to sharing data and control between interacting services. Web services [3] form the most widespread infrastructure to enable components to collaborate dynamically. XML-based notations provide mechanisms to define standardized data exchange notations. The approach on using meta modeling technologies to enable service interoperability on data level is independent from the technological space used for service interaction. Thus, the approach is presented in terminology of web services and XML, but it should also hold for other middleware frameworks like CORBA [23].

Example: Visualizing Software-Architecture

In the following, service interoperability of software *re-engineering* tools is used as an example.

Starting from the source code of a legacy software system, the *Bauhaus re-architecting tool* [7, 10] extracts a (possible) software architecture. The decomposition view [5] of software architectures shows its structure in terms of components and subcomponents. The decomposition of an object oriented software system into packages, classes, attributes, and methods using the Bauhaus built-in visualization is presented in Figure 1.

The decomposition view of the server subsystem of the Kobold system for visualizing and maintaining product lines [16] contains two classes (*SecureKoboldWebServer*, *SessionManager*) and four subpackages (*Messaging*, *RuleEngine*, *ProductManager*, *UserManager*) which are subdivided into further packages and/or classes depicted as nested boxes. Each class is composed of members and methods, presented by differently colored diamonds.

The following sections of this paper will show, how the introduction of *metamodel-driven service interoperability* supports visualizing this architecture by means of UML class-diagrams using IBM Rational Software Modeler [25] (Software Modeler for short), a commercial of-the-shelf UML modeling tool, as a visualization service. □

Section 2 will introduce *services* and *components* and their major elements, which provide the base for service interoperability. The metamodel-based approach on service-interoperability is explained in section 3 using the example just introduced.

2 Services and Components

The shift from developing large monolithic systems towards service oriented architectures, including the potential of web service-based implementations, leads to new chances and challenges in software development. Instead of thinking of coupling *concrete software components*, interoperability is viewed as providing and using *abstract services*.

2.1 Services

Services form the interface to access functionality provided by certain implementations. They hide implementation details enabling service invocation regardless of specific hard- and software environments [19].

According to [28] *services* are viewed as abstract descriptions of coherent functions with public interfaces that are realized by concrete software *components*. They provide all information required to use the functions they describe.

The functionality specified by services usually contains a set of *operations*. Each operation is specified by its *signature* defining input and output data. Interacting with services also requires to agree on the data used by the service to perform its functionality. E.g. web service descriptions (cf. the abstract part of WSDL [2]) contain *types* describing data structures that are shared between interacting services and operations together with their signature to access the service's functionality.

From a more general perspective, *service requesters* have to supply the input data and receive the output data in specific forms specified by the service definition. Its data structure is specified by a *reference schema*, which is a common metamodel of the shared data. The service user has to provide these data and the service implementation has to ensure its use.

2.2 Components

Services are implemented by components which form *service providers* in web services. There may be several alternative components implementing the same service. Appropriate descriptions of components complement the web service description by implementation details like bindings, communication protocols, or address information. Service providers realize the service interface with all its specified operations and reference schemas.

Internal data structures of components, also specified by *schemas*, have to comply with the structure defined by the service's reference schema. Depending on the implementation of the provided operations, different service providers may use different internal implementations of specialized data structures optimized for efficient calculation.

Here, service reference schemas are viewed as an *ontology* which specifies a shared conceptualization between the service and its components [11]. It also serves as data part of the *contract* between the service requester, who has to deliver data according to the reference schemas, and the service provider, who implements the service according to this schema.

Example: Visualizing Software-Architecture

In the re-architecting example, the Bauhaus tool represents software architectures in resource flow graphs [7, 10], and Software Modeler uses a variant of the UML 2 superstructure metamodel [24].

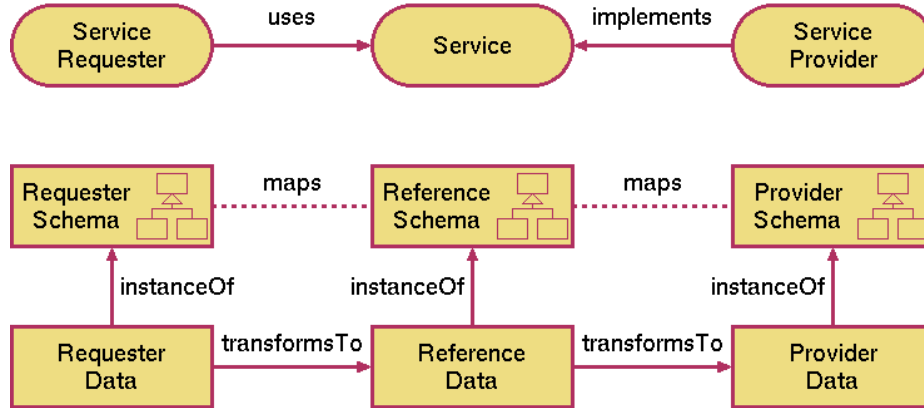


Figure 2: Metamodel-driven approach to Service Interoperability

Here, Bauhaus supplies the re-architecting service and the Software Modeler delivers (among others) the visualization service.

To make Bauhaus and Software Modeler interoperable, these structures have to be synchronized according to the requirements of the service definition.

□

3 Metamodel-based Service Interoperability

Interoperability requires a *contract* between collaborating partners on the operations supplied by the provider and on the data structures shared between the collaborators. This contract contains a *specification of the functionality* defined by the service and a *set of reference schemas* defining the data to be shared.

Service implementations usually employ internal data structures, that are optimized towards efficiency. Hence, collaborating components mostly do not use identical data structures and synchronization of data is required. The following sections show how data synchronization is realized by using the service reference schema as a shared conceptualization and schema transformation as filtering support.

3.1 Metamodel based approach to synchronize data

Interoperable agents have to agree upon exchanged data. *Common reference schemas* supply means to accommodate client data to the data required by service provider, and vice versa. Service interoperability necessitates powerful and adaptable mechanisms to *define*, *transform*, and *synchronize* such data.

Metamodeling and *metamodel based transformation* provide such mechanisms. Metamodels define the structure of data to be exchanged. The reference schema, defining the services data, and the schemas used by the service requester and the service provider offer the formal base to assign metamodel based transformations. Figure 2 sketches this approach.

Data exchange in service oriented environments follows a stepwise approach:

1. **Define schemas**

- (a) Define reference schemas

The reference schemas of the service serve as domain ontologies reflecting the shared concepts among the collaborating components. They are mediators between interoperable partners.

- (b) Define service requester and service provider schemas

Collaborating partners work on their individual own data structures. These have to be made explicit [14] to form the base of data export and import.

2. **Specify data transformations**

Data according to the participating schemas have to be mapped to each other. Mappings from the service requester's data via the reference schema to the service provider's data have to be specified.

These mappings express a common understanding of the semantics of the participating schemas. Concepts and structures of different schemas defining the same content have to be related. Here, different terminology and modeling style has to be normalized according to the services reference schema.

3. Share data

Data provided by the service requester has to be available to the service provider and data calculated by the service provider has to be available to the requester.

Sharing this data can be done either by encapsulating the synchronization within functional interfaces or by physically exchanging the data by a standard exchange form, which conforms to the reference schemas. Since different services use different reference schemas, data has to be exchanged together with its corresponding schema information.

The defined schemas establish a base for both, *data transformation* and *data exchange*. Section 3.2 shows the schemas involved in realizing a software architecture visualization service, with Bauhaus and Software Modeler as collaborating components.

3.2 Example: Visualizing Software-Architectures

Using a COTS-UML tool for visualizing software-architectures recovered by Bauhaus requires to make Bauhaus data accessible to the UML tool. The required synchronization is done here according to a specific architectural view (e.g. [5]). Architectural views serve as reference schemas for services visualizing certain aspects of software architecture [30]. A reference schema for representing a *decomposition view* and the appropriate variants in the collaborating tools are sketched in Section 3.2.1. Schema based transformation and data exchange is described in Section 3.2.2 and 3.2.3.

3.2.1 Define Schemas

A service for visualizing the decomposition view of object oriented software architectures requires to agree on the data to be displayed.

Figure 3 defines a possible decomposition of object oriented software systems. *Packages* contain further *packages* and/or *classes*. *Classes* contain *methods* and *attributes*.

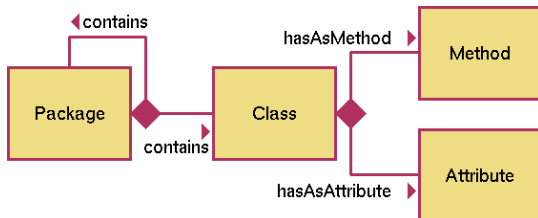


Figure 3: Reference Schema

Together with the signatures of the provided operations the reference schema in Figure 3 defines a contract each service provider for visualizing software architectures according to the decomposition view has to fulfill. A requester calling that service, can be certain, that the data is rendered correctly by a service conformant provider.

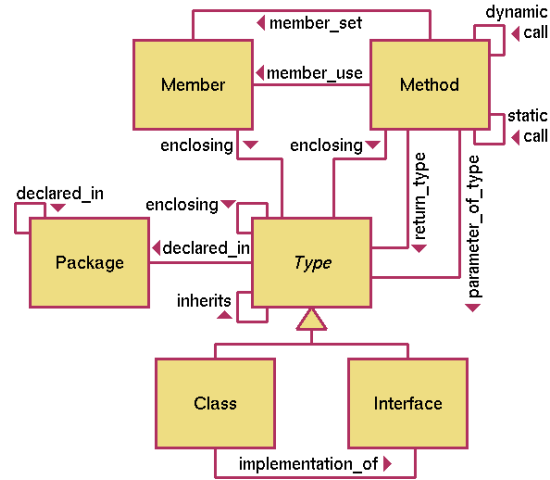


Figure 4: Bauhaus Schema

Using services requires to map the data used by the service requester to the reference schema. The data structure used by the Bauhaus re-architecting tool is shown in Figure 4 [7, 10]. *Packages* declare *classes* and *interfaces*. Both consist of *members* and *methods*. Bauhaus also represents further relationships between components of software architectures which are not considered by our decomposition view visualization service.

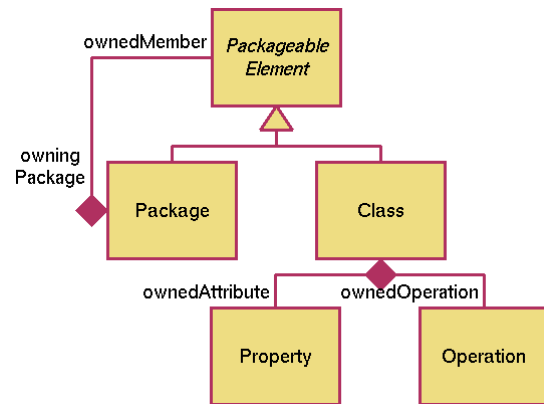


Figure 5: UML Metaschema (extract)

Modern UML tools usually found on the UML 2 superstructure metamodel [24]. Figure 5 shows an extract of the UML metamodel representing the relevant

| Bauhaus Schema | Reference Schema | UML Metaschema |
|--|---|--|
| $p, q : \text{Package}$ $c : \text{Class}$ $a : \text{Member}$ $m : \text{Method}$ | $p, q : \text{Package}$ $c : \text{Class}$ $a : \text{Attribute}$ $m : \text{Method}$ | $p, q : \text{Package}$ $c : \text{Class}$ $a : \text{Property}$ $m : \text{Operation}$ |
| $P \leftarrow \text{declared_in } Q$ $P \leftarrow \text{declared_in } C$ $C \xrightarrow{*} \text{inherits} \leftarrow \text{enclosing } a$ $C \xrightarrow{*} \text{inherits} \leftarrow \text{enclosing } m$ | $P \rightarrow \text{contains } Q$ $P \rightarrow \text{contains } C$ $C \rightarrow \text{hasAsAttribute } a$ $C \rightarrow \text{hasAsMethod } m$ | $p.\text{ownedMember} \ni q$ $p.\text{ownedMember} \ni c$ $p.\text{ownedAttribute} \ni a$ $p.\text{ownedOperation} \ni m$ |

Figure 6: Mapping between corresponding concepts and relations

concepts for visualizing the decomposition view. Here *classes* contain *properties* and *operations*. *Packages* and *classes* are subsumed to *Packageable Elements* which can be substructured into further *packages* and *classes*. Software Modeler provides import facilities for data according to this schema, based on XMI [20].

Apparently, all three schemas are capable to describe the decomposition of a software system into packages, classes, attributes, and methods. But, they use different terminology and modeling styles. The internal schemas of the collaborators are allowed to represent further data. So, e.g. the *Bauhaus schema* also supports inheritance and call relationships and the (complete) *UML schema* supports all other UML notations.

The *reference schema* serves as a mediator normalizing these structures. It constitutes a consistent terminology and defines a common conceptualization required for sharing re-architecting and visualization facilities between the collaborators.

3.2.2 Specify Data Transformations

To enable sharing data between collaborating tools, the data exported by the service requester has to be adapted to the specific import format required by the selected service provider and vice versa. The common conceptualization of both tools is specified in the service reference schema. Thus, in both directions two *transformations* are required: one transforms the requester’s data to data according to the reference schema, the second transformation translates data according to the reference schema into the provider’s notation. If data has to be returned to the requester, proper inverse transformations are needed, which might be realized in a similar way.

Model transformations are a core constituent of model-driven architectures [22]. Model transformation is viewed as the process of converting models into other

models. Those transformations are specified in terms of *mappings* between the corresponding metamodels. Various metamodel based approaches to model transformation are currently under development to enable specification and execution of metamodel based transformations [21].

The required mappings between the Bauhaus schema (cf. Figure 4), the reference schema (cf. Figure 3), and the UML metaschema (cf. Figure 5) are sketched in Figure 6. The mappings define the correspondence between the participating concepts and their relationships.

The Bauhaus schema, the reference schema, and the UML metaschema follow different modeling styles. Associations are oriented and named in the Bauhaus schema and in the reference schema, whereas the UML metaschema uses a more implementation oriented approach, utilizing role defined associations. The mapping in Figure 6 considers these different modeling styles by notating the constraints for the Bauhaus and the reference schema using regular path expressions and stating the constraints for the UML metaschema in OCL predicates.

Interoperation scenarios with very diverse metaschemas might require much more complicated constraints specifying the mapping.

A general and widely applicable reference schema for visualizing decomposition structures should also support generalization of classes. Generalization is ignored in Figure 3 to present an example for a precision loosening model transformation. Generalization of classes in an architecture recovered by Bauhaus has thus to be flattened into a structure according to the reference schema. To preserve as much semantics as possible during transformation, attributes and methods of superclasses are directly associated to *all* subclasses. Thus, the transformed architecture, visualized according to the UML schema does not contain the generalization information, though the UML 2 superstructure meta-

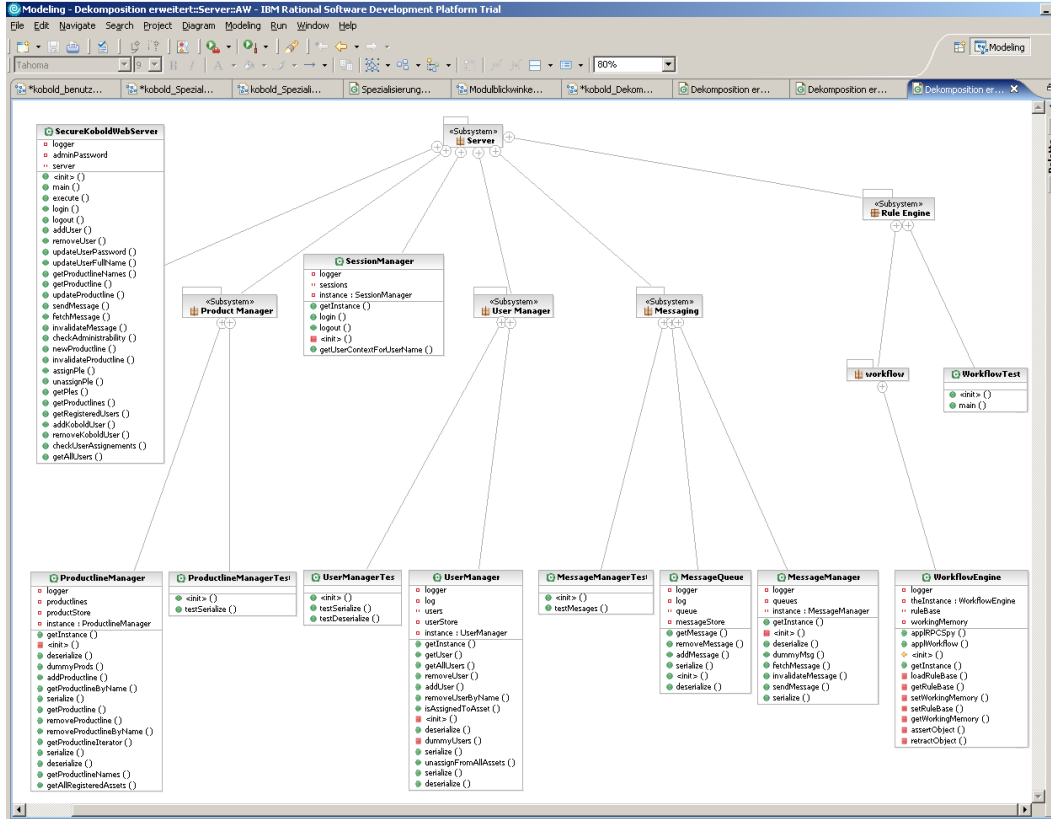


Figure 7: Decomposition View visualized with Software Modeler

model [24] also supports generalization and Software Modeler is capable of visualizing these structures.

This demonstrates the importance of defining a preferably universal schema for services. But, since nobody can anticipate all features or modeling styles used in existing or future services implementations, it is impossible to define these general schemas. Thus, coupling services has always to be aware of loosing precision. Loss of precision is discussed e.g. in [6]. But, defining and comparing reference schemas and tool specific schemas allows to make the loss of information within the collaboration of concrete tools explicit.

The transformations described here were implemented in a case study [30] by XSLT scripts, since Bauhaus and Software Modeler provide XML based import and export facilities. XSLT works rather inefficiently on huge amounts of data, which occur in interoperable reengineering environments. Furthermore, metamodels and their mappings do not influence the XSLT transformation explicitly. Only the source model appears in patterns, the target model is only implicit.

Thus, more efficient and powerful declarative metamodel-driven approaches are needed for model transformation. More promising approaches for

metamodel-based transformation might be BOTL [4], MDI [17], or ATL [1].

3.2.3 Share Data

GXL (Graph eXchange Language) [12] is an XML-based standard exchange format for sharing graph data between tools. GXL comes with a metamodeling approach, such that graph-based data can be exchanged together with its corresponding schema information. In contrast to other XML-based metamodeling approaches, GXL is homogeneously graph-based on all meta-levels, leading to one single and simple notation for instance data, schemas and metaschemas.

In the example, the Bauhaus schema as well as the reference schema were defined by GXL schemas, i.e. they could be exchanged as graphs corresponding to the GXL metaschema. Thus, the mapping between Bauhaus data and the reference schema was directly done by exchanging and transferring GXL documents.

Since Software Modeler only supports XMI, a separate filter was written. Data according to the reference schema were transformed into the XMI imported by the Software Modeler.

Figure 7 shows the resulting UML visualization of the decomposition view from Figure 1 from the running example. The UML class diagram shows the server subsystem of Kobold with its two classes (*SecureKoboldWebServer*, *SessionManager*) and four substructured subpackages (*Messaging*, *RuleEngine*, *ProductManager*, *UserManager*). Member variables and methods of classes are depicted in the usual class compartments.

4 Conclusion

This paper introduced the use of *metamodeling technology for service interoperability*. Interoperability between service provider and service requester is viewed as fulfilling a contract defined by the service. Instead of coupling concrete software components individually, *service interoperability* views tool integration on an abstract level. Services specify the offered *functionality* and a set of *reference schemas*. Collaborating partners have to agree on this service specification.

Within this framework, sharing data among collaborators plays an important role. Due to different internal data structures, transformations between interacting tools are required. Data exchange and the transformations are realized according to the explicit schemas used by the tools and to the reference schema of the services which define a common conceptualization.

Coupling (already existing) tools by providing and using certain functionality in a service oriented setting, requires to define the services schema by a *reference schema*. To provide a wide spectrum of possible implementations the reference schema definition should be done according to the general needs of the intended service and independently of probably existing implementations.

Realizing this service by new components can be done directly on this structure. Using already existing tools or components requires to discover the individual schema of the service provider. These schemas usually differ from the reference schema, and transformers have to be provided for data exchange. Using this service, requires to provide data according to the reference schema. Again, transformers are needed for mapping the service requesters data to the reference schema. These transformers are based on schema transformation between the requesters individual schema and the reference schema.

Standardizing the services interface by a reference schema, defining the individual data structures of requester and provider, and defining schema based transformations facilitates the coupling of already existing software systems in a service oriented environment as

well as the realization and use of new service implementations.

The presented approach is independent of concrete realization strategies and technologies. It can be applied to requests to web services as well as to using encapsulated functionality of monolithic components offering export and import facilities. Similarly, the applied meta modeling approach is not restricted to GXL only. But, it becomes important to adopt powerful *metamodel based transformation techniques*.

Visualizing a recovered software architecture and presenting the results in an UML tool was done with monolithic tools offering export and import facilities and some additional scripts. In the *case study* coupling of the functionality of two commercial, monolithic off-the-shelf tools was not yet implemented in a strong service oriented architecture. But, both tools offer export and import facilities. Bauhaus exports software architectures as GXL graphs and Software Modeler imports XMI streams. The GXL graph generated by Bauhaus was transferred into an GXL graph matching the reference schema. This graph was transferred by another transformer into an XMI. The transformers were realized by XSLT and were based on various reference schemas of the software architecture visualization services and the published metamodels of Bauhaus and Software Modeler.

The current implementation provides reference schemas and transformers for the decomposition view, the specialization view, uses view, implementation view, and module view. Based on the specification of visualization services the Bauhaus re-architecting tool was successfully applied to use Software Modeler to render software architectures. Further activities deal with applying model driven transformation approaches for more elegant and (hopefully) more performant transformation.

Currently approaches on migrating systems towards service oriented architectures (e.g. [18]) are being developed. Enabling data interoperability between communicating components is a prerequisite to service interoperability. The discovery of services in legacy systems, their definition with reference schemas and supplying appropriate transformers will also provide a generic means to service oriented interoperability of legacy systems.

Acknowledgments: This proposal has benefited from various discussions with our students. We thank Kerstin Falkowski, Alexander Kaczmarek, and Julia Wolff for their collaboration, insights and suggestions.

References

- [1] ATL: Atlas Transformation Language, User Manual. http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/ATL/doc/ATL_User_Manual%5Bv00.09%5D.pdf, August 2005.
- [2] Web Services Description Language (WSDL) Version 2.0 Part 0: Primer. W3C Working Draft, W3C, <http://www.w3.org/TR/wsd120-primer/wsd120-primer.pdf>, 3. August 2005.
- [3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services, Concepts, Architectures and Applications*. Springer, Berlin, 2004.
- [4] P. Braun and F. Marschall. Transforming Object Oriented Models with BOTL. *Electronic Notes in Theoretical Computer Science*, 72(3):1–15, 2003.
- [5] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures, Views and Beyond*. Addison Wesley, Boston, 2003.
- [6] J. R. Cordy and D. Jin. Factbase Filtering Issues in an Ontology-Based Reverse Engineering Tool Integration System. *J.-M. Favre, M. Godfrey, A. Winter (eds.): Integrating Reverse Engineering and Model Driven Engineering, 2nd International Workshop on Meta-Models and Schemas for Reverse Engineering. Electronic Notes in Theoretical Computer Science*, 137(3):65–75, September 2005.
- [7] J. Czeranski, T. Eisenbarth, H. Kienle, R. Koschke, and D. Simon. Analyzing xfig Using the Bauhaus Tool. In *7th Working Conference on Reverse Engineering*. IEEE Computer Society, Los Alamitos, pages 197–199. 2000.
- [8] W. Dostal, M. Jeckle, I. Melzer, and B. Zengler. *Service-orientierte Architekturen mit Web Services, Konzepte, Standards, Praxis*. Elsevier, München, 2005.
- [9] ECMA. Reference Model for Frameworks of Software Engineering Environments, Version 2.6. Technical Report ECMA TR/55, European Computer Manufacturers Association, National Institute of Standards and Technology, US Department of Commerce, 1993.
- [10] Thomas Eisenbarth, R. Koschke, and S. Bellon. Bauhaus Software Technologies, Language Guide 5.1.4. Documentation, 28. April 2005.
- [11] W. Hesse. Ontologie(n). *Informatik Spektrum*, 16(6):477–480, Dezember 2002.
- [12] R. C. Holt, A. Schürr, S. Elliott Sim, and A. Winter. GXL: A Graph-Based Standard Exchange Format for Reengineering. *Science of Computer Programming*, 60(2):149–170, April 2006.
- [13] I. Ivkovic and K. Kontogiannis. Interoperability and Integration of Enterprise Applications through Grammar-Based Model Synchronization. In *Y. Zou, M. di Penta: Pre-Proceedings IEEE International Workshop on Software Technology and Engineering Practice (STEP 2005)*, pages 244–247. 2005.
- [14] D. Jin, J. R. Cordy, and T. R. Dean. Where’s the Schema? A Taxonomy of Patterns for Software Exchange. In *10th International Workshop on Program comprehension*. IEEE, Los Alamitos, pages 65–74. 2002.
- [15] D. Jin and J. R. Cordy. A Service Sharing Approach to Integrating Program Comprehension Tools. Proceedings of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003), Workshop on Tool Integration in System Development (TIS 2003), 73–78, September 2003.
- [16] Werkbold Team, University of Stuttgart: Kobold Productline-Manager. <http://kobold.berlios.de/index.html>.
- [17] A. Koenigs and A. Schuerr. Multi-Domain Integration with MOF and extended Triple Graph Grammars, <http://drops.dagstuhl.de/opus/volltexte/2005/22/pdf/04101.KoenigsAlexander1.Paper.pdf>. In J. Bezhin and R. Heckel (eds.) *Language Engineering for Model-Driven Software Development*, Dagstuhl Seminar Proceedings 04101. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany, 2005.
- [18] K. Kontogiannis and Y. Zou. Reengineering Legacy Systems Towards Web Environments. In *K. M. Khan, Y. Zheng (eds.): Managing Corporate Information Systems Evolution and Maintenance, Idea Group Publishing, Hershey, PA, USA*, pages 138–146. 2004.
- [19] H. Kreger. Web Services, Conceptual Architecture (WSCA 1.0). <http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>, IBM Software Group, May 2001.

- [20] OMG XML Metadata Interchange (XMI) Specification. <http://www.w3.org/TR/2000/REC-xml-20001006.pdf>, January 2002.
- [21] Request for Proposal: MOF 2.0 Query/Views/Transformations RFP. <http://www.omg.org/docs/ad/02-04-10.pdf>, October 2002.
- [22] MDA Guide. <http://www.omg.org/docs/omg/03-06-01.pdf>, June 2003.
- [23] Common Object Request Broker Architecture: Core Specification. <http://www.omg.org/docs/formal/04-03-01.pdf>, March 2004.
- [24] UML 2.0 Superstructure Specification. <http://www.omg.org/docs/ptc/04-10-02.pdf>, October 2004.
- [25] IBM Rational Software Modeler. <http://www-306.ibm.com/software/awdtools/modeler/swmodeler/>.
- [26] S. E. Sim. Next Generation Data Interchange, Tool-to-Tool Application Program Interfaces. In *7th Working Conference on Reverse Engineering*. IEEE Computer Society, Los Alamitos, pages 278–280. 2000.
- [27] Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation, W3C XML Working Group, <http://www.w3.org/TR/2004/REC-xml-20040204>, February 2004.
- [28] Web Services Architecture. W3C Working Group Note, W3C Web Services Architecture Working Group,, <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/wsa.pdf>, 11 February 2004.
- [29] P. Wegner. Interoperability. *ACM Computing Survey*, 28(1):285–287, 1996.
- [30] J. Wolff and A. Winter. Blickwinkelgesteuerte Transformation von Bauhaus-Graphen nach UML. *Softwaretechnik-Trends*, 25(2):33–34, May 2005.