The Business Grid: Providing Transactional Business Processes via Grid Services

Frank Leymann[1] and Kai Güntzel[2]

[1] IBM Software Group, Schönaicherstr. 220,
71032 Böblingen, Germany
Ley1@de.ibm.com
[2] Fraunhofer Institut für Arbeitswirtschaft und Organisation IAO, Nobelstr. 12,
70569 Stuttgart, Germany
Kai.Guentzel@iao.fraunhofer.de

Abstract. Web Services provide a suitable technical foundation for making business processes accessible within and across enterprises. The business logic encapsulated inside Web Services often resides in already existing transactional backend-systems. However, the scope of these systems is normally limited to their domain and is not distributed across heterogeneous environments.
In this paper, we investigate the impact of the emerging Web Service technology on transactional backend-systems: Transactional context needs to propagate from activities or even business processes to services they use. Negotiations between service requestors and services on context to be propagate can be done automatically based on policies attached to the corresponding Web Service descriptions. Corresponding standards and mechanisms form the basis of a new computing and middleware paradigm: the Business Grid.
Some exemplary research work to be done to actually build the outlined Business Grid environment is sketched.

1 Introduction

Web Services can be considered as the seminal integration solution for software architecture in information technology. Nearly every software vendor and especially all major suppliers of middleware technology are supporting this new computing paradigm. Before we will explain our ideas concerning the potential of combining Web Service technology, transactional backend-systems and here especially ERP-systems and Grid environments, we will define what we understand by the term Web Service.
We describe the Service Oriented Architecture as the underpinning for Web Service technology and the necessary steps for providing and requesting Web Services which are hosted in traditional backend-systems in section 2. The problems which will arise when integrating transactional backend-systems across heterogeneous environments will be discussed and a possible solution for this challenge namely BPEL4WS (Business Process Execution Language for Web Services), WS-Transaction and WS-Coordination, the specifications for a comprehensive business process automation framework that allows companies to leverage the benefits of the Web Services architecture to create and automate business transactions will be sketched in section 3. The

Web Services Policy Framework which provides a general purpose model and corresponding syntax to describe and communicate the policies of a Web Service – a necessary feature when searching and matching Web Services with special properties – could be augmented in conjunction with WS-Coordination to achieve a transactional handshake. Therefore, WS-Policy together with WS-Coordination helps to select the correct Web Service regarding operational properties which is outlined in section 4. In chapter 5 the evolving Open Grid Services Architecture in which a Grid provides an extensible set of services that can be aggregated in various ways is shortly explained. Further on, the term Business Grid vaguely used before is specified and compared with classical Computational Grids. Classical transactional backend-systems like ERP-systems and the new concept of Business Grids are combined into a new system architecture – the effects on traditional software architectures are presented in section 6. Finally, in chapter 7, we give a short example of a Business Grid architecture which could be used to provide fined-grained SAP R/3-transactions as Grid Services. Some of these provided Grid Services are requested and selected due to operational and business properties. We will show how failures can be compensated and which precautions must be made in SAP R/3 to enable Business Grid-support. We conclude in chapter 8 with a summary and give an outlook on future research.

2 The Service Oriented Architecture and Web Services

A Web Service can be considered as a virtual software component that hides middleware idiosyncrasies like the underlying component model, invocation protocol etc. as far as possible [1]. Web Services are provided (→ publish) by Service Providers, discovered using a Service Directory (→ find) and bound dynamically by Service Requestors (→ bind).

All the action is based on standards-based protocols and formats in order that interoperability is achieved, even when the partners are using heterogeneous hard- and software. The transport medium used by this Service Oriented Architecture (SOA) is (normally) based on TCP/IP and the data exchanged by the involved partners is encoded in XML. The platform-neutral and globally available invocation mechanism for Web Services is the Simple Object Access Protocol (SOAP) [2]. Web Services can be considered as firewall-compliant remote procedure calls (RPC), since the standard transport protocol used by SOAP is HTTP. The Web Service invocation is packed in the payload of a SOAP-message (to be precise in the SOAP body of the SOAP envelope which forms the SOAP payload) and sent to a SOAP Endpoint Reference, the Service Provider.

The Basic Web Services Stack

Web Services can be discovered by browsing a Service Directory, the so called Universal Description, Discovery and Integration or for short UDDI [3]. Service Providers are populating UDDI with information about the Web Services provided by them. This information contains amongst technical descriptions (which methods can be called, what are the parameters etc.) expressed in the Web Service Description Language (WSDL) [4] information about the Service Provider himself, i.e. some background about the person providing the service. The Service Requestor selects an appropriate Web Service after querying the UDDI Business Registry (UBR) and finding a matching service. As a rule, this Web Service is called in a SOAP-message, the XML-based RPC.

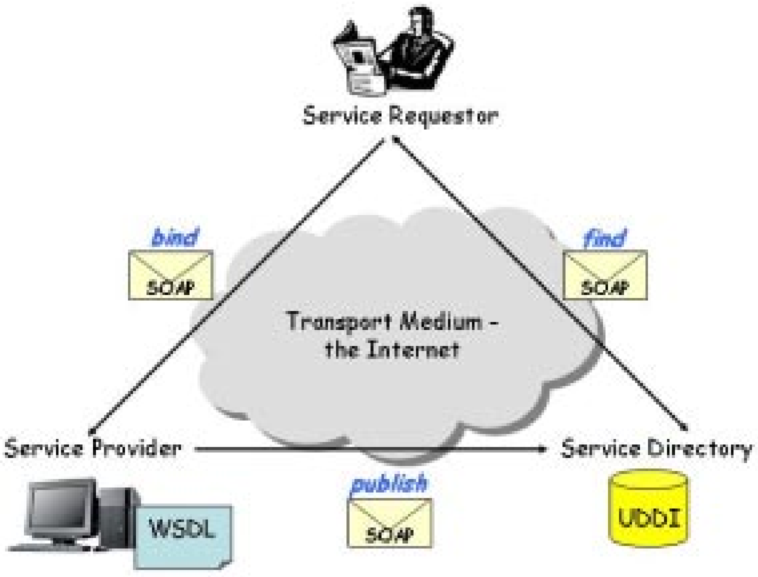

Fig. 1. The Service Oriented Architecture

Web Service Aggregation

The functions and the business logic inside the provided Web Services could be implemented when there is a need for this specific function. In fact, the business logic is often already in place and implemented on some systems, e.g. an inventory check against an inventory control system. Now, this functionality should be provided via standardized interfaces and transport protocols so that other applications can "consume" this feature in their own application without explicit stub programming as e.g. in CORBA [5]. Further on, this functionality could be provided to business partners with two positive effects: first, the business partner is checking the inventory himself before placing an order for his customer and by this reduces the internal outlays of the Service Provider and second the business partner can retrieve the information needed whenever he wants and thus becomes independent of the Service Providers office hours.

As Web Services are not for direct user-interaction as web browsers but for B2B integration we have to distinguish between "stand-alone" Web Services and Web

Services which will be aggregated into process flows. In the latter case, we have to cope with deferred failure or abort situations. This can become tricky, if the process-step performed by the Web Service is already committed, the following or adjacent step must be cancelled and therefore the Web Service must be cancelled, i.e. rolled back, too. This is a quite difficult task, when then different steps are performed in a distributed environment and locking [6] can't be assured due to costs and negative system throughput at the participating resource managers. Instead of taking the risk of inconsistencies after a failed rollback, compensation and compensation spheres [7] are an alternative solution to "classic" ACID-transactions [8].

3 Transactions and Process-Support in a Web Services World

In the real world, companies won't just consume "lonely" offered Web Services. Indeed, these software granules will be aggregated into more complex services or workflows to offer an even more sophisticated service for either internal use or to offer this encapsulated and augmented service as a new Web Service to the outside world. Anyway, the services offered will manipulate resources and these actions – whether they are short- or long-lived – should preserve the consistent state of the involved underlying systems.

In classical system-landscapes, transaction support, even across system boundaries can be achieved with transaction monitors and corresponding transaction protocols, e.g. 2PhaseCommit [9]. These concepts and techniques can't be adopted without additional assumptions or modifications to the Web Services world, on the one hand the involved transaction monitors and protocols are not known a priori when dynamically selecting a Web Service, on the other hand the selected Web Services are often integrated in higher situated workflows [10] and are therefore dependant on the outcome from long-running processes, even when the Web Service itself is a short-lived transaction.

BPEL4WS

Initially, both, IBM and Microsoft, developed their own languages for process-modelling in Web Service environments: WSFL from IBM [12] and XLANG from Microsoft [11]. But soon after that, both companies undertook an effort to combine both languages into a new standard proposal called Business Process Execution Language for Web Services (BPEL4WS) [14]. At the time this paper has been written, BPEL4WS has been published in a second release (BPEL4WS 1.1 [15]), and it has been submitted to OASIS for formal standardization.

Business processes expressed in BPEL4WS export and import functionality by using Web Service interfaces exclusively [14] and by specifying the potential execution order of operations from a collection of Web Services allows the definition of both business processes that make use of Web Services and business processes that externalize their functionality as Web Services.

This Web Service composition language builds directly on top of WSDL. An important difference between WSDL and BPEL4WS are states. WSDL is essentially stateless because the language is unaware of states between operations. The only state supported is the state between sending and receiving a message in a request-response or solicit-response operation. But only by recording the state it becomes possible what action should be taken next and thus enabling business transactions [16].

For a simple BPEL4WS example have a look at the BPEL4WS specification [14] or at the introducing article about "Business Processes in a Web Services World" [17]. As BPEL4WS is a work in progress a number of required features are absent from the current specification, for example, the actual status of a business process can't be queried.

WS-Coordination and WS-Transaction

WS-Coordination (WS-C) [18] describes an extensible framework for providing coordination protocols that coordinate the actions of distributed applications. This framework enables the involved participants to reach consistent agreement on the outcome of distributed activities. WS-C doesn't provide support for processes.

WS-Transaction (WS-Tx) [19] describes the necessary coordination types that are used with WS-C: an atomic transaction (AT) is used to coordinate activities having a short duration and executed within limited trust domains. A business activity (BA) is used to coordinate activities that are long in duration and desire to apply business logic to handle business exceptions. The long duration prohibits locking data resources to make actions tentative and hidden from other applications. Instead, actions are applied immediately and are permanent.

4 Service Arrangements with WS-Policy

Finding and selecting a matching Web Service is the first step in building a business process based on Web Services. But how to negotiate e.g. security, or transactional behaviour? For example, how can the work already done be compensated; does the backend-system providing the selected Web Service support a 2PhaseCommit or does this system work with compensation spheres instead of atomic transactions?

Transactional Handshake between Web Services

The Web Service Policy Framework (WS-Policy) provides a general purpose model and corresponding syntax to describe and communicate the policies of a Web Service [20]. The goal of WS-Policy is to provide the mechanisms needed to enable Web Service Providers to specify their policy information. This information is expressed through an XML-based structure called a policy expression and a core set of grammar elements to indicate how the contained policy assertions apply to a policy subject, i.e. the endpoint or resource to which the policy is bound. These policy information can be either associated with specific instances of services or be referenced from WSDL definitions [21]. Thus, the Service Provider can expose the conditions under which he provides the Web Service.

WS-Policy can be used to achieve a “transactional handshake” between the Service Provider and the Service Requestor. The following two figures show in a simplified manner how a Service Requestor selects in a Contracting Phase a Web Service based on certain policies. During the Binding Phase the two participants interact with a common coordinator as the activity propagates between them:

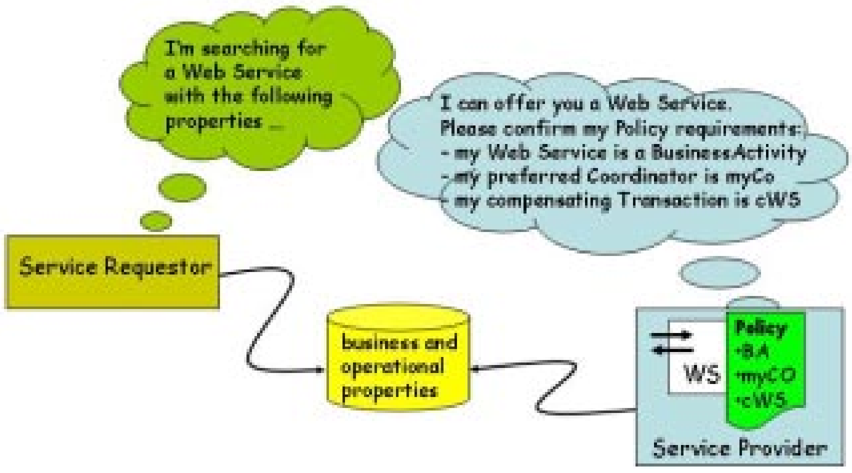
I'm searching for a Web Service with the following properties ...
I can offer you a Web Service. Please confirm my Policy requirements: - my Web Service is a BusinessActivity - my preferred Coordinator is myCo - my compensating Transaction is cWS
Service Requestor
business and operational properties
WS
Policy -BA -myCO -cWS
Service Provider

Fig. 2. Contracting Phase

(1) Due to information provided in the attached policy file, the Service Requestor knows that the requested service is a long-running BusinessActivity and therefore requests the creation of a CoordinationContext at the Activation Service ASmyCo at the specified Coordinator myCo. (2) This CoordinationContext C1 comprises a URI to the Service Provider’s BusinessActivity BA. (3) In a third step, the Service Provider will then register to participate in the BusinessActivityProtocol.
Selecting a Web Service will therefore be done, first, by matching business and operational properties, e.g. the Service is performed under the control of a mission critical transaction monitor and the data produced will be stored in a relational database, and, second, by agreeing on matching policies, e.g. that in case of failures the invoked Web Service has to be compensated with a dedicated compensating Web Service offered by the same Service Provider.

5 The Business Grid

When Web Services will become more and more widespread, the consumption of the Services offered can grow up rapidly due to the standardized integration of the software granules. Inherent, Service Providers have to cope with well know problems as load balancing, dynamic selection of the most suited Service, billing techniques and Service Level Agreements [22].

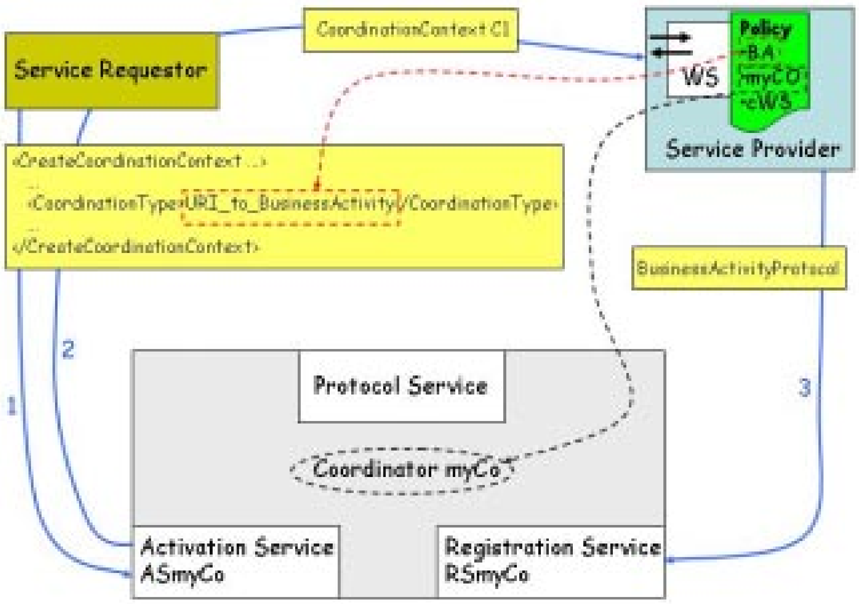

Fig. 3. Binding Phase

Service Oriented Architecture can be considered as new underpinning for a more generic Application Service Providing (ASP) model. Serving finer grained software components augmented with a standardized usage model rather than complete applications (which could be also realised with Web Services) requires to think about a scalable and service(level)-oriented middleware which will provide the necessary basic services to enable the Service Oriented Architecture even across system and enterprise domains.
Therefore it is only consequent, that researchers began to "merge" the concepts and ideas of Grid Computing with the Service Oriented Architecture and thus to adopt the Web Services Architecture [23] to the classical number crunching grid computing environment [24].

In our terminology, a Business Grid is a Grid environment with Web Services as the provided resources rather than processor time, storage area or bandwidth as it is the case with Computational Grids thus resulting in a scalable Service Oriented Architecture. Instead of a few requestors in "classical" Grid computing scenarios the number of potential service requestors in such a Business Grid environment is a priori unknown and can be enormous. There's another observation that helps distinguishing Computational Grids and Business Grids: in classic Grid environments, requestors had to cope with the issue that their software and/or their data-sets had to be split and distributed to the performing IT resources, i.e. to the resource providers in the network. When the job finished, the distributed results had to be assembled by the originating requestor. In contrast, in a Business Grid environment, the Service Requestor calls the Web Service granule over the Grid, gets his job done and continues with other tasks. The challenge in Business Grids is first to find the best Web Service for the job to be done and second to instantiate the selected Web Service by the Grid environment that the work is done as it were the only service provisioned in the Business Grid.

Service Domain

The Business Grid provides the runtime environment for provisioning Web Services across heterogeneous platforms and thus ensures the optimal utilization of resources. The selection process for a suitable Web Service is nearly the same as in a "normal" Service Oriented Architecture just that we are dealing with business and operational properties and that the Service Provider is probably not known thus resulting in a contracting phase done automatically by the environment upon the properties and the requirements.

The notion of a Service Domain [25] introduces a collection of comparable or related Web Services through a common service entry point and thus enables the selection of a specific Web Service instance not just on availability but also on the basis of Service Level Agreements (SLA) and business arrangements. Therefore, the Service Domain architecture can be described as a service sharing and aggregation model through a service entry interface [25].

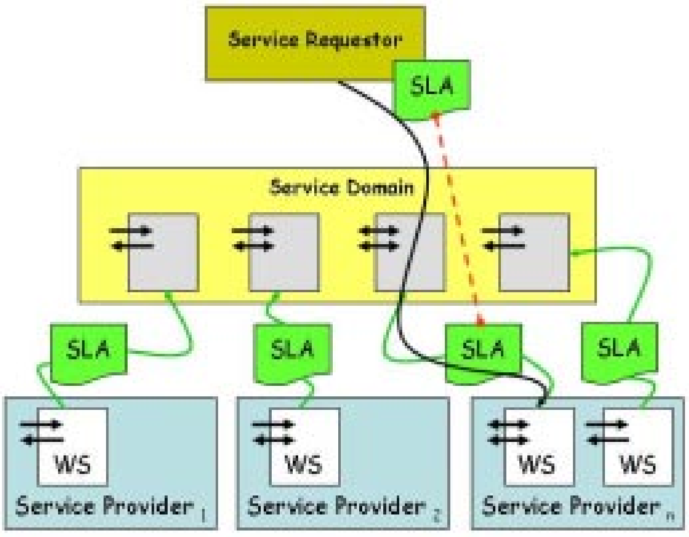

Fig. 4. A Service Domain

The conjunction of Grid technologies and Service Domains therefore forming the new concept called the Business Grid enables a new form of resource sharing and pooling of resources: in the future, whenever a company or a clerk needs some specific functionality, the user won't install an additional server component on its own server, instead he will "ask" for this functionality in the Business Grid environment he subscribed to. Then, it's up to the Service Domain to fetch the most suited service; the Business Grid has to schedule the job and to return the results to the requestor.

Open Grid Services Architecture

Several Grid technologies and toolkits are available today, such as Unicore [26] and the Open Grid Services Architecture (OGSA) [27] for instance. In what follows, we are focussing on OGSA because OGSA together with the GLOBUS Toolkit [28] is providing an infrastructure for Grid Computing that is based on a Service Oriented Architecture.

Building on concepts and technologies from both the Grid and the Web Services communities, OGSA defines a uniform exposed service semantics – the Grid Service [29]. It integrates key Grid technologies with Web Service mechanisms to create a distributed framework based around the Open Grid Services Infrastructure (OGSI) [30].

OGSI defines, in terms of WSDL interfaces and associated conventions, extensions and refinements of Web Services standards to support basic Grid behaviours. These OGSI-compliant Grid services (see figure 5) are intended to form the components of the Grid infrastructure [31]. Associated with each Grid interface is a potentially dynamic set of service data elements which provide a standard representation for information about Grid service instances, facilitating the maintenance of internal state for their lifetime [32].

OGSI is based on Web Services and in particular uses WSDL as the mechanism to describe the interfaces of Grid Services. However, OGSI extends WSDL 1.1 in two areas: interface (portType) inheritance and the ability to describe additional information elements on a portType [30].

6 Backend-Systems and the Business Grid

The Globus Toolkit, as a possible OGSA-compliant implementation base for a Business Grid, can provide the necessary framework to integrate traditional backend-systems.

However, one cannot plug an enterprise system in a Business Grid and anticipate all the benefits mentioned before: scalability, reliability and interoperability. If the concerned backend-system is already decomposed into finer software granules, i.e. Web Services, one has to look at the underlying software architecture of the affected system(s). How to cope with monitoring? Who's responsible for load-balancing: the Grid or the backend? Which system will ensure consistency after a failure of composed Web Services from different backend-systems? These questions must be answered before adjusting the affected software infrastructure.

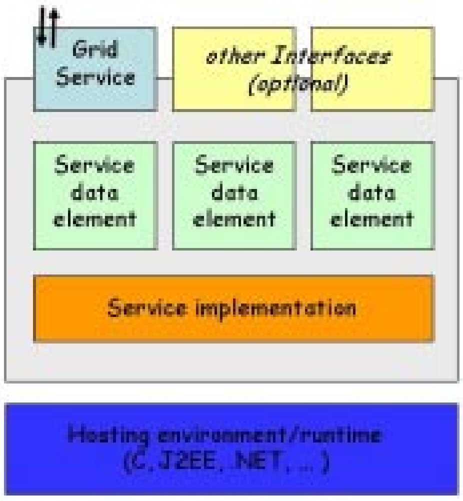

Fig. 5. OGSA Grid Service [29]

The Business Grid – Comparison with Traditional Client-Server-Computing

If we have a look at figure 6, we can observe an analogy to the classical Client-Server-Computing paradigm: a client (Service Requestor) requests a job to be done (Web Service) by a server (Service Provider). In our case, the loadbalancer moved from a dedicated server component to the Business Grid, this means that instead of reporting their actual load to their proprietary dispatching server, the Service Providers are reporting their status to the infrastructure provided by the Grid environment. If a Service Requestor asks for a specific service, the Business Grid selects as in the "classic" SOA an appropriate Service Provider, but considers besides the matching business and operational properties also system load, response time, availability and transactional behaviour.

Monitoring

Even more complicated than selecting the most suitable service and thus Service Provider amongst all services is the issue of how to deal with business processes, composed from different Web Services residing in heterogeneous backend-systems. Even if the exposed services can be triggered from the outside world, is it possible to monitor the progress of the different steps from a general perspective? In the past, research has been done on integrating workflows from existing systems in higher situated workflows [32, 33]. In the world of Web Services, status monitoring especially over different domains, is currently not supported. Therefore, workarounds have to be provided. A Service Provider can send either periodically a message to the Service

Requestor including the current status or every time a status has changed or a Service Requestor "polls" for the current status, i.e. requests an update. This is cumbersome because the Web Service itself has to deal with monitoring. Normally, every backend-system has a monitoring component, which tracks the progress of the involved processes. Nevertheless, we envision that in future Web Service enabled flow technologies facilitate corresponding monitoring features.

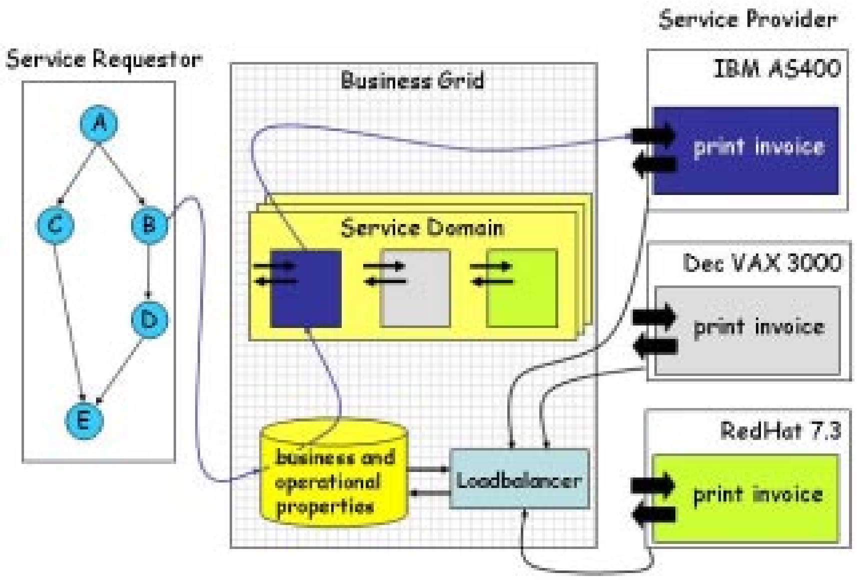

Fig. 6. Client-Server Business Grid-like

Error-Handling and Compensation

If a business process is aborted due to an application error or cancellation from the Service Requestor, already completed activities (i.e. Web Services) typically have to be undone. As mentioned in chapter 3, these Web Services often represent long running computations, and to ensure suitable performance characteristics they do not implement ACID properties. This implies that these Web Services offer compensation actions to undo transactions that they already committed in course of their processing. Since the process' activities will be bound at runtime by the Business Grid, the Business Grid has to maintain a persistent log ("compensation log") specifying pairs of activity implementations and corresponding compensating activity implementations. Often, the latter will be derived from information given in the policy-file of the activity proper.

In figure 7, the Service Requestor cancels his process. This means that the compensating activities have to be bound by the Business Grid. This is based on the information about compensating activity implementations in the compensation log. Therefore, there is no need that the Service Requestor is involved in specifying compensation actions. The Business Grid has all required information resulting in an opaque error-handling and compensation.

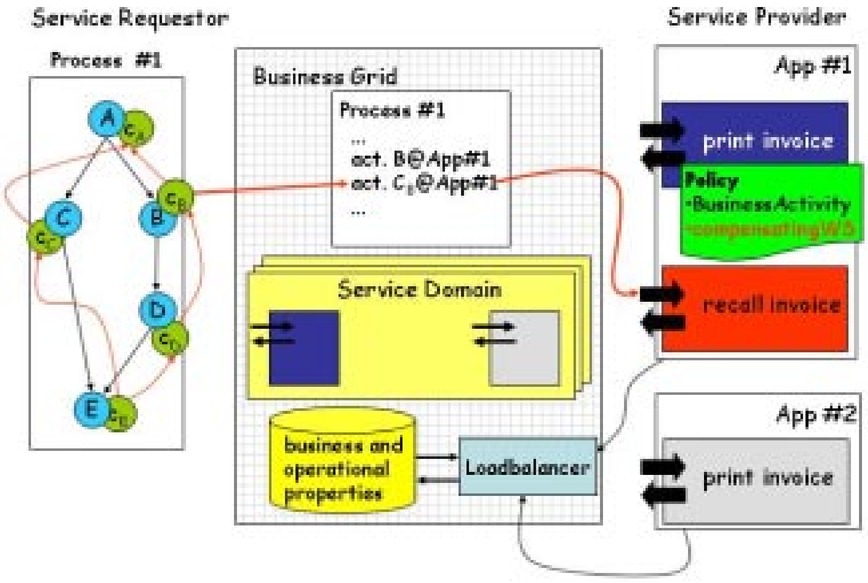

Fig. 7. Error-handling in Business Grids

7 Web Services-Based Integration of SAP R/3 in a Business Grid

SAP R/3 [34], the Enterprise Resource Planning (ERP) solution from SAP, is one of the most commonly used transactional application system for business management purposes. SAP's NetWeaver [35] Architecture makes functional components respective transactions (e.g. BAPI, RFC, IDOC) available as a Web Service.

Nevertheless it's important to understand the implications resulting from integrating SAP R/3 in a Business Grid environment. As mentioned in chapter 6, when integrating a transactional backend-system like SAP R/3 into a Business Grid, information about load and availability of the backend-system should be available to the Business Grid. In case of R/3 this information might stem from the SAP Message Server and the Dispatcher-Workprocess. Otherwise, the selection of the current load-optimized R/3 Application Server within the actual SAP R/3 environment is not possible.

SAP R/3-Transactions via Web Services

As transactions and thus the corresponding Web Services provided by SAP R/3 get long-running rather than ACID transactions, compensation must be kept in mind. Therefore, for each transaction there is a compensating one, undoing the steps as good as possible in case of an undo request. This pair of transactions can be specified in a policy attached to service description, together with the coordination type supported.

Due to performance reasons, not all transactions in SAP R/3 are actually executed and applied to the underlying resource manager at the time of when they are requested to commit. Instead the corresponding requests are written to a special queue from which they processed and committed later, or bundled with other transactions in a batch job.

In the Business Grid scenario, the compensating transaction thus can only be performed after the intended transaction has been finally committed to the database system. Otherwise, inconsistencies may arise. This requirement can be achieved with special SAP R/3-customizing, enforcing the commit of transactions called by Web Services directly after the last step performed in the Logical Unit of Work [36] in SAP R/3. However, this mechanism will likely have performance impacts.

8 Summary

Service Oriented Architecture establishes the base for a new area of distributed computing. We positioned basic Web Service technology as a means for accessing an application and not for implementing it.

The Web Services paradigm therefore enables the transition from tightly coupled applications to loosely coupled services. With the support of transactions and the composition of Web Services into processes of any degree of complexity, BPEL4WS supports orchestration of Web Services.

Contracting between Service Requestor and Service Provider regarding the preferred or even enforced Coordination Service could be realised based on attached policies. Similarly, we have argued that the compensating Web Service of a certain Web Service should published via policies too. Finally, the coordination type, i.e. the information whether the Web Service is realized via an atomic transaction or a long running business activity, for example, has to be known to enable a transactional handshake between the two parties.

Grid concepts and the evolving Open Grid Services Architecture have been briefly introduced, as well as OGSI, integrating Grid technology and Web Service technology, providing a uniform service-oriented architecture for Grid environments.

In the last section, we brought up the potential of integrating SAP R/3 with a Business Grid: every transaction or process inside SAP R/3 can be externalized as Web respective Grid Service, thus enabling a totally new integration aspect – SAP R/3-processes as loosely coupled services in higher-level workflows.

Nevertheless, integrating transactional backend-systems in Business Grids implies a clear concept how to resolve failures: compensation based recovery has to assume that activities really have been committed. Otherwise, inconsistencies will arise.

References

1. Frank Leymann: *Web Services and Business Processes*, Fraunhofer IAO Symposium Collaborative Business, Stuttgart, Germany, July 9, 2002
2. Martin Gudgin et al.: *SOAP Version 1.2 Part 1: Messaging Framework*, W3C, December 19, 2002, http://www.w3.org/TR/2002/CR-soap12-part1-20021219/
3. UDDI.org: *UDDI Version 3.0*, Published Specification, July 19, 2002, http://uddi.org/pubs/uddi-v3.00-published-20020719.pdf
4. Eric Christensen et al.: *Web Services Description Language (WSDL) 1.1*, W3C, March 15, 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315
5. OMG: *Common Object Request Broker Architecture (CORBA)*, December 6, 2002, http://www.omg.org/technology/documents/formal/corba_iiop.htm
6. Philip A. Bernstein et al.: *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987
7. Frank Leymann, Dieter Roller: *Production Workflow*, Prentice Hall, 2000
8. Jim Gray, Andreas Reuter: *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers, 1992
9. Philip A. Bernstein, Eric Newcomer: *Principles of Transaction Processing*, Morgan Kaufmann Publishers, 1997
10. Hans-Jörg Schek et al.: *Workflows over Workflows: Practical Experiences with the Integration of SAP R/3 Business Workflows in WISE*, in Proceedings of the Informatik '99 Workshop, Paderborn, Germany, October 1999
11. Satish Thatte: *XLANG – Web Services for Business Process Design*, Microsoft, 2001, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm
12. Frank Leymann: *Web Services Flow Language*, IBM, 2001, http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
13. Assaf Arkin et al.: *Web Service Choreography Interface (WSCI)*, BEA, Intalio, SAP, SUN, 2002, ftp://edownload:BUY_ME@ftpna2.bea.com/pub/downloads/wsci-spec-10.pdf
14. Francisco Curbera et al.: *Business Process Execution Language for Web Services (BPEL4WS) 1.0*, BEA, IBM, Microsoft, July 31, 2002, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf
15. Tony Andrews et al.: *Business Process Execution Language for Web Services (BPEL4WS) 1.1*, BEA, IBM, Microsoft, SAP, Siebel, March 31, 2003, ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf
16. Will van der Alst: *Don't go with the Flow: Web Services Composition Standards Exposed*, in Web Services: Been There, Done That?, IEEE, 2003
16. Frank Leymann, Dieter Roller: *Business Processes in a Web Services World*, IBM developerworks, August 1, 2002, http://www-106.ibm.com/developerworks/webservices/library/ws-bpelwp/
18. Felipe Cabrera et al.: *Web Services Coordination (WS-Coordination) 1.0*, BEA, IBM, Microsoft, August 9, 2002, http://www-106.ibm.com/developerworks/library/ws-coor/
19. Felipe Cabrera et al.: *Web Services Transation (WS-Transaction) 1.0*, BEA, IBM, Microsoft, August 9, 2002, http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/
20. Don Box et al.: *Web Services Policy Framework (WS-Policy*, BEA, IBM, Microsoft, SAP, December 18, 2002, http://www-106.ibm.com/developerworks/webservices/library/ws-polfram/
21. Don Box et al.: Web Services Policy Attachment (WS-PolicyAttachment), BEA, IBM, Microsoft, SAP, December 18, 2002, http://www-106.ibm.com/developerworks/library/ws-polatt/
22. Alexander Keller, Heiko Ludwig: *The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services*, IBM, May 22, 2002

23. Michael Champion et al.: *Web Services Architecture*, W3C Working Draft, November 14, 2002, http://www.w3.org/TR/2002/WD-ws-arch-20021114/
24. Ian Foster, Carl Kesselman: *The Grid: Blueprint for a New Computing Architecture*, Morgan Kaufmann Publishers, 1999
25. Yih-Shin Tan et al.*: Business Service Grid*, Part 1,2,3, IBM developerWorks, February 1, 2003, http://www-106.ibm.com/developerworks/grid/library/i-servicegrid/
26. The Unicore Forum, http://www.unicore.org
27. Ian Foster et al.: *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, The Global Grid Forum, the latest Version can be found at http://www.globus.org/research/papers/ogsa.pdf
28. The Globus Project, http://www.globus.org
29. Ian Foster et al.: *Grid Services for Distributed System Integration*, IEEE, 2002
30. Steve Tuecke et al.: *Open Grid Services Infrastrucutre (OGSI)*, The Global Grid Forum, the latest version can be found at http://www.ggf.org/ogsi-wg
31. Ian Foster et al.: *The Open Grid Services Architecture (OGSA)*, The Global Grid Forum, the latest version can be found at http://www.ggf.org/ogsa-wg
32. Andre Naef et al.: *Monitoring komplexer Dienste in unternehmensübergreifenden Prozessen am Beispiel von SAP R/3 Business Workflow*, in Proceedings 9. Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft, Oldenburg, Germany, March 1999
33. Crossflow: *Cross-Organizational Workflow Support in Virtual Enterprises*, ESPRIT Project 28635, http://www.crossflow.org
34. SAP AG: *SAP R/3 Enterprise*, Walldorf, Germany
35. SAP AG: *SAP NetWeaver*, Walldorf, Germany
36. SAP AG: *ADM100 SAP Web AS Administration I*, Education, Walldorf Germany