Federating Heterogeneous Workflow Systems

Andreas Geppert, Markus Kradolfer, Dimitrios Tombros

Technical Report 98.05

Department of Computer Science, University of Zurich
{geppert,kradolf,tombros}@ifi.unizh.ch

Abstract

Heterogeneous workflow management systems often have to cooperate in order to execute inter-organizational or inter-departmental processes, i.e., processes that involve more than one enterprise/department. The problem of workflow interoperability is aggravated by the fact that the autonomy of participating workflow management systems has to be preserved. In this paper, we present an approach towards workflow interoperability that allows to specify and execute workflows over a federation of heterogeneous workflow management systems. This approach is based on a global repository to which members can export workflow types they define or from which they can import external types as subworkflows. On the software architecture level, we show how workflow system federations can be modeled in such a way that the glue needed for interoperability is reusable and leveraged to a more abstract level than the usually applied technique of point-to-point cooperation based on APIs.

Keywords: workflow management, workflow interoperability, federated workflow systems

1 Introduction and Motivation

Workflow management has recently found great attention in the information systems field, as it allows to capture knowledge about business processes, to define workflows in a formal language/framework, and to enact workflows according to their specification. Hereby, a *workflow specification* defines the structure of workflows (e.g., atomic activities/steps), processing entities responsible/capable of executing these activities, and further constraints such as execution or temporal dependencies. *Workflow management systems* (WFMS) [5, 8] are the software systems that support the specification and the execution of workflows.

In many organizations different WFMS have been installed, and even a single enterprise can operate different WFMS in different departments or branches. Often, enterprises or departments need to cooperate in the context of at least some business processes, and thus the various heterogeneous WFMS must be integrated into a federated inter-enterprise or enterprise-wide WFMS. Nevertheless, in the current state of the art, this kind of interoperability is not yet possible to achieve [1, 19].

The problem of achieving WFMS interoperability is similar to that of the integration of heterogeneous database systems into a database federation [14], since both aim at integrating heterogeneous local/component systems into one global system. Just like component database systems in a federation can have different data models, the component WFMS can have different (conceptual) workflow models, and it is generally not feasible to assume a single model that applies to all component systems. Similar to database federations, a WFMS-federation might require the maintenance of autonomy among participating WFMS. For instance, the execution of workflows can be required to remain under full control of the component WFMS, and requestors might not have the possibility to monitor or even influence such executions. Likewise, component WFMS might be willing to publish the interfaces to workflows they offer globally, but would hide the detailed workflow specification from other WFMS

Thus, the aim of workflow interoperability is to enable execution of workflows in an environment of heterogeneous WFMS in such a way that "global" workflows can trigger subworkflows anywhere in the WFMS federation. This integration should occur so that

- a homogeneous global workflow model is not assumed,
- workflow specifications remain private to the component WFMS where they are defined,
- workflow execution remains under control of component WFMS.

In this way, WFMS can cooperate in the context of enterprise-wide workflows, even though the component WFMS use different workflow specification languages (or even specification paradigms), different formats (e.g., for input/output parameters), different enactment techniques, and different implementations (e.g., with respect to the underlying data storage systems).

In this paper, we present an approach to workflow interoperability. A service-oriented view of the global system allows to refer to workflow types of a component system. Thus, a component WFMS can "export" some of its workflow types, meaning that it is able and willing to execute these workflows in the context of a global workflow. Component WFMS are represented in the global system as processing entities (PE). The services these PE offer are exactly those workflow types that the component WFMS export. These PE also implement the necessary functionality to invoke the corresponding workflows (at the component WFMS) and to convert parameters if necessary. In other words, these PE act as mediation components in the global workflow system.

Ultimately, the lowest layer provides for workflow enactment. Because access to and invocation of other component WFMS is completely hidden in the PE mentioned above, the existence of heterogeneous WFMS is completely hidden from the perspective of any local execution engine. Therefore, it is possible for a component WFMS to execute enterprise-wide workflows without further modifications.

The contribution of this paper is thus that it provides a solution how to achieve interoperability in enterprise-wide and inter-organizational workflow management. Particularly, it extends previous work [e.g., 4] in that it does not assume that all component WFMS share the same workflow model, and in that it respects schema and execution autonomy of component WFMS.

The remainder of this paper is organized as follows. In the next section, we introduce the necessary terminology, discuss architectural choices, and give a brief overview of our approach. Section 3 discusses global workflow specification, section 4 introduces our approach to model the architecture of federated workflow systems, and section 5 presents execution of global workflows. Section 6 discusses related work, and section 7 concludes the paper.

2 Overview

In this section, we first introduce the relevant terminology, present architectural choices how to federate workflow systems, and finally give an overview of our approach.

2.1 Terminology

A workflow management system consists of a buildtime and a runtime component. The buildtime component supports the definition of *workflow types* in some language or notation (e.g., a script language, Petri Nets, or statecharts). The notation/language is henceforth referred to as *workflow model*. For a workflow type, we assume that it defines an *interface* (consisting of a unique name and sets of typed formal input and output parameters) and a *body* (which in turn specifies in terms of the workflow model the internal structure of the workflow, i.e., subworkflows/activities, task assignment rules, involved processing entities, etc.). The runtime component of the WFMS supports the enactment of workflow instances as prescribed by the corresponding workflow type.

A *workflow system* (WS) consists of a WFMS and a set of workflow types. The latter is called the *workflow schema* of the WS. A *federated WS* (or *federation*, for short) consists of a set of individual, possibly heterogeneous yet interoperable WS. The individual WFMS may be heterogeneous with respect to the workflow models they use, workflow enactment techniques, used data stores, etc.

2.2 Architectural Choices

Several (architectural) choices are conceivable to design and implement a WFMS-federation (see Fig. 1). Wherever applicable, we subsequently reference systems and proposals that have chosen the respective approach; for details see section 6 on related work.

In the first case, one WS directly invokes another one via an API (this approach corresponds to level 4 of the WfMC interoperability specification [20]). The basic disadvantage of this approach is that each WFMS must know the precise API of each other one in the federation. This ultimately leads to a high number of point-to-point connections resulting in high implementation and maintenance complexity. An additional complication is that component WFMS must be extended in order to be able to start workflows in other ones.

The second choice is based on data integration between different WS in that the federated WFMS uses a common platform, i.e., information is actually exchanged via the database system (as suggested for Wide [4]). The practical difficulty, however, is that enterprises/departments are usually reluctant to allow access to their database. Thus, this choice is not considered practical for inter-enterprise or enterprise-wide workflow management. Furthermore, this variant assumes homogeneous component WFMS at least with respect to their workflow engines; it is otherwise not appropriate to assume that WFMS manufacturers allow the access to their internal platform for other (competing) WFMS.

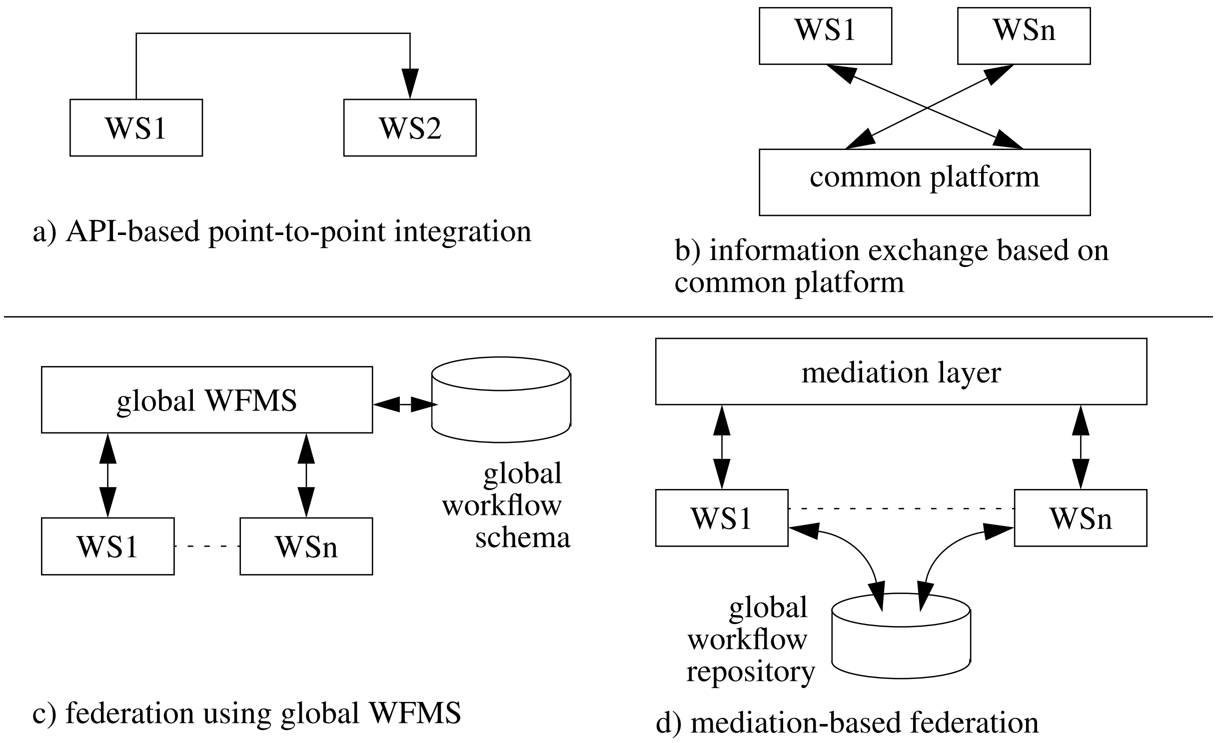

Figure 1. Architectural choices for WFMS-federations

In the third alternative, a global integration layer—as is typical for federated database systems [14]—is provided that in turn is a full-fledged WFMS. In this approach, the buildtime component of this global WFMS supports the specification of workflow types, which are then mapped to workflow types defined in the local WS. The runtime component of this global WFMS is able to execute workflow instances, including subworkflows executed in the component WS. Since most departments or enterprises have preexisting WFMS/WS anyway, this solution implies that a second WFMS needs to be installed and operated.

In the fourth option, the federation is constructed based on a mediation layer that allows each component WS to invoke workflows defined in some other component WS. Unlike the third alternative, this layer is not itself a full-fledged WFMS. Furthermore, unlike the first alternative relying on API-based point-to-point integration, it does not require extensions to local WFMS and the mediation components can be reused. Subsequently, we investigate this option.

2.3 Overview on Workflow System Federation

A WS whose schema contains a workflow type (i.e., the WS owns that type) can selectively export it to the federation by publishing its interface. Exported workflow types are contained in the *repository* of the federation, whereby for each exported type, information about its interface, its owner, and other WS using the type is maintained. Using a workflow type from the repository actually means that a WS (different from the owner) imports the interface. Imported workflow types can be instantiated just as any other workflow types that are defined by the importer itself.

In order to abstract from API-based point-to-point connections in federations, WS are represented by special mediation-components. On the client-side (i.e., where a global workflow executes), other component WS are represented by software components as processing entities that offer imported workflows in the form of services. On the server side (i.e., the remote component WS where subworkflows of a global workflow can be executed), specific clients mediate between the client WS and the local WFMS.

Before we discuss our approach towards workflow federation in more detail, we present a running example. Assume a globally operating bank (say, "Swiss Asset Management", SAM), with branches/subsidiaries in Zurich, New York, and Frankfurt. One workflow to be defined is the purchase of stocks. It should thereby be possible for Swiss customers to buy stocks that are traded at (say) NASDAQ in such a way that customers only interact with the branch in Zurich, which in turn cooperates with the corresponding broker in New York (note that this broker can belong to a subsidiary or to a completely independent company).

3 Specifying Global Workflows

In this chapter the specification of global workflows is addressed. The overall architecture we pursue is as shown in Fig. 1d.

A workflow `wf` is called *global* if at least one of its subworkflows is executed by a component WS that is different from the WS `wf` is enacted by. Simply stated, a global workflow can be specified by a WS using workflow types provided by other WS. In order to enable the specification of global workflows by a WS, the following three prerequisites have to be met:

(1) it must be possible for a WS to find out which workflow types are offered by other WS,
(2) the workflow types offered by a certain WS can be represented in another WS, and
(3) software components must be representable as processing entities, and task assignment rules must be specifiable (note that this functionality is fairly standard).

The first prerequisite is met by a federation repository that contains information about the workflow types exported by workflow systems. Each WS may choose to export some of its workflow types to the federation. An exported workflow type is inserted into the workflow repository, which contains the following information for each exported workflow type:

- its name,
- a set of typed formal input parameters,
- a set of typed formal output parameters,
- a set of WS that implement and export this workflow type,
- a set of WS that import the workflow type.

In order to meet the second prerequisite, a WS must be able to import workflow types from the repository such that they can be instantiated by the WS as any other workflow type defined by the WS itself. Such imported workflow types are declared as automated, atomic activities. "Atomic" means that the activity has no internal structure, or at least its structure is not known to the WFMS. "Automated" means that the activity can be executed by a software system without interaction with humans.

Note that the fact that such an activity represents an imported workflow is no longer explicit in the specification. In order to render this information visible in the specification, uses of imported workflow types could be labeled with an additional keyword (e.g., "external workflow"). This would however require an extension of the workflow specification language or model. Since this is not desirable in most cases, we do not consider this alternative further. In order to at least implicitly identify workflows as imported, appropriate naming conventions can be used.

The information about which resource can execute an activity is specified by *task assignment rules*. In our case, this executing resource is another component WS, so that an importing workflow system has to invoke the remote WFMS. In order to abstract from the technical details such as calling API-functions, converting parameters, and communicating with remote WFMS, workflow specifications define a processing entity that—from the perspective of the importing WFMS—is able to execute these activities (i.e., external workflows). Thus, when a workflow type `WT` is imported by a WS, a representation as a (server) processing entity of the component workflow system supporting `WT` has to be created (cf. the third requirement). The representation of this PE solely depends on the importing WFMS, so that it needs to be implemented only once. Upon importing additional workflow types, the corresponding PE can be extended automatically to also offer the new types as activities.

In our example, assume that the aforementioned subsidiaries in New York and Frankfurt both operate a WS, and both export workflow types named `BuyStock` and `SellStock` (see Table 1) for purchasing and selling stocks.

WFType	In-Parameters	Out-Parameters	Providers	Importers
BuyStock	customer: string, stockID: integer, number: integer, maxPrice: real, deadline: Date	effectivePrice: real, fee: real, number:integer date: Date	WS-SAM-NY, WS-SAM-FR,	WS-SAM-ZH
SellStock	customer: string, stockID: integer, number: integer, minPrice: real, deadline: Date	effectivePrice: real, fee: real, number:integer date: Date	WS-SAM-NY, WS-SAM-FR,	

Table 1: Repository Information about Workflow Type BuyStock

Assume the Zurich branch wants to define a workflow type that uses the `BuyStock` workflow type of the repository. For this purpose, `BuyStock` has to be imported from the federation repository into the WS of the Zurich branch. Fig. 2 shows the example workflow type `BuyStockTransaction` specified with our workflow modeling tool that is based on the TRAMs workflow specification approach [9][1]. `BuyStockTransaction` defines the process of buying stocks for a customer in either Zurich, New York or Frankfurt. Note that `BuyStock` of the repository is represented as `EXTERNBuyStock` in order to indicate that this step will be executed by an external WS.

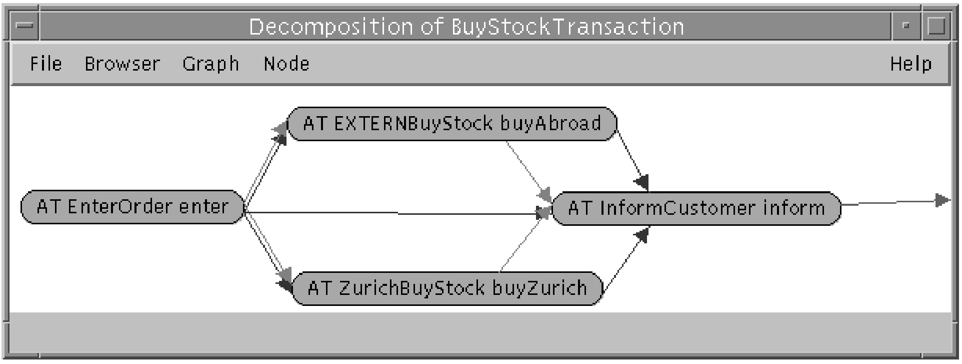

Figure 2. Global Workflow Type BuyStockTransaction

In an enterprise-wide federation (even if it comprises subsidiaries), one might assume that the federation repository is managed by a well-known server in the enterprise, and that component WS "know" where and how to access this federation repository. This assumption no longer holds in inter-enterprise workflows, especially when the component WS cooperate rather loosely and occasionally. We therefore propose to manage the repository by one or more trading services [e.g., 11, chapter 16], i.e., agencies providing for workflow type "yellow pages". Such an agency allows to exporters to announce offered workflow types. Clients (i.e., importers) can query the repository via the trading service.

Note that in our approach only workflow type interfaces are exported and imported. Thus, the "body" of a workflow, i.e., its definition in terms of activities, subworkflows, task assignment rules etc. is not visible to importers (this is in contrast to other proposals [2, 3, 4, 14, 20]). In practice, workflow specifications define how an enterprise conducts its business, and an optimized, efficient workflow specification will represent a competitive advantage for an enterprise. Hence, workflow specification internals represent a strategic asset that enterprises would not like to exhibit to other companies, especially not to competitors.

4 Modeling Workflow Federations

In this section, we describe the software architecture model of a workflow federation. The motivation for introducing this additional level of abstraction in addition to workflow specification and the execution engines are

- ❐ an abstraction of the needed components mediating between component systems,
- ❐ fostering reusability of these components.

1. Dark arrows represent control flow and grey ones data flow. "AT" stands for "activity type".

In order to interoperate at runtime, mediation components are necessary that form the “glue” between the local WS. This type of mediators is formed by two types of software components: specialized *WFMS-clients* and *processing entities*. A processing entity is a resource (human or software component) that can execute one or more of the (atomic) activities requested in a workflow. In our context, a specialized processing entity is a software component that offers to a local WS the execution of subworkflows defined in another WS of the federation. The (local, client) WS defines task assignment rules specifying that the processing entity is invoked whenever one of the imported workflows has to be executed. The reaction of the PE to such a request is to trigger the execution of the imported workflow at the WS that actually implements the workflow.

In order to spawn such a subworkflow at another WS, the PE communicates with the WS-specific client. This one is able to automatically start workflows via the client-API of its WFMS and—as soon as the workflow is terminated—receive results and pass them back to the PE (which in turn forwards results to the WFMS-engine that originally requested the subworkflow).

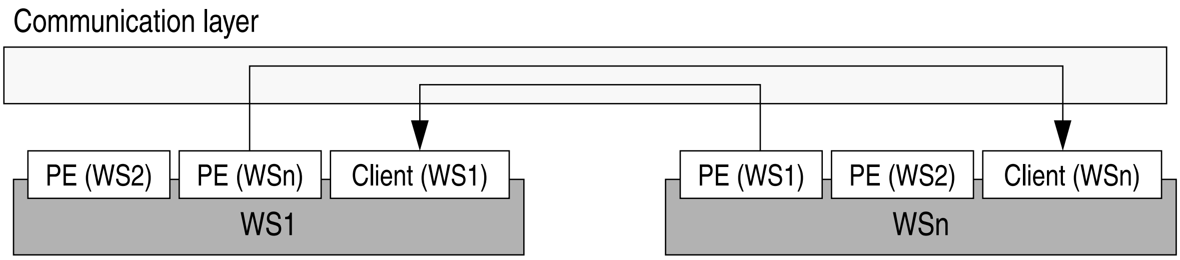

Figure 3. Software Architecture of Federated Workflow Systems

The two types of components are plugged into each of the component WS that should participate in the federation. WS-PE and -clients communicate via a communication layer. They thereby use a common model of messages and parameters. Each PE (client) is able to convert parameters from/to the representation in terms of this uniform model. The distinction into PE and clients means that PE do not have to know the concrete interface of the WFMS offering workflows—they simply have to know service interfaces of the respective clients. Similarly, clients to not have to know interfaces/formats of their clients WFMS, but only need to know the API of their local WFMS. In this way, PE and clients have to be written once for each type of WFMS.

The concrete choice how to represent and implement PE and clients as well as the concrete communication software is of minor concern—provided that PE, clients, and the communication system run on all platforms involved in the federation. For instance, an ORB [12] could be used for communication; in this case it would be quite natural to use the OMG object model for representation and transmission of parameters. The component WFMS would then be represented as CORBA application objects, clients would implement skeletons and PE would implement stubs (in CORBA terminology).

We illustrate this approach by using the Broker/Services Model (B/SM) [16, 17] to represent specialized clients and PE. In this model, brokers are reactive entities that offer services to their peers. Services are described by their signature (name, input parameters, replies, and optionally exceptions). Service provision is implemented in brokers by *event-condition-action rules* (ECA rules), and brokers interact with each other by generating events. Brokers define a persistent state and a set of state operations to describe the wrapping of external components. Responsibility rules are defined for events; they can refer to event parameters, broker states, etc. and determine which broker has to react to occurrences of the event.

Services in B/S systems are requested by service clients that raise a service request event with actual event parameters corresponding to the input parameters defined for this service. The server broker of a particular service is determined by special responsibility and task assignment functions. Subworkflows that are executed by component WFMS in a federation are defined as services with the following signature:

- the service name is the exported workflow type name,
- the service input parameters correspond to the exported type formal input parameters,
- a reply corresponding to completed external subworkflow execution with formal parameters as defined for the return type of the exported workflow,
- an exception corresponding to abnormal termination of the remote workflow.

Consider the situation depicted in Fig. 3. A broker $\texttt{PE}(\texttt{WS}_n)$ is defined in the local workflow system $\texttt{WS}_1$ which represents the external WFMS $\texttt{WS}_n$ and has a single ECA rule whose event is the exported workflow service request. This broker implements in its state operations the communication and information exchange mechanisms required to interoperate with the external WFMS client `Client(WSn)` which spawns the subworkflow at the other WS. Their implementation depends on the interface provided by the external WFMS.

For each service corresponding to an exported workflow in the repository a responsibility function is defined which returns a set with a single element, namely $\texttt{PE}(\texttt{WS}_n)$. After the remotely executed workflow terminates a reply event is generated by $\texttt{PE}(\texttt{WS}_n)$. Remotely executed workflows are in general long-lasting processes. Consequently, communication breakdowns as well as crashes of involved local and remote components must be appropriately handled. Persistence and recoverability of the state of the broker representing the remote WS as well as reliable communication mechanisms such as those described in [6] can be used to achieve these properties.

5 Executing Federated Workflows

In this section, we briefly address workflow execution in federated workflow systems. Provided that workflows have been specified and federations modeled as described above, global workflows can be executed without any required extensions to a workflow engine. In particular, external workflow execution is seamlessly possible in the usual workflow execution cycle:

(1) the local WFMS determines that an atomic activity should be executed (how this fact is determined depends on the enactment paradigm applied by the WFMS),

(2) it performs task assignment (and in the case of external workflows discovers that the PE representing the remote WS is responsible),
(3) it commands the chosen PE with the activity execution.

The PE then forwards the request to the WFMS-specific client at the remote federated component WS. At some point in time, the PE will receive a reply and forward it to its local WFMS. Again, the local WFMS does not need to be aware of the fact that the reply represents the result of an external workflow and make any special provisions.

Consider the example shown in Fig. 4. At one WS (`WS-SAM-ZH`, which runs under control of our own WFMS EVE [6, 18]) a workflow of type `BuyStockTransaction` is executing. One of the parameters in this workflow identifies the stock exchange where the customer wants to buy the stocks. Control flow rules define that if the exchange is the one in Zurich, then the subworkflow `buyZurich` will be executed. Otherwise, the alternative path containing the activity `buyAbroad` will be followed. Additionally, there is one specific PE represented by a broker (called `WS-SAM-NY-PE`), and task assignment rules specify that this broker is responsible for the activity execution whenever the parameter representing the stock exchange equals “NASDAQ”. Thus, whenever stocks from NASDAQ are requested, this broker is notified by EVE, and then forwards the request to another broker at `WS-SAM-NY` (which in turn runs under control of the WFMS ProFlow[2]). Note that `WS-SAM-NY` might run under any WFMS that satisfies the aforementioned requirements (i.e., allows workflows to be started via an API).

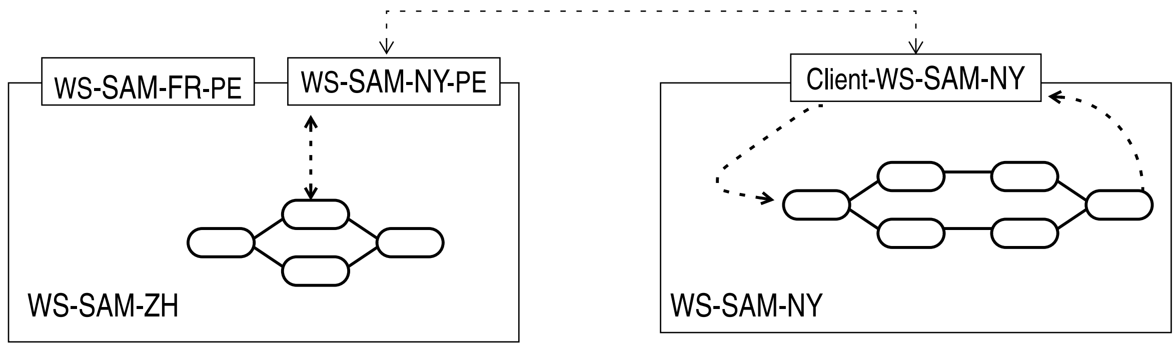

Figure 4. Global Workflow Execution in the BuyStock Example

6 Related Work

Interoperability has been investigated in the Oz-project. In [2], the question of how to establish interoperability between different process-centered software engineering environments (PCSE) is addressed. The various component systems can have different process models (i.e., workflow types), while heterogeneity of execution engines and process modeling formalisms is

2. ProFlow is a registered trademark of OpenInfo (http://www.openinfo.ch/).

not addressed there. In [3], the same authors address architectural issues in federating PCSE. This paper addresses mainly homogeneous federations (i.e., those where the component systems are instantiations of the same PCSE), while the heterogeneous case in which components are instances of heterogeneous PCSE is only briefly sketched. In both works, components can coordinate their processes and thus share information on their process models. This is certainly necessary in design applications such as software engineering, while it is not feasible in the general workflow applications we consider.

Sheth et al. [15] discuss workflow execution in heterogeneous environments. However, they focus on the access to heterogeneous resources, such as databases, using a variety of infrastructure technologies (such as ORBs and the Web). They use a single WFMS ($METEOR_2$), and therefore do not consider workflow system federations in the sense of this paper.

Some prototypes and commercial products (e.g., FlowMark [10]) allow to call executable programs or scripts from within a workflow execution. In this way, external subworkflows can be executed. In contrast to our approach based on a global repository and mediation components, this solution implies a more complex, low-level and hard-wired implementation of interoperability, which must be implemented manually and is less well maintainable.

Casati et al. [4] also address workflow interoperability. They assume a global conceptual workflow model for the federation, and also a homogeneous enactment engine. Information between component engines can be exchanged via the underlying database. Their main focus is on the synchronization of active workflows.

The Workflow Management Coalition has devoted one of its interfaces to workflow interoperability [20]. This interface allows the exchange of workflow type definitions; as already mentioned, we do not consider this feature as viable in inter-enterprise workflow management. Interface 4 defines 8 “levels of interoperability” (where actually the first one represents the absence of interoperability, and the highest one—the same standard user interface for all WFMS—might never we achieved [20]). Several WFMS currently comply to level 4 [19], which is characterized by a limited common API subset. It is questionable whether compliance to higher levels is realistic, because these levels assume a complete standardized API (level 5, product-specific extensions are excluded), a shared process modeling language (level 6), and standardized client/server communication (level 7, including transmission of definitions, and workflow transactions).

The workflow facility under definition of the OMG also addresses workflow interoperability. In this case, component systems can interoperate via the CORBA middleware. Nevertheless, this requires that the underlying WFMS complies to the workflow facility standard, and in order to be able to invoke a subprocess at another component of a federation, the WFMS has to be modified. In [13], it is possible to retrieve workflow type definitions from other federation components.

7 Conclusion

We have presented an approach to federating workflow systems, whereby the workflow systems can be heterogeneous to different degrees (underlying WFMS, workflow schemas, etc.).

The proposed approach allows workflow systems to interoperate, without assuming that workflow specifications are exchanged. This is in practice an aspect of paramount importance especially in inter-organizational workflow management. We have also shown how the architecture of a workflow federation can be modeled in such a way that the mediation software is leveraged from a point-to-point, API-based connection.

The contributions of this approach, thus are that:

- it allows the specification of global workflows which can contain subworkflows offered by different federation components,
- it maintains encapsulation and autonomy in the sense that the definitions of workflow types defined in component WS remain private to their owners,
- it leverages the interaction of federation components from point-to-point, API-based connections to a more abstract level of reusable mediation components,
- it allows the execution of global workflows without modifications to the engines of the component WFMS.

In our future work, we will extend our approach by allowing a more flexible task assignment whenever multiple other component WS offer a specific workflow type (e.g., in the example, multiple component WS export a workflow type to buy stocks at the same stock exchange). Parameters relevant for the decision on which WS to choose in such cases can be the expected execution time and cost (as we have previously suggested for assignment of activities [7]).

8 References

1. G. Alonso, D. Agrawal, A. El Abadi, C. Mohan. Functionality and Limitations of Current Workflow Management Systems. *IEEE Expert*, 1996
2. I.Z. Ben-Shaul, G.E. Kaiser. An Interoperability Model for Process-Centered Software Engineering Environments and its Implementation in Oz. Technical Report CUCS-034-95, Columbia University Department of Computer Science, 1995.
3. I.Z. Ben-Shaul, G.E. Kaiser. Federating Process-Centered Environments: the Oz Experience. *Automated Software Engineering* 5:1, January 1998.
4. F. Casati, S. Ceri, B. Pernici, G. Pozzi. Semantic WorkFlow Interoperability. *Proc. 5th Intl. Conf. on Extending Database Technology*, Avignon, France, March 1996.
5. D. Georgakopoulos, M. Hornick, A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases* 3:2, April 1995.
6. A. Geppert, D. Tombros. Event-based Distributed Workflow Execution with EVE. Technical Report 96.05, Department of Computer Science, University of Zurich, May 1996.
7. A. Geppert, M. Kradolfer, D. Tombros. Market-based Workflow Management. *Proc. Int'l IFIP Working Conf. on Trends in Distributed Systems for Electronic Commerce*, Hamburg, Germany, June 1998.
8. S. Jablonski, C. Bussler. *Workflow Management. Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London 1996.

9. M. Kradolfer, A. Geppert. Modeling Concepts for Workflow Specification. Technical Report 97.05, Departement of Computer Science, University of Zurich, April 1997.
10. F. Leymann, W. Altenhuber. Managing Business Processes as an Information Resource. *IBM Systems Journal* 33:2, 1994.
11. CORBA Services: Common Object Services Specification. OMG, July 1997 (http://www.omg.org/corba/sectran1.htm).
12. The Common Object Request Broker: Architecture and Specification. Revision 2.1, Object Management Group, August 1997.
13. Workflow Management Facility. OMG Business Object Domain Task Force, BODTF-RFP 2 Submission, August 1997.
14. A.P. Sheth, J.A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases. *ACM Computing Surveys* 22:3, September 1990.
15. A. Sheth, K. Kochut, J. Miller, D. Worah, S. Das, C. Lin, D. Palaniswami, J. Lynch, I. Shevchenko. Supporting State-wide Immunization Tracking using Multi-Paradigm Workflow Technology. *Proc. 22nd Int'l Conf. on Very Large Data Bases*, Bombay, India, September 1996.
16. D. Tombros, A. Geppert, K.R. Dittrich. Brokers and Services: Constructs for the Implementation of Process-Oriented Environments. In B. Freitag, C.B. Jones, C. Lengauer, H.-J. Schek (eds). *Object-Orientation with Parallelism and Persistence*. Kluwer Academic Publishers, 1996.
17. D. Tombros, A. Geppert, K.R. Dittrich. The Broker/Services Model for the Design of Cooperative Process Oriented Environments. Technical Report 97.06, Department of Computer Science, University of Zurich, June 1997.
18. D. Tombros, A. Geppert, K.R. Dittrich. Semantics of Reactive Components in Event-Driven Workflow Execution. *Proc. 9th Intl. Conf. on Advanced Information Systems Engineering*, Barcelona, Spain, June 1997.
19. D. Tombros, A. Geppert. Managing Heterogeneity in Commercially Available Workflow Management Systems: A Critical Evaluation. SWORDIES Project Report No. 5, Department of Computer Science, University of Zurich, June 1997
20. Interface 4 - Interoperability - Abstract Specification. WFMC-TC-1012, Workflow Management Coalition (http://www.aiai.ed.ac.uk/project/wfmc/), October 1996