Web Service Composition with O'Grape and Osiris

Roger Weber Christoph Schuler Patrick Neukomm Heiko Schuldt
Hans-J. Schek

Database Research Group, Swiss Federal Institute of Technology (ETH), CH–8092 Zurich, Switzerland
Email: {weber,schuler,schuldt,schek}@inf.ethz.ch,neukommp@student.ethz.ch

1 Introduction

Services are well known building blocks in modern information systems. Technologies and standards like XML, SOAP (Simple Object Access Protocol) [8], and WSDL (Web Service Description Language) [10] provide a simple means to describe these services and to make them accessible to a large community in a distributed environment. Yet, the full potential of web services becomes only apparent if we can combine several service invocations in a well-defined order and with well-designed execution guarantees to establish even more powerful composite services. Among others, processes provide a simple mechanism to compose services [1]. A process defines the logical dependencies between independent services by specifying an invocation order (control flow) as well as rules for the transfer of data items between different invocations (data flow). In addition and following the model of transactional processes [7], we can define the transactional behavior and execution guarantees to ensure a correct execution of processes in case of concurrency and failures. An infrastructure for transactional processes has to support all these run-time features. Furthermore, a graphical process modeling tool should support the specification of all these features. An important aspect is that such a modeling tool is transparently integrated into the process management environment.

In addition to the transactional semantics of service composition, several other run-time aspects are crucial. Usually, several semantically equivalent web services are available at different places. An infrastructure for process execution should equally distribute the load over all web service providers. Similarly, process executions should take costs and expected execution times into account to optimize response times. To this end, the infrastructure should bind services dynamically at run time instead of at hard-coding this binding at process definition time. This decoupling provides a high degree of flexibility since new providers and services are seamlessly integrated (following the ideas of autonomic computing [9]).

With the Osiris (**O**pen **S**ervice **I**nfrastructure for **R**eliable and **I**ntegrated process **S**upport) prototype infrastructure for the execution of transactional processes and the O'Grape (Osiris GRAphical Process Editor) modeling tool, the aspects of modelling and execution of transactional processes can be seamlessly combined. O'Grape allows to graphically specify processes by combining web services. This includes both control and data flow aspects as well as transactional semantics. An application developer can then export new/revised processes from O'Grape to the Osiris system, where they can be executed. Osiris combines the advantages of several infrastructures, namely: i.) discovery and invocation of web services from frameworks like .NET [3], ii.) process support and execution guarantees from workflow management systems, iii.) late binding of service calls and load balancing from grid infrastructures [2], and, finally, iv.) Peer-to-Peer systems. With the peer–to–peer based execution of processes, navigation costs can be accumulated only on nodes that are directly involved in the execution of a process.

In the following, we first give a brief overview of Osiris in Section 2. Section 3 then describes O'Grape, our modeling tool. Finally, Section 4 enumerates the subjects of our demonstration.

2 The Osiris System

The Osiris architecture consists of two parts: besides a small software layer on every node, the so-called hyperdatabase layer, Osiris runs a set of global system services that manage global settings and meta data (cf. middle box in Figure 1).

2.1 The Hyperdatabase Concept

Osiris follows the vision of a *hyperdatabase system* [4, 5, 6]. In short, a hyperdatabase (HDB) pro-

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

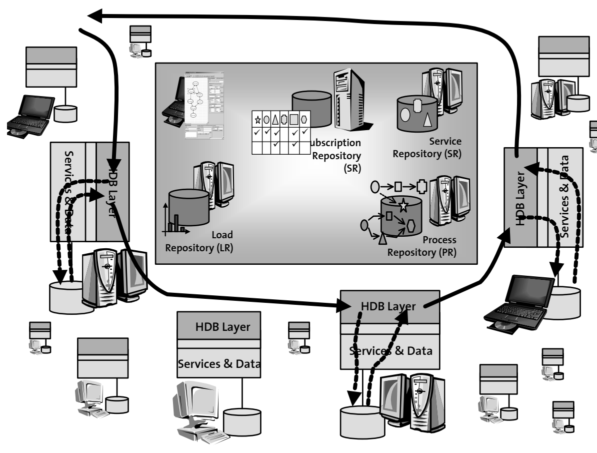


Figure 1: Peer–to–peer process execution

vides transactional guarantees for processes over distributed components using existing services. The HDB provides sophisticated routing strategies to dynamically choose among the available providers. Prior to the invocation of a service a hyperdatabase requires that each service provider locally installs an additional software layer, a so-called *hyperdatabase layer* (HDB layer). Hence, a hyperdatabase does not follow a monolithic system architecture but consists of a set of additional thin software layers. Ideally, this HDB layer comes together with the operating system much like the TCP/IP stack does (comparable to the .NET framework).

2.2 Architecture of Osiris

The main emphasis of our architecture is to avoid any central component for process execution. Instead, a process instance should run only on those nodes of the Osiris community that contribute a service to that process (cf. Figure 1). We call this *Peer-to-Peer Execution of Processes* (P2PEP). With P2PEP, a node works off its parts of the process, and then directly migrates the instance data to nodes offering a suitable service for one of the next steps in the process. To do so, the HDB layer requires global meta information about the system, e.g., about providers and their services, about load information, and about process definitions. A set of central repositories collect, maintain, and store this information. The local HDB layers replicate from these sources those pieces of information that are locally required to drive process executions. It is important to distinguish between the flow of process execution and the meta-data flow. While process execution follows a true peer-to-peer pattern, the meta-data flow has a centralized organization. To enable P2PEP, it is crucial to decouple process navigation from meta-data replication. In other words, process navigation always relies on the currently and locally available meta information (to this end, it has to be garatieed by the Osiris system that this load information is sufficiently consistent). Replication of meta information runs in the background and exploits some week constraints on the freshness of the data. For instance, load information changes permanently. However, an HDB layer should only receive significant changes to balance the load. For that purpose, Osiris deploys a publish/subscribe based replication scheme with freshness predicates. A freshness predicate defines when to publish changes on subscribed meta-data.

2.3 Core Services

The services essential for process navigation are:

The **Subscription Repository** keeps the primary copy of the service subscription lists. These lists subsume providers offering semantically equivalent services. The HDB layers replicate those lists from the subscription repository, that are needed to migrate subsequent steps within the set of processes that run through this layer. In general, an HDB layer only requires a portion of information out of the subscription list.

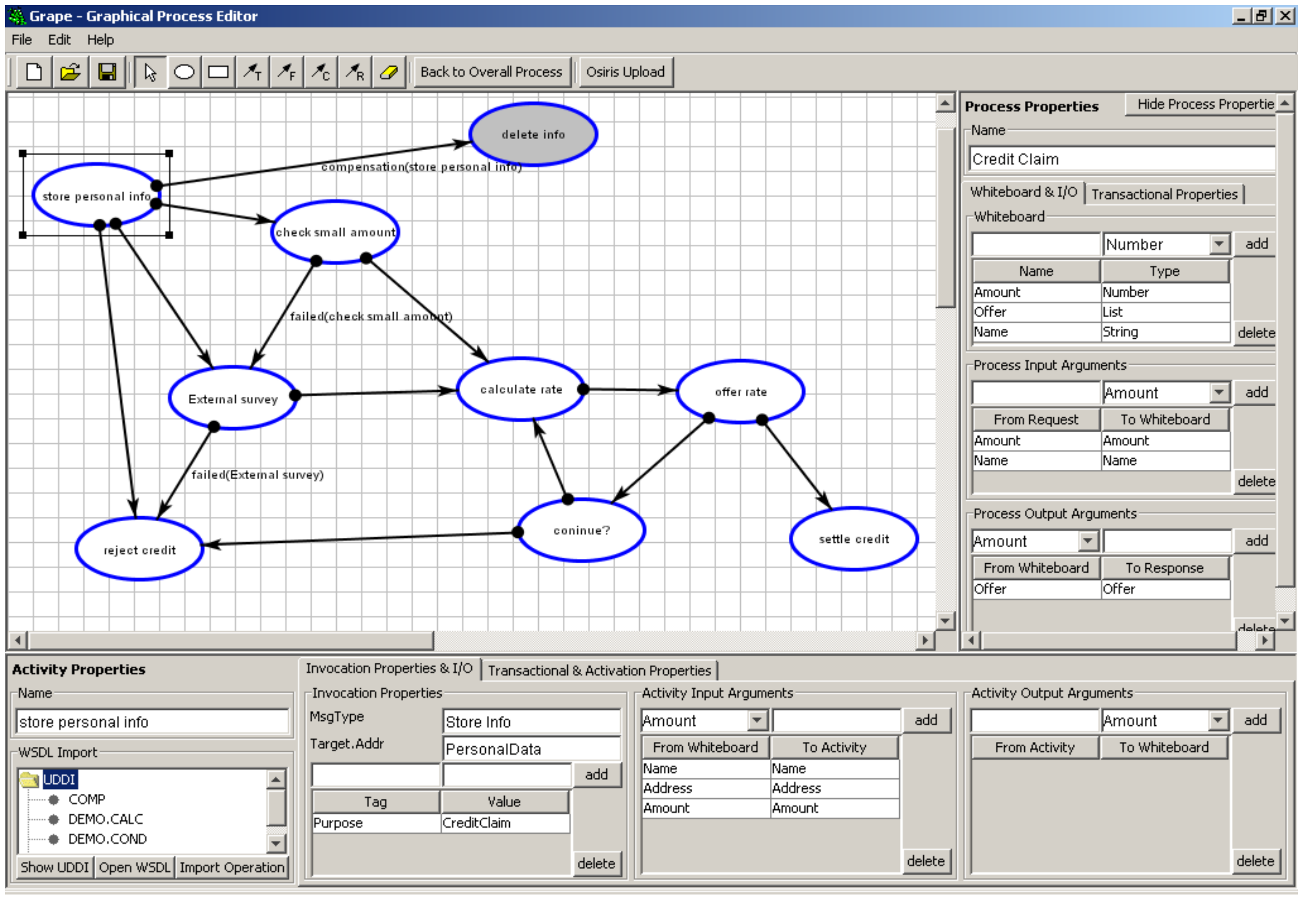
The **Service Repository** maintains the interface definitions (WSDL) of all services available in the Osiris system. The modeling tool O'Grape downloads these descriptions to offer them to the application developer as basic steps of a process. Apart of this, the repository contains the executables of services for on-demand installation of hot-spot services.

The **Process Repository** holds the global definitions of all processes of the community. Application developers upload new/revised processes with O'Grape. As a result, pieces of the process description are distributed to all HDB layers that potentially may receive an instance of this new process type (again using publish/subscribe mechanisms).

Monitoring Repositories store the history and states of recently and currently executed processes. A monitoring repository subscribes itself for events related to process execution. These events are published by the HDB layers whenever a process instance advances or a significant change of its state occurs. Furthermore, the monitoring component condenses information to provide monitoring clients a coherent view of the current state of a process instance.

Web Service Proxy Gateways offer a hyperdatabase layer for external services. To integrate such services into the Osiris system, their WSDL descriptions are uploaded into the proxy service. As a side effect, the proxy service registers these services in the system. For the Osiris system, external services look like ordinary Osiris services with the exception that process navigation takes place on the proxy node.

Apart of these services, there exists a number of further core services related to concurrency control and generic tasks like, for instance, XSLT transformation.



Process Properties

Name: Credit Claim

Whiteboard

Name	Type
Amount	Number
Offer	List
Name	String

Process Input Arguments

From Request	To Whiteboard
Amount	Amount
Name	Name

Process Output Arguments

From Whiteboard	To Response
Offer	Offer

Activity Properties

Name: store personal info

WSDL Import: UDDI, COMP, DEMO.CALC, DEMO.COND

Invocation Properties

MsgType: Store Info

Target.Addr: PersonalData

Tag	Value
Purpose	CreditClaim

Activity Input Arguments

From Whiteboard	To Activity
Name	Name
Address	Address
Amount	Amount

Activity Output Arguments

From Activity	To Whiteboard

Figure 2: OSIRIS Process modeling tool O'GRAPE

3 Modelling with O'GRAPE

A crucial part of the OSIRIS system is the graphical modelling tool O'GRAPE for easy composition of web services into processes. O'GRAPE is implemented in Java and runs as a standalone application as well as an applet. Figure 2 shows a screen shot of the standalone version. On the right hand side, the figure depicts the so-called *whiteboard* of the current process (tool box "Whiteboard"), which contains the global variables of a process instance. During process execution, the whiteboard of a process instance is first filled with the process arguments (tool box "Process Input Arguments"). Whiteboard parameters might be modified during process execution by service invocations. Finally, the return parameters of a service invocation are fed back to the whiteboard and the result of the process invocation is assembled from its contents (tool box "Process Output Arguments").

The ellipses in Figure 2 denote the activities of the process, i.e., service invocations. For each activity, the designer can choose among the list of services and processes known in OSIRIS. To this end, O'GRAPE downloads WSDL descriptions of services and processes from the global service repository and process repository (see the selection box at the left hand side at the bottom of the figure). A service invocation retrieves its input parameters from the whiteboard (see tool box "Activity Input Arguments" at the bottom), and maps output parameters back to the whiteboard (tool box "Activity Output Arguments"). Further, arrows define the control flow of a process. The OSIRIS process model allows to fork and join execution paths, and to introduce loops. In the case of loops, it is the task of the application designer to guarantee the termination of the process. To this end, our model supports conditional edges between activities. Finally, the application developer can upload new/refined versions of a process into the process repository of OSIRIS, and immediately test the process using a standardized input form to enter process input parameters.

So far, O'GRAPE provides the same functionalities as other modelling tools. Beyond these basic methods, OSIRIS and O'GRAPE feature sophisticated failure handling strategies at the application level. In terms of failure handling, our model provides compensation activities and alternative execution paths. An example of the later one is depicted in Figure 2: if the activity "check small amount" fails, the alternative execution path over activity "External survey" is followed. Only if this activity succeeds, the process continues with the usual path over activity "calculate rate". Hence, alternatives are a means to react on failures of service invocations. If no alternatives are provided and a service invocation fails, Osiris rolls back the process instance until an alternative path is found or the entire process is undone. Furthermore, the model allows to specify explicit compensation activities that undo the side effects of an earlier service invocation even in absence of an automatic compensation provided by the service itself (for an example, see the shaded activity "delete info" at the top).

Other interesting aspects of Osiris that is enabled with O'Grape are transactional and activation properties. Each activity is described by a set of properties like execution costs, compensatability, retriability, and failure probabilities. Moreover, the commutativity characteristics of all activities are given. Osiris consults these properties to ensure a correct execution of processes as described in [7]. Furthermore, costs for service executions and success probabilities are taken into account to optimize the overall costs of processes and to avoid the compensation of costly activities due to concurrency control. These transactional properties come together with the service interface definitions. To this end, we have extended the WSDL format to further encompass these properties.

Finally, O'Grape supports the application designer in deriving these properties for the entire process (recall, a process may be again a service in an other process). A toolbox assistent checks the integrity of the specified process, and derives, as far as possible, transactional properties and expected execution cost for the overall process.

4 Demonstration

The demonstration shows how easily existing services are integrated into processes with a graphical tool, and how peer-to-peer process execution combines the advantages of existing service infrastructures. Our current prototype system Osiris provides standardized service interfaces and fully-fledged peer-to-peer execution of transactional processes, and it runs a number of central repositories. Among these repositories, the service and process repository play a central role of the demonstration. The smooth integration of O'Grape into Osiris allows application developers to choose and download WSDL descriptions of known services (and processes) from these repositories. Among many possible scenarios, the demonstration of O'Grape features the design and implementation of processes in the context of banking, e-business, and image retrieval. In the latter case, external services (like Google search) and internal services are seamlessly combined.

A second part of the demonstration features the process engine of Osiris. We have implemented a number of services for content-based and region-based image retrieval like feature extraction, segmentation, and indexing. These services are connected by several processes to implement an image retrieval application and to maintain consistency between repositories and indexes. We demonstrate during process execution how the system is able to automatically install and remove service instances whenever needed. Finally, a monitoring client displays the current state of executed process instances in the system.

References

[1] M. Schmid F. Leymann, D. Roller. Web services and business process management. *IBM Systems Journal*, 41(2):198–211, 2002.

[2] I. Foster, C. Kesselmann, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. `http://www.gridforum.org/ogsi-wg/`.

[3] Microsoft .NET. `http://www.microsoft.com/net/`.

[4] H.-J. Schek, K. Böhm, T. Grabs, U. Röhm, H. Schuldt, and R. Weber. Hyperdatabases. In *Proceedings of the 1st International Conference on Web Information Systems Engineering (WISE'00)*, pages 14–23, Hong Kong, China, June 2000.

[5] H.-J. Schek, H. Schuldt, C. Schuler, and R. Weber. Infrastructure for information spaces. In *Proceedings of Advances in Databases and Information Systems, 6th East European Conference, ADBIS 2002*, volume 2435 of *Lecture Notes in Computer Science*, pages 23–36, Bratislava, Slovakia, September 2002. Springer.

[6] H.-J. Schek, H. Schuldt, and R. Weber. Hyperdatabases – Infrastructure for the Information Space. In *Proceedings of the 6th IFIP 2.6 Working Conference on Visual Database Systems (VDB'02)*, Brisbane, Australia, May 2002.

[7] H. Schuldt, G. Alonso, C. Beeri, and H.-J. Schek. Atomicity and Isolation for Transactional Processes. *ACM TODS*, 27(1), March 2002.

[8] SOAP – Simple Object Access Protocol. `http://www.w3.org/TR/SOAP/`.

[9] I. Wladawsky-Berger. Advancing the Internet into the Future. Talk at the *International Conference Shaping the Information Society in Europe 2002*, April 2002. `http://www.ibm.com/de/entwicklung/academia/index.html`.

[10] WSDL – Web Service Description Language. `http://www.w3.org/TR/wsdl/`.