Towards Self-Optimization of Message Transformation Processes

Matthias Böhm[1,2,3], Dirk Habich[2], Uwe Wloka[3], Jürgen Bittner[1], and Wolfgang Lehner[2]

[1] SQL GmbH Dresden, Germany
[2] Dresden University of Technology, Database Technology Group
[3] University of Applied Sciences Dresden, Database Group

Abstract. The Message Transformation Model (MTM), for modeling complex message transformation processes in data centric application scenarios, provides strong capabilities for describing the data and control flow, transactional behavior and even the interaction with external systems. Thus, this general model could be used for different integration platforms like EAI-Servers, Message-Brokers and Subscription-Systems, as well as for ETL-Tools. In this paper, we describe self-optimization strategies for MTM processes to determine an optimal executable process. Our proposed strategies can be distinguished into rule-based and workload-based techniques. Aside from theoretical consideration, we describe implementation aspects within the integration platform TransConnect®. Furthermore, we present some first evaluation results.

1 Introduction

The number of heterogeneous information systems within business processes is continuously increasing. This offers the problem of many different data representations and data schemes. A widespread integration approach of such information system is the horizontal integration by message based communication. The advantage of this strategy is to ensure an adequate loosely coupling of participating systems and applications.

As described in [1], the conceptual modeling of such complex message transformation processes (also called integration processes), could only be realized adequately as a set of control flow, data flow, transactional, and even interaction oriented process steps. Because of the absence of a generally accepted model, we defined the Message Transformation Model (MTM) [1]. This MTM includes a conceptual message model with the intent of a conceptual data independence and a conceptual process model. The proposed approach realizes the independence of concrete process description languages.

In this paper, we focus on self-optimization of MTM processes caused by several facts: (1) suboptimal modeled processes by the process designer resulting from missing knowledge about internal processing concepts, (2) missing information about the dynamic workload characteristics of incoming messages during modeling time and (3) total costs of ownership to maintain an integration

Y. Ioannidis, B. Novikov and B. Rachev (Eds.): Local Proceedings of ADBIS 2007, pp. 116-125

system. Therefore, self-optimization of message transformation processes is an interesting and complex field for itself. The contribution of this paper is a first step towards self-optimization of processes. Fundamentally, our proposed strategies can be distinguished into rule-based and workload-based techniques. These strategies are implemented within the integration platform TransConnect®.

For the sake of a better description, the specification of the techniques are illustrated on a complex example process. Figure 1 shows this suboptimal modeled integration process. A SAP IDOC [2] "ORDERS05" document is received by a SAP Inbound Adapter and will be translated into an internal XML representation. This internal message is passed to a `SWITCH` node. Within this part, the message is translated to a specific schema of a CRM database and in two cases enriched with additional data. Afterwards, the message is transformed to the TPC-H [3] schema. The next steps are schema validation and the construction of a savepoint. After this steps, the current message is passed to a subprocess synchronously. This subprocess distributes the message to multiple systems by starting three concurrent flows. The presented process is suboptimal with regard to the evaluation of the `SWITCH` conditions, redundant query preparations, redundant control flows, worse modeled data translations, redundant validations and also the subprocess invocation. However, such situations are found in real-world modeled processes.

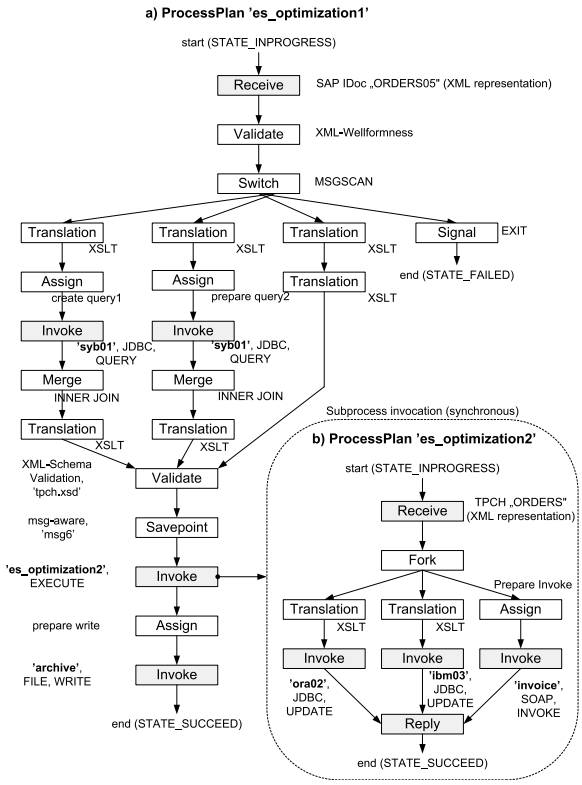

Fig. 1. Unoptimized Example Process

Therefore, this paper makes a clear contribution towards self-optimization of message-based application integration processes. Before we describe our optimization strategies in more detail in Section 4, related work and the integration platform TransConnect® are presented in Section 2 and 3. The benefit of our proposed strategies are demonstrated based on performance measurements comparing not optimized and optimized process executions in Section 5. The paper closes with an conclusion and further research aspects.

2 Related Work

Basically, functional oriented process description languages, like WSBPEL [4], and their deficits concerning the modeling of the data flow, but also the absence of a fully accepted conceptual model for data intensive integration processes, are the main motivation for the previous definition of the Message Transformation Model (MTM) [1]. This model have strong capabilities describing the control flow as well as the data flow and thus contributes to the systematic modeling for complex message transformation. However, their are other work, like the BPEL extension II4BPEL [5], the Enterprise Integration Patterns [6] and the IBM ESB Mediation Patterns [7], which are also contribute to this topic, but on an other level of abstraction. As previously mentioned, there are also plenty publications concerning data streaming operator optimization [8], [9], workflow optimization [10] and optimization of webservice composition/invocation in datacentric processes [11]. Furthermore, the approach of self*-techniques are currently discussed widely. Representative for these works, the "IBM MAPE concept" [12] and the "soft index creation" [13] should be mentioned.

3 Integration Platform TransConnect®

The mentioned real-life integration platform TransConnect® comprises several subprojects, including the `Server`. It offers a set of `Inbound Adapter`, which listen passively for incoming messages, translates them into the internal format and forwards them to the `Dispatcher`. In case of asynchronous processing, the message stream pass the `MessagePool`. The `Dispatcher` routes the incoming messages, asynchronous to `MessageQueues`, respectively synchronous, directly to the `WorkflowProcessEngine` (WFPE). Beside the event model of incoming messages, also time-based events, initiated by the `Scheduler`, could be applied. The

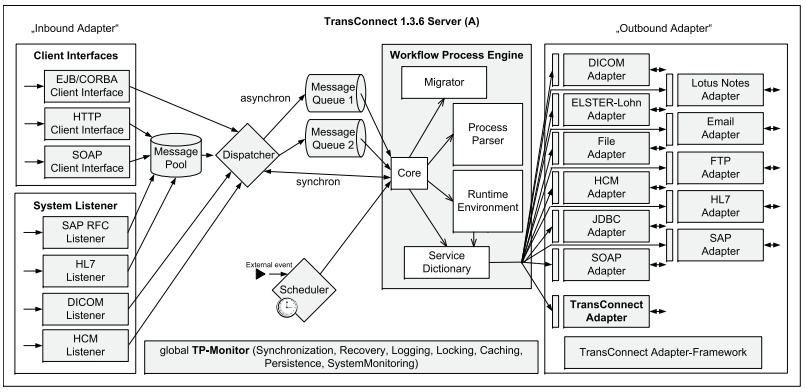

Fig. 2. Coarse-Grained Server Architecture

`WFPE` allows the translation of process descriptions into an executable form and manages its processing. The translation is realized by the `ProcessParser`, which is stratified into the four logical layers: *External Mapping (1)*, *Internal Analysis and Optimization (2)*, *ProcessPlan Generation (3)* and *ProcessPlan Instantiation (4)*. The processing includes the invocation of services in form of `Outbound Adapters`, local services and other process types. The `Outbound Adapters` allow for active reading interactions (pull) as well as writing interactions (push) with external systems. Furthermore, the global `TP-Monitor` should mentioned as a component for persistence, caching, configuration, resource management, transactional behavior and even system auditing.

The workload-based process optimization, realized within the mentioned `ProcessParser`, is generally based on the following universal cost model. In this context, costs are used as an expression for the CPU-usage, the Main-Memory-usage and especially the normalized processing time. Thereby, we focus a cost segmentation into three cost categories.

Definition 1. *Let C(p) be the costs of a process p, we define, that this costs are composed with* $C(p) = a * b * (C_c(p) + C_p(p) + C_m(p))$ *of the communication costs* $C_c(p)$*, the processing cost* $C_p(p)$ *and the internal management costs* $C_m(p)$*, depending on the message size with* $a = sizeof(m)$ *and the hierarchical deepness of the message with* $b = dim(m)$*.*

Definition 2. *Let* $C_c(p)$ *be the communication costs, we define, that these are composed with* $C_c(p) = \Sigma C(ino_k) + \Sigma C(s_l)$ *of the costs for all interaction-oriented operators ino and of the execution costs for the external systems s.*

Definition 3. *Let* $C_p(p)$ *be the processing costs, we define, that these are composed with* $C_p(p) = \Sigma C_{cno_{ia}}() + \Sigma C_{dno_{ib}}() + \Sigma C_{tno_{ic}}()$ *of the costs for controlflow-oriented operators cno, dataflow-oriented operators dno and transaction-oriented operators tno.*

Definition 4. *Let* $C_m(p)$ *be the internal management cost, we define, that these are composed with* $C_m(p) = C_q(m, count(M)) + C_{pi}(p, count(M)) + C_{tx}(undo, redo, tid)$ *of the costs for transactional message queuing* C_q*, for process instances management* C_{pi} *and for the transactional recoverability* C_{tx}*.*

Statistics are required for an adequate workload-based cost estimation. These are collected using the following monitoring concepts. First, for *implicit performance propagation*, the approach of monitor events is used. Thus, an abstract node comprises the publishing of these monitor events to the `SystemMonitor`. Second, a *transparent monitor selection* between the `MONITOR_LEVELS` "EmptyMonitor", "TransientMonitor" and "PersistentMonitor" is supported. Third, in case of the mentioned "PersistentMonitor", *prepared analysis queries* are available to efficiently join the monitor statistics with the metadata repository. Fourth, the *Self-Optimization trigger mechanism*, similar to control loops, should be mentioned. It is realized in accordance to the IBM MAPE-concept (Monitor, Analysis, Plan, Execute). After the `ANALYSIS_INTERVALL` is reached an asynchronous background analysis and recompilation is started. Thereby, the runtime environment is only locked during the exchange of processplans.

4 Process Optimization Techniques

Having introduced the integration platform TransConnect® and its execution concept in short, we now want to illustrate several optimization techniques and explain selected ones in detail. Thereby, these techniques could be distinguished into rule-based and workload-based approaches.

4.1 Rule-Based Process Optimization

The rule-based optimization of MTM-processes was designed in analogy to the static analysis of procedural language compilers. Thus, some techniques could be adopted and others are quite specific to transformation processes.

Controlflow Optimizations: The identification of *Redundant Control Flows*, which occurs mainly in concurrent subgraphs, is realized with a simple pattern matching. So all concurrent subgraphs following a `FORK` operator, are compared, whether they have particular the same parameterization. As shown in Figure 3, the `TRANSLATION` operators of the first two subgraphs are redundant, cause of the same input message and the same XSLT stylesheet, and thus could be optimized. If not all forklanes could be optimized symmetric, a further `FORK` operator has to be inserted, in order to ensure the semantic correctness after process rewriting. Finally, some parameters of the following operators, like the input message of the `INVOKE` operators, have to be updated.

The identification of *Unreachable Subgraphs* is much more complex. Basically, there are two groups of such unreachable subgraphs. The first group is identified by searching for operators following terminating operators, like the `SIGNAL` or `REPLY` operator. In contrast to that, the second group comprises unreachable `SWITCH`-paths and operators following an endless iteration. In order to determine anomalies of the second group the given XPath-conditions have to be evaluated. For example, in order to identify an unreachable `SWITCH`-path, conditions have to be found, which are impossible by itself or already be included in a previous condition. Finally, an algorithm is executed, deleting all identified subgraphs.

Although, the *Local Subprocess Invocation* is necessary, it would be more efficient to realize an inline compilation. The performance overhead is caused by additional internal management costs, invoking the subprocess. Also, there are additional processing cost, caused by the used `RECEIVE` and `REPLY` operators.

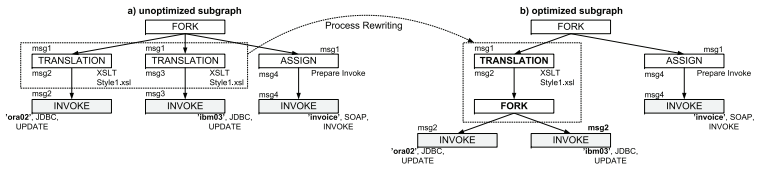

Fig. 3. Optimizing Redundant Concurrent Control Flows

During the recursive process of inline compilation all operators of the subprocess, except for leading `RECEIVE` and ending `REPLY` operators, are inserted in the main process.

Dataflow Optimizations Also the dataflow optimization uses a pattern matching concept. The following enumeration shows the main dataflow anomalies, which should be identified.

- *Double Variable Assignments:* If a message gets an assignment to the same part multiple times, the first assignment could be removed.
- *Unnecessary Variable Assignment:* All assignments, which are never used in the following processing could also be removed.
- *Unnecessary Variable Declaration:* If a variable is never used, the whole declaration for this variable could be removed.
- *Two sibling TRANSLATION operators:* Two sibling `TRANSLATION` operators (type XSLT) could be aggregated to one operator, processing a Two-Phase-Transformation.
- *Unnecessary SWITCH-paths:* If a `SWITCH` operator, comprises disjoint `SWITCH`-paths, each path containing less than one operator could be removed.
- *Two sibling Validations:* `VALIDATE` operators following `TRANSLATION` or other `VALIDATE` operators should be removed.

The whole rule-based optimization-process has to be executed, until no more process type changes could be determined, cause one optimization rule could produce further anomalies. Although, the rule-based optimization lead to a high performance improvement, there are open issues. For example the determination of unnecessary `TRANSLATION` operators, as well as the fine-grained optimization of transformation stylesheets within the process context is a big challenge.

4.2 Workload-Based Process Optimization

The monitored processing statistics are used to determine several workload characteristics. Basically there are two groups of workload-based optimizations. First, implicit optimizations, like the message indexing, are realized. Second also explicit optimizations, whereby the processplan is rewritten, are focused.

Message Indexing: The message indexing taking place in an implicit manner, in order to make the processplans most robust, by preventing processplan rewriting. Furthermore, we only use transient indexes. Thus, after server restart the indexes are empty.

Access to extracted single values: Because of the dynamic data aspect, it is not useful to manage messages in a fine-grained and attribute-oriented manner. Thus, accessing single values of one message, e.g. within the `ASSIGN` or `SWITCH` operator, results in cost-intensive message scans. We prevent this, using more efficient index scans. Single values, identified by the message ID and a XPath

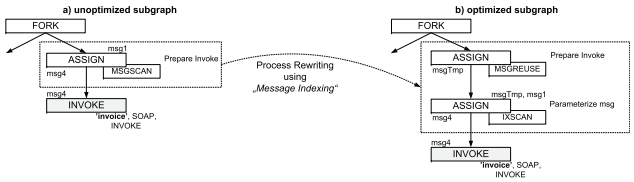

Fig. 4. Splitting the `ASSIGN` Operator for Partial Indexed Message Reuse

expression, are extracted at the `Inbound Adapter` side. These values are put into the 2-layer-hash-index. After a message was successfully processed, the whole index subtree for this message could be removed. The optimizer determines, whether the single value extraction is optimal for a specific usecase.

Reuse of static messages: The `ASSIGN` operator could be used for assignment of static value expressions to messages. If all copy elements in one `ASSIGN` operator are such static value expressions, the produced message could be reused across instances of one process type. This is realized by an index structure for complete message objects, identified by the process type ID, the node ID of the `ASSIGN` operator and the message name. This index is never cleaned during server runtime. Furthermore, the `ASSIGN` operator often uses both, static value assignments, as well as message value assignments. Figure 4 shows an optimization approach of splitting such an operator into two operators, separating message reuse and value assignments. Our process optimizer realize this at every possible position.

Controlflow Optimizations The workload-based controlflow optimization is reached by explicit rewriting of processplans. As an example for such optimizations, the `FORK` operator optimization will be discussed. By definition, the `FORK` operator starts several subprocesses in sequence, execute them concurrently, and finishes, when all subprocesses were executed. Because of the sequenced start of these subprocesses, the first forklane starts earlier than the second. Thus, it is most optimal to start the subprocess, with the highest processing time, first, in order to prevent waiting time of subprocesses. Therefore, the average normalized processing time of all subprocesses is evaluated and the subprocesses are ordered ascending by its average. The performance impact, reached by this control flow optimization, depends on the platform-specific costs for thread management.

Dataflow Optimizations The workload-based dataflow optimization comprises the optimization of the dataflow-oriented operators of the MTM. Furthermore, also the controlflow-oriented operator `SWITCH`, which evaluates XPath expressions over messages, is under observation. Following, only this `SWITCH` operator will be discussed, in order to explain one operator optimization in depth.

This operator comprises a set of `SWITCH`-paths, where each contains a condition and optionally also an ending `SWITCH`-path, which does not contains a

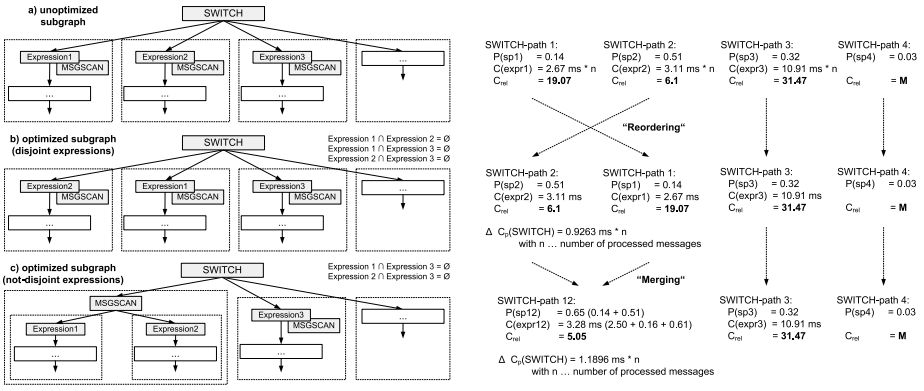

Fig. 5. Example `SWITCH` Operator Optimization

condition. By definition one or none, of these paths is executed. Thus, the problem is the evaluation of the conditions, which are in fact XPath expressions over messages. Because of the sequenced evaluation of these conditions it is suggesting, that the path with the lowest relative costs (costs relative to likelihood) should be executed first, in order to prevent unnecessary evaluations.

Figure 5 illustrates a subgraph of the introduced example, assuming costs and likelihoods. Imagine, the first three `SWITCH`-paths contains disjoint and not-disjoint conditions, while the fourth `SWITCH`-path contains none. With the aim of performance improvement, first the relative costs for each condition is computed. Thereby, end-paths without a condition get an theoretical maximum value M. Second, the paths are ordered ascending by these relative costs.

In order to rearrange the sequence of switch-conditions, the modeled semantic have to be maintained. Basically `BooleanExpressions` are used as `SWITCH`-path-conditions. These expressions contain one operator and two operands, which are represented by `ValueExpressions` or absolute values. Coming back to the maintenance of the process semantic, all disjoint expressions could be rearranged. Thereby, two expressions are disjoint, if they use the "equals" or "not equals" operator or if they use different `ValueExpressions`.

Definition 5. *In order to formalize that problem, we define the disjoint SWITCH operator is optimal in terms, where* $\frac{C(condition_{spath_i})}{P(spath_i)} \leq \frac{C(condition_{spath_{i+1}})}{P(spath_{i+1})}$ *with* $i = 1, \ldots, k$ *and* $k \ldots$ *number of switch paths. Thereby, the performance improvement* $\Delta C = C_{unopt} - C_{opt}$ *reached by rearrangement could be computed with* $C = \sum_{i=1}^{k} \left(P\left(spath_i\right) * \sum_{j=1}^{i} \left(C\left(condition_{spath_j}\right)\right)\right)$*. Furthermore, the determination of the relative costs requires the computation of execution likelihoods. They are computed with* $P_{(spath_i)} = \frac{count(p \in n_{nid\_spath})}{count(p \in n_{nid\_switch})}$*.*

Additionally, also the optimization of the not-disjoint `SWITCH` operator should be mentioned. As Figure 5 illustrates, there is the requirement, that the sequence

of not-disjoint expressions have to be excluded from the rearrangement. Furthermore, the characteristic of the non-disjoint expressions is, that they use the same `ValueExpression` in their `BooleanExpressions`. Thus, it is suggestible to reuse this value for both `BooleanExpressions`. In consequence the not-disjoint expressions are used as a compound `SWITCH`-path in order to rearrange this compound-path with other `SWITCH`-paths, which are disjoint to the compound one. Thus, the sum of all subpaths are used to determine the likelihood and costs for this compound-path.

4.3 Performance Measurements

The performance of the introduced example process was measured using no optimization, only rule-based optimizations, only workload-based optimizations, but also full optimization. Actually, the performance measurements do not comprise all optimization aspects introduced in this paper, cause there are plenty of side effects needed by a full-fledged implementation of these concepts. However, especially the implementation of rule-based optimization is almost realized. Figure 6 shows the average performance values, depending on the number of executed processes and the optimization approaches. This performance values were measured on a desktop pc (Athlon64 1800+, 1GB RAM, Java 1.5 [JVM -Xms300m -Xmx700m]). As Figure 6 shows, especially the rule-based optimization reached an enormous performance improvement. The small impact of the workload-based optimization is cause by the small message size of 7 KB. Thus, the impact of message indexing and of `SWITCH`-path condition evaluation optimizations is small, relative to the overall processing costs. However, there are some incisive results, which should further be discussed. So firstly the difference between the first and second phase should be mentioned, in which performance optimizations of the integrations system as well as of the grounding DBMS and environment taking place. Furthermore, especially on using the message indexing, performance values with a low standard deviation should be noticed. Also the scale factor in form of the number of executed processes should be mentioned. Here, no performance deficit could be measured. Finally, also the fault treatment should taking place. The processing performance was measured, by using the introduced "PersistentMonitor". Thus, the monitoring of performance events, as well as the whole TransConnect `Server` maintenance has influenced the performance results. Further side effects like the index maintenance, which is monitored for the internal management costs and also the communication costs of the external systems, were excluded and thus not analysed.

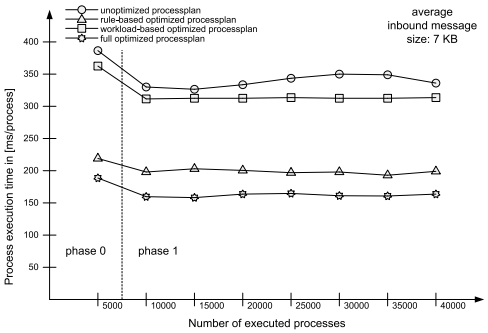

Fig. 6. Measured Performance Values

5 Summary and Conclusion

Basically, this paper has illustrated optimization techniques of message transformation processes, using a self-optimization approach, comprising rule-based optimizations, as well as workload-based optimizations. Furthermore, the theoretical suggestions and the practical performance measurements proved the high optimization potential of such integration processes. Because of the complexity of integration processes, this paper could not claim to discuss all optimization aspects of such processes, but it could give some ideas about the approach of self-optimization in such environments.

Furthermore, some readers would see the similarities to DBMS and DSMS. Thus, there are lots of further research items along with the optimization of MTM processes. These items for instance include, the adoption of adaptivity concepts from the field of data integration.

Finally, it could be stated, that similar to the well-known distributed databases and the currently discussed mash-ups the horizontal integration of systems by integration platforms is also a diffusive way of application design. Thus, also the research of optimization techniques will be displaced from the grounding systems to the integration processes, to which this paper tries to contribute to.

References

1. Böhm, M., Wloka, U., Habich, D., Bittner, J., Lehner, W.: A message transformation model for data centric integration processes. Technical report, University of Applied Sciences Dresden (2007)
2. SAP: SAP Interface Repository. (2006)
3. Transaction Processing Performance Council: TPC-H - ad-hoc, decision support benchmark, Revision 2.3.0. (2005)
4. OASIS: Web Services Business Process Execution Language Version 2.0. (2006)
5. IBM: Information Integration for BPEL on WebSphere Process Server. (2005)
6. Hohpe, G., Woolf, B.: Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions. Addison-Wesley (2004)
7. Hutchison, B., Schmidt, M.T., Wolfson, D., Stockton, M.: Soa programming model for implementing web services, part 4: An introduction to the ibm enterprise service bus. (2005)
8. Viglas, S.D., Naughton, J.F.: Rate-based query optimization for streaming information sources. In: SIGMOD'02. (2002) 37–48
9. Krämer, J., Yang, Y., Cammert, M., Seeger, B., Papadias, D.: Dynamic plan migration for snapshot-equivalent continuous queries in data stream systems. In: ICSNW'07. (2007)
10. Li, H., Zhan, D.: Workflow timed critical path optimization. Nature and Science **3(2)** (2005)
11. Myerson, J.: Work with web services in enterprise-wide soas, part 5: Optimize web service applications with websphere business integration tools. (2005)
12. Kailing, K., L A.: (Challenges and trends in information management)
13. Lühring, M., Sattler, K., Schmidt, K., Schallehn, E.: Autonomous tuning with soft indexes. In: SMDB'07. (2007)