Invisible Deployment of Integration Processes

Matthias Boehm[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@tu-dresden.de
wolfgang.lehner@tu-dresden.de

Abstract. Due to the changing scope of data management towards the management of heterogeneous and distributed systems and applications, integration processes gain in importance. This is particularly true for those processes used as abstractions of workflow-based integration tasks; these are widely applied in practice. In such scenarios, a typical IT infrastructure comprises multiple integration systems with overlapping functionalities. The major problems in this area are high development effort, low portability and inefficiency. Therefore, in this paper, we introduce the vision of *invisible deployment* that addresses the virtualization of multiple, heterogeneous, physical integration systems into a single logical integration system. This vision comprises several challenging issues in the fields of deployment aspects as well as runtime aspects. Here, we describe those challenges, discuss possible solutions and present a detailed system architecture for that approach. As a result, the development effort can be reduced and the portability as well as the performance can be improved significantly.

Key words: Invisible Deployment, Integration Processes, Virtualization, Deployment, Optimality Decision, Heterogeneous Integration Platforms

1 Introduction

Integration processes—as an abstraction for workflow-based integration tasks—gain in importance because data management continuously changes towards the management of distributed and heterogeneous systems and applications. There, the performance of complete IT infrastructures depends on the central integration platforms. In this context, different integration system types are used. Examples for those types are Federated DBMS, EAI (Enterprise Application Integration) servers, ETL (Extraction Transformation Loading) tools, and WfMS (Workflow Management Systems). However, the boundaries between these different classes of systems begin to blur due to overlapping functionalities of concrete products. Major problems in this context are posed by the high development effort for integration task specification, the low degree of portability between those integration systems, and the possible inefficiency. The inefficiency problem (optimization potential) is caused by system-inherent assumptions about the primary application context. If the actual workload characteristics (process types, data

size) differ from those assumptions, the execution performance can be significantly improved by changing the used integration system.

Our main hypotheses are (1) that a typical IT infrastructure comprises multiple integration systems with overlapping functionalities, and (2) that we can generate platform-specific integration task specifications from platform-independent models. The opportunities arising from these hypotheses led us to our idea of *invisible deployment*. Here, a user models an integration process in a platform-independent way and deploys it using a central deployment interface. Now, there is the general possibility to decide on the optimal integration platform to execute the specified integration process. This decision should consider workload execution statistics in order to be based on objective online statistics with regard to changing workload characteristics. Clearly, this general idea can overcome the problems of high development effort and low portability (generation of process descriptions) as well as inefficiency (optimality decision, load balancing), but it comes with several inherent challenges.

In order to overcome the described problems and to convey the core idea of *invisible deployment*, in this paper, we make the following contributions.

- In Section 2, we introduce the vision of *invisible deployment* and describe the major challenges that arise when realizing that vision.
- Subsequently, in Section 3, we describe our approach for the deployment of integration processes. Here, we focus on the selected aspects of integration process generation and functional candidate set determination.
- Furthermore, in Section 4, we discuss a possible runtime approach, where we investigate cost modeling and cost normalization, optimality decisions and the heterogeneous load balancing.
- Based on the proposed solution, in Section 5, we present a system architecture for the realization of *invisible deployment* in the context of integration platforms.

Finally, we survey related work in Section 6 and conclude the paper in Section 7.

2 Vision Overview

Based on the described problems, in this section, we pose our main hypotheses and present the resulting conceptual architecture for the vision of *invisible deployment*. Additionally, we point out the major challenges that arise here.

2.1 Assumptions and Hypotheses

In fact, our vision is based on empirically evaluated assumptions. Here, we conclude the following two hypotheses.

Hypothesis 1 Model-Driven Generation: *Integration processes can be modeled in a platform-independent way. Based on those models, we can generate proprietary (platform-specific) integration task specifications for concrete integration platforms.*

Hypothesis 2 Optimality Decision: *A typical IT infrastructure comprises multiple integration systems with overlapping functionalities. Hence, there is the possibility to decide on the optimal integration system based on functional and non-functional properties according to given integration processes.*

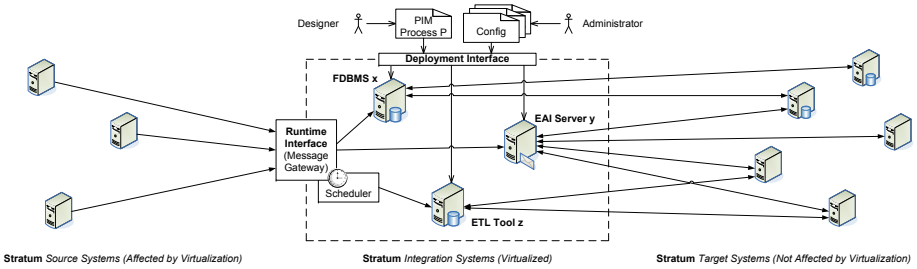

Fig. 1. Vision of *Invisible Deployment*

Here, the model-driven generation of integration processes is the precondition for the *invisible deployment*. It has been shown in several projects (GCIP project [1, 2], Orchid project [3], and ETL process management [4]) that process generation can be done using concepts from the model-driven software development. Furthermore, a typical IT infrastructure comprises multiple integration systems with overlapping functionalities [5] such as specific operators, supported external systems, possibilities to react to external events or transactional functionalities. Hence, an integration processes can be executed using different integration systems, without changing the external behavior. Thus, the decision on the optimal integration system can be made based on non-functional properties such as efficiency, scalability and resource consumption.

2.2 Conceptual Architecture

If there are multiple integration systems with overlapping functionalities, we can decide on the optimal system for given integration processes. Usually, this decision is made based on subjective experience and certain workload assumptions. Hence, there is the need for a conceptual architecture that allows for the objective optimality decision as well as for transparency (hiding) of the used integration system. Clearly, this is a virtualization approach. In contrast to typical virtualization, where multiple logical systems are mapped to one physical system, the *invisible deployment* addresses the inverse problem, where one logical system has to be mapped to multiple physical systems. In fact, this is similar to shared-nothing architectures, where only meta data (about the distribution) has to be centrally maintained.

Figure 1 shows the conceptual architecture for our vision of *invisible deployment*. Here, we have to distinguish three strata: the stratum of source systems, the stratum of integration systems and the stratum of target systems. Clearly, a source system can be a target system at the same time. Furthermore, we have two major types of integration processes that are deployed into and executed by the integration systems. First, there are data-driven integration processes where instances are initiated by messages sent from the source systems to the integration systems (synchronous as well as asynchronous execution model). Second, there are scheduled integration processes that are initiated by an internal time-based scheduler of the integration system.

The vision of *invisible deployment* describes the transparent deployment of integration processes into a logical integration system that consists of a set of heterogeneous physical integration systems. In order to realize this kind of transparency, we

need to focus on deploytime and runtime challenges. Assume that the designer has modeled integration processes in a platform-independent way (PIM); then we need to determine the subset of integration systems that can realize the modeled integration process. Further, we need to generate platform-specific integration tasks based on the platform-independent model. Those aspects are realized by the deployment interface. In order to allow for transparency, we now focus on the runtime requirements. Here, we must provide a runtime interface for the source systems. Thus, we do not specify the physical integration systems as target systems for message propagations but we specify the single logical integration system. Furthermore, the transparency for external target systems is pretty simple because here, we only need to distribute the configurations to all used physical integration systems. This runtime interface and a central scheduler for all physical integration systems allow for the optimality decision on used integration systems. Therefore, we need to monitor execution statistics, normalize those costs into a platform-independent cost model and allow for cost estimation over heterogeneous physical integration systems. In fact, this opens the opportunity for optimization approaches such as dynamic optimality decisions and heterogeneous load balancing.

2.3 Problems and Challenges

After having introduced the conceptual architecture, we now want to focus on the major challenges that arise with regard to process deployment and runtime scheduling.

Deployment Challenges The deployment challenges address the generation of platform-specific integration tasks and the deployment into the integration systems.

Challenge 1 Generation of Integration Processes: *The precondition for the* invisible deployment *is the platform-independent modeling of integration processes and the generation of platform-specific integration tasks. Especially, the bi-directional model transformation without information loss poses a fundamental challenge.*

Challenge 2 Functional Candidate Set Determination: *Based on Challenge 1, there is the need to evaluate whether or not a given integration process can be executed with a specific integration system. Here, we need to derive functional requirements from the integration process and match those with feature sets of the specific integration systems.*

Challenge 3 Configuration Management: *Due to the virtualization of multiple heterogeneous physical integration systems (each with its own configuration), also the platform-independent configuration management is a tough challenge. Here, the specification of transactional behavior and the adapter/wrapper configurations are important. In particular, this is required for the application in practice.*

Challenge 4 Reliability: *On top of candidate set determination and configuration management, there is the need to ensure functional (semantical) correctness. Hence, model checking techniques must be investigated in order to prove the conformance of certain integration system configurations with the semantics of specified integration processes.*

Runtime Challenges In addition to the deployment challenges, we further see the following runtime challenges. In particular, the platform-independent cost modeling as well as the serialization and transactional behavior are important.

Challenge 5 Cost Modeling and Cost Normalization: *In order to adapt to changing workload characteristics and allow for comparison over heterogeneous integration systems, we need to monitor execution statistics. There is a need for a platform-independent cost model and cost normalization approaches to allow for comparability.*

Challenge 6 Optimality Decision: *Based on the comparability of integration systems, we will be able to decide on the optimal integration system for a given integration process. Here, a periodical re-decision seems to be advantageous. In fact, such a decision can be made for one given process, for a set of k given processes or for arbitrary subgraphs of processes. Clearly, this is a challenging combinatoric problem.*

Challenge 7 Heterogeneous Load Balancing: *The heterogeneous load balancing extends the challenge of the optimality decision with the aim of an optimal utilization of all physical integration systems rather than only finding the optimal integration system. Here, different optimization objectives are present.*

Challenge 8 Serialization and Transactional Behavior: *Although the heterogeneous load balancing has high optimization potential, it poses a problem of serialization and transactional behavior. If a sequence of two messages is forwarded to different physical integration systems, we need to serialize those process executions in order to prevent the message outrun (avoid changing external behavior).*

In order to be concise, we try to highlight core ideas of possible solutions for a selected subset of those challenges in the following two sections.

3 Deployment

In this section, we want to explain possible solutions for selected deployment challenges. In fact, the generation of integration processes and the functional candidate set determination are the most important deployment aspects.

3.1 Integration Process Generation

The generation of integration processes has been investigated intensively. The GCIP project (Generation of Complex Integration Processes) [1, 2] and the correlated GCIP Framework allow for the platform-independent modeling of integration processes as well as the generation and optimization of platform-specific integration tasks for concrete integration systems. Figure 2 illustrates the general GCIP approach as well as its current project state. In general, the generation framework comprises the four layers of certain platform-independent models (PIM), a central abstract platform-specific model (A-PSM), platform-specific models (PSM) and the declarative process descriptions (DPD). Currently, the framework supports five concrete integration systems (for the types FDBMS, ETL and EAI) as target of our generation.

On a platform-independent level, integration processes can be modeled with different languages, such as UML (Unified Modeling Language) activity diagrams or BPMN (Business Process Modeling Notation) process specifications. In fact, those specifications are directed graphs that can be annotated in order to provide details and other parameters. Finally, those models can be imported and transformed to an abstract platform-specific model. In the case of UML, XMI documents are imported, while in the case of BPMN, we can use XPDL as well as WSBPEL specifications, respectively.

In contrast to typical model-driven approaches, a central abstract platform-specific model has been introduced, where the *Message Transformation Model* [6] and its defined operators are used. This central model is independent from any integration system type and reduces the transformation complexity between m PIMs and n PSMs from $m \cdot n$ to $m + n$; additionally, it provides the possibility to apply platform-independent optimization techniques.

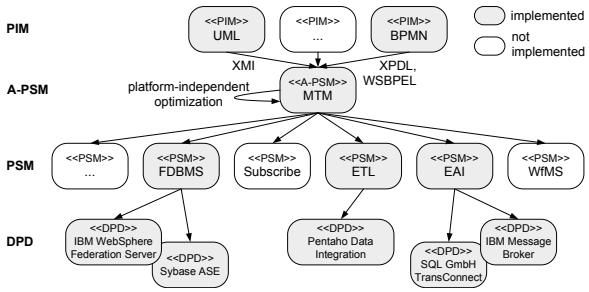

Fig. 2. Overview of GCIP Generation Approach

Based on the A-PSM, platform-specific models can be generated. Those models are specific to the integration system type but not specific to the concrete integration systems. Currently, the groups of FDBMS, EAI and ETL are supported. Here, XML representations are used for those internal models.

Finally, declarative process descriptions are generated from the single platform-specific models. In the model-driven architecture, this is called the `code` layer. However, we explicitly separate this from the internal code layer of the integration systems. Hence, we use the name DPD. As an example, we can generate stored procedures (scheduled integration processes) or triggers (data propagations) of different SQL dialects. Further, also EAI processes (message flows) and ETL jobs can be generated.

In conclusion, integration tasks for concrete integration systems can be generated based on platform-independent specifications. For *invisible deployment*, this is the foundation for all other subsequent challenges. In fact, it has been shown that this is realizable for fully different types of integration systems.

3.2 Candidate Set Determination

Based on the generation of integration processes, we need to determine a candidate set in the sense of a subset of integration systems that are able to realize a given integration process P (Challenge 2). Hence, we determine candidates for the optimal integration system. Therefore, a two-phase approach is meaningful. In the first phase, the functional requirements are derived from the given integration process (e.g., the ability to receive messages). Based on this feature set $F(P)$ and the defined feature sets of all supported integration systems $F(S)$, in the second phase, a matching between the single feature sets is computed in order to determine the logical candidate set C of integration systems that can be used to execute P.

Algorithm 1 illustrates the details of the system candidate set determination. First, for each operator o_i with $o_i \in P$, the functional requirements are derived and added to the feature set $F(P)$ (lines 3-5). Then, deployment policies $D(P)$—such as the choice of execution model (synchronous/asynchronous) or certain transactional requirements—are also added to the feature set (lines 6-8). After this first algorithm phase, the second phase realizes the matching of feature sets. Therefore, we iterate over all systems s_i and the single feature sets $F(s_i)$ in order to compare those features with the feature set $F(P)$. If we determine two equal features, we set $fflag$ to true (line 16). Further, if we have determined that there are no equal features, $sflag$ is set to false (line 21) and the current system s_i is not included into the candidate set C. Finally, C contains all systems that fully conform to the required functionalities. Clearly, this algorithm has a worst-case (in the case of $C = S$) complexity of $O(\sum_{i=1}^{|S|}(|F(s_i)| \cdot |F(P)|))$, where $|F(P)| = m + |D(P)|$ and m denotes the number of operators.

Algorithm 1 Candidate Set Determination

Require: process plan P, deployment policy $D(P)$, feature sets $F(S)$, system set S

```
 1: C ← ∅, F(P) ← ∅
 2: // part 1: derive functional properties of P
 3: for i = 1 to |P| do                                   // foreach operator o_i
 4:    F(P) ← F(P) ∪ f(o_i)
 5: end for
 6: for i = 1 to |D(P)| do                                // foreach policy dp_i
 7:    F(P) ← F(P) ∪ f(dp_i)
 8: end for
 9: // part 2: match feature sets
10: for i = 1 to |S| do                                   // foreach system s_i
11:    for j = 1 to |F(s_i)| do                           // foreach system feature f_j
12:       sflag ← true
13:       fflag ← false
14:       for k = 1 to |F(P)| do                         // foreach plan feature f_k
15:          if f_j = f_k then
16:             fflag = true
17:             break 14
18:          end if
19:       end for
20:       if NOT fflag then
21:          sflag = false
22:          break 11
23:       end if
24:    end for
25:    if sflag then
26:       C ← C ∪ s_i
27:    end if
28: end for
29: return C
```

4 Runtime

Aside from the deployment aspects, there are also runtime challenges, and we want to use this section to explain possible solutions for them. Here, we discuss the cost modeling as well as static and dynamic optimality decisions.

4.1 Platform-Independent Cost Model and Cost Normalization

In fact, if we have multiple physical candidate integration systems and if we want to decide on the optimal integration system, there is a need for a platform-independent cost model as well as algorithms for the cost normalization of monitored statistics into that model. We can only normalize statistics from platform-specific models into the platform-independent model but not vice versa because there is an infinite number of denormalized forms of one normalized form. Actually, our cost model contains two types of costs: the abstract cost $C(P)$ that is defined by cardinality formulas as well as the weighted cost $C'(P)$ that is computed by $C'(P) = \frac{t_e(P)}{C(P)}$ as the ratio of execution time $t_e(P)$ and abstract cost $C(P)$. Hence, we can overcome the problem of possible different hardware and disjoint process instances in the sense of computing tuple rates. In fact, we only need to monitor and normalize cardinality and execution time statistics at the operator granularity.

Figure 3 illustrates the general concept of statistic extraction and its normalization into the described platform-independent model. Execution statistics (cardinalities and execution times) are extracted using the system-specific statistic APIs of the physical integration systems. Those statistics are annotated at PSM level and finally mapped to the platform-independent cost model. This mapping comprises the cost normalization. Here, we use three algorithms in order to overcome the sub-challenges of cost normalization. The *base normalization algorithm* overcomes the problems of parallelism of process instances, different resource utilization and different execution models by computing normalized part times (execution time, number of parallel instances, effective and maximal resource allocation). Further, the *semantic transformation algorithm* overcomes the problem of different semantics of extracted statistics. Here, we need to be aware of 1:1, 1:N, N:1 and N:M mappings between platform-specific and platform-independent operators. Obviously, for 1:N mappings and N:M mappings, we cannot aggregate the measured statistics and hence, the result is missing statistics for parts of the process. Finally, there is the *statistical correction algorithm*. It overcomes the problems of inconsistent statistics and missing statistics by checking operator sequences as well as computing missing statistics by linear extrapolation.

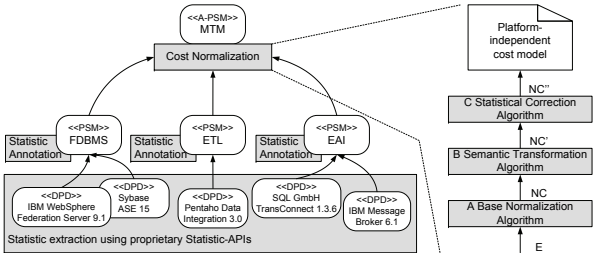

Fig. 3. Cost Normalization Overview

Finally, we can use the platform-independent cost model as well as the normalized statistics in order to decide on the optimal integration system in a cost-based fashion (aware of workload characteristics).

4.2 Optimality Decision

The optimality decision addresses the static decision on the optimal integration system, similar to an advisor decision. In conclusion of such a decision, a given integration process should be executed with this integration system.

In order to do so, we need to deploy the integration process P into all candidate integration systems s_i with $s_i \in C$. Then, we require several reference runs of P on each of those systems in order to gather statistics. Subsequently, we can normalize the statistics as described and finally, we decide on the optimal integration system. In order to adapt to changing workload characteristics, we need to execute those reference runs periodically. In fact, the application areas for such a decision are mainly scheduled integration processes or static decisions for data propagations.

The most obvious problem in that context is the optimality decision on exactly one integration process P. If we generalize this problem, we need to decide on a set of integration processes $k \cdot P$. Clearly, here we can choose one integration system for all k integration processes (trade-off between different processes) or use the simple decision problem for each of those processes. Furthermore, we can also consider different combinations of subgraphs of all integration processes. Clearly, we get an exponential number of alternative distributions to decide on.

Obviously, the major problem of this static optimality decision is that we do not utilize all resources (hardware in the case of physically separated systems). In fact, we always use the optimal integration system (which may change over time), but we do not use multiple systems at the same time.

4.3 Heterogeneous Load Balancing

In order to overcome the previously mentioned problem of suboptimal resource utilization, we introduce the concept of heterogeneous load balancing over multiple heterogeneous physical integration systems. Hence, this results in a dynamic and continuous optimality decision. Here, we preferably use the optimal integration system but the other candidate systems s_i with $s_i \in C$ may be used as well. Therefore, the optimality decision is changed from a deployment approach (periodically re-executed) to a dynamic runtime approach where the optimality decision must be made continuously.

In this context, the major problem is the assurance of serialization and transactional behavior with respect to the serialization according to the incoming message order and the observable external behavior. Assume, for instance, a process type P (with process instances p_i) that executes messages in sequential order; thus, we have to ensure the serialized order of $end(p_i) \leq start(P_{i+1})$. If we distribute those messages to two different integration systems that use the asynchronous execution model, the anomaly problem of message outrun [7] can occur. If the integration systems use the synchronous execution model, the serialization of process instances is simple but inefficient because—due to scheduling overhead—the resource utilization of the overall architecture is even worse than for the static optimality decision.

Our general solution for this heterogeneous load balancing problem is the distribution of fully disjoint message sequences according to their correlated integration processes across multiple physical integration systems. Therefore, we need to predict the costs in a platform-independent manner once more but now, we also need to predict the future workload and we must solve the balancing problem. This problem comprises the search for the optimal distribution of k integration process types across $|C|$ integration systems s_i such that the globally optimal solution is used (with regard to the optimization objectives (1) throughput maximization, (2) latency minimization or (3) load balance maximization). Clearly, we can extend this to a more fine-grained decision model, where subgraphs (similar to the challenge of optimality decision) are distributed across the integration systems.

If the workload changes over time, we need to exchange the execution context between integration systems. Hence, our optimality decision must be aware of the costs that are necessary for switching integration systems (due to synchronization efforts).

5 System Architecture

Figure 4 illustrates an architecture realizing the vision of *invisible deployment*. Basically, the message propagation is supported by an execution interface. Further, the integration task specifications $PD_i(x)$ (process types, configurations) are also possible using a deployment interface. For deployment purposes, process transformers as well as process deployers are required to generate platform-specific models and to deploy those into the different integration systems used. Furthermore, an `Optimizer` component for rewriting is needed. Deployed processes are registered within a central `Repository`. Here, all types of systems, except for client systems, as well as the time schedules are managed. Concerning the execution of the integration tasks, the synchronous events are directly forwarded to the `Runtime Environment`, while asynchronous events are appended to a specific `Request Queue`. Independent from this, the `Scheduler` also invokes the `Runtime Environment` directly, based only on the defined time schedules. Within the `Core Execution Service`, the integration task is split into subtasks.

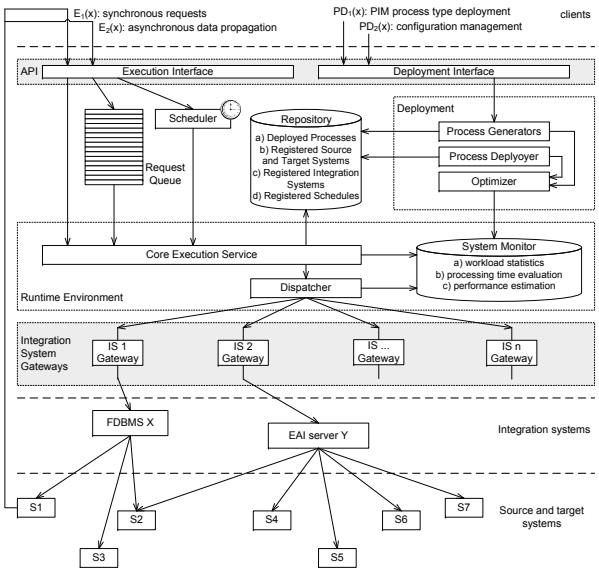

Fig. 4. Hybrid Gateway EIP Micro-Architecture

Here, the transactional behavior is ensured as well. The `Dispatcher` decides about the optimal integration system for each integration task and invokes the registered integration systems via `IS Gateways`. These decisions are based on functional as well as non-functional properties, including monitored workload characteristics. Finally, such an environment provides a central deployment of integration tasks, using a distributed system infrastructure (of integration systems) during execution. This is the core requirement of a service-oriented architecture (SOA). Finally, this type of gateway integration system can realize the *invisible deployment* in a transparent manner.

6 Related Work

In fact, the *invisible deployment* is a novel vision and no comparable work exists. Here, we want to survey application areas as well as correlated virtualization approaches.

6.1 Application Areas

In the context of the generation and deployment of integration processes, we need to emphasize three projects and approaches, respectively.

The Orchid project [3] addresses the generation of ETL jobs based on declarative mapping specifications. There, a so-called Operator Hub Model (OHM) is used in order to transform the execution semantics of ETL jobs into a platform-independent form. The Orchid project is restricted to the generation of ETL processes for IBM ETL tools. In general, an extension to vendor-independent semantics seems to be possible without conceptual problems. While Orchid addresses only the generation of ETL jobs, the approach presented in [4] focuses on the combination of ETL process generation and model management. There, the authors presented platform-independent operators for the deployment of ETL processes. In contrast to those two approaches, the GCIP (Generation of Complex Integration Processes) Framework focuses on the modeling of platform-independent integration processes [1], the generation for numerous different integration system types (such as FDBMS, EAI servers and ETL tools) as well as the application of optimization techniques [2, 8] during model-driven generation.

The major similarity of those three projects is the possibility of deciding on the optimal target integration system to use (target of the generation). Hence, the vision of *invisible deployment* can be applied to all of those approaches.

6.2 Virtualization Approaches

Clearly, the *invisible deployment* is a virtualization approach. In the context of *software as a service*, in particular, the database virtualization seems advantageous. Here, several approaches exist for the realization of multi-tenant databases [9, 10] where multiple logical databases are maintained within one physical database. Obviously, there are more general approaches for multi-tenant software [11] as well as for IT service provision [12]. By now, the famous term *cloud computing* [13] has been established for a superset of those virtualization approaches. Even more, there is an approach [14] on how to provide EAI as a service. Those approaches virtualize multiple logical systems into

one single physical system. In contrast to this, according to the scalability terminology [15], we virtualize one logical integration system into a farm of heterogeneous, physical integration systems.

7 Summary

To summarize, in this paper, we introduced our novel vision of *invisible deployment* that is applicable in many different areas and that exhibits a high optimization potential as well as numerous challenging research aspects. In general, the *invisible deployment* is based on the hypothesis that a typical IT infrastructure comprises multiple integration systems with overlapping functionalities. Hence, the core idea is to virtualize a number of heterogeneous physical integration systems by one logical integration system.

Here, we identified the main challenges and explained the conceptual overall architecture. Subsequently, we provided details on specific aspects of that vision and we described a system architecture to realize our vision. However, there are lots of open research aspects and huge optimization potential; hence, further detailed investigation is absolutely required. In conclusion, the major problems in the area of integration processes (the high development effort, the low degree of portability, and the inefficiency) can be overcome by the general concept of *invisible deployment*.

References

1. Boehm, M., Habich, D., Lehner, W., Wloka, U.: Model-driven development of complex and data-intensive integration processes. In: MBSDI. (2008)
2. Boehm, M., Wloka, U., Habich, D., Lehner, W.: Model-driven generation and optimization of complex integration processes. In: ICEIS (1). (2008)
3. Dessloch, S., Hernández, M.A., Wisnesky, R., Radwan, A., Zhou, J.: Orchid: Integrating schema mapping and etl. In: ICDE. (2008)
4. Albrecht, A., Naumann, F.: Managing etl processes. In: NTII. (2008)
5. Stonebraker, M.: Too much middleware. SIGMOD Record **31**(1) (2002)
6. Boehm, M., Habich, D., Wloka, U., Bittner, J., Lehner, W.: Towards self-optimization of message transformation processes. In: ADBIS. (2007)
7. Boehm, M., Habich, D., Lehner, W., Wloka, U.: An advanced transaction model for recovery processing of integration processes. In: ADBIS. (2008)
8. Boehm, M., Habich, D., Lehner, W., Wloka, U.: Workload-based optimization of integration processes. In: CIKM. (2008)
9. Aulbach, S., Grust, T., Jacobs, D., Kemper, A., Rittinger, J.: Multi-tenant databases for software as a service: schema-mapping techniques. In: SIGMOD. (2008)
10. Jacobs, D., Aulbach, S.: Ruminations on multi-tenant databases. In: BTW. (2007)
11. Tsai, C.H., Ruan, Y., Sahu, S., Shaikh, A., Shin, K.G.: Virtualization-based techniques for enabling multi-tenant management tools. In: DSOM. (2007)
12. Shwartz, L., Ayachitula, N., Buco, M.J., Grabarnik, G., Surendra, M., Ward, C., Weinberger, S.: It service provider's multi-customer and multi-tenant environments. In: CEC/EEE. (2007)
13. Ramakrishnan, R.: Cloud computing - was thomas watson right after all? In: ICDE. (2008)
14. Scheibler, T., Mietzner, R., Leymann, F.: EAI as a Service - Combining the Power of Executable EAI Patterns and SaaS. In: EDOC. (2008)
15. Devlin, B., Gray, J., Laing, B., Spix, G.: Scalability terminology: Farms, clones, partitions, packs, racs and raps. CoRR **cs.AR/9912010** (1999)