A Message Transformation Model for Data-Centric Service Integration Processes

Matthias Böhm and Uwe Wloka
University of Applied Sciences Dresden
Friedrich-List-Platz 1
01069 Dresden, Germany
{mboehm,wloka}@htw-dresden.de

Jürgen Bittner
SQL GmbH Dresden
Franklinstrasse 25a
01069 Dresden, Germany
juergen.bittner@sql-gmbh.de

Dirk Habich and Wolfgang Lehner
Dresden University of Technology
Nöthnitzer Strasse 46
01187 Dresden, Germany
dbinfo@mail.inf.tu-dresden.de

ABSTRACT
The horizontal integration of systems by message-based communication via middleware products is a widespread method of application integration to ensure an adequate loose coupling of participating systems and applications. For the description of such service integration processes, the use of functionally oriented process description languages, like WSBPEL, is gaining in importance. However, these languages reveal deficits when describing data-centric application scenarios. Due to these deficits and the lack of a model for service integration processes, this paper contributes to the systematic modeling of complex message transformations in data-centric integration processes. The practicability of the model is shown with a prototypical implementation within the service integration platform TransConnect® of the SQL GmbH Dresden.

1. INTRODUCTION
Nowadays, WSBPEL—as a process description language for service orchestration in a Service-Oriented Architecture (SOA)—has become widely accepted within the layer of process integration. Furthermore, it reflects the current state of the art to use similar workflow descriptions in data-centric integration scenarios of message broker systems, EAI (Enterprise Application Integration) servers and ETL (Extraction, Transformation, Loading) tools as well. However, there is a lack of a semantic-based conceptual external view on data-centric service integration processes.

According to [17], it could be stated that in workflow as well as in ETL descriptions, aspects of control flow and data flow modeling are used. The tuple-oriented workflow systems have strong capabilities for the control flow specification but deficits concerning the description of the data flow. In contrast to this, the dataset-oriented ETL tools include broad data flow functionalities but often disregard the control flow. Thus, there is the need for combining the advantages of both execution concepts in one universal model. This is especially required for Message Broker and EAI servers because these work in a tuple-oriented as well as dataset-oriented fashion.

This motivation and the mentioned problems have also significantly influenced the requirements for the extension of TransConnect® as a message-based service integration platform. Hence, there was the need for the definition of a conceptual model and its external description with process description languages in order to reach the highest possible flexibility by modeling service integration processes on the one side and to ensure the independence of concrete process description languages on the other side. In this context we refer to services, in the meaning of interaction types with external system types or subprocesses, respectively.

In accordance with the classification of integration forms, as described in [7], the focus of this work is the definition of a generic conceptual model for service integration process composition and orchestration and the explanation of its implementation. This model is used for the description of service integration processes within the information integration and application integration using methods and standards of the process integration. In order to achieve the suitable loose coupling—preventing the necessity for a global schema or ontology—this model addresses the message-based service integration and thus the physical integration of systems by sending data from system A to system B. In this context, service integration processes are provided as composed services, hiding the distribution and heterogeneity.

Figure 1 shows an example scenario for the approach of physical integration in the context of ETL processes. The problem comprises the extraction of master and movement data from source systems of two independent sales and purchase organizations into one central data warehouse and into the organization-specific physical data marts. Therefore, the example scenario covers tuple-oriented transactions (1, 2), which are initiated by business transactions in the source systems, but also dataset-oriented transactions (3), which a scheduler initiates once per day. Due to the different execution and time models, the use of a “consolidated database” and the division into subprocesses is advantageous.

To summarize, in this paper, we focus on the physical integration. Due to the lack of a conceptual model for data-intensive service integration processes, we first contribute to
the systematic modeling of complex message transformation processes by defining the Message Transformation Model (MTM) in Section 3. Aside from the introduced context of ETL-processes, this model may be used in all application areas of EAI. Second, our contribution also comprises the explanation of the prototypical implementation within the commercial platform (EAI server) TransConnect®. This includes the description of architectural aspects in Section 4 as well as selected processing aspects in Section 5.

2. RELATED WORK

The process description language WSBPEL 2.0 [21] and its deficits concerning the modeling of the data flow could be seen as the starting point of this work. In fact, with HiBPEL [13] there is a proposed extension which addresses similar aims as this paper does, but these suggested enhancements are limited to the interaction with DBMS. However, nowadays, these functionalities could be plugged into the IBM Information Server [15]. Aside from WSBPEL, there are alternative languages which allow the explicit separation of the control flow and the data flow and which are thus assumed to be more suitable. However, after critical consideration, the choice has to be limited to execution layer languages (implementable). The languages XPDL [29] and ebXML BPSS (ebBP) [20], which belong to this layer, also have to be evaluated as inappropriate because of their degree of abstraction and missing language constructs.

Moreover, methods of process integration are also used increasingly in data-centric processes. Although ETL systems, like the Business Objects Data Integrator [4], allow for declarative workflow modeling, the development of proprietary solutions is observable. Within the context of message-based EAI systems, there is no system realizing a sophisticated process orientation. So, for instance, the IBM WebSphere Message Broker [14] should be mentioned as representative for such systems. There, the message routing and processing is also specified with a proprietary user interface rather than with a process model.

Concerning the systematic modeling, it has to be said that no fully specified model for complex message transformation exists as of now. However, there is related work according to [11, 12], which comprises the same problem in the form of "message patterns" and "Enterprise Service Bus (ESB) mediation patterns" but with a higher degree of abstraction. Thus, these patterns are rather general design patterns than concrete implementable operators.

As the first publication of the Message Transformation Model (MTM), [2] should be mentioned. It was published in German and presents an early version of the MTM. Thereby, this work rather addressed the core model definition than the architectural and processing concepts which are included in this paper.

3. MESSAGE TRANSFORMATION MODEL (MTM)

In this section, we define the Message Transformation Model (MTM) as a conceptual model for complex message transformations in the context of integration processes.

First, an explanation of the term complex message transformation should be given. The term "transformation" is generally used to describe the deformation of a structure. The message transformation consequently addresses the deformation of messages or message sequences. Before the separation of elementary and complex message transformations can be discussed, the layers of transformation according to [11] will be introduced with the following enumeration:

- 3. Data structures: Semantic description of application object transformations with awareness of relationships and other dependencies,
- 2. Data types: Syntactical description of message data fields, including their data types,
- 1. Data representation: lossless format converting, such as the translation from CVS into XML, but also compression and encryption,
- 0. Transport: lossless transformation for communication purposes concerning specific transformation protocols.

The meaning of elementary message transformation, in the closer scope of transformation, is the realization of layers 0 to 2. In the wider scope of transformation, the complex message transformation addresses all four transformation layers. Due to the complexity of such transformations, they have to be described as a sequence of control-flow- and data-flow-oriented process steps.

3.1 Model Requirements

The requirements of the model for describing transformation processes can be divided into functional and non-functional requirements. Functional requirements describe the explicit basic functionalities, which have to be realized with a defined message transformation model:

- Interaction with an arbitrary number of source systems and target systems,
- Support for the synchronous and the asynchronous execution model,
- Enabling the content-based routing by providing an appropriate query language,
- Handling of unstructured, semistructured and structured data,
- Handling of different amounts of data, up to the message set,
Transformation of the semantic aspect of messages by adding, updating and removing message attributes,

Support for specific methods of elementary message transformation, and

Implicit and explicit validation of messages.

In contrast to the functional requirements, non-functional requirements describe general conditions and characteristics of message transformations processed by a data-aware service integration platform:

- Efficient and scalable processing (amount of data and parallelism),
- Reliable processing using a transaction concept,
- Robustness and failure tolerance, and
- High flexibility by the abstraction of concrete system types.

3.2 Model Classification

The necessity of a generic model for the description of complex message transformations is caused by the data representation heterogeneity of messages but also by the different process models of alternative process description languages. Although a conceptual model generally describes both the structure and the operations, the MTM is consequently separated into a conceptual message model and a conceptual process model in order to illustrate the different perspectives.

With the message model, arbitrary messages and data formats may be used. Thus, this submodel describes the static aspects of the message transformation with the aim of data independence. Indeed, the process model describes the actual transformation process and uses the defined message model. Hence, the process model describes the dynamic aspects of a complex message transformation, independent from concrete process description languages. Figure 2 classifies the MTM into an adopted three-layer architecture, in accordance with the three-layer architecture by ANSI/SPARC.

1. The external layer comprises the different representations of messages and also of data on one side. On the other side, it also encloses alternative process description languages. Thus, this layer represents the view of a user on messages and transformation processes.

2. In contrast to the external layer, which could be seen as a standard- and language-oriented layer, the conceptual layer describes the requirements of complex message transformations concerning their static and dynamic aspects. Thus, this layer is a generalization of data and process descriptions and thus ensures the desired data independence and the independence of specific process descriptions.

3. The internal layer represents the physical realization of the conceptual model, where different alternatives exist as well.

Basically, the conceptual message model is oriented at the relational data model [6] and the evolutionary molecule atom data model (MAD) [9]. Since these models exclusively describe structured data, they are enriched with concepts from models for describing semistructured data, e.g., the “Object Exchange Model” (OEM) [22] and the “Yet Another Tree-based Data Model” (YAT) [5] but also the XML data model.

In analogy to the message model, the conceptual process model is also designed in accordance with the relational data model and uses set operations together with relational operations. Due to the fact that in transformation processes, the control flow also has to be describable, these operations are insufficient. Hence, constructs of the mathematically funded petri net model but also of current process description languages, such as WSBPEL [21], are adopted.

3.3 Conceptual Message Model

The conceptual message model should describe the static aspects of a transformation process. Therefore, it has to be adequately generalized in order to represent the different message formats and data representations.

In accordance with the molecule atom data model (MAD), the MTM meta message model is illustrated in Figure 3.a as a molecule type “Message”. The MAD ensures the descriptiveness of recursive, hierarchical, object-oriented and redundancy-free structures, where the freedom of redundancy is reached by “overlapping molecules” and the hierarchical structure. With such a structure, the processing of messages of different degrees of abstraction is possible. Both complete message objects and attributes of any hierarchical level could be referenced. Thus, the molecule type “Message” is composed of two atom types: the "header

Figure 2: Adopted 3-layer-architecture

![Figure 2: Adopted 3-layer-architecture](image)

Figure 3: Conceptual message structure

![Figure 3: Conceptual message structure](image)
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”. The logical structure of the header segment and the data segment, illustrated in Figure 3.b, is designed in analogy to the relational data model. Thus, the header segment is composed of k name-value pairs, whereas the data segment is a logical table with l attributes and n tuples. Moreover, the attributes may have atomic types or can be a data segment and thus represent nested tables.

Due to the nested tables, the conceptual message model ensures a dynamic and structure-carrying description of all data representations. The organization in tables reduces the overhead for the management of metadata and thus of the structure. Furthermore, the model allows direct access to single data values.

**Definition 1.** Let \( M \) be a message type which is composed with \( M = (H, D) \) of a header segment type \( H \) and a data segment type \( D \), a message \( m \) is defined with \( m \subseteq M \). Furthermore, let a header segment be defined with \( h = \{a_1, \ldots, a_i\} \) as a set of elementary attributes with \( k > 0 \) and a data segment type with \( D = A_1 \times \ldots \times A_l \) as set of attribute types with \( l > 0 \), where the attribute type \( A_i \) is atomic or a data segment with \( A_i \subseteq D \). Hence, it is defined that an attribute \( a_i : D \rightarrow A_i \) is the description of a data segment type onto an attribute type. In addition, we define for all tuples of a data segment: \( \forall t \in D \times A_i : t[a_i] \equiv t[i] \).

The translation of external data representations into the conceptual message model is also important. While the mapping of relational data could be applied directly to the model, the translation of XML is much more complex. In accordance with SQL:2003 Part 14: SQL and XML [18], an element-oriented, and in particular a structure-oriented decomposition is pursued as a specific approach of the XML shredding. For that reason, the decomposition is realized in a generic way, which is based on self-defined structure-oriented rules and which thus is independent from the concrete content.

Hence, an attribute-oriented fine-grained approach could be used internally, whereby messages are decomposed to atomic attributes as the prerequisite for direct access to single attributes. Aside from this illustrated approach, there are alternative concepts like the DeweyIDs [10] and the Pre-Post-Order [8] for the fine-grained management of XML documents. Alternatively, two more approaches are possible. Within the document-oriented approach all messages could simply be managed as BLOB. The advantage is the simple internal use of such messages. However, the disadvantages of this approach predominate; this includes the bad performance when accessing single values and the difficulties when adding, updating and removing message fragments. In contrast to the document-oriented approach, the attribute-oriented coarse-grained approach even has a separation in attributes but exclusively manages a set of atomic or complex subfragments. Thus, direct access to single values is possible but has some other disadvantages such as the data dependence.

### 3.4 Conceptual Process Model

The conceptual process model addresses the dynamic aspects of a transformation process. Therefore, an execution model for the conceptual message model is defined. According to the transformation layers in [11], which have already been introduced, the process model comprises layers 2 and 3 in the form of process steps and processes, whereas layers 0 and 1 (format- and system-specific transformations) should be abstracted by special adapters for external systems.

![Figure 4: Process model design dimensions](image)

1. In accordance with the structural form of the process model, the graph-oriented process model is used. In fact, with the "process", a hierarchical element is integrated. Within this approach, all process steps, also known as nodes, are elementary. These nodes are related to any directed transitions in order to model the whole process. The particular advantage of this approach is the flexible and redundancy-free modeling. Alternatively to the graph-oriented approach, there is the hierarchical approach, whereby elementary process steps are nested in structured process steps. Such a model needs particularly deep hierarchies and offers very restrictive modeling capabilities.

2. For the dimension functional orientation of the process model, the declarative semantic-oriented type seems to be the most suitable approach because of the mightiness of modeling. Moreover, this also ensures the language independence, as the single process steps are oriented at the semantic model requirements. Alternatively, a procedural language-oriented process model would have a tight coupling to the activities of a process description language. An orientation at a block-oriented language would consequently implicate a significantly hierarchical structure.

3. Concerning the dimension of internal representation, the compiled type was used for most efficient processing and most robust execution. Therefore, a static process plan is generated from a template, and by parametrization, a concrete process instance is created. In contrast to that, an interpreted representation of a process is also possible by executing an object graph of parametrized objects during the runtime.
Within this approach, the sequence of execution is determined during runtime and thus can be dynamically updated, with the disadvantage of lower performance.

Furthermore, the actual definition of the conceptual process model should be discussed. Basically, it is realized by the defined base model “Directed Graph”. The concept of “JBOSS Graph Oriented Programming” [16] was adopted and adequately extended, according to the specifics of transformation processes. The base model is limited to three components. The first component is the node, which represents a generalized process step. The second component is the edge and thus, a transition between two nodes. Even with these two components, a directed graph and thus, a graph-oriented process model could already be defined. This is extended by the hierarchical element process.

A single node could have multiple leaving transitions, and during 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, a current and an end node. In fact, such a process is also a specialized node, so that a recursive execution with any hierarchy level is possible.

Definition 2. A process type \( P^x \) is defined with \( P^x = (N^x, S^x, F^x) \) as a 3-tuple representation of a directed graph, whereas \( N^x \), with \( N^x = \{n_1, ..., n_k\} \) and \( k > 0 \), is a set of nodes, \( S^x \), with \( S^x = \{s_1, ..., s_l\} \), \( l > 0 \) and \( s_i = \{o_1, ..., o_m\} \) with \( m > 0 \) is a set of services, including their specific operations, and \( F^x \), with \( F^x \subset (N^x \times S^x) \), is a set of flow relations. \( P^x \), with \( P^x \subseteq N^x \), is also a node type. A process \( p^x \) with \( p^x \subseteq P^x \) has, in this case, a specific state \( z(p^x) \) with \( z(p^x) = \{z(n_1), ..., z(n_k)\} \). Thus, the process state is an aggregate of the specific single node states \( z(n_i) \) with \( z(n_i) = \{M[i]\} \) and \( (((M[i] = \neg\circ) \lor (M[i] = \circ)) \).

According to this definition, a node receives a set of input messages, further executes several processes specified by its node type and its parametrization, and finally returns a set of output messages. The mentioned messages are conform to the defined conceptual message model.

The declarative requirement-oriented 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 three categories: interaction-oriented, control-flow-oriented and data-flow-oriented operators.

The interaction-oriented operators, illustrated in Table 1, allow for interaction with external systems, which should be abstracted, to reach an independence of concrete system types. Thus, transformations of layers 0 and 1 have to be realized implicitly. Indeed, the division of the synchronous and the asynchronous execution model should take place explicitly in the operators. Finally, these operators ensure the suitable modeling of the five basic interaction patterns: initi-
ating receive, not-initiating receive, request-response-invoke, request-invoke and reply.

Through the specification of the control-flow-oriented operators special cases of control flow description are addressed because simple sequences and patterns are already realizable with the defined base model “Directed Graph”. Due to the decision for a compiled representation, these operators are internally used as simple control structures of a procedural programming language.

The data-flow-oriented operators represent the core of complex message transformations. They receive a set of messages, transform them in dependence on their specific node type, and finally forward a possible changed message set.

### 3.5 Example Scenario “ETL Process”

In the following, the introduced example scenario will be discussed in more detail. It will be used to illustrate the modeling with the MTM on the conceptual layer.

Initiated by business transactions within the SAP R/3 system, data from company X is brought into the consolidated database. Therefore, the specific IDOC types [23] DEBMAS05, CREMAS03, MATMAS03 and ORDERS05 from the modules SD (Sales and Distribution) and MM (Materials Management) are used. The movement data of purchase and sales processes of company Y are exported from a proprietary system and stored as XML files into the file system. Before loading the data into the consolidated database, special data have to be extracted from a CRM database, which is physically managed within a MS SQL Server. The schema of the consolidated database (CDB) conforms to the TPCH-Schema [28], which is extended by several flag and timestamp attributes. While plenty of tuple-oriented transactions are executed between the various source systems and the CDB, the data from the CDB is brought into the data warehouse (DWH) only once. Just before such a loading, the process of data cleansing [19] has to be executed. The schema of the DWH conforms exactly to the TPCH schema. Finally, the delta of data which has been brought into the DWH has to be distributed to both data marts (DM), whereby their schemas conform to a reduced TPCH schema. While plenty of tuple-oriented transactions are executed between the various source systems and the CDB, the data from the CDB is brought into the data warehouse (DWH) only once. Just before such a loading, the process of data cleansing [19] has to be executed. The schema of the DWH conforms exactly to the TPCH schema. Finally, the delta of data which has been brought into the DWH has to be distributed to both data marts (DM), whereby their schemas conform to a reduced TPCH schema. During the loading time it is necessary that the movement data from company X and company Y is exclusively brought into the organization-specific data marts. The master data, indeed, should be distributed into both DMs.

The process type es_process1, an example of which is illustrated in Figure 6, will now be discussed in detail on the process step level. As mentioned previously, this process type is used within the asynchronous execution model. Thus, incoming messages are enqueued into the message queue instance sap.mq and processing tasks are initiated. Within one process instance, first, the sequentially next message is dequeued from the message queue by an INVOKE operator, which also implicitly opens a transactional context for this message. After that, a SWITCH operator, as a logical alternative is processed, evaluating the message type by simple XPath expressions. If the message is of type DEBMAS, CREMAS, MATMAS or ORDERS, a specific XSLT schema transformation and the preparation of INSERT statements is processed. Otherwise, the process instance, and thus, also the transactional context are immediately terminated. After the specific TRANSLATION and ASSIGN operators have been executed, a schema validation is used to prevent unnecessary service invocations. If the produced message is valid, the INSERT statements are executed, and finally, the logical message queue transaction is committed using two INVOKE operators. Also, the signal handling should be mentioned. If one of the service invocations fails, the message queue transaction is explicitly rolled back in order to suspend the message queue and to start the deescalation processing.

The MTM transformation processes can be described on the external layer, for example, with WSBPEL, and after that, they can be translated to the conceptual layer or to the internal layer, respectively. Finally, it could be executed by a service integration platform, as it is discussed in the following sections.

### 4. TRANSCONNECT® ARCHITECTURE

The practicability of the MTM has been proven by a prototypical process engine implementation and its integration into the middleware product TransConnect®. This service integration platform has been developed since 1998 and is used as EAI-server or MessageBroker but also as ETL tool.

#### 4.1 Overall System Architecture
TransConnect® is an integration platform and is basically separated into eight subprojects, which are illustrated in Figure 7. This overall system architecture realizes a three-layer architecture. The presentation layer comprises an Eclipse-based rich client workbench as well as a Web-based management console. Furthermore, within the business logic layer, a J2EE application server is used. Finally, the data layer is realized by an SQL-92-conform RDBMS, where this datastore is divided into the “Operational Datastore”, which is characterized by a high update-rate, and the “Metadata Repository”, which is designed with the aim of read optimization.

Within this section, the core structure of the prototypical process engine implementation and its integration into TransConnect® is discussed and our emphasis is on the subproject TransConnect® Server.

4.2 TransConnect® Server Design

The design of the TransConnect® Server is embossed significantly by the extensive adapter concept, the integrated global TPMonitor and the Workflow Process Engine (WFPE). First, this server architecture is illustrated in Figure 8; second, selected components will be discussed further. The TransConnect® server offers a set with so-called, Inbound Adapter, which listens passively for incoming messages and translates them into the internal format. In case of the asynchronous execution model, the internal messages are first appended to the MessagePool in an inbound transactional context and then forwarded to the Dispatcher. This persistence storage ensures the decoupling of the complete inbound side in order to compensate workload peaks. In contrast to that, within the synchronous execution model, the messages are directly forwarded to the mentioned Dispatcher.

This Dispatcher distributes the incoming messages—depending on locally defined meta data—indirectly to MessageQueues or directly to specific process types, which are managed by the WFPE, respectively. The direct forwarding to the WFPE implicates a synchronous processing, while an indirect forwarding to MessageQueues—as a special element for serialization—allows an asynchronous processing.

Thus, a process instance execution within the WFPE is initiated by an incoming message. Furthermore, there is also the possibility of time-based events initiating such a process execution. This is realized by the component Scheduler, which delivers timetable-based events but also forwards external events.

The WFPE allows the translation of process descriptions into an executable form and manages their processing. This processing also includes the invocation of services in the form of Outbound Adapters, local services and other process types. The introduced Outbound Adapters allow scalable and high-performance active interactions with external systems, which are initiated by the TransConnect® Server. Thus, the Outbound Adapters allow reading interactions (pull) as well as writing interactions (push) with external systems.

Finally, the component TPMonitor should be mentioned. It comprises functionality for system monitoring, message persistence and indexing, resource management and caching, high availability and also for transactional behavior and recovery processing.

4.3 Coarse-Grained Comparison of Integration Platforms with DSMS

Even though there are some similarities between integration platforms and DBMS—and DBMS are often used as “Operational Datastore” and “Metadata Repository” by the integration servers—the differences between them do predominate. However, integration platforms, like EAI servers, message brokers and subscription systems, are very similar to Data Stream Management Systems (DSMS)—or synonymous event processing systems—like the QStream project, which was already discussed in [24, 25, 26]. Due to this similarity, now, the several analogies and differences between these two system types should be discussed in a really coarse-grained manner.

The main similarities between DSMS and integration systems are the stored process plans (“standing queries”) and
the dynamic data aspect in contrast to DBMS, where query plans are generated dynamically, while the data is statically stored with a low update rate. Furthermore, restrictions concerning the main memory limitation and the serialization of incoming events have a high impact on processing concepts of both system types. Thus, there are similarities concerning the asynchronous (jitter, message queues)/synchronous processing model, scheduling algorithms, streaming data processing over multiple operators, serialization and synchronization concepts, data transformation, but also robustness and monitoring.

Aside from the numerous similarities, there are also some differences to notice. The main difference can be found at the inbound data side. While DSMS deal with event data streams (ES), continuous data streams (CS) and discontinuous data streams (DS), integration systems deal with message streams (MS), which could be interpreted as event data streams (ES) where one event represents a message or a message set. Thus, integration systems have a higher degree of abstraction. Furthermore, in DSMS, data is usually transient, while messages in integration systems are stored persistently for recovery and scalability purposes. This time-limited persistence causes a very high update rate within the ‘Operational DataStore‘.

As a conclusion of this comparison we can state that there are lots of concepts within DSMS which could be adopted and reused in the context of integration systems. However, the extension of integration platforms with a ‘DSMS facility‘ seems to be interesting as well.

5. SELECTED PROCESSING ASPECTS

Within this section, essential and interesting subaspects concerning the processing of messages and the MTM operator implementation are selected and explained in detail.

5.1 Internal Process Plan Creation

Concerning the MTM, the subcomponent ProcessParser, which allows for the translation of external process descriptions into internal executable process types, is of high importance. The basic requirements for the ProcessParser are the independence from concrete description languages but also the creation of efficiently executable process definitions. Due to the necessity of independence, the ProcessParser presents a logical stratification into four layers, where each layer realizes a well-defined step of translation. Interestingly enough, the analogy to the processing of SQL queries according to [27] should be noticed. In order to realize the highest possible flexibility for process descriptions and to allow extensive monitoring, the result of each parser layer is visible, and also, the parsing could be started on each layer.

1. The first layer realizes the mapping of external process descriptions into the internal XML process type description of the MTM. In case of XML based external process description languages, this could be realized with simple XSL transformations. Thus, the dependence on external description languages is reduced to an absolute minimum. Furthermore, the internal process description is instrumented by identifying NodeIDs. This NodeID concept is the basis of recovery processing, system monitoring and cost-based optimizations. Finally, all services used by the modeled process are determined for integrity purposes.

2. Within the layer of internal analysis and optimization, the internal XML process type is initially analyzed in a rule-based way. These rules are simple Boolean XPath expressions which are evaluated. After the successful analysis, a subsequent rule-based optimization is processed. These optimization rules are XSLT scripts which are used to transform the process type but where the semantics of the process type has to be preserved. In case of the process type recompilation, a workload-based optimization is executed, with the help of monitored statistics. The rule-based optimizer as well as the workload-based optimizer could be controlled by using specific hints, for example, \texttt{NOINLINE\_SUBPROCESS\_COMPILATION}.

3. The third layer of translation is the Java generation. Java classes, representing process plans, are generated in dependence on the internal XML process type. The generator follows a template-based approach, where templates, including placeholders, are defined for a process and the specific process steps. After all placeholders have been replaced by the extracted information, the generated class is physically stored. With such a generated process plan the operators of the MTM are used in the form of a simple class library. Similar to the optimizer, the generator could, also be influenced by hints; \texttt{STATIC\_NODE\_COMPILATION} is an example for that.

4. Within layer 4, an object instantiation is realized. Therefore, the Java class is first compiled and loaded in the JVM. Second, a new process plan instance could be instantiated and parameterized.

In order to explain the concrete translation in depth, the translation of an external WS-BPEL 2.0 process description example into an internal Java process plan will be discussed. For that, a very simple process type, whose external WS-BPEL representation is shown in Figure 10, is used as an example. There, an internal message is received, an STX transformation [1] is executed, and finally, a target system is invoked.

This standard-conform BPEL process is now transformed into an internal XML representation of the conceptual Message Transformation Module. To reach this, an XSL transformation is used. The interaction-oriented and control-flow-
oriented activities are mapped and transformed with the intent of redundancy elimination. In contrast to that, the data-flow-oriented operators are extensions to WSBPEL. Thus, they have to be embedded into the WSBPEL extension-Activity in order to be standard-conform for portability reasons. These data-flow-oriented operators are finally simply copied into the conceptual process description. During this translation within ParserLayer 1, the NID instrumentalization is also executed in order to identify each node. At the end of this parser layer, an XML representation of the conceptual process, which is called process type was created, as illustrated in Figure 11.

After that, the process type is analyzed and optimized in ParserLayer 2. Since there are plenty of concepts concerning the analysis and optimization of message transformation processes, we only refer to [3] at this point.

Furthermore, the information of the XML process type is now used to generate a Java process plan. As mentioned previously, this is realized by a template-based generator, which uses templates for empty process plans and the defined MTM operators. Additionally, the multiple operators are implemented as a class library. Figure 12 shows such a generated process plan.

The illustrated process plan mainly comprises the instantiation, parametrization and execution of the operator sequences. For example, upon creation of an object of the class Translation, the specified stylesheet is loaded from the underlying persistent data store. Considering process plans as “standing queries”, it is suboptimal to instantiate the operators for each process plan instance. At this point, the generatorhint STATIC_NODE_COMPILATION should be mentioned. It forces the generator to compile the operators and their parametrization as static members. Thus, they are initialized only once for all following process instances. Hence, returning to the example, the stylesheet is also only loaded once.

5.2 Internal Message Representations

The translation of external message representations into the internal message model—transparent for transformation processes—is realized by the specific Inbound Adapters and Outbound Adapters. Therefore, the internal representation is the basis for the realizability of the requirements of content-based routing, the handling of different data representations, the transformation of message semantics as well as the flexibility concerning the data aspect. The message is basically composed of a header segment and a data segment. In order to realize the data independence of process plans, the internal message representation should either follow a fine-grained approach, where a message is split into its atomic values, or a coarse-grained approach, where messages are handled in a uniform format, for example XML. These two approaches for internal message representations are distinguished in the following enumeration.

1. Fine-grained message concept: This approach handles a data segment of a message as an unidirectional tree of logical tables. The first important step is the translation of arbitrary data representations into the internal format. For specific formats, such as XML, CVS or relational data, translation rules are defined. The translation of XML will now be explained by an example. Basically, there are three rules: mapping of different child elements (a), mapping of equal child elements (b), and mapping of XML attributes (c). In a second step, the transient internal representation could be translated into a persistent internal representation by bringing it into the underlying TransConnect® DataStore. There is the problem that, at development time, the structure of the message is unknown and DDL transactions should be prevented. Thus, a generic relational structure for saving these representations is necessary. Of course, efficient access to this structure is important. So, the concept of batch inserts is used.
datastream-oriented way. This streaming processing ensures the full scalability; for example, the message content could already be made persistent while reading the rest of the message from the network stream. Thus, very large messages could also be processed. Figure 14 illustrates this second message concept. In order to provide efficient access to single values of the messages, this coarse-grained approach creates the need for the so-called message indexing. There, single values used within process plans, are identified by XPath expressions and are already extracted by inbound and outbound adapters. Due to various aspects, demanding consideration when implementing such a message indexing, this concept should also be discussed explicitly in forthcoming papers.

5.3 Transaction and Recovery Concept

The transaction concept was designed in order to ensure the recoverability of process type states, the avoidance of double message processing and the prevention of message lost. Therefore, the transaction concept is based on the NodeID-concept for the identification of several nodes within one process type, which has already been introduced. Hence, mainly the following three different transaction levels are distinguished.

1. The TX level NONE is the lowest level and is not suitable for the specified aims. However, there are process types where it is acceptable to go without transactional behavior in order to reach the highest possible performance. At this level, no images are stored and no TID management concerning writing interactions with external systems takes place.

2. Within the TX level USER_DEFINED, the user may explicitly use the SAVEPOINT operator, where UNDO images of the process context or single messages should be stored. Thus, the mentioned goals for transactional behavior are satisfied with limitations, depending on the concrete process type.

3. The TX level FULL represents the highest level of transactional behavior because, at this level, UNDO images will be stored implicitly for each reading interaction but also explicitly for each SAVEPOINT. It is unnecessary to store images after each process step because local processing steps, such as an XSLT transformation, could be repeated without getting different results. Furthermore, the TID is managed implicitly upon each writing interaction to prevent the double processing of writing interactions.

Having introduced the core transaction concept, we now want to give a general overview of the recovery process, for which the UNDO images are stored in correlation to the specific NodeIDs. If a process fails or in case of a server breakdown, the latest image of the unfinished process is loaded and a special recovery process type is generated starting at the NodeID the image correlates to. Thus, when a process is run in TX level NONE, the whole process is executed a second time. In contrast to that, Figure 15 illustrates a generated recovery process for a process type which runs with TX level FULL.
5.4 Process Self-Optimization

The defined message transformation model allows for rule-based and workload-based process optimization techniques. The first results and experiments [3] show a high optimization potential but also the need for further research in this area. Therefore, a cost model—with the three cost categories: Communication costs, Internal management costs and Processing costs—has been specified. This cost model and the statistics collected by the TPMonitor-subcomponent SystemMonitor are the prerequisites for workload-based process optimization. In this context, costs are used as an expression for the CPU usage, the main-memory usage and especially for the normalized processing time.

In order to collect the mentioned statistics, the processing costs have to be monitored. Basically, the approach of implicit performance event propagation is used, as shown in Figure 16. Thus, the abstract super class Node comprises the publishing of these monitor events to the SystemMonitor. Due to the fact that all processes and MTM operators are inherited from Node, such events are implicitly published each time a whole process or a single operator is executed. Furthermore, the SystemMonitor collects incoming events using an event buffer. If the event buffer size reaches the MAX_BUFFER_SIZE, all buffered events are handled in a batch manner. For this, there is the possibility to configure the SystemMonitor with a special MONITOR_LEVEL. This level determines if the EmptyMonitor, the TransientMonitor or the PersistentMonitor should be used. These special monitors are subclasses of the SystemMonitor; thus, the event producer does not know which monitor is currently used. The EmptyMonitor is used if no monitoring should take place in order to get the best performance after a time-based optimization phase. The TransientMonitor, in fact, uses the event buffer as a simple unidirectional ring-buffer. Thus, the advantage is a better event monitoring performance compared to the PersistentMonitor. The disadvantage of this monitor is the limited event buffer size but also the requirement for transient event analysis. The last monitor to mention is the PersistentMonitor, which is used by default. Using the event buffer, the events could be made persistent efficiently with a batch insert. The advantage of this is the possibility of long analysis intervals but also the elegant event analysis with relational queries. Finally, the different monitoring levels allow for time-based monitoring as well as for continuous monitoring of events.

The mentioned PersistentMonitor ensures a really efficient evaluation of monitored events, using a special DB-structure and predefined analysis queries. To illustrate this, a simple example should be given. The processing performance events are made persistent in a table which consists of the attributes: EID, PID, NID, NType, StartTime, EndTime. Upon joining this table with the metadata repository, information relevant for the workload-based optimization but also for administration interfaces or auditing the integration system can be created. As an example for such a query, the following statement selects the average processing time (not normalized) of a specific process type in milliseconds.

\[
\text{SELECT AVG(EndTime - StartTime) / 1000000}
\]

\[
\text{FROM ProcessingPerformance}
\]

\[
\text{WHERE NType = 0 AND PID IN (}
\]

\[
\text{SELECT PID FROM Process WHERE PTID = 7 )}
\]

Similar to well-known "control loops", the initiation of self-optimization is realized in accordance to the IBM MAPE concept (Monitor, Analysis, Plan, Execute). This is shown in Figure 17. Thus, in this loop, performance events are published (Monitor) from the RuntimeEnvironment to the SystemMonitor. After the ANALYSIS_INTERVAL has been reached, an asynchronous analysis (Analysis) of the specific process plan within the ProcessParser is initiated. If the requirement for recompilation is determined, a process plan recompilation—also asynchronous—is started. As shown in Figure 9, this recompilation is started in ParserLayer 2c. After the process of recompilation has finished, the specific ProcessContainer within the RuntimeEnvironment has to be locked in order to exchange (Plan) the process plan. Finally, after unlocking the ProcessContainer, the message stream is processed with the recompiled process plan and the loop continues monitoring performance events. Notice that the execution is only interrupted for the short period of the process plan exchange.
6. SUMMARY AND CONCLUSIONS

Due to the lack of a generic model or a standardization for service integration processes, respectively, the Message Transformation Model was defined and proven by its implementation. Furthermore, the advantages of workflow descriptions as well as ETL descriptions were combined in order to define a model, which covers the specification of the control flow, the data flow and even the interaction with external systems.

Thus, we followed a three-layer approach according to the ANSI/SPARC architecture. On the external layer, extended standardized process description languages are used to model the application integration processes. They are translated to the conceptual layer using the defined Message Transformation Model.

The results of this work comprise the Message Transformation Model (MTM) definition, its external description with WSBPEL, and its implementation in TransConnect®. Thereby, the MTM combines strong capabilities of control flow specification as well as data flow specification and could be used in numerous application environments. With the prototypical implementation in TransConnect®, the practicability of the MTM was proven. Concerning further research and development, the MTM could be a generic starting point for special models within other application environments. There is a set of aspects which need further research.

Within the environment of complex message transformation in data-intensive service integration processes, a consolidation can be expected. The descriptions of workflows, EAI and ETL processes but also of message queuing processes are converging. From this convergence and from the lack of generic models, an extensive modeling must be expected, which this paper tried to contribute to.

With the complexity of message transformation processes in mind, the optimization of such integration processes has a high potential. Thus, we basically work on the two optimization perspectives “intra-system process optimization” and “inter-system process optimization”. The first perspective comprises the rule-based as well as the workload-based process optimization. Furthermore, future papers will present experiments and performance measurements which are explicitly excluded from this work in order to present the core model and its implementation.

7. REFERENCES