Matthias Böhm, Dirk Habich, Wolfgang Lehner, Uwe Wloka

Model-Driven Development of Complex and Data-Intensive Integration Processes

Erstveröffentlichung in / First published in:

Model-Based Software and Data Integration: First International Workshops. Berlin, 01.-03.04.2008. Springer, S. 31-42. ISBN 978-3-540-78999-4.

DOI: https://doi.org/10.1007/978-3-540-78999-4_5

Model-Driven Development of Complex and Data-Intensive Integration Processes

Matthias Böhm[1], Dirk Habich[2], Wolfgang Lehner[2], and Uwe Wloka[1]

[1] Dresden University of Applied Sciences, Database Group
mboehm@informatik.htw-dresden.de,
wloka@informatik.htw-dresden.de

[2] Dresden University of Technology, Database Technology Group
dirk.habich@inf.tu-dresden.de,
wolfgang.lehner@inf.tu-dresden.de

Abstract. Due to the changing scope of data management from centrally stored data towards the management of distributed and heterogeneous systems, the integration takes place on different levels. The lack of standards for information integration as well as application integration resulted in a large number of different integration models and proprietary solutions. With the aim of a high degree of portability and the reduction of development efforts, the model-driven development—following the Model-Driven Architecture (MDA)—is advantageous in this context as well. Hence, in the GCIP project (Generation of Complex Integration Processes), we focus on the model-driven generation and optimization of integration tasks using a process-based approach. In this paper, we contribute detailed generation aspects and finally discuss open issues and further challenges.

Keywords: Model-Driven Architecture, Integration Processes, GCIP, Federated DBMS, Enterprise Application Integration, Extraction Transformation Loading.

1 Introduction

The scope of data management continuously changes from centrally stored data to the integration of distributed and heterogeneous systems. In fact, integration is realized on different levels of abstraction: we distinguish between information integration (function integration and data integration), application integration, process integration, and partially also GUI integration. Due to missing standards for information integration and application integration, numerous different integration systems with overlapping functionality exist. In conclusion, only a low degree of portability of the integration task specification can be reached. However, portability is strongly needed, in particular, in the context of extending existing integration projects. Typically, in an enterprise IT-infrastructure, there are multiple integration systems. Assume that two relational databases are currently integrated with a federated DBMS for reasons of performance. If this integration process should be extended with the aim of integrating an SAP R/3 sys-tem, functional restrictions make it necessary to transfer the integration process to the existing EAI server, with the lowest possible development efforts. Aside from the main problem of portability, there are three more major problems. First, extensive efforts are

Final edited form was published in "Model-Based Software and Data Integration: First International Workshops. Berlin 2008", S. 31-42, ISBN 978-3-540-78999-4
https://doi.org/10.1007/978-3-540-78999-4_5

needed in order to specify integration tasks. This includes, for example, mapping specifications between heterogeneous data schemas. Although there are approaches to minimize these efforts, like automated schema matching or data integration with uncertainty (e.g., dataspaces), these approaches are not applicable in real-world enterprise scenarios, due to insufficient exactness. Second, very little work exists on the optimization of integration tasks. However, this is crucial, e.g., in the context of application integration, where complete business processes depend on it. Third, the decision about the optimal integration system (w.r.t. performance, functionality and development effort) is made based on subjective experience rather than on objective integration system properties.

Hence, our GCIP project (Generation of Complex Integration Processes) addresses the mentioned problems using a model-driven approach to generate and optimize integration processes. Basically, integration processes are modeled as platform-independent models (PIM) using graphical process description notations like UML and BPMN. Those are transformed into an abstract platform-specific model (A-PSM). Based on this central representation, different platform-specific models (PSM) can be generated with the help of specific platform models. Currently, we provide platform models for federated DBMS, EAI servers and ETL tools. Finally, each PSM can be transformed into multiple declarative process descriptions (DPD), which represents the code layer.

The contribution of this paper mainly includes three issues. First, after having surveyed related work in this research area in Section 2, we show how integration processes can be modeled in a platform-independent manner in Section 3. Second, in Section 4, we discuss generation aspects including the A-PSM, PSM and DPD specifications. Third, in Section 5, we enumerate open issues and research challenges correlated to our approach. Finally, we conclude the paper in Section 6 and give an overview of our future work on the optimization of integration processes.

2 Related Work

A lot of work on MDA techniques already exists [1,2]. Further, sophisticated tools and frameworks for model-driven software engineering have been developed. Therefore, we survey related work in three steps. First, we point out major MDA techniques. Second, we show that for application development as well as for data modeling, suitable MDA support exists. Third, we illustrate the lack of model-driven techniques and approaches in the context of integration processes.

The core concepts of a model-driven architecture (MDA)—specified by the Object Management Group (OMG)—are MOF [3], standardized meta models like UML [4], and model-model transformation techniques. In accordance to [5,6], it could be stated that *QVT (Query/View/Transformations)* and *TGG (Triple Graph Grammars)* are currently the most promising techniques for such model-model transformations. Due to the bidirectional mapping and the possibility of incremental model changes, TGG in particular is seen as the most suitable solution for model transformations. TGG comprises three graphs: the left-side graph (representing the first model), the right-side graph (representing the second model) and finally, a correspondence graph between the two models. Based on the given correspondence graph, a TGG rule interpreter is able to process the graph transformations for both directions.

Further, the MDA paradigm is widely used in the area of database applications for database creation. First, the model-driven data modeling and the generation of normalized database schemas should be mentioned. This approach is well known and many different systems and tools exist. Second, there is the generation of full database applications, including the data schema as well as data layer code, business logic layer code, and even user interface code. In this area, CASE tools in particular have to be named. However, there are still research items in this application area. For example, the logical database optimization is not realized adequately. The GignoMDA project [7,8] addresses this research item by allowing hint specification. Third, it can be stated that in the application area of data warehouse schema creation, MDA is also used increasingly. In accordance to the MDA paradigm, the *CWM (Common Warehouse Metamodel)* [9] should be mentioned. However, in contrast to normalized data modeling, there is not as much MDA support for data warehouse modeling.

Within the context of integration processes, there is insufficient support for model-driven development. Basically, two different groups of related work should be distinguished in this area. First, there is the modeling of workflow processes. With WS-BPEL [10], there is a promising standard and several extension proposals. However, this language has deficits concerning the modeling of data-intensive integration processes; plus, it is a platform-specific model. In accordance to the MDA paradigm, the existing process description languages can be classified into three layers: (1) *graphical representation layer*, (2) *description layer* and (3) *execution layer*. The graphical notations of UML 2.0 and BPMN 2.0 [11] are included in (1). Further, in (2), only WS-CDL [12] should be mentioned. Finally, (3) comprises WSBPEL, XPDL [13] and ebBP [14]. With this logical stratification in mind, the model-driven development of workflow processes is possible. In contrast to this, there are only few contributions on generic integration process generation. One of these is the RADES approach [15], which tries to give an abstract view on EAI solutions using technology-independent and multi-vendor-capable model-driven engineering methodologies. However, this approach is very specific to EAI solutions. Although some work on ETL process modeling [16,17,18] and ETL model transformation [19,20,21] exists, most data integration research addresses schema matching automation [22,23] or static meta data management [24] rather than the model-driven generation of integration tasks for different target integration systems. Thus, we are not aware of any solution for the generic generation of data-intensive integration processes. However, we are convinced that such a solution is required as a logical consequence of the historical evolution in this area. Finally, we want to refer to the Message Transformation Model (MTM), a conceptual model for data-centric integration processes in the area of message-based application integration, which was introduced in short with [25]. This model is used as our abstract platform-specific model and thus, as the core of the whole generation process.

3 Integration Process Modeling

Due to the lack of solutions for generic model-driven integration process generation, the main approach of modeling complex integration processes is introduced in this section. As already mentioned, integration processes can be specified with platform-independent models (PIM). Therefore, the PIM has to be derived from the computer-independent model (CIM), which is represented by textual specifications. Further, it is the first formal specification of the integration process and thus the starting point for the automated generation. In order to allow different target platforms, at this level, no technology is chosen. Thus, the platform independence of the model is reached by restricting the modeling to graphical notations like BPMN [11] and UML [4].

The procedure for PIM modeling is equal to the procedure of structured analysis and design (SAD). The single steps—which are introduced in the following—are used in order to allow for different degrees of abstraction during modeling time. Note that it might occur that, different persons will model the different degrees of abstraction.

1. *Determination of terminators (external systems)*: First, all external systems, also known as the terminators of a process description, and their types are determined. This is derived from the overall system architecture.
2. *Determination of interactions with the terminators*: Second, the type of interaction is specified for each interaction between the integration system and an external system. As a result, this comprises the determination of whether these are read (pull), write (push) or mixed interaction forms.
3. *Control flow modeling*: Third, the detailed control flow is modeled, including structural aspects (e.g., alternatives), time aspects (e.g., delays and asynchronous flows) as well as signal handling (e.g., errors).
4. *Data flow modeling*: Fourth, the abstract data flow modeling is applied. This means the general specification of data flow activities like filters, data transformations and the transactional behavior.
5. *Detailed model parameterization*: Finally, all control flow and data flow activities are parameterized in detail using annotations. Thereby, condition evaluations are specified and the data transformation is described on the schema mapping level.

Those five modeling steps result in a single platform-independent model which represents the integration process. It would be possible to separate aspects with different model views (terminator interactions, control and data flow, parameterization and configuration). Due to an increasing complexity, we explicitly do not use multiple models.

We want to introduce the example processes `P13` and `P02` from the DIPBench (Data-Intensive Integration Process Benchmark) specification [26,27]. Obviously, these are not complex integration processes, but we use these process types as running examples throughout the whole paper. Figure 1 shows the PIM `P13` and PIM `P02`, modeled with the help of StarUML, using the supported UML activity diagrams. The process type `P13` basically describes the extraction of movement data from a consolidated database and its loading into a global data warehouse. Let us use the introduced procedure for PIM process modeling: First, the two external systems `cs1.cdb.DBA` and `cs1.dwh.DBA` are determined ("service") and identified as `RDBMS`. Second, the interaction types ("operation") have to be detected. So, at the beginning of the process, a stored procedure is `called` on `cs1.cdb.DBA` in order to realize the data cleansing [28]. Furthermore, two different datasets are `queried` from the `cs1.cdb.DBA`. These datasets are finally `inserted` into `cs1.dwh.DBA`. Third, the control flow may be determined as a simple sequence of process steps. Fourth, the data flow—specified by

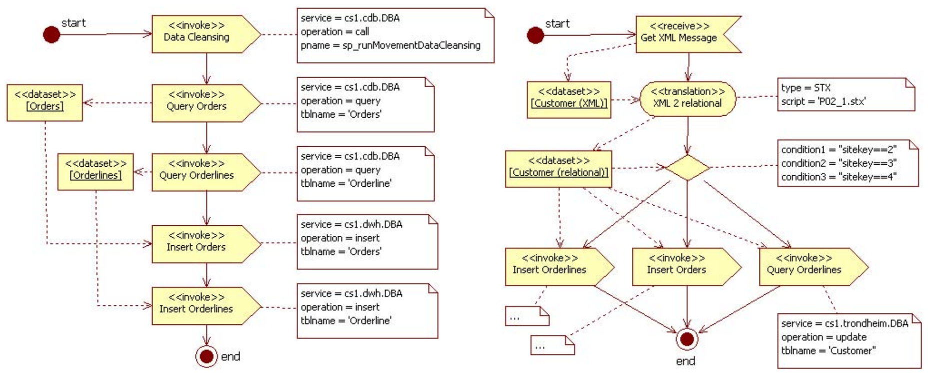

Fig. 1. Example PIM `P13` and PIM `P02`

rectangles and dashed arrows—has to be provided. As illustrated, the specific datasets are transfered from the extracting to the loading activities. Fifth, and thus finally, the model is enriched with technical details (in the form of UML annotations) like table names, procedure names and so on. The process type `P02` focuses on the reception of Customer master data (xml messages), its translation to relational data, and the content-based insertion into one of three target systems, based on the extracted `sitekey`.

4 Integration Process Generation

Based on the modeled platform-specific integration processes, our GCIP Framework supports the generation of several platform-specific models. Due to the complexity of this generation, we describe this from various perspectives.

4.1 A-PSM Generation

In contrast to other approaches, we use an abstract platform-specific model (A-PSM) between the PIMs and the PSMs. This is reasoned by four facts. First, it reduces the transformation complexity between n PIMs and m PSMs from $n \cdot m$ to $n + m$. Second, it separates the most general PIM representations from the context-aware A-PSM of integration processes, still independent of any integration system type. Third, it offers the possibility for applying context-aware optimization techniques in a unique (normalized) manner. And fourth, it increases the simplicity of model transformations, using small, well-defined transformation steps.

Basically, the Message Transformation Model (MTM) is used as A-PSM. This model represents the starting point of all transformations into and from platform-specific models. Obviously, we are only able to generate integration processes which can be expressed adequately with the MTM. Although the MTM was already introduced in short with [25], its importance drives us to give a short overview of this meta model. The MTM is a conceptual model for data-intensive integration processes and is separated into a *conceptual message model* and a *conceptual process model*.

The *conceptual message model* was designed with the aim of logical data independence and represents the static aspects of a message transformation. In accordance with the molecule atom data model (MAD), the MTM meta message model can be seen as a molecule type `message` and thus as a recursive, hierarchical, object-oriented and redundancy-free structure. The molecule type `message` is composed of two atom types: the `header segment` and the `data segment`. Furthermore, there is a unidirectional, recursive self-reference of the atom type `data segment`, with a 1:CN cardinality, which represents the molecule type `data segment`. There, the `header segment` is composed of k name-value pairs, whereas the `data segment` is a logical table with l attributes and n tuples. The option of nested tables ensures a dynamic and structure-carrying description of all data representations.

The *conceptual process model*—the execution model for the defined messages—addresses the dynamic aspects of a transformation process and was designed with the aim of independence from concrete process description languages. Basically, a graph-oriented base model *Directed Graph* is used. It is limited to the three components: `node` (generalized process step), `transition` (edge between two nodes) and the hierarchical `process`. A single node may have multiple leaving transitions, and during the runtime of one process instance, there may be multiple active leaving transitions but not more than the total number of its leaving transitions. Thus, one transition has exactly one target node. Indeed, multiple transitions could refer to one node. The process is a hierarchical element and contains a start node, several intermediate nodes and an end node. In fact, such a process is also a specialized node, so that the recursive execution with any hierarchy level is possible. A node receives a set of input messages, further executes several processing steps specified by its node type and its parameterization, and finally returns a set of output messages. The actual process model is defined—with the aim of a low degree of redundancy—on top of the base model *Directed Graph*. Operators are defined as specialized process steps and thus as node types. Basically, these are distinguished into the three categories: interaction-oriented operators (`Invoke`, `Receive` and `Reply`), control-flow-oriented operators (`Switch`, `Fork`, `Delay` and `Signal`) and data-flow-oriented operators (`Assign`, `Translation`, `Selection`, `Projection`, `Join`, `Setoperation`, `Split`, `Orderby`, `Groupby`, `Window`, `Validate`, `Savepoint` and `Action`).

Definition 1. *A process type P is defined with $P = (N, S, F)$ as a 3-tuple representation of a directed graph, where $N = \{n_1, \ldots, n_k\}$ is a set of nodes, $S = \{s_1, ..., s_l\}$ is a set of services, including their specific operations $s_i = \{o_1, ..., o_m\}$, and $F \subseteq (N \times (S \cup N))$ is a set of flow relations between nodes or a node and a service. Each node has a specific node type as well as an identifier NID (unique within the process type) and is either of an* atomic *or a* complex *type. Each process type P, with $P \subseteq N$, is also a node. A process p with $P \Rightarrow p$ has, a specific state $z(p) = \{z(n_1), \ldots, z(n_k)\}$. Thus, the process state is an aggregate of the specific single node states $z(n_i)$.*

Basically, two different event types initiating such integration processes have to be distinguished. The specific event type has a high impact on the process modeling and on the generation of platform-specific models for different target integration systems. The main event types could be distinguished as follows:

- *Message stream*: Processes are initiated by incoming messages. According to the area of data streaming, such a stream is an event stream. Processes of this event type have a RECEIVE operator and are able to reply to the invoking client.
- *External events and scheduled time events*: Processes are initiated in dependence on a time-based schedule or by external schedulers. These processes do not have a RECEIVE operator and are not able to reply synchronously to an invoking client.

An XML representation of the conceptual MTM was defined. Thus, the external XML representation of the PIM can be transformed into the XML representation of the A-PSM using transformation templates. Figure 2 shows the A-PSM P13, derived from the PIM P13. Basically, the MTM is message-based and thus uses message variables instead of the explicit data flow used in the PIM. Furthermore, detailed parameters like table names and stored procedure names have to be assigned to the input messages. That is why the logical sequence is extended with several ASSIGN operators. In contrast to this, the interaction-oriented operator INVOKE is simply mapped. Finally, note that technical details, like schema mappings and configuration properties, can be annotated on the PIM as well as the A-PSM level, while all following models are generated in a fully automated manner.

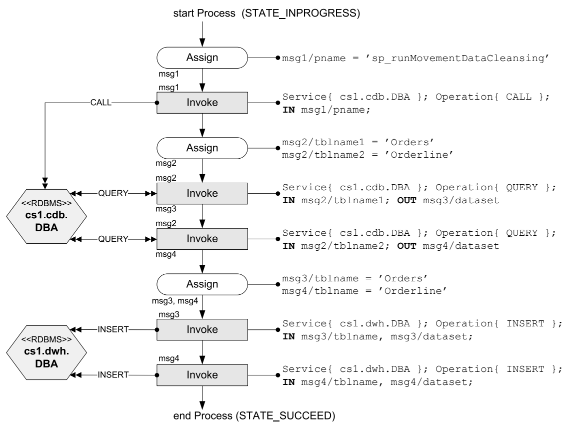
start Process (STATE_INPROGRESS)
Assign
msg1/pname = 'sp_runMovementDataCleansing'
msg1
msg1
CALL
Invoke
Service{ cs1.cdb.DBA }; Operation{ CALL };
IN msg1/pname;
Assign
msg2/tblname1 = 'Orders'
msg2/tblname2 = 'Orderline'
msg2
msg2
<<RDBMS>>
cs1.cdb.
DBA
QUERY
Invoke
Service{ cs1.cdb.DBA }; Operation{ QUERY };
IN msg2/tblname1; OUT msg3/dataset
msg3
msg2
QUERY
Invoke
Service{ cs1.cdb.DBA }; Operation{ QUERY };
IN msg2/tblname2; OUT msg4/dataset
msg4
Assign
msg3/tblname = 'Orders'
msg4/tblname = 'Orderline'
msg3, msg4
msg3
<<RDBMS>>
cs1.dwh.
DBA
INSERT
Invoke
Service{ cs1.dwh.DBA }; Operation{ INSERT };
IN msg3/tblname, msg3/dataset;
msg4
INSERT
Invoke
Service{ cs1.dwh.DBA }; Operation{ INSERT };
IN msg4/tblname, msg4/dataset;
end Process (STATE_SUCCEED)

Fig. 2. Example A-PSM P13

4.2 PSM Generation

From the unique A-PSM, multiple platform-specific models (PSM), including PSMs for FDBMS, ETL tools and EAI servers, could be generated. For this transformation, the defined specific platform models (PM) are used. Here, a PM represents a meta model for an integration system type, like the type ETL tool. We describe in detail only the PM for FDBMS, including the resulting PSM, while the integration system types ETL tools and EAI servers are only discussed from a high-level perspective.

Federated DBMS PSM. The PSM for federated DBMS comprises structural as well as semantic differences to the A-PSM (MTM). In contrast to the MTM, it is rather hierarchically structured and not a graph-based model. Thus, some structured components recursively include other structured components as well as atomic components. Furthermore, the two different process-initiating event types are expressed with two different possibilities for the root component. First, there is the Trigger, which is bound to a queuing table and represents the event type *message stream*. Second, there is the

Table 1. Interaction-Oriented Operators

Name	Description
Call	Invoke a persistently stored module (e.g., procedures, functions)
Insert	Load a dataset into a specified relation
Delete	Delete a specified dataset
Update	Set a specified attribute value
Resource Scan	Extract a dataset from a specified relation

Table 2. Control-Flow-Oriented Operators

Name	Description
If	Execute all path children if the path condition is true
Delay	Interrupt the execution based on a timestamp or for a specified time
Signal	Terminate the integration process or raise a defined error
Iteration	Loop over all children while the specified condition is true

Table 3. Data-Flow-Oriented Operators

Name	Description
Assign	Value assignment of atomic or complex objects (different query language)
Translation	Execution of elementary schema translations
Selection	Choice of tuples in dependence on a specific condition
Projection	Choice of attributes in dependence on a specific attribute list
Join	Compound of multiple datasets depending on conditions and types
Setoperation	Use of the set operations `union`, `intersection` and `difference`
Split	Decomposition of a large XML document into multiple rows
OrderBy	Sorting of a dataset depending on a specified attribute
GroupBy	Partitioning of tuples with grouping attributes and aggregate functions
Window	Partitioning of tuples for ordering and correlations, without grouping
Validate	Constraint validation on a dataset depending on a specific condition

Table 4. Transaction-Oriented Operators

Name	Description
BeginTX	Start a new transactional context
CommitTX	End the current transactional context in a successful way
RollbackTX	End the current transactional context in a failed way
Savepoint	Write intermediate results for recovery processing

`Procedure`, representing the event type *external events and scheduled time events*, where no data-intensive parameters are specified for such a procedure. Both of these root component types implicitly include the handling of the transactional context.

Before revisiting our used example, the components of the PM FDBMS s hould be mentioned. Basically, they are distinguished into four groups: interaction-oriented operators, control-flow-oriented operators, data-flow-oriented operators and transaction-oriented operators. These groups are explained in detail by Tables 1 to 4.

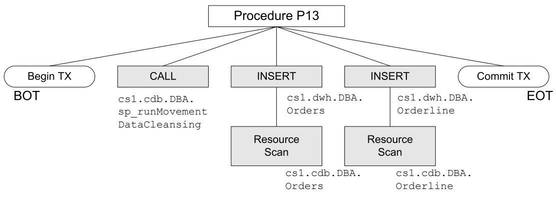

Fig. 3. Example FDBMS PSM `P13`

Using the introduced platform model, the PSM FDBMS is derived from the A-PSM. Figure 3 shows this PSM FDBMS representation of the example process `P13`. The process is transformed into the hierarchical structure `Procedure`. Here, a `BeginTX` is implicitly inserted at BOT and a `CommitTX` is inserted at EOT. The first invoke and all correlated properties are represented by the `Call` element. Finally, there are two query trees, constructed of `Insert` and `Resource Scan` elements, which copy the two movement data relations from the consolidated database to the data warehouse.

ETL Tool PSM. Similar to the first mentioned PSM, the PSM for ETL tools has some semantic differences to the MTM. This platform model is based on tuple routing between data-flow-oriented process steps, where the edges between these steps contain—on the conceptual layer—queues for buffering of tuples. Furthermore, it is an acyclic model, which is typical for the concentration on data flow semantics. In our generic platform model of ETL tools, no special process steps are included directly in the model because they are almost proprietary definitions. Using the mentioned platform model, the introduced example A-PSM `P13` can be transformed to the ETL PSM. The derived model includes specialized process steps, where these are specific to the used source and target systems. Further, the parameter specifications are directly included in the used steps. This approach promises high performance but causes a higher data dependence than an EAI server would do.

EAI Server PSM. Many commercial EAI servers use XML technologies and standard workflow languages like WSBPEL [10] or XPDL [13] for integration process specification. This abstract definition and the well-known adapter concept realize the needed data independence. Thus, the derivation of the EAI server PSM implies the mapping from A-PSM process types to WSBPEL process descriptions. Although this is a very easy mapping for interaction and control flow operators, it is recommended that the specific target integration systems support the WSBPEL extension *WSBPEL-MT*. Otherwise, the data-flow-oriented operators defined for the MTM could not be used. However, for the example process `P13`, this extension support is not required because interaction-oriented operators, control-flow-oriented operators and the data-flow-oriented operator `Assign` are used exclusively.

4.3 DPD Generation

In contrast to the MDA paradigm, where the lowest representation level is the *Code* layer, we name the lowest—MDA-relevant—level of the integration process representation the *Declarative Process Description (DPD)*. This decision was made in order to distinguish (1) the integration process specifications (DPD) deployed in the specific integration system and (2) the actual generated integration code, internally produced by the integration system. However, the mentioned process descriptions are usually specified in a declarative way because this leaves enough space for physical optimization techniques. Let us face the concrete DPDs for federated DBMS, ETL tools and WSBPEL-conform EAI servers. In case of FDBMS, the PSM `P13` is generated in a DPD, represented by several DDL statements and an SQL stored procedure. In contrast to that, `P02` is generated in the form of several DDL statements and a trigger due to the different event type. Both examples are included in appendix A.

The majority of ETL tools work based on XML specifications, which may be directly used or specified by an additional GUI. Also, the EAI server works similar to this, except for the standardized language specification. Further, there are two main types of XML specification usage for these two integration system types. Some tools statically generate imperative code in the form of internal execution plans. Other tools dynamically interpret the XML specifications, where created object graphs are used. The decision is made based on performance aspects as well as flexibility requirements.

5 Open Issues and Challenges

Due to the complexity of model-driven generation, there are major challenges and open research issues to be solved. Basically, we see the following five points:

- *Schema mapping extraction*: We extract schema mapping information from `XSLT` and `STX` stylesheets. Due to the complex functionality of these languages, there are cases, where schema mapping extraction cannot be realized. Hence, sophisticated techniques for the schema mapping generation have to be developed, including levels of uncertainty when no exact information can be provided.
- *Round-trip engineering*: Our approach works in a top-down direction, so all changes should be made on the PIM or the A-PSM level. In order to support incremental changes and the migration of legacy integration tasks, reverse engineering is advantageous. However, there are lots of challenges correlated to this issue.
- *Intra-system process optimization*: The model-driven generation leaves enough space for logical process optimization, rewriting process plans with rule-based and workload-based optimization techniques.
- *Inter-system process optimization*: Aside from the aforementioned optimization, the most powerful global optimization technique is the decision on the optimal integration system (w.r.t. performance). Thus, the major challenge is the workload-based decision on the chosen PSM and DPD.
- *Supporting different execution models*: There are integration system types which have a completely different execution model (e.g., subscription systems like replication servers) and thus, supporting those, creates some challenges.

6 Summary and Conclusion

In this paper, we addressed two major problems within the area of integration processes. First, there was the problem of a low degree of portability. Second, large efforts were needed in order to set up and maintain integration scenarios. We conceptually showed (and evaluated with the implemented GCIP Framework) that a model-driven generation approach can dramatically reduce these two problems and is thus, advantageous for this application context as well. However, there are two more major problems and a lot of open research challenges. In our future work, we will focus on the logical optimization of integration processes using our model-driven generation approach.

References

1. Kleppe, A., Warmer, J., Bast, W.: MDA Explained. The Model Driven Architecture: Practice and Promise. Addison-Wesley, Reading (2003)
2. Thomas, D., Barry, B.M.: Model driven development: the case for domain oriented programming. In: OOPSLA (2003)
3. OMG: Meta-Object Facility (MOF), Version 2.0 (2003)
4. OMG: Unified Modeling Language (UML), Version 2.0 (2003)
5. Königs, A.: Model transformation with triple graph grammars. In: MODELS (2005)
6. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Syst. J. 45(3) (2006)
7. Habich, D., Richly, S., Lehner, W.: Gignomda - exploiting cross-layer optimization for complex database applications. In: VLDB (2006)
8. Richly, S., Habich, D., Lehner, W.: Gignomda - generation of complex database applications. In: Grundlagen von Datenbanken (2006)
9. OMG: Common Warehouse Metamodel (CWM), Version 1.0 (2001)
10. OASIS: Web Services Business Process Execution Language Version 2.0 (2006)
11. BMI: Business Process Modelling Notation, Version 1.0 (2006)
12. W3C: Web Service Choreography Description Language, Version 1.0 (2005)
13. WfMC: Process Definition Interface - XML Process Definition Language 2.0 (2005)
14. OASIS: ebXML Business Process Specification Schema, Version 2.0.1. (2005)
15. Dorda, C., Heinkel, U., Mitschang, B.: Improving application integration with model-driven engineering. In: ICITM (2007)
16. Simitsis, A., Vassiliadis, P.: A methodology for the conceptual modeling of ETL processes. In: CAiSE workshops (2003)
17. Trujillo, J., Luján-Mora, S.: A UML Based Approach for Modeling ETL Processes in Data Warehouses. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, pp. 307–320. Springer, Heidelberg (2003)
18. Vassiliadis, P., Simitsis, A., Skiadopoulos, S.: Conceptual modeling for ETL processes. In: DOLAP (2002)
19. Hahn, K., Sapia, C., Blaschka, M.: Automatically generating OLAP schemata from conceptual graphical models. In: DOLAP (2000)
20. Mazón, J.N., Trujillo, J., Serrano, M., Piattini, M.: Applying mda to the development of data warehouses. In: DOLAP (2005)
21. Simitsis, A.: Mapping conceptual to logical models for ETL processes. In: DOLAP (2005)
22. Dessloch, S., Hernandez, M.A., Wisnesky, R., Radwan, A., Zhou, J.: Orchid: Integrating schema mapping and ETL. In: ICDE (2008)
23. Melnik, S., Rahm, E., Bernstein, P.A.: Rondo: A programming platform for generic model management. In: SIGMOD (2003)
24. Göres, J., Dessloch, S.: Towards an integrated model for data, metadata, and operations. In: BTW (2007)
25. Böhm, M., Habich, D., Wloka, U., Bittner, J., Lehner, W.: Towards self-optimization of message transformation processes. In: ADBIS (2007)
26. Böhm, M., Habich, D., Lehner, W., Wloka, U.: Dipbench: An independent benchmark for data intensive integration processes. In: IIMAS (2008)

27. Böhm, M., Habich, D., Lehner, W., Wloka, U.: Dipbench toolsuite: A framework for benchmarking integration systems. In: ICDE (2008)
28. Rahm, E., Do, H.H.: Data cleaning: Problems and current approaches. IEEE Data Eng. Bull. (2000)

A Example FDBMS DPD

```
EXEC sp_addserver cs1, ...
EXEC sp_addexternlogin cs1, ...
CREATE PROXY_TABLE rOrdersCDB EXTERNAL TABLE      AT "cs1.cdb.DBA.Orders"
CREATE PROXY_TABLE rOrderlineCDB EXTERNAL TABLE AT "cs1.cdb.DBA.Orderline"
CREATE PROXY_TABLE rOrdersDWH EXTERNAL TABLE      AT "cs1.dwh.DBA.Orders"
CREATE PROXY_TABLE rOrderlineDWH EXTERNAL TABLE AT "cs1.dwh.DBA.Orderline"

CREATE PROCEDURE P13 AS
BEGIN
  BEGIN TRANSACTION

    EXEC cs1.cdb.DBA.sp_runMovementDataCleansing
    INSERT INTO rOrdersDWH       SELECT * FROM rOrdersCDB
    INSERT INTO rOrderlineDWH   SELECT * FROM rOrderlineCDB

  COMMIT TRANSACTION
END
```

Listing 1.1. Example FDBMS DPD P13

```
...
CREATE TRIGGER P02 ON P02_Queue FOR INSERT
AS
BEGIN
  DECLARE
    @customerkey   BIGINT,   ...
    @sitekey       INTEGER,  ...
  BEGIN TRANSACTION

    SELECT @tid = (SELECT TID FROM inserted)
    SELECT @i = 1
    SELECT @xpath = "//customer[1]"
    WHILE NOT (SELECT xmlextract(@xpath, (SELECT convert(VARCHAR(16384),MSG) FROM P02_Queue WHERE TID=@tid)
                               RETURNS VARCHAR(16384))) IS NULL
    BEGIN
      SELECT @rowptr = xmlextract(@xpath, (SELECT convert(VARCHAR(16384),MSG) FROM P02_Queue WHERE TID=@tid)
                               RETURNS VARCHAR(16384))
      SELECT  @customerkey  = xmlextract('/customer/@Customerkey',@rowptr RETURNS BIGINT),
              @lastname     = xmlextract('/customer/@Lastname',@rowptr     RETURNS VARCHAR(40)),
              ...
              @lastmodified = xmlextract('/customer/@LastModified',@rowptr RETURNS DATETIME)
      IF( @sitekey = 2 OR @sitekey = 3)
      BEGIN
        IF NOT EXISTS(SELECT 1 FROM rCompanyBP WHERE Companykey=@companykey )
        BEGIN
          INSERT INTO rCompanyBP (Companykey, Name, ImportanceFlag)
            VALUES (@companykey, @companyname, 1)
        END
        INSERT  INTO rCustomerBP
              (Customerkey, Lastname, Firstname, AddressString, Zipcode,
               Sitekey, Phone1, Phone2, Companykey, Birthday, Created, LastModified)
          VALUES (@customerkey, @lastname, @firstname, @address, @zip,
                  @sitekey, @phone1, @phone2, @companykey, @birthday,  @created, @lastmodified)
      END
      ELSE IF (@sitekey = 4)
      BEGIN
        IF NOT EXISTS(SELECT 1 FROM rCompanyT WHERE Companykey=@companykey )
        BEGIN
          INSERT INTO rCompanyT (Companykey, Name, ImportanceFlag)
            VALUES (@companykey, @companyname, 1)
        END
        INSERT INTO rCustomerT
             (Customerkey, Lastname, Firstname, AddressString, Zipcode,
              Sitekey, Phone1, Phone2, Companykey, Birthday, Created, LastModified)
          VALUES (@customerkey, @lastname, @firstname, @address, @zip,
                  @sitekey, @phone1, @phone2, @companykey, @birthday,  @created, @lastmodified)
      END
      SELECT @i = @i + 1
      SELECT @xpath = "//customer[" || convert(VARCHAR(20),@i) || "]"
    END
    DELETE FROM P02_Queue WHERE TID = @tid
  COMMIT TRANSACTION
END
```

Listing 1.2. Example FDBMS DPD P02