MODEL-DRIVEN GENERATION AND OPTIMIZATION OF COMPLEX INTEGRATION PROCESSES

Matthias Boehm, Uwe Wloka
Dresden University of Applied Sciences, Database Group, 01069 Dresden, Germany

Dirk Habich, Wolfgang Lehner
Dresden University of Technology, Database Technology Group, 01062 Dresden, Germany

Keywords: Model-Driven Development, Integration Processes, Message Transformation Model, Federated DBMS, Enterprise Application Integration, Extraction Transformation Loading, Optimization.

Abstract: The integration of heterogeneous systems is still one of the main challenges in the area of data management. Its importance is based on the trend towards heterogeneous system environments, where the different levels of integration approaches result in a large number of different integration systems. Due to these proprietary solutions and the lack of a standard for data-intensive integration processes, the model-driven development—following the paradigm of the Model-Driven Architecture (MDA)—is advantageous. This paper contributes to the model-driven development of complex and data-intensive integration processes. In addition, we illustrate optimization possibilities offered by this model-driven approach and discuss first evaluation results.

1 INTRODUCTION

Based on the trend towards heterogeneous and distributed infrastructures, integration technology is increasingly used in order to exchange data between applications and systems as well as to provide uniform access to multiple data sources. Integration concepts can be classified into the following layers (Dessloch et al., 2003): information integration (data integration and function integration), application integration, process integration and GUI integration. These different fields of integration resulted in the existence of numerous different proprietary integration systems, like federated DBMS, subscription systems, Enterprise Application Integration (EAI) servers, Enterprise Information Integration (EII) frameworks and Extraction Transformation Loading (ETL) tools as well as workflow management systems.

Due to the large number of different integration systems with overlapping functionalities and the lack of a standard for integration tasks, major problems arise. In general, strong efforts are necessary to set up concrete integration scenarios. This obstacle of high development effort is accompanied by the need for well-educated administration teams to handle the specific integration systems. Obviously, a low degree of portability is reached for integration processes. This is a minor problem in static system environments. However, in real-world scenarios which change dynamically, this is unsuitable due to the extensive efforts required for migration. Further, there is the problem of efficiency. In many cases, the performance of complete business processes depends on the performance of the integration system.

Therefore, we focus on the model-driven generation and optimization of complex integration processes using MDA methodologies. Here, integration processes can be modeled in a platform-independent way and platform-specific integration tasks are generated automatically. During this generation, optimization techniques can be applied. Thus, the problems of high development effort and low portability can be eliminated using the model-driven generation approach, while the problem of efficiency is tackled with numerous optimization techniques.

To summarize, our contribution comprises the discussion of the model-driven generation (Section 3) and optimization (Section 4) of integration processes. Section 5 explains implementation details of the GCIP Framework (the realization of our approach) and provides the experimental evaluation. Finally, we conclude the paper in Section 6.

Boehm M., Wloka U., Habich D. and Lehner W. (2008).
MODEL-DRIVEN GENERATION AND OPTIMIZATION OF COMPLEX INTEGRATION PROCESSES.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - DISI*, pages 131-136
DOI: 10.5220/0001679301310136

2 RELATED WORK

Although there is a large body of work on MDA techniques and tools in general, this is not true for the model-driven generation and optimization of integration processes. Hence, we analyze related work from two perspectives.

Model-Driven Architecture (MDA): MDA is a paradigm for the model-driven development of software and hardware. The core approach leads from a computer-independent model (CIM) over the platform-independent model (PIM) to the platform-specific models (PSM). These PSMs, which are created in dependence on the platform models (PM), are used as input for template-based code generation. Here, standardized meta models like UML and model-model transformation techniques (Königs, 2005) are used. In the context of database applications, MDA is widely used but open research challenges still exist. Those are partly addressed in the GignoMDA project (Habich et al., 2006).

Model-Driven Integration Processes: In contrast to this, there are only few contributions on the platform-independent integration process generation. The RADES approach (Dorda et al., 2007) gives an abstract view on EAI solutions using technology-independent and multi-vendor-capable model-driven methodologies. However, this is an EAI platform-specific approach. Further platform-specific work on ETL process modeling (Trujillo and Luján-Mora, 2003; Vassiliadis et al., 2002) and ETL model transformation (Hahn et al., 2000; Mazón et al., 2005; Simitsis, 2005) exists. We are not aware of any solutions which allow the model-driven generation of integration processes. In the context of EII frameworks (data integration), sophisticated techniques like query scrambling (Urhan et al., 1998), bushy tree enumeration, dynamic rescheduling (Urhan and Franklin, 2001), plan partitioning (Kabra and DeWitt, 1998) and adaptive inter- as well as intra-query optimization techniques (Bruno and Chaudhuri, 2002; Stillger et al., 2001) has been developed. However, there is very little research on the optimization of integration processes. Preliminary results are shown on the optimization of ETL processes (Simitsis et al., 2005), the self-optimization of message transformation processes (Böhm et al., 2007), the optimization of data-intensive decision flows (Hull et al., 2000), the query optimization over Web services (Srivastava et al., 2006), and the rule-based rewriting of SQL activities in business processes (Vrhovnik et al., 2007). In fact, these techniques are not sufficient for the logical platform-independent optimization.

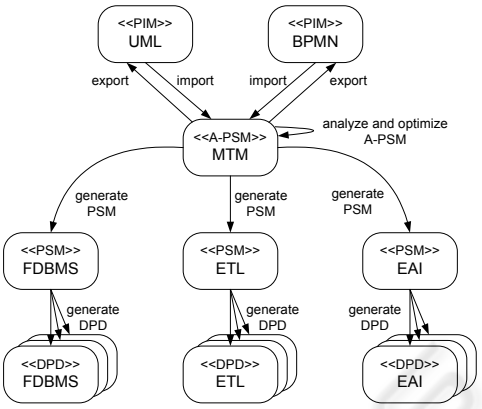

Figure 1: Main Generation Approach.

3 GENERATION ASPECTS

In this section, we introduce our main approach—shown in Figure 1—for the model-driven generation of integration processes (Böhm et al., 2008c) in short.

Here, integration processes are specified with a platform-independent model (PIM) in order to describe the overall process flow. Further, we base our development on one abstract platform-specific model (A-PSM) using the Message Transformation Model (MTM) (Böhm et al., 2007). Based on this A-PSM, several platform-specific models (PSM)—specific to an integration system type—can be generated using the specified platform models (PM). In the next step, the concrete declarative process descriptions (DPD)—specific to an integration system—can be created. Therefore, a number of specific dialects could be applied to each single PSM.

Platform-Independent Model (PIM): The PIM is 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. The platform-independence (no technology choice) of the model is reached by restricting the modeling to graphical notations like BPMN and UML activity diagrams. The procedure for PIM process modeling (determination of terminators, determination of interactions with terminators, control-flow modeling, data-flow modeling, detailed model parameterization) is equal to the procedure of structured analysis and design (SAD). The single steps are used in order to allow for different degrees of abstraction during modeling time. Be aware that, potentially, different persons might model the different degrees of abstraction.

Abstract Platform-Specific Model (A-PSM): In contrast to the original MDA approach, 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 PSMs of integration processes, still independent of any integration system. 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. Basically, the Message Transformation Model (MTM)—introduced in (Böhm et al., 2007)—is used as A-PSM. The MTM is a conceptual model for data-intensive integration processes and is separated into a *conceptual message model* (hierarchical data model) and a *conceptual process model* (directed graph and specific activities). Two different event types—initiating such integration processes—have to be distinguished. First, there is the *message stream*, where processes are initiated by incoming messages. Such a stream is an event stream. Here, processes have a `Receive` operator and can reply to the invoking client. Second, there are *external events and scheduled time events*, where processes are initiated in dependence on a time-based schedule or by external schedulers. These processes do not have a `Receive` operator and cannot reply synchronously to an invoking client. The external XML representation of the PIM can be transformed into the defined XML representation of the A-PSM. During this transformation, several structural changes are applied to the integration processes.

Platform-Specific Model (PSM): From the unique A-PSM, multiple platform-specific models (PSM), including PSM for FDBMS, ETL tools and EAI servers, can be generated. Therefore, the platform models (PM) are used. Here, we only pick the PSM for federated DBMS. In contrast to the MTM, the PSM for federated DBMS is rather hierarchically structured and not a graph-based model. Further, the two process-initiating event types are expressed with two different root components. The `Trigger` (bound to a queuing table) represents the event type *message stream* and the `Procedure` represents the event type *external events and scheduled time events*.

Declarative Process Descriptions: In contrast to the MDA paradigm, we name the lowest generation level the *Declarative Process Description (DPD)* in order to distinguish (1) the integration process specifications (DPD) deployed in the integration system and (2) the generated integration code internally produced by the integration system. Those are usually specified in a declarative way. In case of FDBMS, a DPD (represented by DDL statements) is generated.

4 OPTIMIZATION ASPECTS

The model-driven generation opens several optimization opportunities on different levels of abstraction. We distinguish the *Intra-System Process Optimization* (rewriting of process plans for execution time minimization) and the *Inter-System Process Optimization* (decision on the most efficient integration system).

4.1 Optimization-Influencing Factors

Both optimization types are influenced by several factors. We classify these factors into the three categories: *optimization aim*, *execution knowledge*, and *optimization technique* (see Figure 2). All of these have an impact on both intra-system as well as inter-system process optimization.

The *optimization aim* distinguishes between *execution time minimization* and *throughput maximization*. Here, the former requires a minimal latency, which is mainly used in combination with the synchronous execution model. For the latter, latency is acceptable but the number of executed processes per time period is maximized. There, multi-process optimization techniques (e.g., one external system lookup for multiple process instances) are used. Furthermore, the *execution knowledge* describes whether or not *workload statistics* exist or that *ad-hoc estimation (no statistics)* has to be applied. The former comprise average execution costs for process types and single process steps as well as probabilities for the execution of alternative process paths, statistics about average data sizes, selectivities and cardinality information but also network and external system delays. Finally, the factor *optimization technique* addresses *realization aspects*, while the other factors focus on *precondition aspects*. Here, we distinguish between *cost-based plan comparison* and the *rule-based plan rewriting*. The former creates a set of semantically equal process plans and evaluates those with empirical cost functions in dependence on the given workload characteristics. The latter applies algebraic rewriting rules based on pattern matching algorithms.

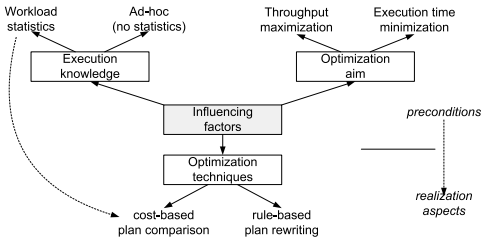

Figure 2: Influencing Factor Dimensions.

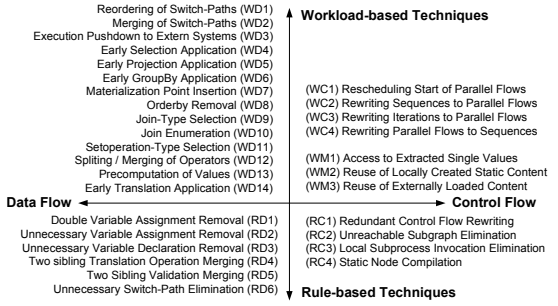

Figure 3: Intra-System Process Optimization Techniques.

4.2 Intra-System Process Optimization

In the context of *Intra-System Process Optimization*, several process rewriting and re-parameterization techniques—as shown in Figure 3 (technique classification)—could be applied on different levels of abstraction. Here, the abstraction levels (1a) Logical Process A-PSM Optimization, (1b) Logical Process PSM Optimization, (1c) Logical Process DPD Optimization, and (2a) Physical Process Code Optimization have to be distinguished, while optimization on PIM level is not suitable. We illustrate examples for each of these abstraction levels.

Logical A-PSM Optimization: Integration-system-specific hints and techniques as well as a concrete cost function could not be used due to missing knowledge of the target integration system. However, on this level, techniques like *WC2: Rewriting Sequences to Parallel Flows* can be applied with respect to existing data dependencies.

Logical PSM Optimization: Here, integration-system-type-related optimizations can be used. For example, the technique *WD7: Materialization Point Insertion* can be applied in order to optimize a federated DBMS PSM. This technique is used for distributed sub-queries, initially executed n times in order to share and execute them only once (reusing the materialized result n times). This also ensures a scalable processing. A second example is the technique *WD3: Execution Pushdown to External Systems*, where the execution can be delegated to external systems. This reduces the amount of transfered data and also prevents local processing steps.

Logical DPD Optimization: Further, vendor-specific tuning mechanism specifications can be used. As an example for that, we want to refer to *WM1: Access to Extracted Single Values* of document-oriented integration processes, where message indexing is used for single-value extraction.

Physical DPD Optimization: On this level, the most implementation-specific optimization techniques can be applied. These are realized within concrete integration systems and are explicitly not the subject of our model-driven approach. Examples for that are the techniques *RC3: Local Subprocess Invocation Elimination* (inline compilation of subprocesses) and *RC4: Static Node Compilation* (reuse of operators across multiple process instances). From our perspective, these optimization techniques could be used by specifying *physical hints*.

The main criticism of the *Intra-System Process Optimization* might be that optimizations on less abstract levels are more powerful. There are four advantages of the model-driven process optimization. First, this central optimization and distributed deployment reduces the development effort and further reduces the optimization time. Second, the abstract integration process specifications allow for an overall view of applicable optimization strategies. Third, the logical optimization achieves the independence from concrete integration systems. Fourth, the logical optimization allows to take physical optimizations into account during *Inter-System Process Optimization*.

4.3 Inter-System Process Optimization

The decision on the most efficient target integration system focuses on the search for the integration system with the maximal performance for a given process plan. Our solution consists of two steps. In the first step, functional properties (like the need for transactional behavior) are derived from the process plan. All integration systems are then evaluated to see whether or not they support these functional properties. In the second step, a performance comparison of the applicable integration systems is executed. For that, we transform the informal optimization aim into the mathematical model of DEA (Data Envelopment Analysis) (Seol et al., 2007), which is an operations research technique for analyzing the relative efficiency of two units. It is based on the idea to get the maximal output by providing the minimal input.

$$\max\left(\frac{\sum_{o=1}^{m}(w1_o \cdot x_o)}{\sum_{i=1}^{n}(w2_i \cdot y_i)}\right) \text{ with } \sum w1_o = \sum w2_i = 1$$

This mathematical model maximizes the ratio of weighted output factors $w1_o \cdot x_o$ and weighted input factors $w1_i \cdot y_i$. From this general DEA model, the GCIP decision model is derived.

$$\max\left(\frac{-w1_1 \cdot AVG(NC(p_x)) + w1_2 \cdot AVG(NT(p_x))}{CF_{istype}(p_x)}\right)$$

There, the output factors are chosen in accordance to the optimization-influencing factors. Thus, the output factors *normalized throughput* $NT(p_x)$ and *normalized processing time* $NC(p_x)$ are used. These factors are complementary, expressed by different algebraic signs. Further, there is only the input factor $CF_{istype}(p_x)$ which represents the cost function

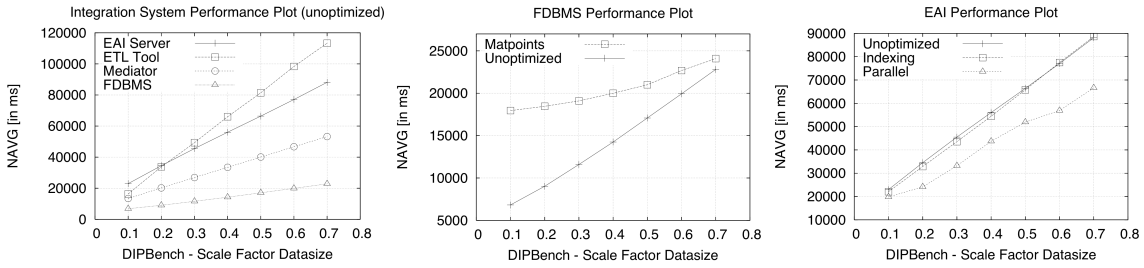

Figure 4: Plots of Different Integration Systems and Comparison of Unoptimized and Optimized Process Plans.

of the specific integration system type. Each cost function has platform-specific parameters, including main-memory constraints, target-system types, target-system interactions, dataset sizes, the degree of parallelism as well as CPU and disk usage. When changing the weights $w1_1$ and $w1_2$, the mentioned *optimization aim* could be specified. Thus, with a weight $w1_1$ lower than 0.5, the decision is made in an execution-time-oriented manner; otherwise, the optimization is based on the throughput of the system.

Aside from the optimization aim, the decision further depends on the kind of optimization model. Here, an input-oriented model (minimization of input) as well as an output-oriented model (maximization of output) may be applied. Although the inter-system process optimization has huge potential, it includes several problems. Thus, with the influence of dataset sizes in mind, it is obvious that the decision is influenced by workload statistics like dataset sizes.

5 EVALUATION EXPERIMENTS

The generation strategies as well as the performance of the different integration systems were evaluated in several experiments. We focus on the performance comparison of integration systems using the introduced example process type `P13` from the DIPBench (Böhm et al., 2008a; Böhm et al., 2008b).

5.1 Framework Implementation Details

Our approach was implemented within the GCIP Framework. There, we currently support the mentioned integration system types FDBMS, ETL tools and EAI servers. The macro-architecture of the framework consists of a swing-based GUI, some core components, a persistence layer as well as three main subcomponents: the `Transformer`, the `Dispatcher` and the `Optimizer`. First, the `Transformer` allows for model-model as well as model-code transformations using defined model mappings and code templates. Second, the `Dispatcher` realizes the *Inter-System Process Optimization* deciding on the optimal integration system based on functional properties and platform-specific cost functions. Third, the `Optimizer` subcomponent optimizes A-PSM models and thus realizes the *Intra-System Process Optimization* by rewriting process plans using rule-based and workload-based optimization techniques; it can be influenced with hints.

5.2 Setup and Measurements

The process type `P13` was used to generate the DPD for the integration system types FDBMS, ETL tools, EAI servers and a standalone mediator (generated reference). The experimental setup comprises three computer systems. On the first system (ES [Athlon64 1800+, 2GB RAM]), all external systems are located. On the second system (IS [Athlon64 1800+, 2GB RAM]), the integration system resides and the DIPBench toolsuite is located on the third system (CS [Dual Genuine Intel T2400, 1.5GB RAM]). We show seven different experiments (Figure 4 a-c). In order to show the datasize impact, we used different scale factors, reaching from 0.1 to 0.7, where a scale factor of 1 implies 70,000 *Order* rows and 350,000 *Orderline* rows. The process type `P13` was executed 100 times (with newly initialized external systems) for each datasize configuration with the aim of statistical correctness. As performance result metrics, we used the average of the normalized costs.

5.3 Performance Result Discussion

The comparison of unoptimized process executions on the different integration systems shows that the FDBMS always outperforms all other integration systems. This is caused by the semantics of `P13`, where only RDBMS are used as external system types. The mediator (generated standalone JDBC mediator, only used for comparison purposes) is always superior to the EAI server because the latter serializes the incoming tuples to XML, executes streaming transformations and deserializes the tuples, while the mediator

was generated as tuple-based (in-memory) mediator using plain JDBC. The EAI server shows a much better scaling than the ETL tool, while the ETL tool has a lower initial overhead. The resulting break-even point may be reasoned by two facts. First, the EAI server stores the temporary data in a coarse-grained way (XML document), while the ETL tool uses a fine-grained (tuple-based) temporary datastore. Second, the overhead for XML serialization decreases with increasing datasizes because it is executed asynchronously while reading and writing data to/from the external systems. In Figure 4b, there is the comparison between unoptimized and optimized (using materialization points) FDBMS DPD, which shows an interesting scaling. In Figure 4c, the unoptimized EAI server DPD is compared with optimized versions.

6 SUMMARY

The motivation behind this work and the enclosing research project *GCIP* was the lack of model-driven techniques for integration process generation. There was also little work on process optimization using rewriting and comparison techniques. Thus, our main approach was the adaptation of MDA to the context of integration processes and the application of optimization techniques on different levels of abstraction. To summarize the paper, we illustrated our approach of generating platform-specific integration processes for FDBMS, ETL tools and EAI servers. Further, we classified the optimization techniques which could be applied during model-driven process generation as well as the optimization-influencing factors. Finally, we described the framework implementation in short and discussed example performance evaluations.

REFERENCES

Böhm, M., Habich, D., Lehner, W., and Wloka, U. (2008a). Dipbench: An independent benchmark for data intensive integration processes. In *IIMAS*.

Böhm, M., Habich, D., Lehner, W., and Wloka, U. (2008b). Dipbench toolsuite: A framework for benchmarking integration systems. In *ICDE*.

Böhm, M., Habich, D., Lehner, W., and Wloka, U. (2008c). Model-driven development of complex and data-intensive integration processes. In *MBSDI*.

Böhm, M., Habich, D., Wloka, U., Bittner, J., and Lehner, W. (2007). Towards self-optimization of message transformation processes. In *ADBIS*.

Bruno, N. and Chaudhuri, S. (2002). Exploiting statistics on query expressions for optimization. In *SIGMOD*.

Dessloch, S., Maier, A., Mattos, N., and Wolfson, D. (2003). Information integration - goals and challenges. *Datenbank-Spektrum*, 3(6):7–13.

Dorda, C., Heinkel, U., and Mitschang, B. (2007). Improving application integration with model-driven engineering. In *ICITM*.

Habich, D., Richly, S., and Lehner, W. (2006). Gignomda - exploiting cross-layer optimization for complex database applications. In *VLDB' 06*.

Hahn, K., Sapia, C., and Blaschka, M. (2000). Automatically generating olap schemata from conceptual graphical models. In *DOLAP*.

Hull, R., Llirbat, F., Kumar, B., Zhou, G., Dong, G., and Su, J. (2000). Optimization techniques for data-intensive decision flows. In *ICDE*.

Kabra, N. and DeWitt, D. J. (1998). Efficient mid-query reoptimization of sub-optimal query execution plans. In *SIGMOD*.

Königs, A. (2005). Model transformation with triple graph grammars. In *MODELS' 05 workshops*.

Mazón, J.-N., Trujillo, J., Serrano, M., and Piattini, M. (2005). Applying mda to the development of data warehouses. In *DOLAP*.

Seol, H., Choi, J., Park, G., and Park, Y. (2007). A framework for benchmarking service process using data envelopment analysis and decision tree. *Expert Syst. Appl.*, 32(2).

Simitsis, A. (2005). Mapping conceptual to logical models for etl processes. In *DOLAP*.

Simitsis, A., Vassiliadis, P., and Sellis, T. (2005). Optimizing etl processes in data warehouses. In *ICDE*.

Srivastava, U., Munagala, K., Widom, J., and Motwani, R. (2006). Query optimization over web services. In *VLDB*.

Stillger, M., Lohman, G. M., Markl, V., and Kandil, M. (2001). Leo - db2's learning optimizer. In *VLDB*.

Trujillo, J. and Luján-Mora, S. (2003). A uml based approach for modeling etl processes in data warehouses. In *ER*.

Urhan, T. and Franklin, M. J. (2001). Dynamic pipeline scheduling for improving interactive query performance. In *VLDB*.

Urhan, T., Franklin, M. J., and Amsaleg, L. (1998). Cost-based query scrambling for initial delays. In *SIGMOD*.

Vassiliadis, P., Simitsis, A., and Skiadopoulos, S. (2002). Conceptual modeling for etl processes. In *DOLAP*.

Vrhovnik, M., Schwarz, H., Suhre, O., Mitschang, B., Markl, V., Maier, A., and Kraft, T. (2007). An approach to optimize data processing in business processes. In *VLDB*.