An Advanced Transaction Model for Recovery Processing of Integration Processes

Matthias Böhm¹, Dirk Habich², Wolfgang Lehner², and Uwe Wloka¹

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

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

Abstract. Integration processes are increasingly used in order to integrate distributed and heterogeneous systems. Although transactional behavior of workflows has been discussed extensively, recovery processing has been disregarded so far. Due to the huge number of different integration systems and models, there are also different transaction concepts with overlapping functionalities available. However, there is the need for a discussion of problem categories and guarantees in order to consolidate the existing transaction concepts. In this paper, we survey possible anomalies of recovery processing in message-oriented middleware and—in conclusion—we define the comprehensive transaction model SIR for data-intensive—but instance-based—integration processes. This model includes the definition of specific transaction levels, which are the precondition for the integration process optimization under transactional restrictions.

Keywords: Recovery, Integration Processes, Transaction Model, EAI, Message-Oriented-Middleware, Workflow Management, Web Service Orchestration

1 Introduction

The scope of data management continuously changes from the management of centrally stored data to the management of distributed and heterogeneous information. Here, a huge number of different integration systems exist, which realize the integration on different levels of abstraction and with different technologies [1]. In the context of process integration and application integration, the specification of message-based workflows (instance-based)—representing integration processes—is increasingly used.

In this context of a central integration platform that integrates numerous heterogeneous systems, one of the main challenges is to ensure transactional behavior, even in cases where external systems do not provide transactional functionalities. The main standards of process description languages, like WSBPEL 2.0 [2], only specify the compensation concept and disregard a full transaction model. Thus, the transactional behavior strongly depends on the used integration system and its implementation. In order to initiate a consolidation process towards a widely accepted transaction model, in this paper, we discuss the main anomalies and a transaction model for recovery processing within integration platforms.
Here, we explicitly address only the recovery processing (R2 and R4) because the aforementioned compensation concepts already address the undo (R1 and R3) of successfully executed activities. Although the compensation concept is specified, the interaction with external systems makes it a necessity for the user to explicitly model these compensations. For instance, as a compensation for an `INSERT` statement, an appropriate `DELETE` statement must be specified, which can be used to `UNDO` the `INSERT`. In contrast to that, our transaction model for recovery processing addresses the `REDO` in case that (a) an integration process fails (R2) or in case that (b) the integration system itself crashes and all pending integration processes have to be recovered.

Although there is a lot of work on transaction management in distributed environments and on transactional behavior of workflows, there is no transaction model for central integration platforms, where the realization of transactional guarantees is transparent to the integrated external systems or services. Hence, our novel transaction model can be the starting point for standardization within the area of enterprise application integration (EAI) and other workflow-based integration concepts, where a loosely coupled infrastructure is one of the most important goals in order to ensure maintainability.

The contribution of the paper mainly comprises four items. First, we describe the problem, considering the architectural as well as processing concepts, and define three problem categories including nine anomaly problems in Section 2. Second, in Section 3, preliminary concepts are mentioned which can be partly used to prevent the highlighted problems. Third, based on these definitions, in Section 4, we define the transaction model SIR (Serialization, Interaction Awareness, Repeatability/Restorability), including different transaction levels. Fourth, in order to simplify the implementation of this model, we provide selected processing aspects and open research challenges we see in Section 5. Finally, we highlight related work in Section 6 and conclude the paper in Section 7.

2 Problem Description

Based on some architectural and processing considerations, in this section, we define the main anomalies possible in integration processes and explain them in very detail. This is the motivation and the requirements specification for the transaction model at the same time.

2.1 Architectural and Processing Considerations

In order to make the problem easier to grasp, we first want to briefly describe a generalized architecture of an integration platform. Typically, such a platform provides a set of so-called Inbound Adapters, which translate incoming messages of external systems and applications into a unique internal message representation. Based on specific routing configurations, these messages are then either directly forwarded to the Process Engine (synchronous execution model) or appended to a Message Queue (asynchronous execution model). Aside from these data-driven events, processes can also be initiated by a time-based Scheduler. However, the Process Engine allows for instance-based workflow execution (one initiating event cause one
process instance thread) using specific **Outbound Adapters** as services in order to interact actively with external systems. For scalability and recovery reasons, messages are made persistent within the **Operational Datastore** for a specified period of time. Basically, this architecture—which is shown in Figure 1—allows for the synchronous as well as the asynchronous execution model.

According to the transactional behavior, the synchronous execution model and the asynchronous execution model should be distinguished in more detail. Within the synchronous execution model, a client is blocked, while the integration process is executed. Thus, the invoking client is able to get result messages but also error messages in case of a server breakdown or integration process failure. Within the asynchronous execution model, messages are made persistent and the invoking client is not blocked until the integration process is executed. Typically, the asynchronous execution model is used for throughput maximization, where latency time is acceptable. In conclusion, only integration processes which are executed within the asynchronous execution model and time-based scheduled integration processes need transactional behavior, while in the synchronous execution model, the invoking client is responsible for **REDO** processing. In contrast to this, the compensation-based **UNDO** processing must occur in both execution models.

In order to distinguish the two types of transactional behavior more clearly, in the following, we define the **Recovery-Based Transaction Concept** (REDO) and the **Compensation-Based Transaction Concept** (UNDO).

**Definition 1.** Recovery-Based Transaction Concept: A transaction concept $T_{CR}$ is called recovery-based if it focuses on the preconditions for recovery processing in case of an integration process failure or a server breakdown. Thus, this type is very similar to R4, and partly also to R2, in DBMS.

**Definition 2.** Compensation-Based Transaction Concept: A transaction concept $T_{CF}$ is called compensation-based if it focuses on the **UNDO** of successfully executed process steps in case of integration process failure. Here, compensations have to be defined in advance, due to the missing control of the external systems, which might not support transactional behavior. Thus, this type is very similar to R1 and R3 in DBMS.
Although the Compensation-Based Transaction Concept is important, the concrete de-escalation procedures depend on the integration process specification, modeled by a user. Thus, in this paper, we only focus on a Recovery-Based Transaction Concept and on how to implement this in a real-world integration platform. Here, in particular, the central integration system must ensure the transactional behavior of the overall integration process, even for external systems which do not support transactional behavior themselves.

2.2 Problem Categories of Recovery-Based Transaction Concepts

In this subsection, we first name the three main problem categories and align the single anomalies within those categories. Second, we formally define the anomaly problems in very detail.

- **Temporal Aspects:** This category comprises anomalies which are caused by the temporal gap between the server breakdown and the recovery processing. Here, the Message Outrun, the Changed Configuration and the Lost Message Update have to be named.
- **Interaction Aspects:** The second category describes anomalies, which are caused by wrong interactions with external systems. Here, the Incomplete Rollback, the Message Double-Processing and the Message Lost must be mentioned.
- **Data Aspects:** The last category comprises problems which show inconsistent data states. Here, Unrepeatable Read, Non-Restorable Data and Outdated Read are aligned.

Now, let us consider the single anomaly problems in detail. Here, we first define those problems and then provide core concepts to prevent the specified problems.

**Temporal Aspects**

**Anomaly Problem 1** Message Outrun: Assume two messages msg\textsubscript{x} and msg\textsubscript{y}, where msg\textsubscript{x} arrives earlier in the integration system than msg\textsubscript{y}, with \(T_{\text{in}}(\text{msg}_x) \leq T_{\text{in}}(\text{msg}_y)\), but the transformed message msg\textsubscript{y}\textsuperscript{'} is sent to an external system s\textsubscript{1} before msg\textsubscript{x}\textsuperscript{'} is sent, with \(T_{\text{out},s_1}(\text{msg}_x') > T_{\text{out},s_1}(\text{msg}_y')\).

Such a message outrun can be explained by two cases. First, if messages are made persistent and forwarded to the Process Engine in a deferred way, the incoming order is not ensured due to the relational set definition. Second, if processes are executed in a multi-threaded way, one process instance might need less execution time than the other. To overcome this, messages—within the asynchronous execution model—should be queued in incoming order and executed in a serialized way.

**Anomaly Problem 2** Changed Configuration: Assume that a configuration c\textsubscript{1} is active during the processing of process instance p\textsubscript{x}. If the configuration was changed to c\textsubscript{2}, it will become active in case of a server restart. In conclusion, if p\textsubscript{x} did not finish successfully (caused by a server breakdown) with c\textsubscript{1} \in \text{active} and c\textsubscript{2} \in \text{pending}, the recovery process rp\textsubscript{x} will be executed with configuration c\textsubscript{2} on a server restart.
Thus, the recovery process execution will lead to unspecified behavior because even external system connections as well as routing properties may be changed. To overcome this problem, two steps are needed. First, a version system for configuration properties must be used. Second, after a server restart, first, all recovery processes should be executed and shortly after that, it is allowed to change the active configuration.

**Anomaly Problem 3** Lost-Message Update: Due to inter-operator parallelism (multiple concurrent subflows) as well as inter-process parallelism (multiple concurrent process instances of different process types), applied message updates can be lost if concurrent operators or processes update the same message.

For instance, if we consider a Fork operator, where Assign operators of two concurrent subflows update a message \( \text{msg}_x \), one of the applied update operations can be lost if they both address the same part. To overcome those raise conditions, messages have to be locked on part level, where at least the states NONE, READ and WRITE must be supported to achieve mutual exclusion.

**Interaction Aspects**

**Anomaly Problem 4** Incomplete Rollback: If a process instance \( p_x \) fails due to a server breakdown or processing failure, a recovery process \( r_{p_x} \) has to be generated and executed. In case that a successfully executed part of \( p_x \) must be rolled back, the specified compensation is executed. Due to missing compensation specifications or external system dependencies, such a rollback might be incomplete, therefore, the reprocessing might fail.

The incomplete rollback is a major problem because suitable compensations often cannot be specified when integrating proprietary systems and applications. Another problem is created by the external system dependencies. For instance, there is an insert trigger which triggers the replication of inserted data to a second site. If the delete compensation is processed and there is no appropriate delete trigger specified, the replicated data of site 1 and site 2 are inconsistent. To overcome this problem, the started process should not be rolled back. Instead, the recovery process should start (with recovered process context) after the last successful process node. With this approach, the Incomplete Rollback can be fully eliminated but there must be awareness of the two problems Message Double-Processing and Message Lost.

**Anomaly Problem 5** Message Double-Processing: If there is a server-breakdown during the processing of an integration process, a recovery process has to be generated. If the external system does not support transactional behavior and the breakdown occurs during interaction with an external system, this problem can be observed. Since the interaction has not been finished, the recovery process includes the interaction with the external system. When executing the recovery process, the same message may be sent to the external system twice. From the perspective of the central integration platform, it is not possible to determine whether or not the interaction with the external system has been finished.
**Anomaly Problem 6** Message Lost: Assume a message $msg_x$, which is processed by the process instance $p_x$. In case that $p_x$ fails, the transformed message $msg'_x$ is not sent to the external system $s_1$. If that happens and $msg_x$ is removed from the integration system, a message lost occurs. Such a message lost may be caused if the message has been sent but not committed while the internal state was set to successful, and thus, the integration system assumes a successfully sent message.

In order to overcome the Message Double-Processing and the Message Lost, the TID concept should be used. With that concept, the temporal gaps in which the Message Double-Processing and the Message Lost can occur are eliminated (if the external system supports transactional behavior) or at least strongly shortened (if the system does not support transactional behavior). This concept is explained in Subsection 3.2.

**Data Aspects**

**Anomaly Problem 7** Unrepeatable Read: Assume a process instance $p_x$, which queries an external system $s_1$ and gets a message $msg_{x,1}$ as result. If $p_x$ fails, a recovery process $rp_x$ is created and executed. In case that the same query to $s_1$ will result in another return message $msg_{x,2}$, the problem unrepeatable read occurs.

In order to prevent this problem, one solution is the maintenance of REDO images. For efficiency reasons, such images only have to be stored for reading interactions or writing interactions with result sets. However, with those REDO images—which are used during recovery processing—the problem of Outdated Read comes along.

**Anomaly Problem 8** Non-Restorable Local Data: Similar to the unrepeatable read problem, local processing steps might also produce unrepeatable results also. If so, the local variables of the process instance cannot be restored by the recovery process.

For instance, if a Translation operator uses XSLT time functions, the produced results cannot be restored. In order to overcome this problem, REDO images should be stored not only for reading interaction but also for all data-flow-oriented operators which update the message content.

**Anomaly Problem 9** Outdated Read: If REDO images are stored for a specific reading interaction or writing interaction with results, the recovery process $rp_x$ will be executed with outdated query results because the data within the external system might be changed.

With the aim to weaken this problem, stored REDO images should be annotated with timeout limits. If the limit is exceeded, the recovery manager is not allowed to use that REDO image but it is possible to re-query the external system. Obviously, Unrepeatable Read/Non-Restorable Local Data and Outdated Read are contrary problems.

### 3 Preliminaries

We need to explain two concepts before defining our transaction model. First, the NodeID instrumentalization as a precondition for REDO images and process resuming is introduced. Second, the TID concept as a foundation for interaction awareness is explained also.
3.1 NodeID Instrumentalization

The core concept of our integration process monitoring approach is called *NodeID instrumentalization*. Basically, a unique ID is assigned to each node of the specified integration process. Therefore, we initially use INTEGER values in ascending order. Figure 2(a) shows an example integration process and how it is instrumentalized with the specific NIDs. In fact, this simple concept is the foundation for system monitoring as well as for recovery processing of integration processes.

However, there are some problems to be handled during integration process recovery generation. First, the problem of *invalid NID order* has to be mentioned. So, during process rewriting (optimization, recovery process generation), nodes may be removed or added. Our solution is that we only instrumentalize the process once. After that, new nodes are enumerated in an alphanumeric manner (e.g., 2a ... 2bc). Second, there is the problem of *temporal order of NodeIDs*. So, with the aim of recovery processing, the NodeIDs within a sequence of nodes must be in ascending order. If we apply node exchanges, this could not be ensured. So, we follow the approach of removing and adding the specific node rather than executing an exchange. The hierarchical operators SWITCH, FORK and ITERATION need further identification of their child operators. Here, we also use a hierarchical enumeration approach.

3.2 TID Concept

The *TID Concept* (in this context) stands for the correlation of successfully executed interactions with unique transaction identifiers (TID). This concept was originally used in SAP R/3 in order to prevent duplicated accounting transactions (see anomaly problem *Message Double-Processing*). Be aware of the different terminology of the TID (tuple identifiers) concept used within storage systems of DBMS in order to achieve logical indirection of tuples.

Basically, the interaction of an integration system *is* with a target system *ts* (without TID concept) contains four steps, as illustrated in Figure 3. First, the core interaction is executed by *is* sending a message to—or using the API of—the *ts*. Second, *ts* sends an acknowledgment, returns data or simply gets the control back. Third, if the target
The TID concept adds two more steps to the core interaction model: the TID check and the TID writing. In fact, there are two possibilities of TID usage, which are illustrated by Figure 4(a) and Figure 4(b). In case (a), where the TID is maintained within the target system, obviously write access is required. In contrast to that, in case (b), no write access is required because the TID is maintained locally. The TID check is used to check whether or not the message has already been processed. If yes, the interaction is skipped and only the specific state is written. Only if the target system in case (a) does not support transactions, Message Double-Processing is possible if the server breakdown of the integration platform occurs between the interaction execution and the TID writing. In case (b), this is more difficult. If the target system does not support transactions, Message Double-Processing is possible (crash between execution and TID writing); otherwise Message Lost can occur (crash between TID writing and commit). In conclusion, the TID concept works in the remote case and under the condition that the target system supports transactional behavior.
Table 1. Overview of SIR Principles

<table>
<thead>
<tr>
<th>Principle</th>
<th>Problem Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serialization</td>
<td>Temporal Aspects</td>
<td>This means the serialized execution of message sequences as well as local processing in sequential order.</td>
</tr>
<tr>
<td>Interaction Awareness</td>
<td>Interaction Aspects</td>
<td>This principle describes the awareness of interactions with external systems with the aim of recovery processing.</td>
</tr>
<tr>
<td>Repeatability/Restorability</td>
<td>Data Aspects</td>
<td>The repeatability addresses the awareness of data accuracy, while the restorability focuses on the persistence of intermediate process results. Both describe the aim of achieving consistent data results on recovery processing.</td>
</tr>
</tbody>
</table>

4 SIR Transaction Model Definition

According to the mentioned anomaly problems, in this section, we introduce the SIR transaction model, which is the foundation for adequate recovery processing in the context of integration processes managed by a central integration platform. Hence, we first define the three principles of the SIR (Serialization, Interaction Awareness, Repeatability / Restorability) transaction model and then we discuss applicable transaction levels that conform to SIR.

The SIR transaction model mainly comprises three main principles, which are named in Table 1. An integration system which implements this model must operate with respect to those principles and must support a subset (at least two of each category) of the mentioned transaction levels.

Further, the SIR transaction levels—illustrated in Figure 5—are separated into a three-dimensional decision, according to the mentioned SIR principles. In the fol-

Fig. 5. SIR Transaction Levels
lowing, we describe the single transaction levels in very detail. Note that a concrete transactional configuration will always consist of three transaction levels (one of each dimension), where side effects and dependencies between dimensions are present.

### 4.1 Serialization Transaction Level Dimension

The principle *Serialization* addresses the temporal aspects and the associated anomaly problems. Here, we distinguish the transaction levels *None*, *Input serialized*, *Local processing serialized*, *Global processing serialized* and *Operational integer*.

The weakest *Serialization* transaction level is *None*. Here, all incoming messages are appended to a global pool with set semantics. The *Process Engine* gets these messages and executes them in a multi-threaded way. Hence, neither the incoming order nor the problems of *Message Outrun* or *Lost Message Update* are prevented. However, even at this level, the problem of *Changed Configuration* must be avoided by the use of a version system for changed configurations.

Based on *None*, the transaction level *Input serialized* additionally ensures that the incoming messages are logically serialized with *Message Queue* components. Thus, the start of local processing within the *Process Engine* is serialized and in most cases, a fully serialized processing is the result. However, there is still the problem of *Message Outrun* due to the multi-threaded execution of process instances.

On top of *Input serialized*, there is the transaction level *Local processing serialized*, where all process instances of a specific process type are executed in a serialized way. Hence, the problem of *Message Outrun* is solved for the case that messages outrun messages of the same type. Nevertheless, the *Message Outrun* problem is not avoided in general because a message processed by process type $p_x$ can outrun another message processed by $p_y$.

One step further, the transaction level *Global processing serialized* additionally serializes all process instances which depend on one specific *Message Queue* (might include different message types which are forwarded to different process types). Thus, the problem *Message Outrun* is fully prevented. However, there is still the problem of *Lost Message Updates*.

The strongest *Serialization* transaction level is *Operational integer* (based on *Global processing serialized*) due to locking of messages which are processed within concurrent subflows of one process instance. Due to the assumption that concurrent access to one message is rare and often results from modeling mistakes, the *optimistic concurrency control* (read version, check and increment version on commit) seems to be most suitable.

We proposed such a fine-grained separation of transaction levels for this dimension because the serialization has very high impact on the throughput and the overall performance of the integration system.
4.2 Interaction Awareness Transaction Level Dimension

The second principle, Interaction Awareness, focuses on the anomaly problems of interaction aspects and comprises the transaction levels None, Restartable, Resumable, Interaction aware and Interaction save.

Similar to the Serialization dimension, the weakest Interaction Awareness transaction level is None. Here, messages are not stored persistently, so the anomalies Incomplete Rollback and Message Lost are present. Especially, the Message Lost should be mentioned because not only a server breakdown but also a server shutdown (with loaded queues) will cause lost messages, which is completely unacceptable.

So, at least the transaction level Restartable should be used, where incoming messages are stored persistently for restart as well as scalability purposes. However, with this, only complete process instances can be re-executed, which causes the problems of Incomplete Rollback and Message Double-Processing.

Based on top of Restartable, the transaction level Resumable specifies that there is a protocol on successfully executed nodes of failed process instances. Hence, the recovery process can restart at a specific position of the original process instance. To do so, the introduced NID concept should be used in order to identify certain positions of one process instance. Due to the fact that no rollback (in the form of compensations) is required, the Incomplete Rollback problem is solved. Anyway, additional to the Message Lost problem, the Message Double-Processing also arises.

In order to prevent the two problems of Message Lost and Message Double-Processing, the mentioned TID concept should be used. Basically, remote as well as local TID maintenance is possible. However, due to different integration system properties, we have to distinguish two transaction levels. The interaction aware transaction level is reached if TIDs are used but the external system does not support transactional behavior. Here, the temporal gap where Message Lost and Message Double-Processing can occur is extremely shortened but is still present. The strongest Interaction Awareness transaction level is Interaction save, where the TID is used similar to Interaction aware. In contrast to the latter, here, the external system supports transactional behavior and thus, the problems Message Lost and Message Double-Processing are solved for the case of remote TID management. Note that if an integration process contains multiple interactions with external systems and at least one of these systems does not support transactional behavior, the transaction level Interaction save is impossible.

4.3 Repeatability / Restorability Transaction Level Dimension

With the third principle, Repeatability / Restorability, all data aspects are addressed, including the transaction levels None, Repeatable read and Repeatable transformation. Here, not all anomalies can be eliminated because there are contrary problems. As a consequence, the user may choose which anomaly to accept.

Similar to the first two dimensions, the weakest Repeatability / Restorability transaction level is None, where no images are stored, so that successfully executed reading operations have to be repeated in order to create the local context which was present at the time of the server breakdown. Thus, this transaction level is insufficient.
The transaction level **Repeatable read** specifies that REDO images are stored after each reading interaction and after each writing interaction with results. So, even at recovery processing, the query results of the original process instance can be reused. In conclusion, the problem **Unrepeatable Read** is solved, while the problem **Outdated Read** arises. This highlights the choice for the user between repeatability and accuracy. Note that there is a dependency between the dimensions **Repeatability / Restorability** and **Interaction Awareness**. If **Repeatable read** is used, at least **Resumable** must be used and vice versa.

The strongest **Repeatability / Restorability** transaction level is **Repeatable transformation**, where REDO images are not only stored for reading and writing interactions but also after each single process step which changes the local variables or messages. Hence, it is much more cost-intensive with respect to execution time and storage requirements. However, with this, the problem of **Non-Restorable Local Data** can be solved, while the problem **Outdated Read** is still present (even for locally transformed data). Similar to **Repeatable read**, at least the transaction level **Resumable** must be used for for **Repeatable transformation** as well.

## 5 Selected Processing Aspects and Open Challenges

Due to the model-based scope of this paper and the lack of comparison opportunities, we do not provide any performance measurements of our model implementation. However, we provide selected processing aspects related to the transaction model and we highlight open problems and further research challenges.

### 5.1 Transaction Logging

Note that our integration process model contains several interaction-oriented, control-flow-oriented and data-flow-oriented operators. All of these operators are specializations of an abstract node. Thus, the optimal (w.r.t. minimal redundancy and ease of use) place of transaction logging is the abstract node. Hence, we integrated logging into the execution method, which also calls an abstract processing method. The latter is overridden by each operator. As a consequence, we have only a single point of transaction logging.

First, the NID logging is executed there. Therefore, if an **Interaction Awareness** transaction level higher than **Restartable** is used, we write a general log entry at the start of the processing (state **INPROGRESS**) and another general log entry at the end of the processing (state **FINISHED**). Second, in case of the interaction-oriented operators **Invoke** and **Receive** and for a **Repeatability / Restorability** transaction level higher than **None**, we write REDO images if the output message of the called operator is not **NULL**. If a **Repeatability / Restorability** transaction level higher than **Repeatable read** is used, we write a REDO image for each operator whose type is in \{**Assign**, **Translation**, **Selection**, **Groupby**, **Window**, **Action**\}. Third, also the local TID management is realized here. The schemas of the different log tables are proprietary, so we omit the details here.
Finally, note that our integration process model also contains a specialized **Savepoint** operator. Here, two possibilities—the context savepoint and the message savepoint—are distinguished. The former stores a **REDO** image of the complete process context, while the latter stores only a **REDO** image of the specified message. This was a compromise in order to allow for a more fine-grained distinction of repeatability within one process type, with the aim of performance optimization.

### 5.2 Recovery Processing

Having surveyed the transaction logging, we want to provide a brief explanation of the recovery processing which is executed at the server startup. First, all invalid processes (states **NEW**, **INPROGRESS**, and **FAILED** _RECOVERABLE_) are determined.

For each process in state **NEW**, the initial message is loaded from the data store. After that, the correlated process type is loaded and deployed. Due to the fact that processing has not started in the past, the process type is simply executed (without any recovery generation). Finally, the state is set to **FINISHED** and the process type is removed (if no more process instances of this type have to be executed).

Similar to that, for each process in state **INPROGRESS** or **FAILED** _RECOVERABLE_, we determine the last recoverable NID _rnid_. After having partitioned the processes according to their _rnid_, a new recovery process (with _nid > rnid_) is generated. Further, these recovery processes are deployed. Now, the last **REDO** image is loaded for each process in order to execute the generated recovery process. If the execution of the recovery was successful the state is set to **FINISHED**. If all processes of one partition are finished, the generated recovery process is removed from the **Process Engine**. In case that a recovery process fails or the server crashes, new recovery processes are generated recursively from the current recovery process. This is only limited by the maximal number of transitive recovery processes. If this number is reached, the state of the process is set to **FAILED_FINAL**, which needs user deescalation.

### 5.3 Open Problems and Challenges

Although the main anomaly problems of a central integration platform, executing integration processes, are solved by the advanced transaction model **SIR**, there are still open research challenges. These challenges are not solvable under the conditions of a central integration platform which is loosely coupled with the integrated external systems. Hence, the user must be aware of these issues.

- **Dirty-Read Problem**: In contrast to traditional DBMS, it is not possible to ensure the **ACID** property _isolation_. There might be indirect dependencies of process types over external systems. Although it would be possible to prevent this via globally (for all process types) serialized access to concrete instances of external systems, the problem is still present. Due to the **Incomplete Rollback** problem, the existence of systems without transactional behavior and due to the semantics of compensation processing (temporal gap between commit and the compensation for rollback), the dirty-read problem (similar to DBMS) is an open challenge.
– **Automated Compensation:** As already earlier, the use of an *Interaction Awareness* transaction level lower than *Resumable* means that specified compensations have to be executed in order to realize a rollback. However, due to the fact that this must be explicitly modeled by the user, this can cause the *Incomplete rollback* problem. Thus, it is an open challenge to automatically generate compensations for specified integration processes. Note that for a subset of external systems, this might be possible, while in general, this cannot be realized independently from the used external systems.

– **Interaction Safeness:** The problems *Message Lost* and *Message Double-Processing* cannot be solved if the external system does not support transactional behavior. However, especially in the area of application integration, where applications often disregard transactional functionalities, concepts should be developed to overcome this (even if very small) temporal gap.

– **Optimization:** Aside from the functional challenges, there is also the non-functional challenge of integration process optimization under transactional restrictions. Thus, there are some global optimization possibilities but also possibilities which are specific to each transaction level combination.

### 6 Related Work

There is a large body of work on transaction support for recovery processing in DBMS [3–6]. However, there are specific requirements for transactional behavior in workflow management, which has also been studied for a long time and many approaches [7–16] and implementations [17–20] do exist. Further, there are also interesting approaches on the transaction coordination of dynamic processes [21], with dynamic participants and very complex interaction scenarios, where a dynamic service negotiation takes place.

Even in the context of service-oriented architectures (SOA), with WS-Coordination (WS-C) [22], Business Transaction Protocol (BTP) [23] and Web Service Composite Application Framework (WS-CAF) [24, 25], there are widely known specifications for distributed transaction management. These specifications were surveyed in [21, 26]. They all work with participant/coordinator semantics, which is unsuitable for the application integration.

Unfortunately, none of the approaches in the mentioned fields address the existing problems of recovery processing observed for central integration platforms in the context of application integration, where a loose coupling between the integration platform and the external systems is required. So, we analyzed the recovery processing of available EAI servers, message-oriented middleware and workflow management systems. Here, many different concepts and models were observable, which reach from the SAP TID concept to fully proprietary implementations. In conclusion, we developed the SIR transaction model to bridge the gap of a missing unified transaction model for recovery processing in the context of integration processes.

### 7 Summary and Conclusions

Due to the changing scope towards distributed data management and the increasing use of integration process specifications, and also due to the numerous different transaction
concepts for central integration platforms, there is the need for a consolidation towards a widely accepted transaction model for recovery processing. Although there is plenty of work in related areas like workflow and distributed query processing, in the context of enterprise application integration, no transaction model is available so far.

Hence, in this paper, we first provided a detailed problem description including the definitions of existing anomaly problems. Second, required concepts were explained as preconditions for the transaction levels. Third, we defined the SIR transaction model and the three-dimensional space of applicable transaction levels. Fourth, we also considered selected processing aspects and open research challenges we see in this context. Finally, the proposed transaction model can be applied to EAI servers, message-oriented middleware, workflow management systems and subscription systems. Thus, we tried to contribute to the consolidation process of transaction models for recovery processing in this context of integration processes.

References


### A Transaction Level Combination Matrix

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td><strong>10</strong></td>
<td>AP1, AP3, AP4, AP5, AP6, AP7, AP8</td>
<td>AP1, AP3, AP4, AP5, AP6, AP7, AP8</td>
<td>AP1, AP3, AP4, AP5, AP6, AP7, AP8</td>
<td>AP1, AP3, AP4, AP5, AP6, AP7, AP8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AP1, AP3, AP4, AP5, AP6, AP7, AP8</td>
<td>AP1, AP3, AP4, AP5, AP6, AP7, AP8</td>
<td>AP1, AP3, AP4, AP5, AP6, AP7, AP8</td>
<td>AP1, AP3, AP4, AP5, AP6, AP7, AP8</td>
</tr>
<tr>
<td>I1</td>
<td></td>
<td>AP1, AP3, AP4, AP5, AP6, AP7, AP8</td>
<td>AP1, AP3, AP4, AP5, AP6, AP7, AP8</td>
<td>AP1, AP3, AP4, AP5, AP6, AP7, AP8</td>
<td>AP1, AP3, AP4, AP5, AP6, AP7, AP8</td>
</tr>
<tr>
<td>R1</td>
<td><strong>12</strong></td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
</tr>
<tr>
<td>I2</td>
<td><strong>12</strong></td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
</tr>
<tr>
<td>I1</td>
<td><strong>12</strong></td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
<td>AP1, AP3, AP4, AP5, AP6, AP8, AP9</td>
</tr>
</tbody>
</table>

1 partially solved anomaly problem, but subproblem still exists