Enhancing Workflow Data Interaction Patterns by a Transaction Model

Sebastian Schick, Holger Meyer, and Andreas Heuer

Database Research Group
University of Rostock
{schick,hme,heuer}@informatik.uni-rostock.de

Abstract. Todays process-aware information systems (PAIS) provide little support for explicit specification of transactional aspects. PAIS have to integrate events and data from various external sources as workflow relevant data. Furthermore, it should be aware of changes made externally and write consistently back data used and altered to external sources. To avoid inconsistencies within redundantly maintained data, transactional aspects within process and data perspective have to be supported. We present a layered architecture which overcomes most of these problems by extending a workflow management system (YAWL) with facilities to access external data sources, to associate the control flow perspective with transactional properties like isolation, serializability and recovery. To ensure a better data integrity, we define synchronization strategies and integrity constraints beyond single objects and tasks. Furthermore, we integrate transactional and non-transactional sources to offer better data security, data persistence and data recovery within our workflow model.

Keywords: Data access; Integrity constraints; Transactional workflows; YAWL

1 Introduction

In Process-Aware Information Systems the integration of external data sources is still a challenging problem. This drives many ongoing initiatives to improve data integration within workflow systems. In the Perikles project[1] [3] we build a PAIS supporting the operating room (OR) manager in large clinical, peri-operative centers. The system is driven by events like information from different sources e.g. patient record or clinical information system. The workflow system has to integrate events and data from external sources as workflow relevant data to keep track of the different ORs' status, from scheduling an OR until the patient leaves the peri-operative center or the hospital. In clinical environments usually the operating room (OR) is the facility with the highest costs and revenues.

Generally the perioperative process can be divided into three sub-processes pre-, intra- and postoperative process, indicating the phases before, during, and

[1] www.perikles.org

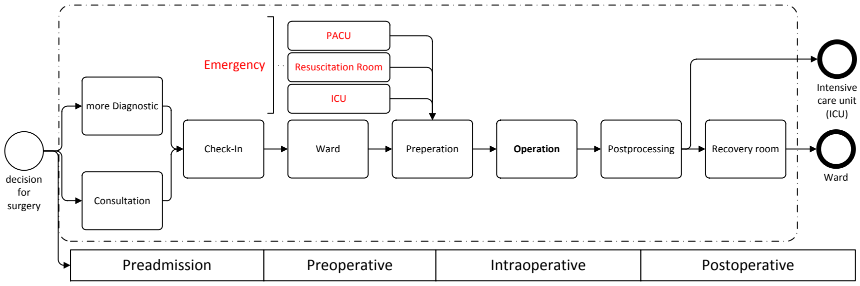


Fig. 1. Perioperative Process at a Glance

after the surgery (cf. Fig. 1). On the whole, the complete surgery-related business processes must be well managed and scheduled in order to be cost-efficient while at the same time meeting the patients expectations of timely service delivery. These challenges call for a supporting system that is process-oriented, resource-centric, and schedule-aware.

Since the data in the external sources, like in the clinical information system or patient record management systems, is altered independently from being used within PAIS, the workflow system should be aware of changes made externally and write consistently back data used and altered to external sources. Furthermore, the data has to be mapped from external systems to workflow internal data, i.e. onto workflow variables. By this, activities can make use of data from external sources without arranging the access by itself.

In case of Perikles data source integration of several heterogeneous systems leads to various problems. Current systems allow explicit data integration only when executing a task and its related application function [14]. The missing relation between application data and workflow relevant data, therefore, has to be modeled in an appropriate way. Approaches like [6] propose an extension to current systems using plug-ins. Nevertheless, issues like global consistency or isolation where not discussed. The access to external data from within the workflow system is addressed by some of the workflow data patterns [20], namely Pattern 15, 16, and 19, 20. Within the data-based routing, different kinds of data conditions are evaluated within the control flow perspective. But the approach does not define any data integrity constraints or isolation properties for external data. This is left to the control-flow view. Only few systems partially implement them, none fully. The YAWL engine can handle some of the state-based conditions (e.g. case initiation, case completion).

To avoid inconsistencies within redundantly maintained data, transactional aspects within the control flow and data perspective have to be supported. Consequently, transactional properties like consistency, isolation, durability, reliability, robustness, and correctness have to be provided by PAIS.

However, many PAIS provide little support for explicit specification of transactional aspects. In the course of database research many advanced transactions models were proposed, investigated, and developed further [8]. Many approaches

[22, 5, 12, 13] have tried to combine concepts from both, workflow and transactional systems. Grefen [10] presents a taxonomy of combining transactional systems with workflow engines and positioned existing approaches. However, transactional properties are desired in PAIS, too. Therefore, transactional types have to be integrated within the control flow perspective. Process parts, apart from atomic tasks should be atomic and isolated building blocks. Another requirement is to define integrity constraints over process parts rather than single task parameters.

We present an approach, which allows for a synchronized, consistency-aware access to data of external sources. Therefore, we extend the workflow language YAWL [1] with the concepts, to model nested transactions and ease the task implementors access to external data sources within the control flow perspective. To ensure a better data integrity, we define synchronization strategies and integrity constraints beyond single objects and tasks. Furthermore, we integrate transactional and non transactional sources to offer better data security, data persistence and data recovery within our workflow model. Besides the data flow described in the workflow model [21] also the data source integration is described in more detail. According to the taxonomy in [10], our approach refers to the class of transactional workflows. In that way, our approach is different from others, which manage and orchestrate long running transactions across different coordinated web services but did not support and control data access within the web service or task implementation [17].

We end with the essence of requirements for transactional workflow support:

- *Req. 1:* There is a urgent need for transparent access to external data sources from within the workflow systems.
- *Req. 2:* The workflow system should manage and control external data sources and allow access through workflow variables.
- *Req. 3:* Workflow activities/tasks should access external data through (externally bound) workflow variables.
- *Req. 4:* The workflow system should allow for defining transactional spheres and inherently support them.
- *Req. 5:* Transactional spheres should assure integrity, correctness, and recoverability over externally bound workflow data.
- *Req. 6:* The workflow system should support integrity constraint over workflow variables, (even if they are bound to external data sources).
- *Req. 7:* The workflow system should offer suitable integrity violation and exception handling mechanisms in combination with transactional spheres.

2 Related Work

In the course of time, many advanced transaction models were introduced. They relax certain properties of the classical ACID transaction like isolation or allow for nested transactions and different kind of structures within a global transaction. We build our concept mainly upon open nested and multi-layered transactions [4] and used multi-version concurrency [16] control for coping with the recoverability problem of open nested transactions. The idea of non-vital, contingent, and compensating transactions as alternative building blocks to atomic ACID transactions are described in detail in [8, 23].

Different approaches in integration transactions and workflows are investigated for the last 20 years. Worah and Sheth [24] and Grefen [10] gave a overview on different integration concepts and systems for *transactional workflows*. The latter suggests a taxonomy for a conceptual and system view on the topic. Grefen et al investigated different dependencies of transactional building blocks within a workflow [11]. They present also a formal transaction model based on graph theory. Long-running transaction are used in [17] and a pervasive workflow approach is presented in [15]. Long running workflows, Sagas [9] and their extensions give up isolation and have to cope with recoverability. That's why, they have introduced compensating transaction.

A better data integration within PAIS is tackled by different approaches. In [14, 18] the aspect of a close integration of the data control flow perspective where described. One main demand in [14, 18] are compliant business processes with the underlying data structures. Hence, different *challenges* where defined to summarize *Object-aware Process Management Systems* [14]. One requirement is the integration of application data within the control flow perspective, so that data is manageable and accessible as complex objects. A generic component for process management is proposed there, which enables data-driven processes. Therefore, an integrated view on the process and the data is introduced. However, the data exchange with external data sources is not tackled. Neither aspects regarding data source integration nor the combination with transactional models are discussed.

In [6, 7] Lehmann and Eder present a comprehensive approach for integrating external data sources. This approach describes an architecture in a way very similar to ours, e.g. the integration of external data sources into the control flow perspective is based on XML, too. The integration is also done using *data access plug-ins* which are controlled by a *data management service*. But the approach just considers data integrity constrains on single variables (data sources). Read and write operations on data sources are under control of user defined policies as isolation or correctness of the data access do. Consequently, the approach didn't support any kind of global integrity constraints or transactional concepts required to avoid inconsistencies within redundantly maintained data.

3 Transactional Workflows — The Concepts of *tx+YAWL*

Now, we present a conceptual extension of the workflow system YAWL [1] called *tx+YAWL*. It combines the integration of external data sources into the workflow systems with transactional properties. The transactional model provided is based on open nested transaction. In fact we exploit the multi-layered transaction approach [4]. It allows for structuring different operations on different layers and providing mappings between two consecutive layers. The transaction concept also make use of multi-version concurrency control [16] to cope with recoverability

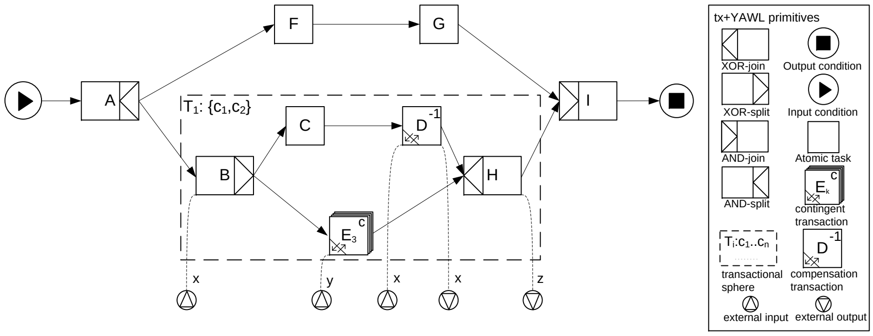


Fig. 2. Transactional workflow with sub-transactions and constraints

by providing a consistent view on versions of database objects and avoiding cascading aborts. The approach is structured into four different layers:

– Layer L_0 is responsible for accessing external data source through workflow variable. The externally managed data can be described by XML Schema types and by views established on XML instances using XQuery. (*Req. 1 and Req. 3*)
– On Layer L_1 workflow tasks are the transaction building blocks. They provide the basis for different transaction types, like contingent, compensating, or non-vital (sub-)transactions. (*Req. 1, Req. 3 and Req. 7*)
– Layer L_2 describes the overall control flow and transactional spheres. Integrity constraints, consistency, and recoverability are provides at this layer. (*Req. 2 and Req. 4-7*)
– Layer L_3 associates workflow cases with the top level transactional sphere or global transaction. (*Req. 4*)

Figure 2 presents a transactional workflow net as described by the tx+YAWL workflow language. The workflow consists of nine workflow tasks $(A, \ldots, I)$. Task A contains an AND-split and tasks I the corresponding AND-join. The two branches consist of a sequence F, G and a sub-structure building a *transactional sphere* [2] T_1. The sphere T_1 encompasses tasks which access external data through input/output variables x, y, z and integrity constraints $\{c_1, c_2\}$ to be ensured for this sphere. All tasks within a sphere are part of a transaction. Some may be sub-transactions with an associated type. Task D represents a sub-transaction with compensation and E consists of three contingent sub-transactions E_1, E_2, and E_3. Only one out of this three sub-transactions must succeed to let E commit. All other tasks are steps of the transaction represented by the sphere.

The concepts of transactional workflows and the details of *tx+YAWL* model are now explained along the four layers.

Layer L_0 (Basic data access) is responsible for the access to data from external sources through specific plug-ins (denoted by the triangles in Fig. 2).

[2] Dashed rectangles are used to depict spheres in Fig. 2.

Actually, the basic operations provided are reading a certain version $T_i.r(x_j)$, writing $T_i.w(x_j)$ and enforcing a transaction boundary by committing $T_i.c()$, or aborting $T_i.a()$ a local transaction at the data source if support. The behavior of these operations depends on the transactional spheres of T_i and is controlled by the layers above. A plug-in (referenced by an identifier pId) encapsulates the data source. The structure of the data is described by a XML Schema XSD_{pId}. The plug-in is responsible for mapping source specific structures to and from valid XML data. Additionally, the plug-in provides functionality to establish a connection, transfer data and exchange service information. In Fig. 2 tasks B, D, E, H accesses external data using plug-ins connected to variables x, y, z.

Layer L_1 (Workflow tasks) describes concepts of tasks as the building blocks of the transactions. Tasks have input and output parameters, e.g. external variables x and y are bound to input parameters of task B, D and E, z is bound to an output parameter of task H in Fig. 2. A task should not directly perform read and write operations an external data from within the tasks implementation. In fact, tasks must access external data through its input and output parameters. Parameters v are described with their own schema XSD_v. The Layer L_1 is responsible for associating the these parameters of atomar or complex tasks with the read and write operations from Layer L_0 defined over the external variables managed by the plug-ins. In case of a composite task, parameters are mapped from or onto variables of the workflow net implementing the composite task. The order of read and write operations is assumed to be evaluated from left to right, i.e. in the order they appear in the parameter list.

To reuse these external variables by different parameters v a mapping between the related XSD_{pId} and XSD_v has to be defined. This is done using an XQuery expression. A mapping m_v consists of a set of $(pId, distKey, map, r_p, w_p)$, where

- pId is a reference to a data source which is encapsulated by a plug-in.
- map is an XQuery expression, which defines transformation between an external variable and the data source managed by plug-in pId.
- $distKey$ indicates a distinct key value, to unambiguously identify an XML fragment in the data source.
- r_p defines the local *read policy*
- w_p defines the local *write policy*

The mapping allows only unique XML fragments as a result of a query. The result has to be a valid subtype of the data source XSD type ($XSD_v \subseteq XSD_{pId}$).

The read operations on external variables yield values for the input parameters of a task. A *read policy* r_p defines whether the external variable has to be re-read each time it is access or kept isolated from external modification of the original data. That's why, different operation modes were proposed. The mode *immediate read* requires for each read access a synchronisation with the external data source. The mode *consistent read* requires a transactional sphere which determines the version of data object to be read.

Write operations on external variables values write back values of output parameters associated with the variable. A *write policy* w_p defines whether modifications of a copy have to be pushed by the workflow system to the external data source immediately or not. Furthermore, the policy is divided into the modes *immediate write*, which propagates changes after each update. With mode *consistent write* the transactional sphere determines whether the operation is can directly write back a new version of the data object or has to be deferred. The mode *no write* simulates a read only variable an can be used by hypothetic or read-only transactions. In combination with transactional sphere only *consistent read*and *write* modes are used.

At this layer local integrity constraints are defined, managed and checked on a per variable basis. If integrities are violated exception handling takes place. Usually, exceptions are handled be rolling back work, executing a compensating transaction if defined or proceed with another sub-transaction if part of a contingent transaction.

Layer L_2 (Control flow and transactional spheres) describes tasks together with control flow patterns which are the building blocks of (sub-)transactions and the control flow perspective of the workflow description. Aside *basic* sub-transactions build upon sequences of tasks, we support different transaction types to cope with diverse application requirements. For example *contingent*, *non-vital*, and *compensating* (sub-)transactions build upon tasks and appropriate control flow structures. The different types are explained in the following.

Basic transactions T_i have ACID properties and ensure serializability and recoverability as known from standard database transactions.

Nested transactions T_i allow for composing transactions from different sub-transactions $T_{i,j}$ even from other nested transactions.

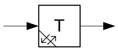
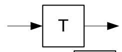
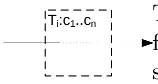
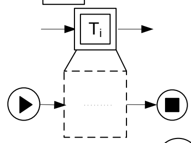
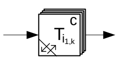
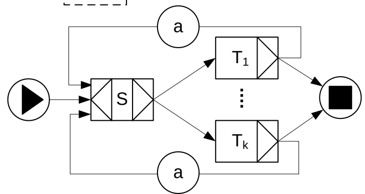
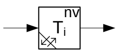
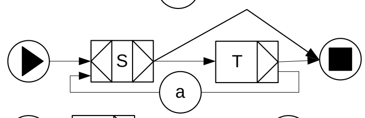
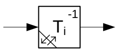
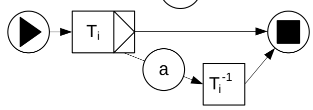
Contingent transactions $T^C_{i,k}$ are a set of transactions $T_{i,1}$ to $T_{i,k}$ out of which only one transaction must succeed. Situations where different, alternative ways of performing a business transaction exist, contingent transactions are the model of choice. In Fig. 2 the task E represents a set of contingent tasks or sub-transactions within the transactional sphere. E has three alternative implementations out of which only one needs to commit for E to be committed.

Non-vital transactions T_i^{NV} may fail. That means the result is an option but not necessary for the overall success. An abort of non-vital transaction has no effect on the transactional sphere it is part of. Nevertheless, atomicity has to be ensured for non-vital transactions, too, i.e. partial results of an aborting non-vital transaction have to be undone.

Compensated transactions are actually a pair of a "normal" transaction T_i and a compensation T_i^{-1}. Normally, the compensation can be done by executing an inverse transaction which rolls back work. Sometimes in practice, it is impossible to withdraw the real impact of a transaction. Then some kind of sufficient compensation specified by the application has to be done. Task D in Fig. 2 is a compensated task which consists of a regular task D and an inverse task or compensation D^{-1} which gets executed if D fails.

Read-only transactions are used for some kind of hypothetic transactions. Changes are just made within the scope of the transaction and kept locally.

Table 1. Concepts of *tx+YAWL* transactional workflows and their YAWL semantics

tx+YAWL	Modeling Concept	YAWL Representation
	Basic (ACID) transaction T.	
	Transactional sphere T_i and enforced set of constraints c_i over a sub-workflow.	
	Contingent transaction $T^C_{i,k}$ consisting of a set of sub-transactions $\{T_1..T_k\}$. System task S arbitrarily give one sub-transaction after another a try. If one succeeds, the contingent transaction commits.	
	Non-vital Transaction T^{NV}, system task S can decide to give T^{NV} several tries or do terminate.	
	Transaction T with compensation T^{-1}, which is executed if T fails or gets aborted.	

These transactions are not allowed to write results back to the data sources. So, they use write policy *no write*.

Non-transactional processing allows to directly work with external data sources. Read and write operations are unconditionally, immediately executed. The application is expected to ascertain integrity in this mode.

Regions of the workflow which are transactional consist of a simple task, a sequence of simple tasks, or are build upon composite tasks. They are called in our approach *transactional spheres*, describing the fact, that they build a sphere within the flow of control which is under transactional control. In fact they are based on the open nested transaction concept. Transactional spheres have:

- A transaction type {*non-transactional, basic, read-only, non-vital, compensating, contingent*}. The different types and their YAWL counterpart are shown in Table 1.
- A read and a write set of external variables $readset(T_i), writeset(T_i)$
- A transaction T is a partial order of steps (actions) of the form $r(x)$ or $w(x)$, where $x \in readset(T_i) \cup writeset(T_i)$ (set of data objects associated with the external variable) and reads and writes as well as multiple writes applied to the same object are ordered. We write $T = (op, <)$ for transaction T with step set op and partial order $<$. The order of read operations and write operations for a single task or net is evaluated in the order they appear in the parameter list of the net or task.

– The semantics or interpretation of a certain step, p_j, of T: If $p_j = r(x)$, then interpretation is assignment $v_j := x$ to local variable v_j If $p_j = w(x)$, then interpretation is assignment $x := f_j(v_{j1}, ..., v_{jk})$. with anonymous function f_j and $j_1..j_k$ denoting T's prior read steps.
– A set of integrity constraints defined over the external variables.

Constraints (e.g. c_1, c_2 in Fig. 2) are checked using the XQuery expressions over workflow variables. Reactions to constraints violations, failures or transaction aborts are dictated by exception handling policies and depend on the transaction type.

The *compensation* is done by executing an associated compensating or inverse transaction. A *rollback* is done for basic (or ACID) transactions. The decision whether to abort or ignore transaction failures depends on the transaction type of the current context. If the current transaction is the top-level transactional sphere, the transaction is aborted, which in turn may cancel the whole case. Failure within non-vital transaction only result in a local rollback and effect no upper-level transactions or spheres. In case of contingent transaction, only one out of the contingent set transactions must succeed, i.e. local aborts a kept local as far as one sub-transaction succeeds.

Usually, it is assumed that the read and write policies within a transactional sphere are *consistent read* and *write*. Accordingly, outside a transactional sphere the policies assumed to be *immediate read* or *immediate write*. If the user bypasses these implications, e.g choosing *immediate read* within a transactional sphere the isolation property may gets violated. Then, effects of this violation have to be controlled on the application level.

Layer L_3 (Case level and global transactions) is a layer of cases and associated global transactions. If a global transaction (sphere) fails, exception handling is done on case level, i.e. it may cause the cancellation of the case. A case may contain several transactional spheres which are independent with respect to their semantics but can interfere each other, e.g. if the first global transaction in a sequence fails and causes the case to be cancelled. The mapping between L_3 and L_2 is defined by the composition or nesting of transactional spheres into a global transaction (sphere) representing the workflow instance or case.

In the next section we will concentrate on how Layers L_0 and L_1 are implemented by our Data Access Framework (DAF) within the YAWL engine.

4 Implementing *tx+YAWL* Concepts with YAWL

To begin with we will explain how to transform *tx+YAWL* elements into a compliant YAWL representation. Then, we introduce a Data Access Framework (DAF) to support operations of Layer L_1 and Layer L_0. The DAF is an extension of the YAWL workflow engine. Additionally, the DAF implements transactional concepts like integrity constraint checking described at Layer L_2.

Layer L_3 + L_2, tx+YAWL modeling concepts are not natively supported by the YAWL engine. First, these concepts must be transformed into a pure

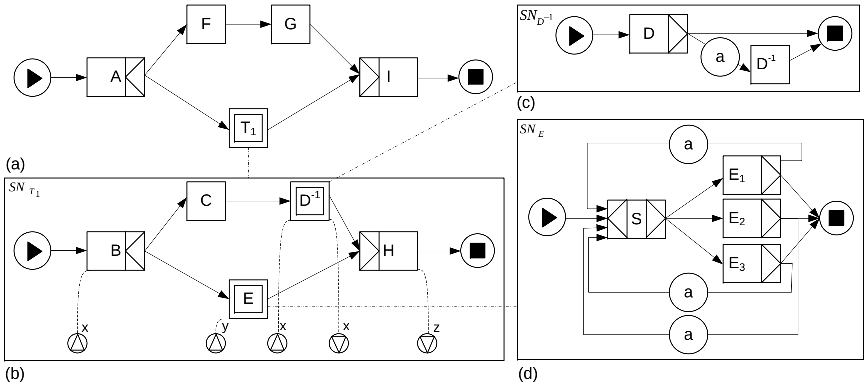


Fig. 3. Rewriting *tx+YAWL* into a YAWL Model

YAWL model. The implementation of Layer L_2 is done by utilizing the transformation rules from Table 1.

To exemplify the transformation process, we have applied the rules to the example from Fig. 2. The result is shown in Fig. 3 (a)–(d): The top-level transaction in Fig. 2 is represented using a YAWL root workflow net (cf. Fig. 3 (a)). We start with applying the rule for a nested transaction on transaction T_1. This results in the corresponding composite task T_1 of Fig. 3 (a). Then the composite task T_1 is decomposed into subnet SN_{T_1}, which contains the transactional tasks B, C, D, E and H of the original transactional region (cf. Fig. 3 (b)). Within the new subnet SN_{T_1} the *compensating task* D^{-1} is transformed into the new composite task D^{-1} using the rule for transactions with compensation. The corresponding subnet $SN_{D^{-1}}$ is can be seen in Fig. 3 (c). It contains task D and its compensation D^{-1}. Further on, the contingent transaction E_3 is transformed into composite task E using the rule for contingent transactions. The resulting subnet SN_E contains the three contingent transactions/tasks E_1, E_2 and E_3 and is depicted in subfigure (d). Net variables of a subnet are task variables of the corresponding composite task in YAWL. Therefore, external variables of the transactional tasks D^{-1} and E_3 appear as task parameters of the composite tasks D^{-1} and E within the subnet SN_{T_1}.

Since integrity constraints are tied to transactional spheres, they may only be verified if transactions are active. To ensure this, we introduce a data controller definition which checks constraints for active transactions. A data controller DC = (E, C, n, t, s) is defined as:

- E is a set of external variables $(e_1, e_2, \ldots, e_n)$, associated with data sources.
- C is a set of dynamic constrains $(c_1, c_2, \ldots, c_n)$ defined on E.
- n is the net the DC belongs to.
- t defines the *transaction type* of the associated transaction.
- s is the state of the DC with $s \in \{active, inactive, deactivated\}$.

The DC belongs to the net n of a transactional sphere, e.g. task T_1 in Fig. 3 (a). A DC may not always be active. In YAWL, state transitions are bound to input and output condition of a net. The initial state s of a DC is inactive. The DC will be activated if the corresponding sphere or rather a net is activated. Then a state transition from inactive to activated take place. Respectively, the DC will be deactivated if the corresponding net will be terminated. However, the data for the DC is not generated at run time, but when the model is instantiated.

The set of constraints C contains the constraints c_1 and c_2 in the example process. Consequently, the set of external variables (E) consists of $\{x, y, z\}$. The transaction type of T_1 is *basic*.

Layer L_1, External variables are not supported by YAWL directly. So, the YAWL engine got extended with the Data Access Framework (DAF) (cf. Fig. 4). The framework connects external bound YAWL variables with data sources. The DAF is responsible for transactional concepts like dynamic integrity constraints described in Layer L_2 as well. Now, the DAF components shown in Fig. 4 are explained in detail.

The *Data Gateway* (DG) is just a simple interface within the YAWL engine where requests to external variables are catched and forwarded to services outside the YAWL engine. It simply passes all read and write requests, i.e. operations of Layer L_1, to the external Data Source Manager and returns results to the YAWL engine.

The *Data Source Manager* (DSM) performs a pre-processing of the received request to control its further processing. The variable mapping m (cf. Sec 3) of the external variable is used to select the plug-in which have to be invoked. To select all required services the read and write policies are extracted as defined in Layer L_1. Also the transaction types and constraints as defined by a data controller (DC) at Layer L_2 are evaluated to configure further processing.

The *Data Integration Chain* (DIC) combines services to process the requests. Typical services are Synchronizing Manager (SM), Transaction Manager (TM) or Recovery Manager (RM). The services have to be deployed at runtime to provide different configurations. For example, with these configurations TM and RM are activated only when consistent read or write and recoverability is required. A service processes the request and passes it on to the next service until

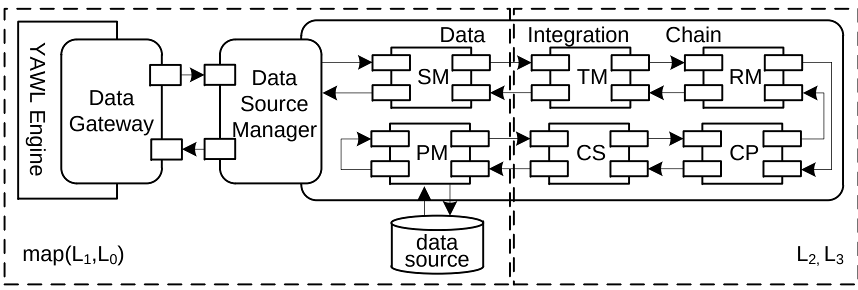


Fig. 4. Data Access Framework Architecture

the plug-in at the bottom of the chain is invoked. After the read ($x_j = T_i.r(x)$) or write ($T_i.w(x_j)$) operation were performed by the plug-in the result will be sent back to the DG in reverse order through the chain. Finally, the Data Gateway in turn passes the result back to the YAWL engine.

The Constraint Service (CS) handles all constraints and is activated by an active DC. The place where constraints are checked, depends on the features the actual used data source supports. Integrity is checked by the CS itself before and after invoking a plug-in. Constraints can be also checked by the data source itself, e.g. using triggers. Nevertheless, the violation has to be catched by the CS. In [19] different exception types and handling strategies for workflows are presented as patterns. A pattern describes a threefold plan to react on an exception. The CS deploys common exception handling features supported by the YAWL engine, especially it uses the Exlet approach [2]. It can detect and handle different kinds of process exceptions. This are for example canceling, suspending, completing, failing and restarting a task, case and/or specifications. Exlets also can directly specify compensatory tasks which eases the implementation of compensating transactions.

Layer L_0, Uniform data access is provided by plug-ins. The **Plug-In Manager** (PM) implements the data access to the data source, which is described in Layer L_0. It calls an appropriate plug-in after the access to the data source has been approved by each service in the DIC. Previously, the *Connection Pool* service (CP) has instantiated the plug-in using *pId* extracted from the variable mapping. The connection pool provides simultaneous data source access and connection management for plug-ins. This the concurrent access of data sources established by various process instances easy.

To allow for easy integration of different plug-ins, we have defined a common plug-in architecture, which is divided into three components: *plug-in driver*, *connection* and *statement*. The *plug-in driver* implements the interface exactly for one data source type and will be managed by a *plug-in driver manager*, which administer all registered plug-in drivers. The *plug-in connection* object is a logical connection to the plug-in, which encapsulate all session information needed by the framework for plug-in communication. If the data source plug-in supports transactions, the $T_i.c()$ and $T_i.a()$ commands are available here and initiate a local commit or rollback at the data source. A *plug-in statement* executes the read $T_i.r(x)$ and write $T_i.w(x)$ operations itself. It sends the request to the data source using the plug-in driver, as well as translating the query (*map*) into the supported data source language and transforming data accordingly as results get returned.

The Data Access Framework Prototype implements our approach in a "proof-of-concept" manner. It is done by a set of Java classes and services extending the YAWL engine. We extended the *YAWL Editor* (modeling tool) to define the mapping between external variables and plug-ins as an XML schema structure within a YAWL net. Other *tx+YAWL* modeling concepts are not supported at the moment.

The data controllers (DC) are defined as XML schema structures within a YAWL net. In the Perikles project [3] the prototype was used for integrating and accessing transactional and non-transactional data sources, e.g. clinical informations systems. Actually, the DAF supports plug-ins for accessing XML native stores and HL7 sources common in healthcare environments. So, the YAWL system is aware of changes to data made externally. Furthermore, It can also write consistently back data used and altered in the workflow system to external data sources. A generic XML-object-relational mapper plug-in is under construction.

5 Summary and Future Work

Supporting access to external data and transactional workflows is still a challenge. We presented an approach which extents an existing workflow engine and the corresponding workflow model by adding external data access, integrity constraints, exception handling methodology, and a transaction concept based on multi-layered transactions. In Sec. 4 the architecture of the Data Access Framework and its design rationals were outlined. The framework is based on the YAWL workflow engine. A prototype implementation is used by a tracking and OR-management system for peri-operative centres in the Perikles project.

The Data Access Framework can not only be used within our transactional workflow framework but everywhere were external data access has to be integrated into a workflow engine, namely YAWL. The Data Access Framework supports different kinds of update strategies to work with transactional, recoverable resources but can deal with non-transactional data stores, too. Data in external source is wrapped by plug-ins and represented as XML data, which can manipulated using mapping defined by XPath and XQuery expressions.

Future work will focus on improved design concepts for modeling transactional workflows based on patterns and anti-patterns for the composition of transactional spheres. For a better understanding of adequate isolation levels of workflow transaction a detailed analysis of the correspondence of data and control flow is under way.

Last but not least, we want the *YAWL Editor* to directly support our modeling concepts. The designer should be assisted in defining transactional spheres, external data mappings and integrity constraints in combination with the control flow definition. Hence, editor extension techniques will be exploited.

References

1. van der Aalst, W.M.P., ter Hofstede, A.: YAWL: Yet another workflow language. Information Systems 30(4), 245–275 (2005)
2. Adams, M., ter Hofstede, A., van der Aalst, W., Edmond, D.: Dynamic, Extensible and Context-Aware Exception Handling for Workflows. In: OTM 2007, LNCS, vol. 4803, pp. 95–112. Springer Berlin / Heidelberg (2007)
3. Bandt, M., Kühn, R., Schick, S., Meyer, H.: Beyond Flexibility – Workflows in the perioperative Sector of the Healthcare Domain. In: WowKiVS 2011. vol. 37. Electronic Communications of the EASST (2011)

4. Beeri, C., Schek, H.J., Weikum, G.: Multi-Level Transaction Management, Theoretical Art or Practical Need? In: EDBT '88. LNCS, vol. 303, pp. 134–154. Springer Berlin/Heidelberg, London, UK (1988)
5. Ceri, S., Grefen, P.W.P.J., Sanchez, G.: WIDE: A Distributed Architecture for Workflow Management. In: RIDE 1997. pp. 76–79 (1997)
6. Eder, J., Lehmann, M.: Synchronizing Copies of External Data in Workflow Management Systems. In: CAiSE 2005, LNCS, vol. 3520, pp. 248–261. Springer Berlin / Heidelberg (2005)
7. Eder, J., Lehmann, M.: Workflow Data Guards. In: OTM 2005, LNCS, vol. 3760, pp. 502–519. Springer Berlin / Heidelberg (2005)
8. Elmagarmid, A.K.: Database Transaction Models For Advanced Applications. Morgan Kaufmann, San Mateo, CA (1992)
9. Garcia-Molina, H., Salem, K.: Sagas. In: SIGMOD Conference. pp. 249–259 (1987)
10. Grefen, P.W.P.J.: Transactional Workflows or Workflow Transactions? In: DEXA 2002. LNCS, vol. 2453, pp. 327–349. Springer Berlin/Heidelberg (2002)
11. Grefen, P.W.P.J., Vonk, J., Apers, P.M.G.: Global transaction support for workflow management systems: from formal specification to practical implementation. The VLDB Journal 10(4), 316–333 (2001)
12. Grefen, P.W.P.J., de Vries, R.R.: A Reference Architecture for Workflow Management Systems. Data and Knowledge Engineering 27(1), 31–57 (1998)
13. Hoffner, Y., Ludwig, H., Grefen, P.W.P.J., Aberer, K.: CrossFlow: integrating workflow management and electronic commerce. SIGecom Exchanges 2(1), 1–10 (2001)
14. Künzle, V., Reichert, M.: Towards Object-Aware Process Management Systems: Issues, Challenges, Benefits. In: BMMDS/EMMSAD. pp. 197–210 (2009)
15. Montagut, F., Molva, R., Golega, S.T.: The Pervasive Workflow: A Decentralized Workflow System Supporting Long-Running Transactions. IEEE Transactions on Systems, Man, and Cybernetics, Part C 38(3), 319–333 (2008)
16. Muro, S., Kameda, T., Minoura, T.: Multi-version concurrency control scheme for a database system. Journal of Computer and System Sciences 29(2), 207–224 (1984)
17. Pottinger, S., Mietzner, R., Leymann, F.: Coordinate BPEL Scopes and Processes by Extending the WS-Business Activity Framework. In: OTM 2007, LNCS, vol. 4803, pp. 336–352. Springer Berlin/Heidelberg (2007)
18. Rinderle, S., Reichert, M.: DataDriven Process Control and Exception Handling in Process Management Systems. In: CAiSE 2006, LNCS, vol. 4001, pp. 273–287. Springer Berlin / Heidelberg (2006)
19. Russell, N., van der Aalst, W., ter Hofstede, A.: Workflow Exception Patterns. In: CAiSE 2006, LNCS, vol. 4001, pp. 288–302. Springer Berlin / Heidelberg (2006)
20. Russell, N., ter Hofstede, A., Edmond, D., van der Aalst, W.: Workflow Data Patterns: Identification, Representation and Tool Support. In: ER 2005, LNCS, vol. 3716, pp. 353–368. Springer Berlin / Heidelberg (2005)
21. Sadiq, S., Orlowska, M., Sadiq, W., Foulger, C.: Data flow and validation in workflow modelling. In: ADC '04. pp. 207–214. Australian Computer Society, Inc. (2004)
22. Wächter, H., Reuter, A.: The ConTract Model. In: Database Transaction Models for Advanced Applications, pp. 219–263. Morgan Kaufmann (1992)
23. Weikum, G., Vossen, G.: Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, USA (2001)
24. Worah, D., Sheth, A.P.: Transactions in Transactional Workflows. In: Advanced Transaction Models and Architectures, pp. 3–34. Kluwer (1997)