An analysis of transactional workflow support

Grefen, P.W.P.J.; Vonk, J.

Published: 01/01/2005

Document Version
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):
An Analysis of Transactional Workflow Support

Paul Grefen and Jochem Vonk

Subdepartment of Information Systems
Department of Technology Management
Eindhoven University of Technology
P.O.Box 513, 5600 MB
Eindhoven, The Netherlands
{p.w.p.j.grefen,j.vonk}@tm.tue.nl

Abstract
Structured business processes are the veins of complex business organizations. Workflows have generally been accepted as a means to model and support these processes, be they interactive or completely automated. The fact that these processes require robustness and clear semantics has generally been observed and has led to the combination of workflow and transaction concepts. Many variations on this combination exist, leading to many approaches to transactional workflow support. No clear classification of these approaches has been developed however, resulting in a badly understood field. To deal with this problem, we describe a clear taxonomy of transactional workflow models in this paper, based on the relation between workflow and transaction concepts. We show that the classes in the taxonomy can be directly related to specification language and architecture types for workflow and transaction management systems. We compare the various classes with respect to their characteristics and place existing approaches in the taxonomy. We cover both 'traditional' workflow approaches and more recent web-based approaches, including inter-organizational workflow approaches. Together, this paper offers a well-structured and concise analysis of the field of transactional business process support.

Keywords: Transactional Workflow; Workflow Transaction; Transaction Management; Workflow Management; Transactional Business Process; Transactional Web Service; Transaction Taxonomy
7 Analyzing the playing field of existing approaches  
7.1 Existing approaches in the taxonomy . . . . . . . . 32  
7.2 Languages versus architecture in existing approaches . . . . . . 33  

8 Conclusion and outlook  

1 Introduction

Structured business processes are the veins of complex business organizations. Workflows have generally been accepted as a paradigm for modeling and supporting processes in complex organizations [vdAvH02, LR00, CARW98, GHS95, Hsu95]. Often workflow processes have a business character, but workflow concepts have also been used for other processes types, e.g., scientific processes [ABJ+04, BMG97] or software production processes [YW03, Gru94]. The use of workflows for core processes of organizations has led to the requirements of clear process semantics and robustness in process execution, both in regular process execution and under exception or error conditions. The notions of transaction management, already used for several decades in the database world, have been combined with workflow notions to satisfy these requirements. Resulting from this, the notion of transactional workflow or workflow transaction has emerged since the term ‘transactional workflows’ was introduced in [SR93]. This notion is both important in ‘traditional’ workflow management approaches and in more recent web-based and inter-organizational workflow approaches to business process management.

Many variations on the notion of transactional workflow or workflow transaction have been developed, however, by merging the worlds of workflow and transaction management in different ways [Gre02, Gre99] - the two more or less synonymous terms are an omen of this. No clear classification has been developed yet that provides a framework for the analysis of transactional workflow models and systems supporting these models. Matching models and systems with application requirements and comparing approaches is therefore not easy.

In this paper, we present a classification framework that provides two main classes for the combination of workflows and transactions, based on the relation between workflow and transaction concepts. The main classes are further refined into subclasses with specific properties, resulting in six basic classes. We show that the conceptual classes can directly be mapped to specification language and architecture classes for workflow and transaction management support. We analyze the six classes - both on the conceptual and architectural levels - with respect to their goal, means to achieve this goal, and advantages and disadvantages. The framework and analysis together provide a clear basis for comparing approaches and selecting specific approaches for specific application classes.

The structure of this paper is as follows. In Section 2, we describe our basic taxonomy that underlies the classification presented in this paper. In Section 3, we present the conceptual point of view of our classification, which is focused around language aspects. Section 4 is devoted to the system point
of view, centered on architecture aspects. In Sections 5 to 7, we apply the
taxonomy we have developed in three ways. First, we present a comparison
of the various classes with respect to their characteristics. Secondly, we
place existing work in our taxonomy, which, thirdly, we analyze to be able to
draw conclusions and find trends among the numerous different approaches.
Section 8 contains conclusions and outlook.
2 The taxonomy

To support transactional workflows, there are two basic approaches: either transactional aspects and workflow aspects are treated as separate issues, or they are seen as one integrated issue. In the former case, separate transaction and workflow models exist that are combined to obtain transactional workflows. In the latter case, one single transactional workflow model is used. These two main classes are refined below.

In the situation where we have separate workflow and transaction models, we need to relate these two models. We have three possible basic relations, based on the abstraction relation between the models:

**Workflows over transactions (WF/Tx)**: workflows are more abstract than transactions, i.e., transaction models are used to provide semantics to workflow models.

**Transactions over workflows (Tx/WF)**: transactions are more abstract than workflows, i.e., workflow models are used to provide process structure to transaction models.

**Transactions and workflows as peers (Tx+WF)**: workflow and transaction models exist at the same abstraction level, i.e., workflow and transaction models can be seen as two submodels of an implicit, loosely coupled process model.

In the case of one single model for both workflow and transaction aspects, obviously there is no relation between models. There are, however, three main variants with respect to the nature of the single model:

**Hybrid transactional workflow model (TxWF)**: a single hybrid model is used that contains both transaction and workflow concepts.

**Transactions in workflows (WF)**: A single workflow model is used, in which transactional aspects are mapped to workflow primitives.

**Workflows in transactions (Tx)**: A single transaction model is used, in which workflow aspects (process aspects) are mapped to transaction primitives.

The resulting taxonomy is depicted in Figure 1, in which the ‘SEP’ main class contains basic classes with separate models for workflows and transactions, the ‘INT’ main class contains basic classes with a single integrated model. We use this two-level taxonomy in the next sections to discuss conceptual and architectural characteristics of each of the classes.
Figure 1: transactional workflow taxonomy.
3 The conceptual point of view

In this section, we discuss the conceptual point of view of our framework. To do so, we will take the specification language perspective, which we explain below. After this, the various classes of our taxonomy are discussed from this language perspective. We group the classes as depicted in Figure 1. We use stylized examples of specification languages to illustrate the various classes.

3.1 The language perspective

In the conceptual point of view, we are interested in the conceptual specification of transactional workflows. For this specification, we use one or more specification languages. Given the two main classes in the taxonomy of Figure 1, we can have two situations:

- A separate language is used for specifying workflow aspects, the workflow definition language (WFDL), and a separate language for specifying transaction aspects, the transaction definition language (TxDL).
- An integrated language is used for specifying both workflow and transaction aspects, the transactional workflow definition language (TxWFDL).

If we have two languages, the languages can have two relations: either one language is a refinement of the other, or the two languages are orthogonal with respect to another.

If the two languages have a refinement relation, we have the following. A language offers primitives to specify transitions in a state space. A language \( L_2 \) is a refinement of a language \( L_1 \) if there is a notion of correspondence (a relation in the mathematical sense) between its state space and that of \( L_1 \), and between its primitives and those of \( L_1 \), such that the transitions specified by the primitives maintain the correspondence between states.

Figure 2 illustrates the refinement relation: states \( S_{1,1} \) and \( S_{1,2} \) correspond to \( S_{2,1} \) respectively \( S_{2,3} \) (as indicated by the dotted arrows), action \( a_{1,1} \) in \( L_1 \) corresponds to the sequence \( a_{2,1}; a_{2,2} \) in \( L_2 \). The correspondence between the states is in the simple case a projection or mapping, but more complicated cases exist as well.

If the TxDL is a refinement of the WFDL, see Figure 3 for an example, the WFDL level contains states with workflow attributes and the intermediate states at the TxDL level are related to transaction states invisible to the WFDL level. If the WFDL is a refinement of the TxDL, the TxDL level contains states with transactional attributes and the intermediate states at the WFDL level are related to control flow states invisible at the TxDL level.
In the integrated approach, all aspects are merged into a single language. This language covers a state space that is the cross product of the two state spaces discussed above.

3.2 Separate languages

In this subsection, we discuss the cases with separate languages for workflow and transaction specification. The main reasons for using two separate languages are flexibility in coupling workflow and transaction models and separation of concerns in dealing with control flow and transaction aspects in complex applications. Below, we discuss the three basic classes of the separate models approach.

3.2.1 WF/Tx

In the WF/Tx case, the control flow aspect is leading in the specification of transactional workflows. Low-level workflow semantics are based on transactional semantics of individual workflow tasks or groups of workflow tasks. Hence, the TxDL is a refinement of the WFDL. Primitives of the WFDL are mapped to primitives of the TxDL.

In the WF/Tx case, transaction semantics are often imported from the data management level - the TxDL is a sublanguage of a data manipulation language (DML) in this case. The WF/Tx approach is taken in most commercial workflow management systems that support (usually limited) transactional behavior of workflows.

A simple example is shown in Table 1. In this example, individual workflow tasks can be parameterized to behave as business transactions (atomic and isolated units of execution). On the left, we see the specification of a
workflow task. The second and third lines of this specification are expanded on a lower abstraction level to the transaction specification shown on the right. Note that some languages allow multiple tasks to be grouped into a single business transaction (see e.g. [GVBA97]).

<table>
<thead>
<tr>
<th>WFDL</th>
<th>TxDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN TASK task_1</td>
<td>BEGIN TRANSACTION</td>
</tr>
<tr>
<td>BUSINESS TRANSACTION</td>
<td>READ form_1.field_1</td>
</tr>
<tr>
<td>USES FORM form_1</td>
<td>READ form_1.field_2</td>
</tr>
<tr>
<td>END TASK</td>
<td>USE form_1</td>
</tr>
<tr>
<td></td>
<td>WRITE form_1.field_1</td>
</tr>
<tr>
<td></td>
<td>WRITE form_1.field_2</td>
</tr>
<tr>
<td></td>
<td>IF status_ok</td>
</tr>
<tr>
<td></td>
<td>THEN COMMIT TRANSACTION</td>
</tr>
<tr>
<td></td>
<td>ELSE ABORT TRANSACTION</td>
</tr>
<tr>
<td></td>
<td>END TRANSACTION</td>
</tr>
</tbody>
</table>

When executed, the TxDL specification will induce intermediate states with respect to the WFDL specification, as illustrated in Figure 3. In the figure, these intermediate states (R_1, R_2, U, and W_1) are states related to the reading (r.f_1, r.f_2), using (u.f), and Writing (w.f_1, w.f_2) of data items. The other states in TxDL, i.e., Uninitialized, Started, W_2, Aborted, and Committed have a direct correspondence to the states in the WFDL: Uninitialized, Running, Completing, Failed, and Completed respectively.

Figure 3: language refinement example.
3.2.2 Tx/WF

In the Tx/WF class, transactional behavior is the leading aspect in the specification of transactional workflows. High-level transactional semantics are specified with a workflow as elaboration of the underlying process structure. Hence, the WFDL is a refinement of the TxDL. The Tx/WF approach is applied for example in workflow management for e-commerce applications. Here, the transaction between two business partners is the starting point and the elaboration of the control flow a refinement of the transaction.

We show a simplified example in Table 2. On the left, we see a TxDL specification of a transaction that states transactional properties. The control flow is seen as an implementation detail to be specified at a lower level of abstraction. This is elaborated in the WFDL specification on the right. Note that the WFDL specification concerns a non-linear process in which \(task_2\) and \(task_3\) can be executed in parallel, which is not easy to specify in traditional TxDLs. The execution of the WFDL specification will introduce intermediate states with respect to the execution of the TxDL specification.

<table>
<thead>
<tr>
<th>Table 2: Tx/WF languages example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TxDL</strong></td>
</tr>
<tr>
<td>BEGIN TRANSACTION (tr_1)</td>
</tr>
<tr>
<td>EXECUTE ATOMIC</td>
</tr>
<tr>
<td>IMPLEMENTATION (wf_1)</td>
</tr>
<tr>
<td>END TRANSACTION</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

3.2.3 Tx+WF

In the Tx+WF approach, there is a balance between control flow and transactional behavior. High-level transactional semantics are defined on the same conceptual level as workflow processes. Hence, workflow and transaction specifications refer to each other on the same level of abstraction.

Table 3 presents a stylized example. On the right, we see a WFDL specification that specifies a control flow and refers to the TxDL specification for the transactional properties. The TxDL specification as shown on the left imports the task list from the WFDL specification and specifies transactional properties over this. In this example, the TxDL specification specifies compensating tasks (as originally proposed in the Saga model [GMS87]) and a
safepoint (as proposed in the WIDE model [GVA01]) to allow flexible rollback by compensation. Note that control flow and specification of compensation functionality can be changed independently of each other, thus creating a separation of concerns between workflow and transaction specification.

Table 3: Tx+WF languages example

<table>
<thead>
<tr>
<th>TxDL</th>
<th>WFDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN TRANSACTION $tr_1$</td>
<td>BEGIN WORKFLOW $wf_1$</td>
</tr>
<tr>
<td>REFERS WORKFLOW $wf_1$</td>
<td>REFERS TRANSACTION $tr_1$</td>
</tr>
<tr>
<td>COMP $ctask_1$ $task_1$</td>
<td>TASK $task_1$ $task_2$ $task_3$</td>
</tr>
<tr>
<td>COMP $ctask_2$ $task_2$</td>
<td>SEQUENCE $task_1$ $task_2$</td>
</tr>
<tr>
<td>SAFEPOINT $task_1$</td>
<td>SEQUENCE $task_2$ $task_3$</td>
</tr>
<tr>
<td>END TRANSACTION</td>
<td>END WORKFLOW</td>
</tr>
</tbody>
</table>

3.3 Integrated languages

In the integrated model class of the taxonomy, workflow and transaction semantics are combined into one single model. This allows a high level of integration between the two aspects on the one hand, but often introduces complexity and inflexibility on the other hand.

3.3.1 TxWF

In the TxWF class of our taxonomy, we find hybrid workflow and transaction models. These models are reflected in hybrid transactional workflow specification languages. These languages contain typical workflow-related primitives - e.g., to express control flows - and transaction-related primitives - e.g., to express atomicity or isolation requirements.

Table 4: TxWF language example

<table>
<thead>
<tr>
<th>TxWFDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN WORKFLOW $wf_1$</td>
</tr>
<tr>
<td>TASK $task_1$ COMP $ctask_1$</td>
</tr>
<tr>
<td>TASK $task_2$ COMP $ctask_2$</td>
</tr>
<tr>
<td>TASK $task_3$ COMP none</td>
</tr>
<tr>
<td>SEQUENCE $task_1$ $task_2$</td>
</tr>
<tr>
<td>SEQUENCE $task_2$ $task_3$</td>
</tr>
<tr>
<td>SAFEPOINT $task_1$</td>
</tr>
<tr>
<td>END WORKFLOW</td>
</tr>
</tbody>
</table>
An obvious way to create a TxWF language (TxWFDL) is to 'merge' a pair of languages of the Tx+WF class. Following this approach, we can obtain the example presented in Table 4 from the Tx+WF example shown in Table 3 above. Clearly, the TxWF and Tx+WF approaches are exchangeable to some extent.

### 3.3.2 WF

In the WF class of our taxonomy, transactional semantics are expressed in workflow processes. Specific process patterns are used in this case to express transaction behavior of workflow processes. A typical example of this is the specification of compensation patterns in workflow definitions to achieve relaxed atomicity characteristics for a workflow.

We show some examples below, ranging from simple (no transactional semantics) to complex (compensation and abort transactional semantics). In the simple example, as illustrated in Table 5 and Figure 4, the WF specification consists of regular tasks and a regular control flow (three consecutive tasks).

#### Table 5: example WFDL specification without compensation

<table>
<thead>
<tr>
<th>WFDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN WORKFLOW $wf_1$</td>
</tr>
<tr>
<td>TASK $task_1$ $task_2$ $task_3$ # regular tasks</td>
</tr>
<tr>
<td>SEQUENCE $task_1$ $task_2$ # start regular control flow</td>
</tr>
<tr>
<td>SEQUENCE $task_2$ $task_3$</td>
</tr>
<tr>
<td>END WORKFLOW</td>
</tr>
</tbody>
</table>

![Figure 4: sequential process without compensation.](image)

Adding transactional properties to the workflow process definition in the form of compensating tasks is illustrated by the example shown in Table 6 and Figure 5. Because the transactional aspects need to be specified in the workflow definition (WF class), all possible compensation patterns need to be explicitly included in the process specification.

The compensating control flow is linked to the regular control flow through or-splits (alternative paths). At an or-split, a condition is evaluated to check
Table 6: example WFDL specification with static compensation pattern

<table>
<thead>
<tr>
<th>WFDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN WORKFLOW $wf_1$</td>
</tr>
<tr>
<td>TASK $task_1$ $task_2$ $task_3$ # regular tasks</td>
</tr>
<tr>
<td>TASK $ctask_1$ $ctask_2$ $ctask_3$ # compensating tasks</td>
</tr>
<tr>
<td>SPLIT $or_1$ $or_2$ $or_3$</td>
</tr>
<tr>
<td>SEQUENCE $task_1$ $or_1$ # start regular control flow</td>
</tr>
<tr>
<td>SEQUENCE $or_1$ $task_2$</td>
</tr>
<tr>
<td>SEQUENCE $task_2$ $or_2$</td>
</tr>
<tr>
<td>SEQUENCE $or_2$ $task_3$</td>
</tr>
<tr>
<td>SEQUENCE $or_1$ $ctask_1$ # start compensation control flow</td>
</tr>
<tr>
<td>SEQUENCE $or_2$ $ctask_2$</td>
</tr>
<tr>
<td>SEQUENCE $or_3$ $ctask_3$</td>
</tr>
<tr>
<td>SEQUENCE $ctask_3$ $ctask_2$</td>
</tr>
<tr>
<td>SEQUENCE $ctask_2$ $ctask_1$</td>
</tr>
<tr>
<td>SEQUENCE $ctask_1$ $task_1$ # start regular control flow</td>
</tr>
<tr>
<td>END WORKFLOW</td>
</tr>
</tbody>
</table>

whether rollback of the workflow is required - if not, the regular control flow is followed - if so, the compensating control flow is followed. The control flow is graphically depicted in Figure 5. Note that in this case the entire workflow will be rolled back. Different rollback patterns can be catered for by introducing safepoints [GVA01].

![Figure 5: static compensation pattern.](image-url)

One limitation to the above approach in the combination of workflow and transaction specification is that the condition whether or not to branch into the compensation path is taken after an activity has finished, while the reason to compensate (or undo) the process is most likely to be found within the execution of a task. So to apply this approach, a compensating activity must not only be able to undo tasks that have finished, but also tasks that were still being executed. This problem can be overcome when individual tasks
are associated with the classical ACID transaction properties. This way, the results produced in a task are reflected in total or not at all, which allows the failing task to be aborted (results not committed, but task ended) and the already committed tasks to be compensated. An accordingly modified example is presented in Table 7 and shown in Figure 6. Note that in this case \textit{ctask}_3 is not needed because of the ACID properties.

Table 7: example WFDL specification with static compensation pattern and support for abort

<table>
<thead>
<tr>
<th>WFDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN WORKFLOW ( w_{f1} )</td>
</tr>
<tr>
<td>TASK ( \text{task}_1 ) ( \text{task}_2 ) ( \text{task}_3 ) # regular tasks</td>
</tr>
<tr>
<td>TASK ( \text{ctask}_1 ) ( \text{ctask}_2 ) # compensating tasks</td>
</tr>
<tr>
<td>SPLIT ( \text{or}_1 ) ( \text{or}_2 ) ( \text{or}_3 ) ( \text{or}_4 )</td>
</tr>
<tr>
<td>SEQUENCE ( \text{task}_1 ) ( \text{or}_1 ) # start regular control flow</td>
</tr>
<tr>
<td>SEQUENCE ( \text{or}_1 ) ( \text{task}_2 )</td>
</tr>
<tr>
<td>SEQUENCE ( \text{task}_2 ) ( \text{or}_2 )</td>
</tr>
<tr>
<td>SEQUENCE ( \text{or}_2 ) ( \text{task}_3 )</td>
</tr>
<tr>
<td>SEQUENCE ( \text{or}_1 ) ( \text{or}_4 ) # start compensation control flow</td>
</tr>
<tr>
<td>SEQUENCE ( \text{or}_2 ) ( \text{ctask}_1 )</td>
</tr>
<tr>
<td>SEQUENCE ( \text{or}_3 ) ( \text{ctask}_2 )</td>
</tr>
<tr>
<td>SEQUENCE ( \text{ctask}_2 ) ( \text{ctask}_1 )</td>
</tr>
<tr>
<td>SEQUENCE ( \text{ctask}_1 ) ( \text{or}_4 )</td>
</tr>
<tr>
<td>SEQUENCE ( \text{or}_4 ) ( \text{task}_1 ) # start regular control flow</td>
</tr>
<tr>
<td>END WORKFLOW</td>
</tr>
</tbody>
</table>

Figure 6: static compensation pattern with aborted \( \text{task}_3 \).

Even more flexibility in the exception handling can be offered using the concept of \textit{retriable task}. When a retrievable task fails, it is possible to retry its execution a desired number of times. A retrievable task might succeed in one of the retries because, for example, the execution of tasks in other parallel
branches might eliminate the reason of the failure. In case the task continues to fail, the compensating branch in the process can be executed to return to another state in the process from which execution might continue again. An example of a process with retriable tasks, i.e., task\textsubscript{2} and task\textsubscript{3}, is shown in Figure 7.

![Figure 7: static compensation pattern with retriable tasks.](image)

### 3.3.3 Tx

In the Tx class of the taxonomy, workflow semantics are (implicitly) expressed in transaction specifications. In this approach, transactions have structured processes as their action specification. An example specification is shown in Table 8 of which a graphical representation is shown in Figure 8. Here we see a transaction consisting of two subtransactions that can be executed in parallel.

<table>
<thead>
<tr>
<th>Table 8: example TxDL specification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TxDL</strong></td>
</tr>
<tr>
<td>BEGIN TRANSACTION \textit{tr\textsubscript{1}}</td>
</tr>
<tr>
<td>SUBTRANSACTION \textit{s\textsubscript{1}}</td>
</tr>
<tr>
<td>\textit{action\textsubscript{1}}</td>
</tr>
<tr>
<td>\textit{action\textsubscript{2}}</td>
</tr>
<tr>
<td>END SUBTRANSACTION</td>
</tr>
<tr>
<td>SUBTRANSACTION \textit{s\textsubscript{2}}</td>
</tr>
<tr>
<td>\textit{action\textsubscript{3}}</td>
</tr>
<tr>
<td>\textit{action\textsubscript{4}}</td>
</tr>
<tr>
<td>END SUBTRANSACTION</td>
</tr>
<tr>
<td>PARALLEL \textit{s\textsubscript{1}} \textit{s\textsubscript{2}}</td>
</tr>
<tr>
<td>END TRANSACTION</td>
</tr>
</tbody>
</table>
Extended transaction models may contain some (implicit) routing primitives. The more advanced the available routing constructs are, the more the Tx class moves towards the TxWF class. Languages in the Tx class are usually block-structured, whereas languages in the WF and TxWF class are usually graph-structured\(^1\). Hence, we can use the language structure as a criterion to distinguish between the two classes.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{example_transaction_graph.png}
\caption{Example transaction with simple process structure.}
\end{figure}

\(^1\)An exception to this is for example BPEL [ACD+03], which is a mixed language (block- and graph-structured). BPEL belongs to the WF class as there is no transaction aspect in BPEL itself. Extensions to BPEL, like WS-Tx [MTR02], however, add transactional semantics, see also Section 6.
4 The system point of view

After having discussed the conceptual point of view in the previous section, we turn to the system point of view in this section. Where the conceptual point of view explains the ‘what’, the system point of view explains the ‘how’ - i.e., the support of workflow and transaction models. We base the system point of view on the architecture aspect, focusing on the high-level structure of transactional workflow support systems. We first explain the architectural perspective. After that, we turn to the various classes of transactional workflows, again organized as depicted in Figure 1, and relate them to the architectural perspective.

4.1 The architectural perspective

In this section, we focus on high-level, abstract architectures to identify the elementary system characteristics of the classes of the taxonomy. We relate these abstract architectures to concrete architectures in Section 6.

We distinguish between design time and enactment (run) time architecture components and the interfaces between them. Design time components include process design, translator and filter modules. Enactment time components include workflow engine and transaction engine. The interface between the design time and enactment time components is a one-way interface in which the specification(s) are passed from the design time to the enactment time component(s).

In the description of the architectures, we place workflow engine and transaction engine modules on top of a function and data support layer. The details of this layer are not relevant in the context of this paper.

In the classes with separate workflow and transaction support, we have an architectural relation between these two modules. This relation can be based on two architectural patterns. In a vertical (hierarchical) architecture pattern, we have a client-server relation between the two modules. In a horizontal architecture pattern, two modules have a peer-to-peer relation (or server-server relation).

4.2 Separate models

In the separate models category of our taxonomy, we have separate workflow and transaction engines in the architecture (WFE respectively TxE). These modules can have three architectural relations: WFE as client of TxE, TxE as client of WFE, and WFE and TxE as peer-to-peer modules. These three
architectures coincide with the three classes WFE/TxE, TxE/WFE, and TxE+WFE.

In discussing the characteristics of the three classes, a first point of interest is the relation between the design time and run (enactment) time modules. A second point of interest is the WFE - TxE interface. This interface is used in all three architectures to synchronize the control flow state in the WFE and transaction state in the TxE.

### 4.2.1 WFE/TxE

The WFE/TxE architecture is depicted in Figure 9. The left hand side of the figure shows the workflow design (WFD) and transaction design (TxD) modules. The right hand side shows the workflow enactment modules (runtime environment): workflow engine (WFE), transaction engine (TxE) and function and data support (FDS) layer. The workflow specification is passed from design to enactment environment in the format of a workflow definition language (WFDL). The transaction specification is dependent on the workflow specification and is designed in the TxD. It is passed from the design time to the run time environment in a transaction definition language (TxDL), which is a refinement of the WFDL, see Section 3.2.1. How the WFDL and TxDL specifications are stored in the enactment environment is not relevant here (they might be stored using the functionality of the FDS layer).

![Figure 9: WFE/TxE architecture.](image)

The interface between WFE and TxE is both a control and a data channel. The WFE uses the TxE interface to open a transaction context and perform data manipulation operations in this context. In this class, the TxE and FDS are often integrated into one database application environment based on a DBMS with built-in transaction management functionality. The WFE/TxE architecture is ‘standard’ for most commercial systems in which every task in
the workflow process adheres to the ACID transaction properties\textsuperscript{2}, because that is the only transactional support offered by the underlying DBMS. As every task has the same transactional properties, usually the transaction specification for the process is implicitly contained in the workflow process specification and the standard mapping to the transaction engine is implemented in the WFE/TxE interface, hence the dashed boxes and arrows in Fig 9.

\subsection*{4.2.2 TxE/WFE}

The TxE/WFE architecture is depicted in Figure 10. The interface between TxE and WFE is both a control and a data channel. The TxE uses the WFE interface to open a workflow context and next to invoke control flow primitives. We observe this architecture class in e-commerce environments where a high-level transaction engine invokes processes supported by workflow management technology.

![Figure 10: TxE/WFE architecture.](image)

At design time, the high-level transactions are modeled (in the TxD module). The specific details of the steps in the high-level transaction are modeled as separate workflows in the WFD module. Both designs are translated into their respective languages (TxDL and WFDL), which can then be used in the enactment time architecture to actually execute and control the business process.

\subsection*{4.2.3 TxE+WFE}

The TxE+WFE architecture is shown in Figure 11. In this architecture, we have a TxE as transaction server and a WFE as process server in a peer-to-

\footnote{An exception to this is IBM’s WebSphere Process Choreographer \cite{Her04} which also offers transaction support for long-running processes in running them as a set of chained navigation transactions, meaning that each navigation step runs in its own transaction.}
peer relation.

Figure 11: TxE+WFE architecture.

The interface between WFE and TxE is strictly a control channel: the WFE communicates process states to the TxE, the TxE communicates transaction contexts and workflow commands to effectuate transactional effects on process states to the WFE. Note that this interface is not a standard interface as defined by the WfMC - its Interface 4 standard is geared towards communication between two workflow servers.[WfM96]

The workflow design module (WFD) and the transaction design module (TxD) work in close cooperation during the process design because the process design will refer to transaction aspects and the transaction design will refer to the process aspects. This illustrates the strong coupling between the two design specifications, but also illustrates the separation of concerns, as two separate specifications (TxDL and WFDL) are the result of the process design.

Figure 12: TxE+WFE architecture supporting TxWF language.

The TxE+WFE architecture is - trivially - fit for Tx+WF language sup-
port. TxWF language support is possible by filtering a TxWF specification into the right parts for TxE and WFE, see Figure 12 for an appropriate architecture.

4.3 Integrated models

In the integrated models class of our taxonomy, we have a single transactional workflow management module in the architecture. This can either be a transactional workflow engine (TxWFE), a traditional workflow engine (WFE), or an advanced transaction engine (TxE).

4.3.1 TxWFE

The TxWFE architecture class, as shown in Figure 13, offers integrated support for transaction management and workflow management.

The one integrated specification (TxWFDL) contains the process and transactional primitives as designed in the transactional workflow design module (TxWFD). Both aspects are designed simultaneously using the same design module. The specification is then passed to the enactment engine (TxWFE) to be executed (and stored via the FDS module).

![Figure 13: TxWFE architecture.](image)

A hybrid transaction and workflow state is maintained within the TxWFE. It supports languages in the TxWF and Tx+WF classes. To handle Tx+WF specifications, the two subspecifications are merged into one specification by a preprocessor.

4.3.2 WFE

Figure 14 shows the basic WFE architecture class. Processes are designed in the workflow design module. Transactional semantics, however, have to be modeled in workflow primitives, as discussed in Section 3.3.2, so that a WFDL specification is the result of the WFD.

During enactment of a process in this architecture class, the state of a transactional workflow is completely maintained by the workflow engine
(WFE). Because the WFE can only interpret specifications in the WF class, transactional attributes of this transactional workflow state are mapped to workflow attributes.

![Figure 14: WFE architecture.](image)

Extending the WFE architecture with a preprocessor makes it possible for this architecture class to support specifications in other taxonomy classes as well. The preprocessor has to translate these other class specifications (typically TxWF) to WFDL format. An example architecture to support this is shown in Figure 15.

![Figure 15: extended WFE architecture.](image)

This architecture class (with or without the preprocessor) is fit for support by commercial workflow management systems.

### 4.3.3 TxE

The TxE architecture is shown in Figure 16. In the TxE class, the state of a transactional workflow is completely maintained by the transaction engine (TxE). Control flow attributes of this state are implemented by transaction attributes. As the TxE can only interpret specifications in the Tx class, process aspects have to be modeled in transaction primitives in the transaction design module (TxD), resulting in a TxDL specification that can be passed to the TxE to be enacted.

Similar to the WFE architecture class, the TxE architecture can be extended with a preprocessor, so that specifications in other taxonomy classes (typically TxWF) can be supported. The preprocessor has to translate those specifications into TxDL format.
Figure 16: TxE architecture.

Note that support of complex process structures requires advanced transaction management functionality. Nested processes can be supported by standard transaction management technology. More advanced structures are typically only supported by research prototypes.
5 Comparing the classes

In this section, we compare the classes of our taxonomy with respect to their characteristics. The comparison is the basis for analyzing existing approaches or choosing approaches for specific application contexts.

In Table 9, we show a comparison of the classes in our taxonomy. For each class, we list the main goal, the means used to achieve this goal, and a brief list of advantages and disadvantages of the class.

The following issues are considered in the (dis)advantage analysis, which are discussed in relation to the classes below:

- Flexibility in model combinations,
- Possibility for separation of concerns between models,
- Consistency between models,
- Complexity of the formal background of the models,
- Availability of (current) system support,
- Expressiveness of the models, and
- Integration of separate models.

Because of the separate models, flexibility in coupling models and separation of concerns between workflow and transaction aspects are main advantages of the first three classes, i.e., WF/Tx, Tx/WF, and Tx+WF. However, problems with integration of models (and support based on these models) form the downside of those classes. Consistency of specifications is a main advantage of the single-model approach: there are no separate models to keep consistent. Consistency can certainly be a problem in the Tx+WF class, as two specifications of a transactional workflow exist without a ’leading’ specification. Limited expressiveness is a clear disadvantage of the WF and Tx classes, as possible semantics of transaction aspects are limited by available workflow primitives and vice versa. The TxWF class does generally not have this problem, but a complex formalism with equally complex semantics usually is the basis for models in this class. Finally, system support is (currently) best for the WF/Tx class (supported by combinations of existing workflow management and transaction management systems) and the WF class (implementable on commercial workflow management systems).
<table>
<thead>
<tr>
<th>class</th>
<th>goal</th>
<th>means</th>
<th>advantages</th>
<th>disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF/Tx</td>
<td>workflow with robust character</td>
<td>data management in workflows</td>
<td>separation of concerns, flexibility, system support</td>
<td>integration</td>
</tr>
<tr>
<td>Tx/WF</td>
<td>transaction with complex control flow</td>
<td>process management in transactions</td>
<td>separation of concerns, flexibility</td>
<td>integration</td>
</tr>
<tr>
<td>Tx+WF</td>
<td>integrated workflow and transaction</td>
<td>coupled process and data management</td>
<td>separation of concerns, flexibility</td>
<td>integration, consistency</td>
</tr>
<tr>
<td>TxWF</td>
<td>integrated workflow and transaction</td>
<td>hybrid process and data management</td>
<td>integration, consistency</td>
<td>complex formalism, inflexibility</td>
</tr>
<tr>
<td>WF</td>
<td>workflow with robust character</td>
<td>advanced process management</td>
<td>simple formalism, consistency, system support</td>
<td>limited expressiveness</td>
</tr>
<tr>
<td>Tx</td>
<td>transaction with complex control flow</td>
<td>advanced transaction management</td>
<td>simple formalism, consistency</td>
<td>limited expressiveness</td>
</tr>
</tbody>
</table>
6 Positioning existing approaches

In this section, we position a selection of existing approaches in our taxonomy. We first discuss approaches that can be classified in the separate models category, then turn to the integrated model category. As approaches exist that combine aspects of both categories, we also pay attention to this type. To illustrate the practical coverage of our taxonomy, Section 7 presents graphical maps in which the existing approaches covered in this section are positioned.

6.1 Separate models

6.1.1 WF/Tx class

The Mercurius reference architecture [GdV98] is a clear example of an architecture in the WF/Tx class: workflow management functionality is placed on top of transaction management functionality.

In the early Web Services approaches, for the Web Services Flow Language (WSFL) a separate language (WSEL - Web Services Endpoint Language) was envisioned, where transactional aspects could be specified [Ley01]. WSFL then refers to WSEL for transaction support, which places this approach in the WF/Tx class. In the architecture, with the absence of WSEL, the (complementary) Business Transaction Protocol (BTP) [PT01] can be used to provide transaction support for WSFL. This would separate the transactional aspects (BTP) from the routing and other business process management related issues. The architecture is thus part of the Tx+WF class.

Long running transactions with compensation are supported as a high-level feature category in XLANG [Tha01], however XLANG considers compensation as a local issue (each web service deals with it itself) and no global transaction support (over the participating web services) is offered. So process control is considered at a higher level then transaction support, which places XLANG in the WF/Tx class.

6.1.2 Tx/WF class

In one of the early works with respect to transactional workflows, a transaction model for long-running transactions is presented in [DHL91] and is called ATM (Activities/Transaction Model). In ATM, control flow and data flow between the steps of an activity may be specified statically in the activity’s script. A proposed architecture is based on a simple nested transaction implementation and uses the services of a reliable queuing facility by which
the atomic steps are connected. Although it is still possible to express the control flow using implicit rules as developed for the predecessor of ATM (Tx class, see below), we consider ATM to be part of the Tx/WF class, for both the language and architecture.

The CrossFlow approach [VG03, GAHL00, VDGK00] can be considered an example of the Tx/WF class. In the CrossFlow language perspective, cross-organizational transactions are specified in an electronic contract that is mapped to workflow definitions. In the CrossFlow architecture perspective, inter-organizational transaction management functionality is placed on top of workflow management systems.

6.1.3 Tx+WF class

The OPERA process support system, as introduced in [HA98], incorporates flexible mechanisms for failure handling in processes. A key aspect therein is the separation of exception detection from exception handling in nested process structures. The transactional aspects of OPERA are embedded in the notion of spheres of atomicity. Parents in nested process structures can have an exception handler defined, while the exception handlers associated with spheres of atomicity can in turn reference processes. This circular referencing places this approach in the Tx+WF class.

Building on top of the OPERA system, in the WISE project inter-organizational aspects of business processes are addressed [AFH+99]. Again, the decomposition in process and exception handlers is a central issue. Activities are grouped to reach desired semantics, which leads to the concepts of spheres of atomicity, spheres of isolation, and spheres of persistency. Transaction coordination agents are used to provide database functionality on top of application systems. Transactional aspects and (inter-organizational) process aspects seem to be on the same conceptual level, as illustrated by the syntax examples in [LASS00], so that it fits in the Tx+WF class in the taxonomy.

In [DDGJ01], an approach to a language in Tx+WF class is proposed in which independence between specification of control flow on the one hand and transactional characteristics on the other hand is a starting point. The approach is based on transactional spheres (hence identified as ‘Spheres’ in the sequel).

Recent work in the Web Services area to provide transactional support for complex (inter-organizational) business processes composed of separate web services is focusing on extending the ‘de facto’ standard BPEL [ACD+03] (Business Process Execution Language) with specific transaction properties. For example, the work in [TMW+04] proposes to attach transac-
tion policies to the BPEL specification (then called T-BPEL) with which transactional semantics can be specified for web services. Based on that, web services can be matched not only on service offer, but also on the supported transactional properties. In the same line of reasoning, by extending BPEL with additional transactional specification languages, the work on WS-Coordination [CCF+04a], WS-AtomicTransaction [CCF+04b], and WS-BusinessActivity [CCF+04c], collectively designated as WS-Tx and the work on WS-CAF [BCH+03a, BCH+03b, BCH+03c, BCH+03d] are placed in the Tx+WF class.

6.2 Integrated models

6.2.1 TxWF class

In the context of IBM’s FlowMark WFMS, an approach for the support of business transactions has been developed [Ley95]. It uses specification of transactional spheres in a workflow process that are interpreted by extended workflow technology - the approach can hence be placed in the TxWF class (identified as FlowMark+ in the sequel).

Support for multi-system workflow applications consisting of transactional and non-transactional tasks has been developed in the METEOR project [KS95]. Transactional semantics is coded in terms of workflow attributes (workflow states) within the workflow specification using dedicated keywords in the language. This places the METEOR workflow specification language in the TxWF class in our taxonomy. However, in the implemented system, the workflow controller functions as the transaction manager, so that the METEOR system belongs in the WF class of our architecture classification.

In the Exotica approach [AAA+96], a language is proposed in the TxWF class. This language is preprocessed to be interpretable by an architecture in the WF class. In the FlowBack project [KMO98], a similar approach has been developed.

In [BMR96] the concept of an INCA is presented. An INCA (information carrier) contains information about requested services and the relevant context for the execution of the services. It also contains rules that are used to specify the control flow and data flow, as well as the failure atomicity requirements of the computation. The locus of control of a computation migrates between the processing stations. This requires an agent and rule management system on each processing station. The agent may exploit the support available from the underlying processing station. This implies that the INCA language belongs to the TxWF class, but the architecture is part
of the WF/Tx class.

The OpenPM middleware system [DDS96] developed at HP Labs supports three kinds of activities: forward, cancel, and compensation, and two kinds of connectors: forward and reset arcs. For each forward activity definition, a compensation scope and compensation activity can be specified. Transaction semantics are thus embedded in the workflow process specification and therefore this system belongs in the TxWF class.

Another example in the TxWF class is the Virtual Transaction Model [KS00]. A Virtual Transaction is a region of a business process designed to preserve ACID properties. It may involve multiple activities, each acting as a sub-transaction within the virtual transaction. The approach seems similar to the WIDE approach, see Section 6.3, except that a choice is offered after compensation has finished to determine the way to proceed: redo, alternative path, or terminate. The system is based on HP Labs’ ChangEngine, which in turn is the successor of OpenPM described above. As ChangEngine relies on the underlying database system (Oracle DBMS) the architecture falls in the WF/Tx class.

BPML (Business Process Modeling Language) can be seen as a recent approach in the TxWF class [Ark02]. BPML supports two transaction models: coordinated (ACID transactions) and extended (isolation requirements relaxed). Transactions are main concepts in BPML, as are process, activities and rules.

6.2.2 WF class

ObjectFlow is placed in the WF class [HK98]. It distinguishes between business-level recovery and system-level recovery. The first is accomplished using backward arrows in the control flow of the process (restrictions on backward arrows apply). This allows going back in the process to redo tasks, which have their state reset and the data restored at that time. The latter is supported by relying on the database management system to preserve the ACID transaction properties, which correspond to task level granularity.

Also in the WF language class, we find work in which transactional characteristics are coded into Petri Nets [Deh01, CPL02], similar to the examples presented in Section 3.3.2.

6.2.3 Tx class

One of the earliest approaches in transactional workflows is presented in [DHL90]. In the presented model, each step of an activity is modelled by a transaction. The control flow among the steps is expressed implicitly by
rules. The rules are triggered by an event or by the rule system detecting that some specified event has occurred in the database. Because transactions and rules are the main concepts in this model, it is part of the Tx class.

The ConTracts model [RS95, RSS97] is a more recent approach in the Tx class. ConTracts provide an environment for reliable execution of long-lived computations. The control flow primitives of the ConTracts language have been used for the realization of transactional workflows. TSME [GHM96] is an approach in the Tx class with comparable goals.

6.3 A combined approach

In the WIDE project, an approach has been developed that combines aspects of both main classes of our taxonomy.[GPS99] The WIDE workflow specification language belongs in the TxWF class, as it is a ‘classical’ workflow definition language extended with (among other things) transactional primitives.

However, the WIDE architecture contains two layers that each offer different transaction support to cater for the differences in transactional semantics for parts of processes [GVBA97]. Processes, themselves, are long-running and require more relaxed transactionality than the short-running subprocesses or activities that are part of the processes. Therefore, the WIDE global transaction support (GTS) is developed to offer extended compensation-based transaction support [GVA01, VGBA99]. The local transaction support (LTS) [BGVA98] provides transaction semantics of the short-running process elements. The LTS, itself, is also separated in two layers; the higher level supports a variant on the nested transaction model and the low-level local transaction interface (LTI) manages the actual execution of interleaved transaction context related to the execution of the nested activities, i.e., managing the concurrent access to shared data items among different activities [BGVA98]. For the taxonomy, this means that the GTS and LTS are part of the Tx+WF class and the LTI belongs to the WF/Tx class.

So, although the language falls into the integrated main class (TxWF), the architecture falls into the separated main class (Tx+WF and WF/Tx), which illustrated that combined approaches in developing transaction support for processes is also a possibility to consider.
7 Analyzing the playing field of existing approaches

In the previous section, we have discussed a variety of existing approaches to transactional workflow support. In this section, we analyze this playing field. First we position a number of discussed approaches in the taxonomy to get an overview of coverage of the taxonomy. Next, we place all discussed approaches in a language/architecture grid to analyze general uniformity in the approaches. Further, we inspect the grid to try and observe trends.

7.1 Existing approaches in the taxonomy

In Figure 17, we graphically place a selection of the discussed approaches graphically in the taxonomy of Figure 1. We distinguish between language and architecture relations in the figure (see legend of the figure).

As can be seen in the figure, the taxonomy is well covered by the existing approaches. One can clearly observe that several approaches are positioned in multiple classes. Noticeable is the WIDE approach, as it is the one approach that uses two different classes for the architecture, while using one class for the language.
7.2 Languages versus architecture in existing approaches

In Figure 18, we place the discussed approaches in a two-dimensional grid\(^3\). The horizontal axis of the grid corresponds to the conceptual aspect of our taxonomy, the vertical axis to the system aspect of our taxonomy. Approaches located on the diagonal of the grid have conceptual and system aspects belonging to the same class of our taxonomy. Both ends of the axis cover the transaction-centric classes of the taxonomy (from the separate and integrated main classes), while the workflow-centric classes are located in the center of the axis.

![Figure 18: approaches in grid.](image)

As can be seen in the figure, most approaches are located on the diagonal of the grid (certainly if one takes into account that many commercial approaches, not shown in the figure, belong to the WF/Tx class in both dimensions - comparable to the Mercurius approach in this context).

Approaches in the lower right hand corner of the grid have integrated language but separate system aspects - here, separation of concerns in the system architecture has been used as a design principle. In the set of dis-

\(^3\)Approaches that only describe language or architecture aspects have been placed in the corresponding architecture respectively language class (assuming uniformity in approach).
cussed approaches, there are none with the inverse (integrated architecture but separate languages).

From Figure 18, we draw the following conclusions:

- If the language and system classes are different in a specific approach, the language is always located in the TxWF class.

- From all the analyzed approaches (25 in total), only five can be considered to be workflow-centric (20 percent). Also, only 20 percent can be considered transaction-centric (five approaches). The majority of the approaches (60 percent) has a balanced transaction/workflow focus.

Figure 19: dates of approaches in grid.

Exchanging the names of the approaches in the grid shown in Figure 18 with the year from which they stem (based on the year in which the main publication of that approach appeared), we get the grid shown in Figure 19 as a result. From this grid we draw the following conclusions:

- Approaches belonging to the Tx class have been developed about a decade ago, and have not received any attention again since that time.
• The same seems to hold for approaches combining languages from the TxWF class with architectures from the WF class.

• The most recent approached to transactional workflows can be found in the Tx+WF class.

• BPML is revisiting an approach that seems to have been abandoned about a decade ago.

• All Web Services approaches (except BPML) to offer transaction support seem to focus on the Tx+WF class, which could mean that separation of concerns is now more important than before.

• All recent approaches (except BPML) belong to the ’SEP’ main class of our taxonomy.
8 Conclusion and outlook

In this paper, we have described a taxonomy for transactional workflow support, paying attention to both the conceptual and system point of view. Characteristics of the conceptual point of view have been described in terms of specification language classes. System characteristics have been discussed in terms of architecture topologies and interfaces. The result is a taxonomy that provides a background for selecting or analyzing transactional workflow support.

Choosing appropriate classes from our taxonomy for the conceptual and system points of view is a basis for the configuration of transactional workflow support in complex applications, where functional requirements and architectural context both play an important role. The choice can be different in both points of view to some extent, but a mapping must be possible.

An analysis of the existing approaches with respect to our taxonomy reveals that the Tx class and the combination TxWF language and WF architecture class have been researched in the past as an approach for transactional workflows, but no recent developments have been discovered for those approaches. Also, any mismatch between language and architecture classes, i.e., not the same class for both aspects, only occurs when the language is of the TxWF class.

The most important trend discovered within the analysis is the increase in importance of the 'separation of concerns', and flexibility characteristics. This is implied by the fact that most recent approaches fall into the separate models classes of our taxonomy. This is well in line with the ideas of the Web Services context, in which modularity and flexibility of functionality are major issues.

The work presented in this paper is the basis for further work. In the upcoming research area of grid computing, grid transactions can be considered the next logical evolution of transactional processes. Analyzing grid transaction approaches[THSS05, SWT, QXZY04] and placing them in our proposed taxonomy is an interesting extension of the work presented in this paper.

Acknowledgments

Maarten Fokkinga is acknowledged for his assistance with respect to concepts related to languages and his feedback on the draft version of this working paper. Ting Wang and Benedikt Kratz are acknowledged for their comments and suggestions to improve the draft version of this working paper.
References


[Ley01] F. Leymann. Wsfl - web services flow language (wsfl 1.0), May 2001. IBM.


