Metamodeling Foundation for Software and Data Integration

Henning Agt, Gregor Bauhoff, Mario Cartsburg, Daniel Kumpe, Ralf Kutsche, and Nikola Milanovic

Technische Universität Berlin
{hagt,gbauhoff,mcartsbg,dkumpe,rkutsche,nmilanov}@cs.tu-berlin.de

Abstract. We propose a model-based methodology for integration of heterogeneous distributed systems, based on the multi-level modeling abstractions, automated conflict analysis and connector code generation. The focus in this paper is on the metamodeling foundation necessary for this process, and consequently we introduce computation independent, platform specific, platform independent and semantic metamodels, which generate a set of domain specific languages used to describe software and data integration scenarios.

1 Introduction

Integration of heterogeneous distributed IT-systems is one of the major problems and cost-driving factors in the software industry today [18]. The problem is not new, and several solution possibilities exist. Schema matching approaches [19] try to detect dependencies and conflicts between data model elements at the model or instance levels. Extract-Transformation-Load (ETL) tools use schema matching methodology to enable easier integration of multiple data sources. Many languages exist that enable specification of transformations between two data models, such as Ensemble [9]. The usability of all mentioned approaches suffers greatly with increased heterogeneity of the underlying data sources.

The second major group of integration approaches is focused on the Service Oriented Architecture (SOA). In SOA, all system interfaces are wrapped as service endpoints and accessible to the Enterprise Service Bus (ESB) engine, which orchestrates data and functional logic. However, it is expected that all service endpoints are compatible and no data or behavior conflicts will occur. If they do, either the endpoint itself has to be modified (frequently impossible), or the XSLT or Java code snippet (BPELJ) has to be written, correcting the conflict at the message level. There are approaches which fill the niche between two main groups such as mapping editors (e.g., Altova MapForce) and extended UML Editors (e.g., E2E Bridge), with very low or no support for the compositional conflict analysis and the code (endpoint) generation.

For these reasons, as a part of the R&D program of the German government for regional business initiatives, the project BIZYCLE (www.bizycle.de) was started in early 2007 in order to investigate in large-scale the potential of model-based software and data integration methodologies, tool support and

practical applicability for different industrial domains[1]. The BIZYCLE integration process [12],[13],[17] is based on multi-level modeling abstractions. The integration scenario is first modeled at the computation independent (CIM) level, where business aspects of an integration scenario are described. The model is then refined at the platform specific (PSM) level, where technical interfaces of the systems that should be integrated are described. The PSM interface descriptions are then abstracted to platform independent (PIM) level, where a conflict analysis process takes place. Based on the result of the conflict analysis, connector model and code are generated and deployed.

As the metamodels behind the integration process are our focus, this paper is structured as follows: first the CIM metamodel is described, followed by one selected PSM metamodel and PIM metamodel. After that, we propose a way to semantically annotate model elements using semantical metamodel. Finally, we show how, based on the proposed metamodels, conflict analysis and connector code generation may be performed.

2 Computation Independent Metamodel

The BIZYCLE integration process captures early stages of an integration scenario at the computation independent model (CIM) level. It describes scenario requirements with an abstract business process and data flow model, regardless of the technical details of the underlying systems. Furthermore, it forms a basis for the conflict analysis which identifies interface interoperability mismatches.

The integration scenario is described in terms of activities and transitions, as well as data exchange aspects and their hierarchical structure (see Figure 1). `BusinessComponent` represents a software or data artifact that is involved in the scenario. Business components do not necessarily represent exactly one physical system, e.g., functional access can be provided as a Web service and data access with an SQL interface. A special business component is the `BusinessConnector`, which handles interconnections between the components. The abstract metaclasses `ActivityNode` and `ActivityEdge` are used to express data and control flow of the integration scenario.

Business components perform different activities. `ControlNodes` are used to determine the beginning and the end of an integration scenario. `ExportInterface` or `ImportInterface` are used to describe the terms of data exchange or function calls. Activities of the type `InternalComponentAction` express actions that are not relevant to the concrete data flow, but are useful for understanding the context of the scenario. The metamodel also defines `BusinessFunction` that represents functionality of the integrated artifacts. It processes incoming data or performs a certain task required by other artifacts. Business functions can only be used inside of business components. `ConnectorFunction` is used to predefine

[1] This work is partially supported by the Federal Ministry of Education and Research (BMBF) under grant number 03WKBB1B.

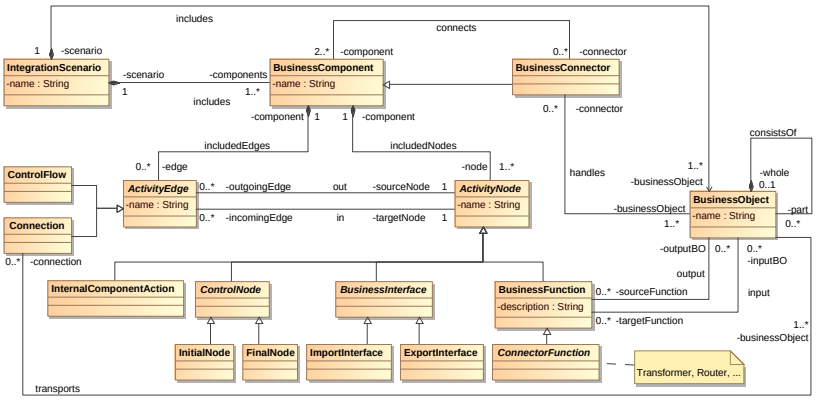

Fig. 1. CIMM – Basic metaclasses for artifact, process and data modeling (excerpt)

functionality of the connector. It can represent control (e.g., `Timer`) or conflict resolution functions (e. g. `Transformer, Router or Aggregator`), as in [8].

Two different kinds of edges can connect activities of the integration process. The `ControlFlow` is used to express the transition from one activity to another inside of a business component. The `Connection` models data and control flow between different business components or connectors and transports `BusinessObject` elements, which can be hierarchically structured independent of any data type of the underlying interfaces. Business objects are produced or consumed by `BusinessFunctions`. Constraints in the metamodel assure the correct combination of activity edges and nodes.

Four CIM views are defined that graphically show relevant aspects of the model: flow, object, connection and function view. The CIM flow view shows the sequence of the integration activities and transitions, similarly to the UML activity diagram. The object view consists of the business object and their structure only. The connection view hides the internal behavior of the components and displays business objects related to their connections. Finally, the function view shows business functions and their input and output business objects.

3 Platform Specific Metamodels

Platform specific metamodels (PSMM) describe the structure, behavior, communication and non-functional properties of the platform-specific system interfaces. The communication part describes information required to establish a connection with the system. The structure part provides information about platform-specific type system. The property part describes non-functional properties, such as performance, authorization, security or logging. It is also possible to annotate model elements using concepts from an existing ontology (see Section 5).

Ecore serves as a metamodel for the PSMMs [2]. Consequently the PSMMs are at the M2-level of the MOF hierarchy. The elements of the PSMMs are lin-

guistic instances (see [11]) of the Ecore metamodel. The PSMMs define a domain specific language (DSL) for the description of software system interfaces at the M1 level. To be more precise, the elements of PSMMs build the vocabulary for the interface descriptions. The system under study is provided on the M0-level. Every metamodel-layer describes only concepts for one layer beneath, therefore one has a linear metamodel hierarchy over four semantic abstraction levels, conforming to the MOF-hierarchy. An overview about the different metamodel layers is given in [6] and different metamodeling roles are defined in [11].

We currently provide metamodel support for the following platforms: ERP systems (SAP R/3 and AvERP), relational and XML databases, Web Services, XML files, J2EE components and .NET applications. As an illustration, in the following subsection we describe the SAP R/3 metamodel in more detail. All other PSMMs have a similar package composition to provide adequate DSLs for the remaining platforms.

3.1 SAP R/3 Platform Specific Metamodel

SAP R/3 is one of the most frequently used enterprise resource planning (ERP) systems. The platform specific metamodel of SAP R/3 consists mainly of classes referring to the remote accessible methods, so-called BAPIs (Business Application Programming Interface) and IDOCs (Intermediate Document). BAPIs are remote accessible functions and IDOCs are structured files. The schema of an IDOC file is defined within an IDOC Type. The IDOC type consists of a hierarchic tree with segments and fields inside the segments. The entry point to the hierarchic tree-based model is the element SAP R3, which represents a concrete SAP R3 system installation. The SAP R3 consist of at least one SAP R3 Interface. This element consist of the elements Access, SAP Business Component and IDOC Type. The whole parameter and structure part can be built automatically through extraction of relevant information from the SAP Business Object Repository (BOR). The core metamodel (excerpt) is shown in Figure 2.

The elements Communication Channel and Access belong to the package communication. This package collects information about the physical access, including user name, password, host name, language, system number, sap client and the communication channel. A communication channel can be synchronous, asynchronous, transactional or queued RFC (remote function control), simple file transfer, or e-mail. The concrete communication channel depends on the type of information exchange. IDOCs will be exchanged over file transfer, e-mail or transactional RFC, and BAPIs will be accessed using synchronous RFC.

Parameters can have the structured, table and field types. Structured and table types can consist of field types, which are atomic. Field types are JCO (Java Connector) types, more precisely they are wrapper classes. JCO offers a library for Java which supports the access to a SAP R3 system. JCO converts the SAP R3 types to Java types and backwards.

The abstract element Method is linked to the package Behavior and the element SAP R3 is linked to the package Property. For more information about behavior and properties, see section 4 about platform independent metamodel.

Parameters and segments have references to the domain object elements of the semantic metamodel (see Section 5) and can therefore be linked with domain object within an ontology. Likewise, all methods can be linked to domain functions.

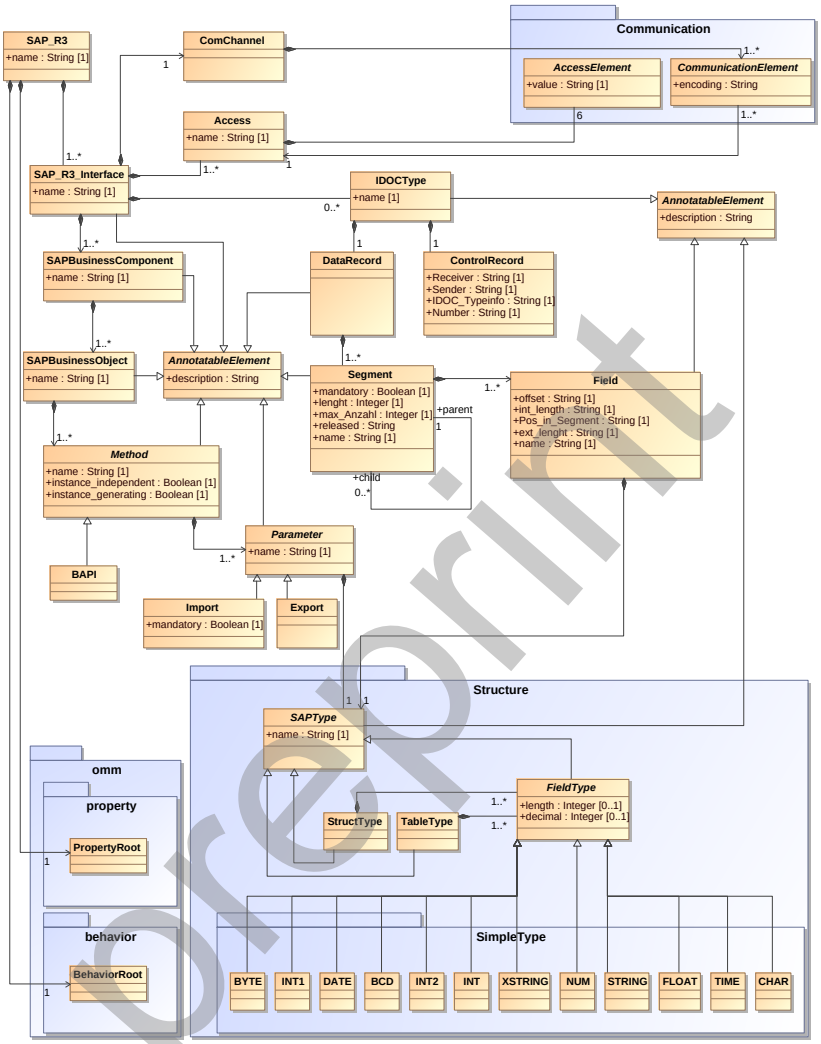

Fig. 2. Excerpt from Core SAP R/3 metamodel

4 Platform Independent Metamodel

The purpose of the platform independent metamodel (PIMM) is to facilitate system interoperability by abstracting all platform specific heterogeneous interface details. The abstraction process is realized by a PSM-to-PIM transformation using ATL [1]. For every PSMM there exists a set of transformation rules which translates the PSM into the common abstraction layer, the PIM. As stated in [14] the PIMM facilitates integration of heterogeneous interfaces. At the PIM level it is possible to represent different interface details on a common basis.

At the PIM level an `Interface` represents a single system gateway which is able to handle data as input and/or output in one single step. Hence PIMM-Interfaces represent an abstraction for all platform specific operations, methods, functions, files etc. PIMM distinguishes between three interface types: `FunctionInterface`, `MethodInterface` and `DocumentInterface`. The first represents a non-object-oriented interface, the second an object-oriented interface and the last a data structure-based interface. All interfaces have different corresponding metaclasses: DocumentInterfaces have only one root `Type` and no parameters, FunctionInterfaces can have different `FunctionParameters`, each with a corresponding type and MethodInterfaces can have object-based parameters with object-identity represented by `IEObject`. Due to this fact, our design decision was not to generalize everything into one single interface type. All interface types are still platform independent from a technical point of view.

Every interface has its own parent container, the system which exposes it, modeled by `IntegratableElement`. Each interface contains associations to different parts of the PIMM which will described in the following subsections (see basic metaclasses in Figure 3).

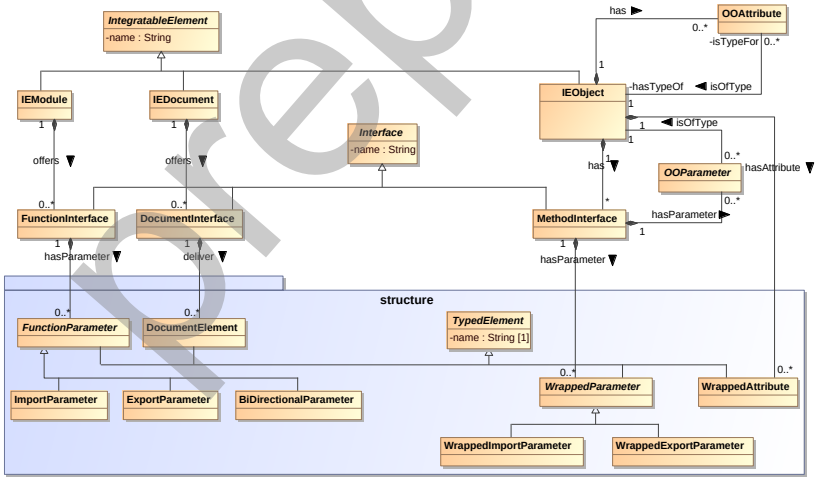

Fig. 3. PIMM - basic metaclasses for interfaces (excerpt)

4.1 Structure and Communication

The structure part (Figure 4) includes the common type system with the abstract super metaclass `Type` and different sub-metaclasses to express `SimpleTypes` (String, Number, Boolean) as well as `ComplexTypes`. The abstract `TypedElement` represents an element with a specific meaning (semantics) and represents a value container at runtime, e.g. `ImportParameter`. A TypedElement has exactly one association to a type but vice versa a type can belong to arbitrarily TypedElements, which means a Type can be 'reused'. For each interface type, different sub-metaclasses of TypedElement are provided: `WrappedParameter`, `FunctionParameter` and `DocumentElement`.

An interface may have its own interaction rules. These can be described by communication patterns such as `MessageExchangePatterns`, or different `Call`- and `Event`-types. Knowing platform independent communication details is essential to be able to interact with every interface in the proper way. During communication conflict analysis, incompatible call mechanisms are detected and fixed (as far as possible automatically).

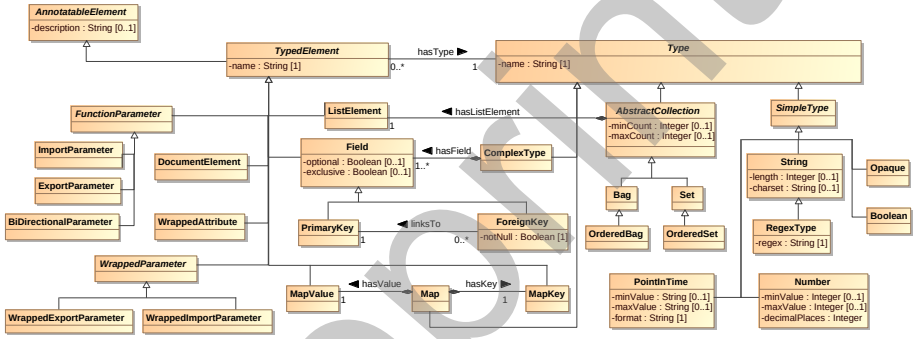

Fig. 4. PIMM - structure part

4.2 Property and Behavior

Non-functional properties are used to characterize interface capability (provided) and expectations (required) properties. The root node `PropertyRoot` is responsible for including every `PropertyContainer` which is derived to `RequirementPC` and `CapabilityPC` element. Every `PropertyElement` is a child of exactly one of these two sub-metaclasses. The metaclass `QualityOfService` is the upper metaclass of every QoS property and has an association to a `Metric`. All metrics have the `AbstractMetric` as their upper metaclass. Based on this metamodel construction every QoS can be combined with every metric. During property conflict analysis, CapabilityPCs and RequirementPCs are related to each other in order to extract incompatible process flow graphs.

Behavior of a system is described using OCL and process algebra metaclasses. The OCL can be used to define parameter and function constraints:conditions between parameter values, pre-conditions, post-conditions and invariants using abstract states. Process algebra interface call order describes the allowed call behavior from a client side view. The connector (mediator) acts as a client and calls different interfaces (e.g., methods) sequentially or concurrently.

5 Semantic and Annotation Metamodel

One of the challenges in software and data integration projects is the (semi)-automatic detection of mismatches in interface semantics. A prerequisite for the semantic analysis is the semantic description of the integration scenario at CIM and PSM/PIM abstraction levels. Our current solution follows the approach of a shared domain ontology [20] to subsume the semantic descriptions that can be used in several integration projects.

The semantic metamodel SMM (Figure 5) allows to create a graph of triples in terms of subject, predicate, and object relations, similar to RDF. Subjects and objects are modeled with the `SemanticConcept` metaclass that can be either a `DomainObject` (knowledge representation of data) or a `DomainFunction` (knowledge representation of functionality). Semantic concepts are linked with `Predicates`. The metamodel includes predefined predicates which we identified as relevant for knowledge relations in integration scenarios: generalization (`IsA`), data processing (`Input`, `Output`), containment (`Has`) and sets (`ListOf`). The `CustomPredicate` allows the creation of user specific relations.

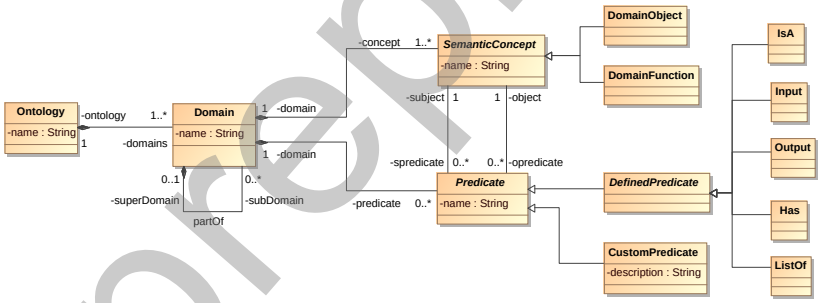

Fig. 5. Semantic metamodel

The association of heterogeneous artifacts such as documents, service interfaces, business processes, web resources and models with semantic concepts is called semantic annotation [10][4]. The goal of the semantic annotation is a distinct semantic characterization of abstract data definitions, underlying interfaces and interface element structures to enable a (semi)-automatic conflict analysis and resolution. We distinguish between data-oriented annotations and function-oriented annotations.

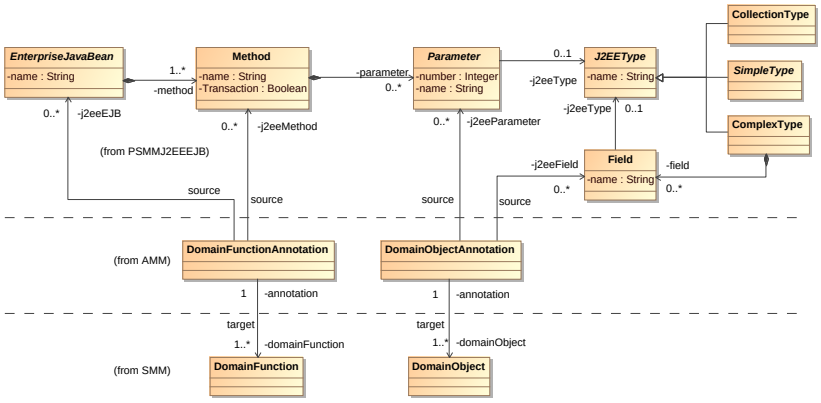

Fig. 6. Excerpt of the annotation metamodel - J2EE annotations

The annotation metamodel (AMM) offers both kinds of annotations. They are realized by a model weaving technique [5]. Figure 6 shows an excerpt of the AMM for J2EE systems to exemplary illustrate the annotation capabilities. The AMM offers the `DomainFunctionAnnotation` metaclass to link the functional parts of the platform specific metamodel with the domain function of the semantic metamodel. On the right side of the picture, all data related elements of the J2EE PSMM are linked to the semantic metamodel using the `DomainObjectAnnotation`. In addition, the AMM contains all further associations to other annotateable metamodel elements (AMM is implemented as Ecore model with EReferences to all the other Ecore models). One of the main advantages of semantic annotation through all levels of abstraction is the traceability of information. Apart from the annotation metaclasses, the AMM also offers means to combine annotations with logical operators. With the operators and varied use of single or multiple references it is possible to build containment, choice and multiple representation expressions.

6 Conflict Analysis

The metamodels forming DSL foundation at CIM, PSM and PIM levels are used for two main purposes: conflict analysis of interface mismatches and connector code generation. The conflict analysis algorithm examines interface descriptions at the PIM level in conjunction with the abstract process and data requirement definitions at the CIM level.

The *semantic* analysis uses the ontology and semantic annotations to check whether the abstract data flow requirements at the CIM level can be fulfilled by interface descriptions at the PSM/PIM level. Additionally, it determines dependencies of interface parameters and verifies their functional annotations according to the business function definitions in the CIM model. Using logical reasoning, mismatched requirements are resolved if possible. The results of the

analysis are requirement mappings of the abstract business objects to exporting and importing interfaces elements at the PSM/PIM level. An in-depth description of the semantic annotation and conflict analysis can be found in [3].

The *behavior* analysis derives a first interface call order. Dependencies and process definitions at the CIM level are checked against the behavior constraints of the interfaces, such as pre- and post-conditions. Closely related is the *property* analysis that examines interfaces' QoS properties and metrics, such as WCET or reliability [16]. The result of both analyses is a refined interface call order. The *communication* analysis then takes into account characteristics of interface interaction [15]. The results of this analysis, together with the information from the PSM level, are used to generate application endpoints that communicate with the system and offer a common access pattern to the connector.

Finally, the *structure* analysis overcomes the structural heterogeneity of the interfaces by performing range of values comparison, identifying required data type converters as well as creating a message processor lists for the e. g., merge, split or filter of data structures. The work of integration specialists is hence supported to a certain degree of automation, but due to the complexity of software systems a fully automated conflict resolution is difficult to accomplish in general case. Remaining conflicts or multiple conflict-free choices are resolved manually.

7 Connector generation

Based on the models describing the integration scenario and results of the conflict analysis, the connector component model and code are generated. A connector is an automatically generated component, which is used to overcome all discovered conflicts and enable technical, semantical and business interoperation. It is based on the principles of message oriented middleware (MOM), and its metamodel is accordingly based on the message passing.

The connector generation starts with the `ChannelAdapter` which comprises `ApplicationEndpoint` and `MessageGateway`. ApplicationEndpoint implements the technical interoperability, and is able to call remote system interfaces. It passes export parameters to the MessageGateway, which serializes them into `Message`. In the other direction, MessageGateway deserializes a message and passes it to the ApplicationEndpoint. Messages are further transported by `Channels` which can be either 1-1 channels or publish/subscribe. `MessageProcessor`s perform conflict resolution by executing aggregation, routing, transformation, enrichment etc. functions. Application endpoints can be generated using standard code generation methods or using model interpretation. Core connector logic (Channels and Message Processors) are interpreted based on the UML Action Semantics description.

The code generation approach is based on Java Emitter Template (JET). The first step is to read all import and export parameters and create Java classes out of them, where each parameter is wrapped within one class. In this step, PSMM types are transformed to the PIMM (Java) types. The second step is to create a Java method for each function modeled in the PSM. This method encapsulates

access to the target system. Import parameters are sent to the application endpoint as a hash map, equivalently, export parameters are also received as a hash map. Alternative is to use model interpretation, where application endpoints are built with Java code at runtime, starting with the interpretation of the platform specific model using the model interpreter component. It reads the Ecore-file of an interface and initiates a connection. Afterwards the interpreter builds a Java object, which implements the Interface *ICallableObject*. It consists of executable functions which conform to the PSM. The endpoint gateway component is responsible for receiving and sending of messages. It provides an interface with methods for registration of the business connector, in order to support the data transfer. Data converter supports translation of data into different formats, allowing a lossless and smooth data exchange within an application endpoint.

UML Action Semantics models are used to describe internal behavior of the connector core components. We extended basic UML Action Semantics metamodel to allow for the following action types: string, mathematical, logical, date and time and code actions (as extensions of computation actions), as well as read and write, type conversion, composite and collection actions [7]. Using this vocabulary, patterns such as Transfomer, Normalizer or Aggregator are built, which are then interpreted as Java code using Java Message Service (JMS).

Using the connector metamodel and code generation techniques, it is possible to exchange the underlying runtime environment without any manual intervention, for example, instead of JMS gateways and Java transformation logic, Web service (WSDL) gateway, BPEL orchestration code and XSLT transformations for the SOAP messages can be generated.

8 Conclusion

The project research results have been prototypically realized as the BIZYCLE Model-Based Integration Framework (MBIF). It is based on the presented metamodels to support different abstraction levels that are part of the BIZYCLE integration process. Beside the modeling platform, which guides a developer through the integration process steps, MBIF consists of two other main components: BIZYCLE Repository (offers all needed persistence services, version and consistency management) and BIZYCLE Runtime Environment (where the generated business connectors are deployed and executable models interpreted).

Based on the practical experience and feedback from our industrial partners, several benefits can be already identified. An important aspect is a degree of automation, achieved through code generation, systematic conflict analysis process and automated technical model extraction. Reuse is supported at the model-level, as interface descriptions, transformation rules and semantic annotations can be shared between multiple projects via BIZYCLE Repository. The evolution is supported at the model level, and code generation methods enable smooth transitions. Metamodeling enables very fast tool prototyping. However, metamodels also improve understanding of the problem domain. One of the major advantages of the proposed solution are multiple abstraction levels, such as

CIM, PSM, PIM and code, which enable business architects not to start at the data model and/or code level right away, as is usual in today's practice. The essential benefit offered by the multi-level modeling environment is based on the capability of performing (to a high degree) automated model transformations, abstracting and refining over the given level hierarchy.

References

1. ATL: Atlas Transformation Language User Manual. *http://www.eclipse.org/m2m/atl/doc/ATL_User_Manual[v0.7].pdf*, 2006.
2. Eclipse modeling framework. http://www.eclipse.org/modeling/emf/, 2008.
3. H. Agt, J. Widiker, G. Bauhoff, N. Milanovic, and R. Kutsche. Model-based Semantic Conflict Analysis for Software- and Data-integration Scenarios. Technical Report http://cis.cs.tu-berlin.de/Forschung/Projekte/bizycle/semca.pdf, 2008.
4. N. Boudjlida and H. Panetto. Annotation of enterprise models for interoperability purposes. In *Proceedings of the IWAISE 2008*, 2008.
5. M. D. D. Fabro, J. Bézivin, F. Jouault, E. Breton, and G. Gueltas. AMW: a generic model weaver. In *Proceedings of IDM05*, 2005.
6. T. Hildenbrand and R. Gitzel. *A Taxonomy of Metamodel Hierarchies*. University of Mannheim, 2005.
7. P. Hoffmann. *Design of a model-based message transformation language*. Diploma thesis, TU Berlin, 2008.
8. G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2003.
9. InterSystems. Ensemble data transformation language. http://www.intersystems.com/ensemble/docs/4/PDFS/DataTransformationLanguage.pdf, 2006.
10. A. Kiryakov, B. Popov, D. Ognyanoff, D. Manov, A. Kirilov, and M. Goranov. Semantic annotation, indexing, and retrieval. In *International Semantic Web Conference*, 2003.
11. T. Kühne. Matters of (meta-)modeling. *Software and System Modeling*, 5(4), 2006.
12. R. Kutsche and N. Milanovic. (Meta-)Models, Tools and Infrastructures for Business Application Integration. In *UNISCON 2008*. Springer Verlag, 2008.
13. R. Kutsche, N. Milanovic, G. Bauhoff, T. Baum, M. Cartsburg, D. Kumpe, and J. Widiker. BIZYCLE: Model-based Interoperability Platform for Software and Data Integration. In *Proceedings of the MDTPI at ECMDA*, 2008.
14. A. Leicher. *Analysis of Compositional Conflicts in Component-Based Systems*. PhD Dissertation, TU Berlin, September 2005.
15. N. Mehta, N. Medvidovic, and S. Phadke. Towards a Taxonomy of Software Connectors. In *Proceedings of the 22nd ICSE*, 2000.
16. N. Milanovic. *Contract-based Web Service Composition*. HU Berlin, 2006.
17. N. Milanovic, R. Kutsche, T. Baum, M. Cartsburg, H. Elmasgunes, M. Pohl, and J. Widiker. Model & Metamodel, Metadata and Document Repository for Software and Data Integration. In *Proceedings of the ACM/IEEE MODELS*, 2008.
18. E. Pulier and H. Taylor. *Understanding Enterprise SOA*. Manning, 2006.
19. E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, Jan 2001.
20. H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information - a survey of existing approaches. In *Proceedings of the IJCAI-01 Workshop: Ontologies and Information Sharing*, pages 108–117, 2001.