(Meta-)Models, Tools and Infrastructures for Business Integration

Ralf Kutsche, Nikola Milanovic
TU Berlin, Computergestützte Informationssysteme CIS
{rkutsche,nmilanov}@cs.tu-berlin.de

Abstract. In the context of the Berlin Brandenburg Business Process Integration and Evolution framework BIZYCLE, we are using modeling and metamodeling strategies in order to achieve a platform for the (semi) automatic peer2peer integration of software components. Our general methodology is based on the principles of the MDA paradigm, distinguishing between platform specific (PSM), platform independent (PIM) and computation independent (CIM) models, and the general philosophy of solving integration problems on higher levels of abstraction than on the concrete software level. Our approach includes previous work on model-based and metadata-based information integration methodologies and on model-based component conflict analysis and resolution. Both approaches essentially require model transformation techniques, which currently are under investigation in our project group. The basic models and metamodels on CIM, PIM and PSM level, forming the pre-requisite for our further project work, are examined in this article. Moreover, a sketch of our industrial application background – and thus the strength of our general methodology in bridging heterogeneity – with our consortial partners in the distinct industry sectors of health, production/logistics, publishing and facility management will be given.

Key words: modeling, metamodels, software & data integration, business applications

1 Introduction

The Berlin-Brandenburg Regional Business Initiative BIZYCLE (www.bizycle.de), a so-called 'Regionaler Wachstumskern', was established in early 2007 in order to examine the potential of model-based software and data integration methodologies, tool support and practical applicability in large-scale, conforming a consortium of six industrial partners and academia (in this case: TU Berlin/ CIS), under guidance of the German Federal Minister for Education and Research. ¹

The basic idea of BIZYCLE is to create a model-based and – more than that, of course – metamodel-based tool generation and interoperability operation platform, in order to allow for improved software and data integration processes

¹This work is partially supported by the Bundesministerium für Bildung und Forschung BMBF under grant number (Förderkennzeichen) 03WKBB1B.
in the area of software component integration. Classically, in daily business, software – and, as an integral part, data – integration tasks are performed by well-experienced software engineers and application experts, manually programming the connections (let us call them connectors in the classical component software terminology). In general, this requires rather professional skills in the software systems/ components/ applications on one-hand side, and integration requirements’ analysis skills on the other. This is a very expensive procedure, and several sources estimate the cost of integration issues up to 80 per cent of the overall IT investments (see e.g. [3]).

In order to reduce this significant cost factor, lots of initiatives in software industry and in science follow the way towards model-based software engineering in order to improve the quality and the process of software engineering by a strengthened use of models.  

The BIZYCLE initiative is based on its core project ISKI, which – as its essence – follows the strategy of model- and metamodel- based abstractions from platform specific aspects towards platform independent and computation independent ones. Our strategy focusses on (semi-)automated conflict analysis and connector generation as competitive advantage.

2 General Background and Related Work: Continuous Software Engineering and Information Systems Evolution

Since the early 90s, integration issues have become a central challenge in many industry sectors: heterogeneous, distributed software solutions under autonomous responsibilities brought up the definite need for data integration in heterogeneous, distributed environment, based on the strongly increasing capabilities of (cheap!) small and medium computer systems. ‘Federated Databases’ FDBMS (since the 80s) [17], and later on ‘Federated Information Systems’ FIS [16] and, their special case of read-only accessible ‘Mediator-Based Information Systems’ MBIS [4] [5] (cf. among others, the workshop series EFDBS/EFIS 1997 thru 2003, [15], [18], and our ‘Reference Model Federated Information Systems’ cf. Figure 1, and [16]), are the examples of results achieved in this context.

Finally, opening this area towards new paradigms like, e.g. ‘Peer Data Management Systems’ have consolidated this very challenging and rapidly emerging research area under the name ‘information integration’ (for an excellent summary

---

2 The term model in science is heavily overloaded: In this context, it is usually interpreted as using graphical oriented editors/ toolboxes for expressing the relevant aspects of the future system by sets of (well-formed) diagrams and additional documents as e.g. constraint specs, following the tradition of RM-ODP [6] and today’s strong initiative of MDA under the coordination of the Object Management Group OMG. In most of the cases, UML with its variety of modeling languages under the control of the UML metamodel in the MOF hierarchy is used in this context.

3 ISKI stands for ‘Methoden, Werkzeuge und Infrastruktur für die Komponenten-integrationsplattform des BIZYCLE Wachstumskerns’, i.e. ‘methods, tools and infrastructure for the component integration platform’.
we refer to the German textbook [19], in the context of the Berlin-Brandenburg Graduate School 'Distributed Information Systems', cf. http://coltrane.wiwi.hu-berlin.de/gkvi/).

Moreover, the 'Semantic Web' (cf. http://www.w3.org/2001/sw/) efforts have added the very important aspects of semantics (commonly running under the notion of 'ontology-based...') to this integration business.

These activities perfectly meet with a tremendous pressure in business, where integration is a crucial aspect of the success (or only: survival) in a global marketplace. However, simply taking 'information integration' as our challenge is, under a holistic view of 'I&C infrastructures', taken too short, because full interoperability, i.e. functional and behavioral integration to full extent in conjunction with structural integration, is needed in real business. This requirement additionally is complemented by an evolution view on software development, in our terminology expressed in the term 'Continuous Software Engineering' [11] (cf. Figure 2), meaning that a consistent view on and continuous development and 'maintenance' of all artifacts in the complex circles of forward/reverse/re-engineering processes must be achieved. These two holistic approaches of software and systems engineering, namely

1. integral view of integration issues across wide-scale I&C infrastructures, combining all informational aspects with full interoperability requirements, and
2. integral view on all artifacts in a continuous software engineering process, yielding benefits wrt. consistency goals

constitute the essence of our BIZYCLE philosophy, methodology and concrete work. Basic paradigm of our approach is the use of models and metamodels as 'containers' of the 'engineering knowledge' on all levels of abstraction, to be
complemented by special model extensions wrt. to semantic issues or metadata management.

3 BIZYCLE Methodology in Detail

In this section internals of the BIZYCLE Interoperability Platform will be investigated. We will describe BIZYCLE integration process, as well as the novel roles involved in the integration lifecycle. The abstraction levels already mentioned (CIM, PIM, PSM) will be examined in more detail, together with the process of model transformation, conflict analysis and connector generation.

3.1 BIZYCLE Integration Process

Four levels of abstractions are supported in the BIZYCLE integration process with appropriate models: computational independent level, platform independent level, platform specific level and deployment/code level (Figure 3).

Computational Independent Model (CIM) captures essentials of the business process that is enacted by the integrated systems. This modeling step consist of identifying business process, components and connectors. Results of this step are flow model and connector template model. They abstract ICT related properties from the integration solution and describe only process- and business-related issues. Flow model describes abstract data and control flow. Integration points and (incomplete) list of potential conflicts are described in the connector template model. This model will be subsequently refined. The following step is identification, definition and modeling of platform-specific system interfaces: input/output
parameters, communication capabilities, and non-functional properties (such as security, dependability or semantics) are described. PSM interface descriptions are abstracted to platform independent level (PIM), for the purpose of conflict analysis, as is not possible to directly define and analyse integration conflicts between heterogeneous platforms. A process of generalization is performed which transforms all PSM interface descriptions to the common PIM level. The process of conflict analysis generates a set of integration conflicts at the PIM level. It is performed iteratively until all conflicts are solved (or only non-solvable conflicts remain). Connector is then generated as a mediator component that matches requirements and capabilities of the observed interfaces. Finally, physical connector is generated, by transforming connector model from PIM to PSM level, generating connector code and deploying the connector into BIZYCLE integration middleware.

3.2 BIZYCLE Roles

Several roles emerge within BIZYCLE integration process (Figure 4). They enable not only modularization and separation of concerns in the integration process, but primarily aim to support automation and optimal (re)use of the domain-expert knowledge.

Integration solution architect develops flow model and connector template model, based on the requirement analysis. Platform specialists create and update metamodels for the platforms that are supported by BIZYCLE interoperability platform. Application specialists refine the work of solution architect, and create detailed platform specific interface models. System integrator uses platform specific interface models, generalizes them to PIMs and guides conflict analysis process. It is his responsibility to manually intervene when the interoperability platform is unable to solve the conflicts automatically. Finally, solution deployer
overssees code generation and connector deployment. He is also responsible for maintaining and managing the runtime environment. The roles can overlap, e.g., it is possible that platform specialist takes the role of the application specialist, or that one or more platform specialists participate in conflict analysis and connector generation process.

3.3 Metamodels and Model Transformation

The BIZYCLE interoperability platform is based on the principles of model-driven engineering and Model Driven Architecture (MDA) [7]. Therefore, metamodels and model transformation form the core BIZYCLE elements. They are defined at the M2 level of the Meta Object Facility (MOF) hierarchy [14]. The MOF-based BIZYCLE (Meta)model hierarchy is shown in Figure 5.

**Computational Independent Metamodels** (CIMM) comprise two metamodels: Flow Metamodel (FMM) and Connector Template Metamodel (CTMM). FMM is based on the UML2 Activity Diagram and describes business data and control flow between the integrated systems. For that purpose a signaling mechanism is used: `SendSignalAction` represents a source system, while `AcceptEventAction` represents target systems (from the flow perspective). Class `ConnectorNode` connects one sending action with one or more receiving actions. CTMM is used to describe potential conflicts and connector logic from the business-process perspective. The core element of CTMM is `Connector` class which connects two or more `IntegratableElement` classes. Connector template defines several `DataProcessor` classes, such as `Aggregator`, `Router`, `Transformer` or `Filter`. They capture abstract semantics of integration, which is the result of requirement analysis process. An example of such semantics is a flow model which uses `Aggregator` to express that customer data from two branch offices should be aggregated in a data warehouse. Technical aspects (e.g., tables to be exported or database drivers) are not addressed at this level. CTMM will
be refined at the subsequent levels. The connection between FMM and CTMM is shown in Figure 6, where one Connector activates one or more ConnectorNodes. CIMM can be perceived as extended UML 2 activity diagram, where activities between send and receive signal actions represent business connector template. The relation between Connector and ConnectorNode thus also has an activation semantics.

**Platform Specific Metamodels** (PSMM) cover a class of metamodels describing technical aspect of systems that are to be integrated. Systems under study, for which we currently provide support within BIZYLE interoperability platform, are relational databases (including extensions for MySQL, Microsoft SQL Server, Oracle, and Firebird), J2EE applications (EJB 2.1 and 3.0), .NET Framework applications (version 1.1 and 2.0), ERP applications (SAP R/3 BAPI/IDOC and AvERP), XML flat files and SOAP-based Web Services. In accordance with the view that all systems are treated as components, that is, as black boxes that expose their functionality via interfaces, PSMM offers a platform specific interface description. Aspects such as remote addresses, function and parameter signatures and structure, communication properties, diverse non-functional properties spe-
specific for each platform (e.g., security or transactions) are supported, as well as interface semantics. The goal of PSMM is to provide enough platform-specific information on interface properties, such that: 1) they can be abstracted to platform-independent level and be used in the process of conflict analysis (design time requirement) and 2) an automatic connector can be generated, that invokes the given interface (runtime requirement). As a short PSMM example, excerpt from SOAP-based Web Service metamodel is given in Figure 7.

All PSMM share the similar structure, where endpoint is described (e.g., WSDL, SQL query or EJB remote interfaces), parameters (e.g., SOAP messages,
Models for Business Integration

SQL tables or Java serializable objects), communication properties (e.g., SOAP over HTTP, ODBC or RMI-IIOP) and type system (e.g., XSD schema describing messages, SQL types or Java type system). Non-functional part of interface description has been left out of this example, as it is orthogonal to all PSMMs and will be commented on in the next section within PIMM context.

Platform Independent Metamodels (PIMM) are used to abstract platform-specific properties from interface descriptions, which is precondition for the conflict analysis process. Core excerpt from PIMM is given in Figure 8, containing interface, structure, property, behavior, communication and semantic packages. Depending on the source PSM, interface can be functional (e.g., SAP remote function), method (e.g., EJB) or document (e.g., Web Service). Each interface (more precisely, parameters which are not shown in this view) has a structure defined within platform independent type system. Platform specific types are then mapped into this type system in the process of model transformation (e.g., SQL VARCHAR(20) can be transformed into PIMM String with length 20). Object mapping is also supported through Object class, where object attributes are further PIMM-typed. Therefore, a Java object can be transformed into PIMM object. Properties of platform specific communication channels are abstracted using AccessPattern and MessageExchangePattern classes, describing function- and document-based communication styles. A synchronous .NET remoting service, for example, is then represented with synchronous AccessPattern, asynchronous EJB event handler with asynchronous Event and SQL query with DataAccess. Behavior is associated with each interface, describing pre-conditions, post-conditions and invariants. Properties define quality of service attributes, such as security, performance, reliability, etc. Using combination of behavior and properties, it is possible to express requirements and capabilities of each interface, e.g., login, encryption or transactional behavior. Interface can carry semantic description via association to BusinessFunction. Domain ontologies are used to describe business semantics, together with business objects that are mapped to physical interface parameters. For example, SAP function can be annotated with a business function stating that it performs tax calculation operation for an order object using domain ontology from the logistics sector. Model transformation rules, that perform generalization from PSM to PIM level, are defined and implemented in ATL language [2].

Connector and Conflict Metamodels represent the final step in the process of interface integration. Connector model is message-based, that is, interface heterogeneity is addressed using messaging as key abstraction concept. Integration data and control flow are determined within a connector, which uses channels to transport messages between interfaces of source and target applications. Message processors operate on the message content, namely, they can route messages (control and data flow) or transform message content. Transformation can take the form of aggregation, filtering, translation, enrichment, normalization, etc. A message-middleware independent transformation language is developed for expressing message content transformation rules. Business goals, as well as tech-
Fig. 8. Platform Independent Metamodel (excerpt)

technical requirements are therefore matched within a connector. Connector also contains an application endpoint, which is an abstract placeholder for the interface client-side code. Thus our definition of a connector departs somewhat from definition established in the component-based software engineering, where connectors (from the architectural point of view) are passive first-class architectural entities that describe communication semantics between mismatched components [13]. Our understanding of a connector is pro-active, where a connector not only describes behavior protocol between two components, but can activate that behavior in runtime. According to the principles of generative programming [8], BIZYCLE connectors do not require glue code, as they are generated up to the code/deployment level and can communicate autonomously with the target interfaces.
The BIZYCLE interoperability platform recognizes the following conflict types: structural, behavioral, communication and property. To represent each conflict type, an appropriate formal abstraction is used. Structural conflicts are type conflicts which are detected and solved using typing and subtyping rules. Behavior conflict can be solved using either simulation and bisimulation [1], or transferring interface descriptions into abstract machine notation and performing invariant preservation proofs [12]. Process algebra [10] is used to perform communication and protocol compatibility check, while frame-logic [9] is suitable to perform semantical and property checks. The overall strength of this approach is the possibility to use model transformation (with PIM as a source), to choose adequate target formalisms and languages for verifying properties of interest.

4 Conclusion and Outlook

We have presented the results of the first phase of our ISKI project within the BIZYCLE initiative, i.e.

- clarifying the general philosophy and strategy of our model- and metamodel-based approach towards (semi)automatization of software and data component integration
- defining the 'new' process and roles which will be used in the future when the BIZYCLE platform has become operational, and, finally
- giving a sketch of our bunch of (meta)modeling activities which constitute the basis for the BIZYCLE framework’s toolboxes and interoperability platform.

At present, detailed specifications of concrete integration scenarios are being finalized by our industrial partners, which already (in the first round of our cyclic procedures) have been used for the platform specific models partially presented in this paper. From that we shall be able to revise and to finalize the PSMs and then integrate them into our tool generator.

Several of the aspects of component conflict analysis (cf. the Ph.D. thesis of Andreas Leicher in our research group [1]) and of the connector generation capabilities are still under investigation. For the time being, we are going to refine and to extend algorithms for conflict analysis under all viewpoints (structure, behaviour, communication and other properties), and to establish the basis for conflict resolution and connector generation on PIM level, and then, for the long term, model transformations towards connector generation on PSM level.

References