Synchronizing Copies of External Data in Workflow Management Systems

Johann Eder and Marek Lehmann

Alps-Adria University Klagenfurt,
Department of Informatics-Systems
{eder, marek}@isys.uni-klu.ac.at

Abstract. Workflow management systems integrate applications and data resources in business processes. Frequently they have to keep local copies of data in the so called workflow repository. We introduce and define synchronization policies for these copies with their external data sources ranging from the provision of fully synchronized replications of external data sources to unsynchronized storage of read results. We analyze in detail the effects of combining various pull and push policies and show in an example how these policies can be used in different situations.

1 Introduction

Workflow management systems (WfMSs) [7, 15] are instrumental in automating business processes, enacting activities due to business logic represented in workflow definitions and documenting the execution history. Workflow systems also frequently integrate separate information systems, using and manipulating data from various sources. Nevertheless, research on access to and integration of data by workflow systems is scarce compared with the intensive research on the control flow aspects [3].

A WfMS coordinates the execution of different activities which can access different data sources. To determine the state transitions of a workflow (e.g. transition conditions) the WfMS uses *workflow relevant data* [15], which may be accessed both by the WfMS and the applications. Usually, the WfMS needs direct access to the workflow relevant data and, therefore, stores these data in an internal workflow repository. This raises the issue of synchronizing the replicas in the workflow repository with their originals, which typically may be altered by other applications unnoticed by the workflow system. Being unaware of possible synchronization problems (e.g. decisions based on stale copies of data, lost updates, unawareness that data needed for past decisions has changed, timely propagation of changes to copies, etc.) leads to potential application errors.

Synchronization of copies of external data in workflow repositories is, however, not trivial. Workflows might be long-running. Some copies need be refreshed, others not. Changes might be local or might need externalization, etc. Mere replication management is not sufficient, since it is frequently not adequate and more sophisticated synchronization policies are required to allow workflow

O. Pastor and J. Falcão e Cunha (Eds.): CAiSE 2005, LNCS 3520, pp. 248–261, 2005.

designer to achieve the required behaviour. We would like to provide synchronization based on policies, such that the designer can chose a synchronization policy and thus select the exact semantics and properties of data access operations.

Currently, access to external data is typically provided by invoking automated activities or other means which are not a part of the WfMS (e.g. Automatic Steps and Scripts in Staffware [13] or Java code attached to the activities in @enterprise [8]). The problem is that the programmers have to hardcode the mechanisms for accessing external data in automated activities. A workflow definition can contain a hundred or more activities, use data from dozens of external data sources like legacy systems, web services etc. Therefore, additional activities responsible only for keeping a copy of some data in the workflow repository increases the complexity of the workflow definition up to date. Such workflow definitions become difficult to maintain, in particular, when it is necessary to add new data sources or replace exiting ones. It may not always be clear whether an activity overwrites a local copy with the data from an external source or pushes local changes from the workflow repository to the environment. If such activities are tailored to one workflow definition, then it is difficult to reuse them in an other definition. These are activities which serve a purely technical purpose and do not represent any step in the business procedure. These activities, furthermore, have subtle relationships and interdependencies with other activities increasing the complexity of maintenance and evolution.

We propose an abstraction layer for WfMS which allows transparent access to any data source [6]. This is achieved by *data access plug-ins*, reusable and interchangeable wrappers around external data sources, which present to the WfMS the content of underlying data sources, manage the access to it, and provide thus also the basic synchronization operations. The functionality of the external data source is abstracted in these plug-ins. We introduced the workflow language WDL-X (an extension of WDL [5]) which makes use of these plug-ins and provides a coherent seamless data view of workflow data and external data.

In this paper we extend WDL-X with a transparent and manageable mechanism for maintaining copies of external data in a workflow repository.

The remainder of this paper is organized as follows: Section 2 describes new mechanisms for copying external data into the workflow repository in our workflow definition language WDL-X. In Section 3 we discuss in detail different maintenance policies for such copies. An example application of these policies is presented in Section 4. We discuss related work in Section 5 and finally draw some conclusions in Section 6.

2 Copying External Data into a Workflow Repository with the WDL-X

2.1 WfMS Architecture Including Data Access Plug-Ins

The work we present here is a continuation of our work in workflow systems. Our workflow system *Panta Rhei* [5] used a form-flow metaphor to provide access to

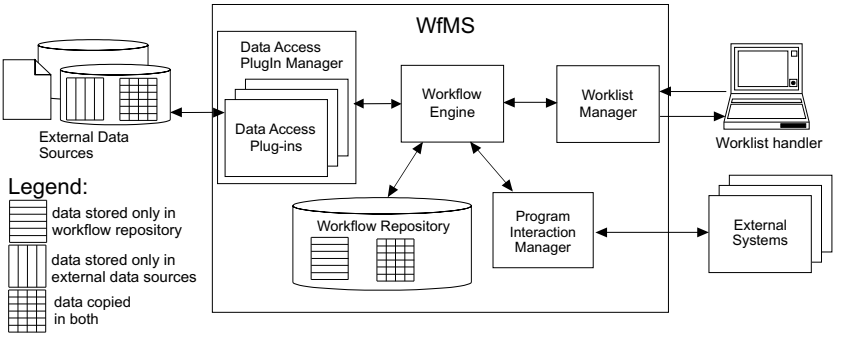

Fig. 1. WfMS architecture

workflow specific data. In [6] we proposed a uniform treatment of all kinds of business data in a workflow definition. We used XML as data access language in our workflow definition language WDL-X [6], which replaced the earlier WDL [5]. The transparency of data location and logical and physical data independence of workflow systems is achieved in WDL-X by using specialized wrappers around external data sources called data access plug-ins. In this paper we describe an extension of the WDL-X and the WfMS functionality, which allows to manage copies of external data in the workflow repository.

The data access plug-ins introduced in WDL-X provide an abstraction layer between the WfMS and the actual storage and format of the data. WDL-X uses XML as unifying data format which allows to integrate data from external sources, data exchange between workflow systems or web services, and internal data handling. A data access plug-in is a wrapper presenting to the WfMS the content of external data sources as documents of a predefined XML Schema type. Data sources might be legacy systems, relational or object relational DBMSs, files stored in a file system or native XML databases etc., both in- and outside of an enterprise. A data access plug-in exposes to the WfMS a simple interface for the creation, selection, update and deletion of an XML document in a collection of many documents of the same XML Schema type and the evaluation of the XPath expressions on a selected document. The task of the plug-in is to translate these operations on XML documents to the underlying data sources. A workflow designer can specify in a workflow definition, which document should be accessed by which data access plug-in. The WfMS calls the associated data access plug-in each time an operation on a specified XML document is performed. This provides the transparency of actual data location and allows to use external data to control the flow of a workflow. Data access plug-ins are collected in a library and might be used in several workflows.

The overall WfMS architecture is presented in Fig. 1. The workflow engine is responsible for execution of instances of workflows based on process definitions. Both process definitions and instances are stored in the workflow repository. The worklist manager is responsible for the worklists of human actors and for the interaction with the client software (worklist handlers). The program interaction

manager calls programs implementing automated activities. The workflow engine can access data stored either in the workflow repository or in external data sources. The engine uses data access plug-ins to transparently access external data.

2.2 Workflow Repository and External Data

Sometimes it is necessary that a WfMS works on a copy of external data stored in the workflow repository. The synchronization needs of these copies are quite different, depending on the application. In some cases it may be required that the WfMS operates on a stale copy, e.g. during a processing of some application the marital status of an applicant at the starting time of the process is important, even if the status has changed in the meantime. In other cases it may be required to restrict the possibility to externalize modifications made locally to the copies in the workflow repository, e.g. these modifications cannot be propagated to the original data until they are proven correct. Therefore, we argue that a WfMS needs a transparent and manageable mechanism for making and synchronizing copies of external data. We propose to use the functionality of data access plug-ins to copy external data into the workflow repository combined with a set of different policies for synchronizing these copies.

External data and their copies in the workflow repository are independent. External data may be read and updated by external systems and their copies in the workflow repository may be read and updated by the workflow instances. A workflow designer has to define in a WDL-X script a desired policy for synchronizing a copy with its original:

- Refresh policy for a copy in the workflow repository:
  - `refresh` – the WfMS automatically refreshes the copy with external data before each read access to the copy made within a process instance,
  - `refreshOnDemand` – the workflow designer has to use explicitly the WDL-X command `forceRefresh` to refresh the copy,
  - `doNotRefresh` – keeps the copy isolated from the changes made in the original data.
- Push policy for changes made locally in the workflow repository:
  - `immediatePush` – the WfMS automatically pushes to the environment each change made to the copy in the workflow repository,
  - `pushOnDemand` – the workflow designer has to use explicitly the WDL-X command `forcePush` to push the copy,
  - `doNotPush` – keeps all the changes locally in the workflow repository.

To refresh and to push a copy the WfMS uses a data access plug-in associated with a document copied from an external data source. The refresh mechanism depends on the capabilities of a data access plug-in and an underlying data source. A *passive data access plug-in* is not capable of monitoring original data for changes and, therefore, the WfMS has to use the plug-in to reload the original document each time it refreshes a copy. An *active data access plug-in* ia able to monitor original data for changes (e.g. using triggers in an underlying database) and reload the original data only if these data were changed.

2.3 Workflow Definitions with Data Access Plug-Ins

In WDL-X the process variables are bound to XML documents of predefined XML Schema types. A variable can be bound either to an existing document by the WDL-X command `openDocument` or to a new document created during a process execution. If no document is bound to a variable, then the variable is not initialized. The process variables are local to a process instance. Activities can receive parameters in one of the following modes: *IN* - as an input parameter, *OUT* - as an output parameter and *INOUT* - as an in- and output parameter. Conditions may be tested on a process variable, e.g. to evaluate the control flow of a workflow.

Based on this description an example variable declaration in WDL-X looks as follows:

```
documents customerData : customerType accessedBy customerDbPlugIn
                         doNotRefresh doNotPush;
```

The semantics of this example is: In a workflow definition a variable named `customerData` of the XML Schema type `customerType` is declared. The variable should be bound after the initialization to an XML document created as a copy of a document accessed by a data access plug-in named `customerDbPlugIn`. The copy stored in the workflow repository should not be refreshed and the changes made to the copy should not be pushed back to the original document.

3 Policies for Synchronizing Copies of External Data in WfMSs

Documents bound to process variables may be stored in the external data sources, be stored in the workflow repository, or the workflow repository contains a copy of external data (see Fig. 1). In this paper we focus only on the latter case.

3.1 Basic Operations on Documents

We refer to the set of documents stored in the external systems as X, and $\tilde{X}$ denotes the set of documents stored in the workflow repository:

$$X = \{x : x \text{ is a document stored in an external data source}\} \tag{1}$$

$$\tilde{X} = \{\tilde{x} : \tilde{x} \text{ is a document stored in the workflow repository}\} \tag{2}$$

On the elements of both X and $\tilde{X}$ read and write operations are allowed. The operation $r(x)$ returns the actual value of the document x. The operation $w(x, val)$ writes the value val to the document x. We define the preconditions and postconditions to the operation $r(x)$ as follows:

$$\begin{aligned} & r(x) \\ & \quad \text{preconditions} : \exists x \\ & \quad \text{postconditions} : \exists x : x = r(x) \end{aligned} \tag{3}$$

We define the preconditions and postconditions to the operation $w(x, val)$ as follows:

$$w(x, val) \quad \begin{array}{r} \text{preconditions : none} \\ \text{postconditions} : \exists x : x = val \end{array} \tag{4}$$

Using these two operations we can pull a document $x \in X$ from the external data source to the workflow repository or push a document $\tilde{x} \in \tilde{X}$ from the workflow repository to an external data source. The operation $op\_pull(\tilde{x}, x)$ is equivalent to $w(\tilde{x}, r(x))$. The operation $op\_push(\tilde{x}, x)$ is equivalent to $w(x, r(\tilde{x}))$.

$$op\_pull(\tilde{x}, x) \equiv w(\tilde{x}, r(x)) \quad \begin{array}{r} \text{preconditions : } \exists x \in X \\ \text{postconditions} : \exists \tilde{x} \in \tilde{X} : \tilde{x} = r(x) \end{array} \tag{5}$$

$$op\_push(\tilde{x}, x) \equiv w(x, r(\tilde{x})) \quad \begin{array}{r} \text{preconditions : } \exists \tilde{x} \in \tilde{X} \\ \text{postconditions} : \exists x \in X : x = r(\tilde{x}) \end{array} \tag{6}$$

If one of either operations $op\_pull(\tilde{x}, x)$ or $op\_push(\tilde{x}, x)$ was performed, then we say that documents $x \in X$ and $\tilde{x} \in \tilde{X}$ are in the copy relation K. The document x is called the original of $\tilde{x}$ and the document $\tilde{x}$ is called a copy of x:

$$(x, \tilde{x}) \in K, K \subseteq X \times \tilde{X} \tag{7}$$

The copy relation K has the following properties:

$$\forall \tilde{x} \in \tilde{X} : \tilde{x} \text{ is a copy of at most one } x \in X \tag{8}$$

$$\forall x \in X : x \text{ is the original of none or many } \tilde{x} \in \tilde{X} \tag{9}$$

3.2 WDL-X Operations on Documents

The WDL-X commands and operations mentioned in Sec. 2 can be now defined with the presented basic operations. Passing a variable to a simple activity in *IN* or *INOUT* mode, or testing a condition on a variable is equivalent to a read operation on a document bound to the variable. Receiving a variable from an activity in *OUT* or *INOUT* mode is equivalent to a write operation on a document bound to the variable. If the variable was not bound to any document, a new document is created and bound to the variable. The exact semantics of these WDL-X operations depends on a chosen policy. In each policy the WDL-X command `openDocument` will create a copy of an external document in the workflow repository and will bind a variable to a newly created document. The refresh policies influence WDL-X read and `forceRefresh` operations as presented in Tab. 1. The push policies influence WDL-X write and `forcePush` operations as presented in Tab. 2.

Table 1. Semantics of WDL-X operations in the refresh policies

WDL-X operations	Basic operations		
	`refresh`	`refreshOnDemand`	`doNotRefresh`
`openDocument(var,x)`	*op_pull*$(\tilde{x}, x)$ variable `var` will be bound to a new copy $\tilde{x}$ of x		
`r(x̃)`	if $\exists x \in X : (x, \tilde{x}) \in K$ then *op_pull*$(\tilde{x}, x)$ $r(\tilde{x})$ else raise error	$r(\tilde{x})$	$r(\tilde{x})$
`forceRefresh(x̃)`	not allowed	if $\exists x \in X : (x, \tilde{x}) \in K$ then *op_pull*$(\tilde{x}, x)$ else raise error	not allowed
`w(x̃,val)`	not influenced by these policies		
`forcePush(x̃)`			

Table 2. Semantics of WDL-X operations in the push policies

WDL-X operations	Basic operations		
	`immediatePush`	`pushOnDemand`	`doNotPush`
`openDocument(var,x)`	*op_pull*$(\tilde{x}, x)$ variable `var` will be bound to a new copy $\tilde{x}$ of x		
`r(x̃)`	not influenced by these policies		
`forceRefresh(x̃)`			
`w(x̃,val)`	$w(\tilde{x}, val)$ *op_push*$(\tilde{x}, x)$	$w(\tilde{x}, val)$	$w(\tilde{x}, val)$
`forcePush(x̃)`	not allowed	*op_push*$(\tilde{x}, x)$	not allowed

The policies `refreshOnDemand` and `pushOnDemand` imply that in a WDL-X script there should be at least one `forceRefresh` and `forcePush` command respectively. Otherwise the policies degenerate to `doNotRefresh` and `doNotPush` respectively. Both problems are detected during the compile time and warnings are generated.

3.3 Combinations of Refresh and Push Policies

A workflow designer may define a different combination of refresh and push policies for each document copied into the workflow repository. In this section we shortly discuss each of the nine possible combinations. Each policy combination has to be analyzed separately for the documents copied into the workflow repository with the `openDocument` command and for the documents created first locally in the workflow repository.

1. In the policy combination `refresh immediatePush` documents copied into the workflow repository will be refreshed before each read operation and pushed after each write operation. Documents created in the workflow reposi-

tory will be pushed immediately after the first write operation and afterwards the situation is the same as for the copied documents.

2. In the policy combination `refresh pushOnDemand` documents copied into the workflow repository from external sources will be refreshed before each read operation. Local modifications of the copy in the workflow repository will be pushed only on an explicit `forcePush` command in a WDL-X script. A workflow designer has to be aware that, if in a sequence of WDL-X operations there is a read operation between a write operation and the `forcePush`, then the effect of this write operation will be overwritten by the refresh policy. If a new document was created in the workflow repository then this policy combination will lead to an error, when before a read operation the refresh policy tries to pull the document, which has not yet been pushed by the `forcePush` command.
3. In the policy combination `refresh doNotPush` documents copied into the workflow repository from external sources will be refreshed before each read access. Effects of all write operations will never be pushed and they will be always overwritten by the pull operations performed before subsequent reads. Any attempt to read a new document created locally in the workflow repository will always cause an error. Therefore, in this policy combination no write operations are permitted and the copy is *read only*.
4. In the policy combination `refreshOnDemand immediatePush` any write operation on a document copied into the workflow repository from an external source will be pushed immediately. Documents created locally in the workflow repository will be pushed immediately after the first write operation and afterwards the situation is the same as for the copied documents. To refresh the copy a workflow designer has to use explicitly the `forceRefresh` command.
5. In the policy combination `refreshOnDemand pushOnDemand` a workflow designer has full control by specifying when to pull and push a document copied into a workflow repository by using explicitly `forceRefresh` and `forcePush` commands respectively. The designer has to be aware that a new document created locally in the workflow repository has to be pushed before it can be pulled (i.e. `forcePush` has to precede `forceRefresh`). Otherwise the `forceRefresh` command will cause an error.
6. In the policy combination `refreshOnDemand doNotPush` any write operation on a document copied into the workflow repository from an external source will never be pushed and a workflow designer has to be aware that the `forceRefresh` command will always overwrite the effects of a preceding write operation. Any attempt to use `forceRefresh` on a document created locally in the workflow repository will always cause an error. Therefore, for all documents created only locally in the workflow repository this policy combination degenerates to the policy combination `doNotRefresh doNotPush`. Such a situation can be detected during the compile time, because in the WDL-X script there will be no `openDocument` command for the documents local to the workflow repository.

7. In the policy combination `doNotRefresh immediatePush` a document copied into the workflow repository from an external data source is pulled only once after the WDL-X command `openDocument`. A document created locally in the workflow repository will be immediately pushed. All write operations are immediately pushed. In this policy combination a workflow instance works on a local copy and externalizes all modifications made to this copy.
8. The policy combination `doNotRefresh pushOnDemand` is similar to the policy combination `doNotRefresh immediatePush` with the difference that to push modifications made to a document in the workflow repository a workflow designer has to explicitly use the command `forcePush`. In this policy combination a workflow instance works on a local copy, which is externalized only on demand.
9. In the policy combination `doNotRefresh doNotPush` a document copied into the workflow repository from an external data source is pulled only once after the WDL-X command `openDocument`. Afterwards the copy is never synchronized. A document created only in the workflow repository is never pushed and always stays a local document.

4 Example

We illustrate our approach with a simplified example of a credit application processing workflow. The workflow graph representing the workflow definition is presented in Fig. 2. The rectangles represent activities. Each activity has a name and an assigned agent specified in brackets underneath the name. Arrows between activities represent the control flow. A circle with a question mark inside and outgoing arrows represents an exclusive choice, whereas a circle with incoming arrows represents a simple merge. The WDL-X definition of the workflow is presented in Fig. 3.

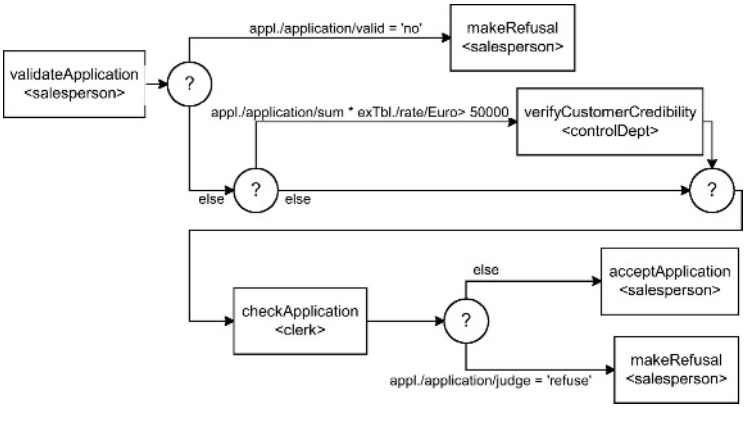

Fig. 2. Workflow graph of a credit application processing

```
1  process processCredit(IN appl  : creditApplicationType)
2  documents customerData : customerType
3                 accessedBy customerDbPlugIn refresh pushOnDemand,
4               exTbl : exchangeRateTableType,
5                 accessedBy centralBankPlugIn refresh doNotPush;
6  begin
7   salesperson validateApplication(appl);
8    if (appl./application/valid/text()='no')
9     salesperson makeRefusal(appl);
10   else
11    openDocument(customerData, appl./application/customer/@id);
12    openDocument(exTbl, appl./application/sum/@currency);
13    if (appl./application/sum/text() * exTbl./rate/Euro/text() > 50000)
14       controlDept verifyCustomerCredibility(customerData);
15       forcePush(customerData);
16    endif;
17    clerk checkApplication(apl,customerData);
18    if (appl./application/judge/text()='refuse')
19        salesperson makeRefusal(appl);
20    else
21        salesperson acceptApplication(appl,customerData);
22    endif;
23  endif;
24 end;
```

Fig. 3. WDL-X script definition of a credit application processing

A credit application is passed to the process as an input parameter (line 1). A salesperson validates the application for correctness. The applications which are not valid are rejected. A sum of the credit in a valid application may be given in one of several currencies (e.g. USD, JPY, GBP). If the sum of the credit recalculated into Euro exceeds a value of 50 000 EUR, then the control department of the credit institution has to check the credibility of the customer. In any case valid applications have to be checked by a clerk. According to his / her judgment the salesperson accepts or rejects the application.

In the process apart from the credit application are used two additional documents: customer data (line 2) and the exchange rate table for a currency of the credit application (line 4). Both documents are stored locally in the workflow repository as copies of correspondent documents received from external data sources by data access plug-ins. According to policies defined by a workflow designer the customer data should be refreshed before each access and local changes should be pushed only after an explicit `forcePush` command, and the exchange rate table should be refreshed before each read and never pushed (lines 3 and 5).

The document describing customer data is copied from the external database according to the customer's id given in the application (line 11) and the exchange rate table is copied for the currency describing the credit sum (line 12). The exchange table is automatically refreshed before recalculating into Euro and checking the credit sum (line 13). Checking of the customer's credibility (line 14) may change the customer data in a significant manner. These changes are

therefore pushed to the original data source (line 15). Changes possibly made to the customer data by any other activity are never pushed and always overwritten from the original source before each read access in the workflow, e.g. before checking the application (line 17) and accepting the application (line 21). No modifications to the exchange rate table are allowed.

The example illustrates the versatility of our simple yet powerful mechanism. Our synchronization policies can model large spectrum of requirements ranging from fully synchronized replicas, through read only data to data local just to the workflow repository. In the example on the one hand the policy combination `refresh pushOnDemand` gives the workflow designer a control when to externalize updates made to the local copy of data. On the other hand the policy combination `refresh doNotPush` allows to model read only data. The former case allows to push to the original source and to take into account in the further workflow processing only important and verified data modifications. This is particulary important if activities and subprocesses are reused in different workflow definitions. If a subprocess reused in a workflow definition modifies the data, this modification can be simple discarded by the refresh policy or preserved by the `forcePush` command. The latter case describes the situation when in a workflow are used data, which are completely managed outside the context of an active workflow instance. These data cannot be changed by the workflow instance, nevertheless, the control flow of the workflow depends on them. Such read only data are currency exchange rate (as in our example), a marital status of a person, credit card validity, position of an employee, tax rates, metal and oil prices and many more.

5 Related Work

Data aspects in workflow systems did not yet receive the same attention as process aspects. There were even questions whether workflows had lost sight of the dataflow [3]. The Workflow Management Coalition (WfMC) in its glossary [15] defines three classes of data in workflows. *Workflow control data* managed by a WfMS and describing workflow execution are not in scope of our interest here. *Application data* are managed by the applications supporting the process instance and generally are never seen by the WfMS. *Workflow relevant data* are used by the WfMS to determine the state transitions of a workflow (e.g. transition conditions). In the traditional WfMSs workflow relevant data must be stored in the workflow repository and the application data are beyond the WfMS. In [16] the WfMC defined environmental data, as data which may be accessed by workflow activities or used by the WfMS in the evaluation of conditional expressions in the same way as workflow relevant data. Unfortunately the WfMC does not specify in [16] anything else in this matter.

The analysis of commercial WfMSs shows that most of the activity programming is related to updating application databases [1]. Some products provide constructs for accessing external data sources. Typically these are specific elements that can be included in a process definition [11]. For example Staffware

provides Automatic Steps and script commands which enable specific items of data to be requested from external systems [13]. In @enterprise [8] to each activity can be attached Java code, which e.g. can fetch external data. In many systems (e.g. MQWorkflow [9]) access to external data occurs within individual simple activity implementation.

On the contrary we present a complete mechanism for accessing the external data by the WfMS. The data from the external systems may be used to control the flow of a workflow or copied into the workflow repository according to the policies defined by a workflow designer in a workflow definition.

The project Exotica/FMQM was extended to incorporate data management capabilities into the WfMSs [2]. Its architecture was based on a fully distributed workflow engine for control flow, and a set of loosely synchronized replicated databases which provided the common distributed repository for all the sites participating in the execution of a process. In this approach only control data were replicated in the distributed repository, whereas we provide the WfMS with a mechanism for copying and synchronizing data between external systems and the workflow repository.

Replication is a well known problem both in distributed systems and in databases [10, 12, 14]. In these domains data are usually replicated to provide better performance and data availability. Our proposal is specific to the workflow environment and has to deal with different requirements and problems. Mere replication management lacks the flexibility of more sophisticated synchronization policies. But according to the terminology in this domain in our case we can say that we have a multimaster replication. Data are stored in an external source and in the workflow repository and can be accessed and modified independently in both of them. Both copies are synchronized in a lazy manner. For simplicity we consider only a state transfer between copies. The main difference to the other work is that the copy conflicts are solved according to the policy defined by the workflow designer (e.g. the policy combination `refresh doNotPush` gives superiority to the external data source).

A more general approach to change propagation in heterogeneous information systems was presented in [4]. Dependencies between data objects stored in different information systems were managed by a separate system called Propagation Manager. This approach used wrappers to connect to external systems and allowed to specify scripts with data transformation definitions. In our proposal the WfMS decides when to propagate changes according to synchronization policies and a workflow definition. External data sources are accessed by specialized wrappers called data access plug-ins, which also provide the required data transformations.

6 Conclusions

This work contributes to a better handling of data in workflow management systems. We argue that data should get more attention in workflow systems and that making data access explicit rather than hiding it in activities has several

advantages, in particular, it improves understandability, maintainability and auditability of workflow definitions. With suitable abstractions and policy based synchronization workflow development will also be improved significantly.

We offer a simple and transparent way for accessing external data and thus, in particular, make the relationship between external data and workflow decision data more visible. We offer a transparent and manageable mechanism for synchronizing copies in the workflow repository with the external data with a clear semantics of operations on data copied into the workflow repository. The synchronization is policy driven relieving the workflow developer from low level programming details. Overall, the concepts introduced in WDL-X are intended to simplify workflow definition and maintenance.

A workflow designer can choose whether to access environmental data in external systems via data access plug-ins directly or to copy these data into the workflow repository. In the latter case he has full flexibility in choosing the synchronization policy. The mechanisms described in this paper builds a solid foundation for further support of workflow definition languages where synchronization specification is more automated by correctness specifications and analysis of workflow definitions.

References

1. Martin Ader. Workflow and business process management comparative study. volume 2. Technical report, Workflow & Groupware Stratégies, June 2003.
2. Gustavo Alonso, Berthold Reinwald, and C. Mohan. Distributed data management in workflow environments. In *Proceedings of the 7th International Workshop on Research Issues in Data Engineering (RIDE '97) High Performance Database Management for Large-Scale Applications*, page 82. IEEE Computer Society, 1997.
3. Christoph Bussler. Has workflow lost sight of dataflow?, 1999. High Performance Transaction System Workshop 1999.
4. Carmen Constantinescu, Uwe Heinkel, Ralf Rantzau, and Bernhard Mitschang. A system for data change propagation in heterogeneous information systems. In *Proceedings of the International Conference on Enterprise Information Systems (ICEIS)*, volume I, pages 51–59, Cuidad Real, Spain, April 2002.
5. J. Eder, H. Groiss, W. Liebhart: *The Workflow Management System Panta Rhei.* In: A. Dogac, L. Kalinichenko, T. Öszu, A. Sheth (Eds.): *Workflow Management Systems and Interoperability*, Springer-Verlag 1998
6. Johann Eder and Marek Lehmann. Uniform access to data in workflows. In Kurt Bauknecht, Martin Bichler, and Birgit Pröll, editors, *Proceedings of the 5th International Conference on E-Commerce and Web Technologies, EC-Web 2004*, number 3182 in LNCS, pages 66–75, Zaragoza, Spain, August/September 2004. Springer-Verlag.
7. Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: from process modeling to workflow automation infrastructure. *Distrib. Parallel Databases*, 3(2):119–153, 1995.
8. Groiss Informatics GmbH *@enterprise* Documentation available at: http://www.groiss.com.
9. IBM Corporation. IBM WebSphere MQWorkflow Concepts and Architecture. GH12-6285, 2003

10. Adbelsalam A. Helal, Abdelsalam A. Heddaya, and Bharat B. Bhargava. *Replication Techniques in Distributed Systems*. Kluwer Academic Publishers, 1996.
11. Nick Russell, Arthur H. M. ter Hofstede, David Edmond, and W.M.P. van der Aalst. Workflow data patterns. Technical Report FIT-TR-2004-01, Queensland University of Technology, Brisbane, Australia, April 2004.
12. Yasushi Saito and Marc Shapiro. Optimistic replication. Technical Report MSR-TR-2003-60, Microsoft Research, October 2003.
13. Staffware plc. *Staffware Technical Overview. Issue 1*, October 2001.
14. Matthias Wiesmann, André Schiper, Fernando Pedone, Bettina Kemme, and Gustavo Alonso. Understanding replication in databases and distributed systems. In *Proceedings of the The 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, page 464. IEEE Computer Society, 2000.
15. Workflow Management Coalition. *Workflow Management Coalition Terminology & Glossary*, 3.0 edition, February 1999.
16. Workflow Management Coalition. *Workflow Process Definition Interface - XML Process Definition Language (XPDL)*, wfmc-tc-1025 edition, 2002.