FROM CORBA TO WEB SERVICES COMPOSITION

Denivaldo Lopes[1,2] and Slimane Hammoudi[2]
[1,2]*Université de Nantes, Nantes, France*
[2] *Ecole Supérieure d'Electronique de l'Ouest- ESEO, Angers, France*

Keywords: Web Service Composition, Transaction, Workflow and CORBA

Abstract: CORBA has some positive aspects to develop applications, but its communication model is limited to accomplish interactions among clients and enterprise servers on the Web. The technologies of Web Service seem to offer a better answer for developing distributed applications on the Web. The first part of this paper is a discussion about the evolution of CORBA and of Web Services, showing their benefits and limitations. The new solutions provided by the technologies of Web Services (XML, WSDL, UDDI and SOAP) are more adapted for the Web than CORBA. However, these technologies are not sufficient to compose Web Services, which represents a real challenge. Workflow Technology seems to be a better answer for this challenge. The second part of this paper deals with this integration of Workflow technology and Web service that is designed in WEWS. An approach for transaction based on conversation plus optimistic commit protocol is also presented. A comparison of our work and other propositions is provided too, highlighting similarities and differences.

1 INTRODUCTION

In the last decade, people thought that Distributed Object Technology (DOT) could be the solution for the development of wide-scale distributed systems on the Web* (Harkey, 1998), (Northrop, 1997). DOT can encapsulate the software components in different levels of detail and makes possible the inter-process communication in a transparent way. This is realized through utilization of middleware, i.e. a software tier among hardware, operating systems and applications. It provides homogeneity of development for heterogeneous platforms. The OMG's Common Object Request Broker Architecture (CORBA) (OMG, 2001), Microsoft's Distributed Component Object Model (DCOM) (it is implemented only for Windows) and Sun's Java Remote Method Interface (RMI) are the fundamental examples of middleware. Amongst these, CORBA is a specification for general distributed system architecture and was the major element to support the utilization of DOT. But CORBA has problems, e.g. it is complex, it requires specialized training and it has a high cost for development.

In the last years, a set of technologies based on XML has been developed to facilitate the development of applications on Web, i.e. SOAP, UDDI and WSDL. These technologies have resulted in the creation of Web Services that are Web applications that interact with other Web applications using these open standards.

However, this set of technologies is limited and it cannot meet all the requirements in Web services development, i.e. reliability, transactions, security, scalability, accountability and management (Zhang, 2001), (Tsur, 2001). Moreover, these technologies are not sufficient to allow service composition. In this paper, we suggest an integration of Web Services and Workflow to meet several of these requirements and take into account service composition.

This paper is organized as follows. Section 2 is a discussion about CORBA and Web Services, detaching their benefits, limitations, similarities and differences. Section 3 presents some modification on WSFL to better compose Web Services. Section 4 is our proposal to enable transactions in the context of Web Services. Section 5 is a discussion about the architecture of Workflow Environment for Web Services (WEWS) to support the composition of Web Services and transaction. Section 6 shows the comparison among WEWS and other frameworks. The last section is the conclusion about our work.

* In this paper, we use the word Web to make reference to the Internet (i.e. the large network) and to the Web (i.e. an Internet service).

Lopes D. and Hammoudi S. (2004).
FROM CORBA TO WEB SERVICES COMPOSITION.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 114-121
DOI: 10.5220/0002635901140121

2 CORBA AND WEB SERVICES

In the DOT context, Web objects (e.g. ORBlets) were the major approach of CORBA on the Web, because they would stimulate the adoption and fast development of applications on the Web (Harkey, 1998). However, these events did not happen because the standardization of the Internet Inter-ORB Protocol (IIOP) had a long delay (OMG, 2001). Moreover, the submission for IIOP firewall traversal is not a consistent standard yet, and the integration of CORBA/IIOP with Netscape browser and server was not well accepted in industrial circles.

The frustrations with CORBA have modified the vision of many enterprises about the future of distributed systems on the Web.

The set of technologies formed of eXtensible Markup Language (XML), Simple Object Access Protocol (SOAP) (W3C, 2001a), Web Services Description Language (WSDL) (W3C, 2001b) and Universal Description, Discovery and Integration (UDDI) (UDDI Project, 2000) is being accepted for the development of the next generation of applications on the Web.

2.1 CORBA and Object Technology

CORBA is the best example of DOT, although their benefits have not yet been totally explored. Many services and facilities designed for it are not very well standardized or implemented. Its qualities are unquestionable, e.g. portability and interoperability across heterogeneous platforms (language, operating system, hardware and other Object Request Brokers-ORBs), transparency, real time specification, uniform error detection, adapted to wrap legacy applications and uniform security mechanism (SAS) (Northrop, 1997), (OMG, 2001).

2.2 The CORBA Stack

CORBA can be seen as a stack composed of Interface Definition Language (IDL), CORBA Services, Stubs/Skeletons, Common Data Representation (CDR) and GIOP/IIOP (OMG, 2001), (OMG, 2000).

IDL is used to describe the interfaces that allow client objects to call up the remote object implementations. In other words, it defines a contract between client and server.

CORBA Services define a set of services, e.g. Trading Object, Security and Transaction. Among these services, we draw attention to the Trading Object Service that enables a service provider to register the description of a service and the location of an interface on the trader. Afterwards, the requester asks for a service with characteristics which are specific to the trader. The trader receives the requisition, searches for a suitable service object based on the described characteristics and responds to the requester passing the location of selected service's interface (OMG, 2000).

The client object uses the stubs to access a specific object implementation. The client Stubs are created from the IDL description to make access to a service operations and data possible. The stubs make calls on the ORB using interfaces that are private or optimized to a particular ORB. The object implementation uses the skeletons as an adapter to make its operations and data available. The ORB calls the operations and has access to data on object implementation through the skeleton.

Common Data Representation (CDR) is a transfer syntax definition that maps OMG IDL data types into a bi-canonical and low-level representation (i.e. an octet stream) for transfer between ORBs and Inter-ORB bridges. In other words, it implements an Electronic Data Interchange (EDI).

General Inter-ORB Protocol (GIOP) is a low-level protocol that specifies standard transfer syntax and a set of message formats to enable communication between ORBs. The Internet Inter-ORB Protocol (IIOP) defines how GIOP messages are exchanged using TPC/IP connections on the Internet.

2.3 Web Services

Web Services can be seen as an evolution of the Web based on lessons learned in past decades with middleware (including CORBA) and markup languages (e.g. XML).

The main characteristics of Web Services are independence of platform (language, operating system and hardware), a world-wide scenario for transactions, the utilization of multiple transport protocol (HTTP, SMTP or FTP), the message encoded in XML, friendly behavior with firewall, easily adaptable with legacy systems, and the localization as Uniform Resource Identifier (URI).

2.3.1The Web Services Stack

The Web Services stack is composed of WSDL, UDDI, SOAP, XML and HTTP (or SMTP or FTP).

WSDL is an abstract definition based on XML grammar to describe the syntax and semantics necessary to call a service (W3C, 2001b). This

abstraction is utilized to build an invocation on a service that is often implemented with SOAP.

UDDI is the universal registry for Web Services and its core is based on XML files that may store information about a business entity and its Web Services. It is similar to white pages (to find a service by contact, name and address), yellow pages (to find a service by topic based in standard taxonomies) and green pages (to find a service by technical characteristic) (UDDI Project, 2000).

SOAP is a protocol based on XML and it is used to exchange information in decentralized and distributed systems (W3C, 2001a).

XML is an extensible markup language that has been used for documents and data representation. In other words, it resolves the problem of Electronic Data Interchange (EDI) and it can be rapidly modified to address new types or requirements.

One advantage of Web Services is the possibility to use multiple protocols to transport the request and response, e.g. Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP) and File Transfer Protocol (FTP).

The concepts around Web Services are not new. CORBA promised a structure that would make it possible to build services (as components), to publish services, to find services and to bind these services; in other words, it would make possible the reutilization of a component by other components. This notion is present in the Trading Object Service designed for CORBA (OMG, 2000) and developed by PrismeTech's OpenFusion software (www.prismtechnologies.com). However, CORBA is not sufficiently loose-coupled to implement final applications on the Web. Table 1 shows the CORBA and Web Services stack.

Table 1: CORBA and Web Services Stack

Stack	
CORBA	**Web Services**
IDL	WSDL
CORBA Services	UDDI
Stubs/Skeletons	SOAP
CDR	XML
GIOP/IIOP	HTTP (or SMTP, FTP)

Analyzing the CORBA and the Web Services stacks, we come to the conclusion that together they can be used to provide a powerful set of technologies to cover a vast scope of domains. In (Gokhale, 2002), the authors have demonstrated that the combination of CORBA and Web Services brings some benefits.

In fact, applications for the Web can be implemented with Web Service technologies or with CORBA technology; the difference will be observed in the efforts to develop the desired service. In other words, the enterprise will be free to choose what technology is more suitable. For example, software that demands high performance, persistency, notion of object and stable QoS will still be constructed with CORBA and IIOP. It is important to observe that any software developed in CORBA can be developed using Web Services technologies, and vice-versa. In order to choose which technology is suitable, it is necessary to know the characteristics of the problem. Recently, there has been a consensus that Web Service is more adapted to Internet, and CORBA is more adapted to Intranets.

3 WORKFLOW AND WEB SERVICES

Organizations have been concerned with the improvement of the quality of their processes. Therefore, the interest for workflow technology has increased significantly in the industrial and academic fields. A workflow is defined as an organized collection of tasks to perform a business process (WfMC, 2002a). A Workflow Management System (WFMS) is a set of tools implementing techniques that allow the definition, management and execution of workflows (WfMC, 2002a).

Recently, the use of workflow to compose services has been explored and developed by industrial and academic laboratories (Dayal, 2001), (Helal, 2002). These efforts have resulted in the creation of workflow languages and meta-models to describe the Web Service composition, e.g. Web Services Flow Language (WSFL) (Leymann, 2001) and Workflow Process Definition Interface (WfMC, 2002b).

3.1 A modified WSFL

WSFL is an XML language designed to compose Web Services. With WSFL, it is possible to describe how to achieve a particular business goal (i.e. business process) and the interaction of a collection of Web Services (i.e. partner interactions).

In WSFL, a Web Services composition consists of the description of the use of functionalities provided by the set of composite Web Services. The composite Web Services will be executed following a flow model used to describe the execution order. A workflow engine based on Web Services executes this flow model. Afterwards, the global models are used to describe how Web Services interact with each other.

In (van der Aalst, 2003), the authors present the patterns for Workflow and compare them with some

languages of service composition, including WSFL. According to this work, WSFL has the following patterns: sequence; parallel split; synchronization; exclusive choice; simple merge; multiple choice; synchronization merge; implicit termination; MI without synchronization; MI with a priori design time; cancel activity; and cancel case.

We have extended WSFL with the following patterns: multiple merge and discriminator.

The fragment of WSFL schema presented in the figure 1 illustrates the patterns introduced in WSFL.

```
<complexType name= "multimergeType">
  <attribute name= "condition" type= "wsfl:NCNameList"
             use= "required"/>
  <attribute name= "when" type= "wsfl:whenType"
              use= "default" value= "deferred"/>
</complexType>
…
<complexType name= "discriminatorType">
  <attribute name= "condition" type= "wsfl:NCNameList"
             use= "required"/>
  <attribute name= "when" type= "wsfl:whenType"
              use= "default" value= "deferred"/>
</complexType>
…
<complexType name= "activityType">
  <complexContent>
    <extension base="wsdl:operationType">
      …
      <element name="multimerge"
       type="wsfl:mutimegeType"   minOccurs="0"/>
      <element name="discriminator"
       type="wsfl:discriminatorType" minOccurs="0"/>
    </extension>
  </complexContent>
</complexType>
```

Figure 1: The extension of WSFL schema.

4 TRANSACTION FOR WEB SERVICES

The joining of transactions and Workflow concepts provides a powerful mechanism for the management of e-business. However, the traditional models of transactions are not sufficient for the context of the Web. A service transaction cannot be seen as a single ACID transaction (i.e. Atomicity, Consistency, Isolation and Durability). In order to implement transactions suited with the Web Services context, we assume a transaction model based on conversation plus optimistic commit protocol that uses transaction compensation (Ouyang, 2001). Conversations are sequences of interactions between multiple Web services that enable the conversational transaction. In this context, a transaction that has sub transactions aborted is able to create a new sub transaction, instead of direct use of all-or-nothing semantics.

Our transaction protocol is based on the XIP protocol (Ouyang, 2001) with some modifications. The main parts of this approach are:

- Component transaction is the unit of business at each service. The component transactions within a conversation create a conversational transaction;
- Services conversation C, $C=\{s_0, s_1, \ldots, s_{n-1}\}$ where s_0 is the root service starting the conversation, and $s_1, \ldots, s_{n-1}$ are the participating children of services;
- Conversational transaction T, $T=\{t_0, t_1, \ldots, t_{n-1}\}$ where t_0 is the root transaction starting the conversational transaction, and $t_1, \ldots, t_{n-1}$ are component transactions corresponding to a service $s_1, \ldots, s_{n-1}$;
- d_{ij} represents an interaction from the service t_i to the service t_j. A service interacts with other exchanging messages;
- Each component transaction has a function denoted $f(d_{ij})$ that performs the business logic, and $j(ti)$ that performs the compensation to cancel the committed operations for ti;
- Each $t_0, t_1,\ldots,t_{n-1}$ has two attributes: a deadline and a status. The deadline defines when the committed transaction cannot be cancelled. The status defines t's life-cycle (i.e. created, prepared, running, waiting for other events, compensating, hard-committed, local-committed, global-committed, aborted and cancelled).

A conversational transaction is organized in a spanning tree formed by component transactions. In this tree, a commit or cancellation is propagated using the correlators. A correlator is a handle to a node that has information about its parent and their immediate children. A correlator is defined in XML schema as following:

```
<complexType  name = "CorrelatorType">
  <element name = "ParentHandle"       type= "HandleType"
           minOccurs= "0"    maxOccurs= "1"/>
  <element name = "TranHandler"  type = "HandleType"/>
  <element name = "ChildrenHandles"  type= "HandleType"
           minOccurs = "0" maxOccurs = "unbounded"/>
</complexType>
<complexType  name = "HandleType">
  <element      name = "URL"        type = "string"/>
  <element      name = "ID"         type = "decimal"/>
  <element      name = "Status"     type = "string"/>
  <element      name = "Time"       type = "string"/>
  <element      name = "IDEngine"   type = "decimal"/>
  <element      name = "Deadline"   type="string"/ >
</complexType>
```

A correlator has information about its parent and their immediate children represented by HandleType. This HandleType has information about endpoint (i.e. URL), transaction ID, transaction status, time statistics and workflow engine identification.

The XIP protocol consists of two parts. The first part has the specification of root transaction that starts the conversation. The second part specifies the behavior of a component transaction. In our protocol, we follow the same perspective that is implemented by the Transaction Assistant in the Workflow Engine. The root transaction coordinator (RTC) implements the root transaction, and the component transactions coordinator (CTC) implements the component transaction behavior.

The RTC has some events that are described as:

- The start event – the root service sends a start event to the RTC that starts the root transaction;
- The handshaking event – when a component transaction is started, it sends a handshaking event to its parent, in order to establish a connection;
- The response event – the CTC sends this event to RTC, when its suitable component transaction ends;
- The timeout event – each timeout event results in a checking in the CTC, using a ping message. If there is an error, the suitable service is informed about the failure;
- The compensation event – this event is sent by a CTC to its parent. If this parent does not have another immediate child executing a compensation event, it sends back to the requester CTC a confirmation in order to call the function $\varphi(t_i)$;
- The end event – this event is generated by the root service that commits or not the conversational transaction. The RTC sends to each immediate child a commit request and waits for a global-committed event;
- The cancel event – this event is generated by the root service that cancels the conversational transaction.

The CTC has dual events to RTC events (i.e. start, handshaking, response, timeout, compensation, end, and cancel event) and more the following events:

- The global commit event – if a local transaction component receives a global commit from their children, then it sends a global commit to its parent/root;
- The ping event – a ping event is generated to test a transaction component. If a component transaction receives a ping, then it checks if its children are in a good state by sending a ping. If all the children are in a coherent state, then a confirmation of execution is sent to its parent, or else an error message is created.

Our protocol differs from other protocols (e.g. optimistic commit protocols (Ouyang, 2001), transactional conversation, Saga (Dayal, 2001), Activity-Transaction (Dayal, 2001)) in terms of complexity and implementation to provide process design, rollback for compensation, and forward execution process to guarantee the success of the global transaction in presence of faults in sub transactions.

5 WORKFLOW ENVIRONMENT FOR WEB SERVICE

Workflow Environment for Web Service (WEWS) is an intermediary infrastructure among service providers and service requesters. The Workflow technology, WSFL and a transaction compensation model are used in order to enable the development and the life-cycle control of e-business process. The WEWS prototype is being developed in Java, and then it will be able to operate in all machines that have a JVM.

5.1 WEWS Architecture

Figure 2 shows the WEWS architecture. The Server Infrastructure is a tool for service composition as business process, a repository of described services based on UDDI, a scalable workflow engine for flow process control, a tool of specialized services and a service-tier to enable CORBA objects on the Web.

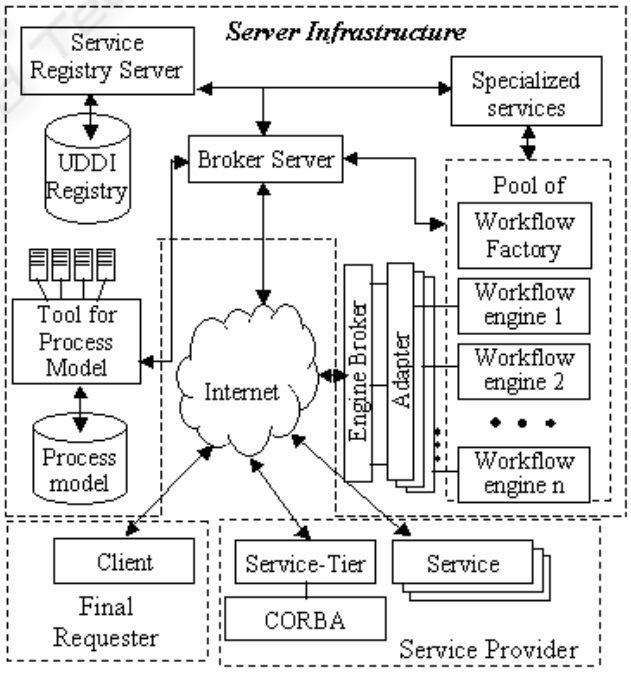

Figure 2: WEWS Architecture

Server Infrastructure is composed of:

- Broker Server – it receives all the requests to find, to publish and to monitor a service;
- Service Registry Server – it allows publication and search of a service in UDDI Registry;
- Process Model Tool – it allows service composition, in other words, it helps to build a business process description using the modified meta-model of WSFL;
- Specialized Services – they include some services as login, payment and authentication;
- Pool of Workflow – it is a Workflow Factory and a set of Workflow Engines that controls all services. The Workflow Factory is responsible for creating the workflow engines. The workflow engines execute a process based on a process model. The pool can be formed for one or more machines, then it can provide scalability in the server infrastructure;
- Adapter – it is formed of a service facility and a wrapper. It is the tier that makes the link between the workflow engines and the engine broker;
- Engine Broker – the pool of workflow can have many machines, but the access for an intranet must be limited for proxies, in order to maintain security notions. Then all workflow engines use an "Engine Broker" as proxy to communicate with the Service Adapters and Service Interfaces that are not in the server infrastructure;
- Service-Tier – it is a layer to adapt CORBA objects as Web Service.

5.2 WEWS Implementation

The workflow engine is shown in figure 3. It is composed of:

- Scheduler – it schedules and controls the execution of tasks according to a defined priority list, an execution time, availability, performance, and resources;
- Query – this functional block allows administration tools to interact with a workflow engine to acquire information about the state of process execution, e.g. individual performance, dependencies and exceptions;
- Active Manager – each task in active state is controlled and monitored by this manager;
- Trigger Manager – it supervises tasks in a waiting state (task waiting for an event) and notifies the scheduler when all resources are available for execution of a suitable task;
- Transaction Assistant – the execution of a service composition can follow a transaction process. In this case, WEWS includes a transaction assistant into the workflow engine to support compensation transaction. It complements the monitor present in the stack of specialized functions;
- DB Layer – this is a database layer responsible to control the information storage, search and recovery. It is composed by three data banks: Instance, History and Profile. When the Server Broker creates a workflow engine to execute a process, it delivers a copy of process model to DB Layer that will create an instance of this process and put it in Instance repository. If there is a fault, the instance repository is used to recover a process instance. An instance is a process ready for execution. The history repository stores all happened events of an already executed instance and it can be utilized for performance analysis or processes debugging. The profile has all information about configuration, participants and necessary resources;
- Handler – the active manager delegates activities to be executed by a handler. It is responsible for binding the requests with the suitable service provider (i.e. service adapter) and controlling the service execution. If a resource is not available for an activity, the handler informs the Active Manager that puts it in a waiting state. After that, the active manager contacts the trigger manager that monitors the resource, performs the allocation of the requested resource and notifies the scheduler, when the resource is available.

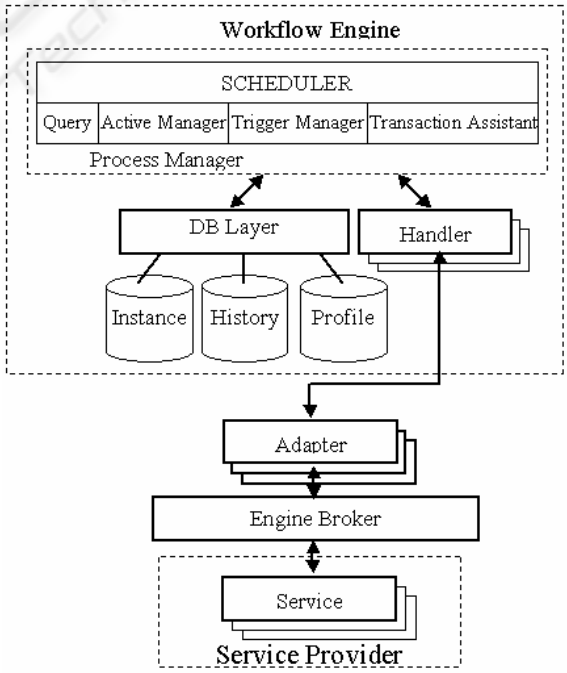

Figure 3: Workflow Engine

The Adapter is presented in figure 4. It is formed of Service Facility and Wrapper. Service Facility

provides support for service monitoring, interaction with its suitable workflow engine, security requirements, negotiation and payment. Wrapper controls a service directly, following the instructions received from the workflow engine. It helps the engine with exception detection and notification support. For the wrapper, a service can be a business application or a legacy application.

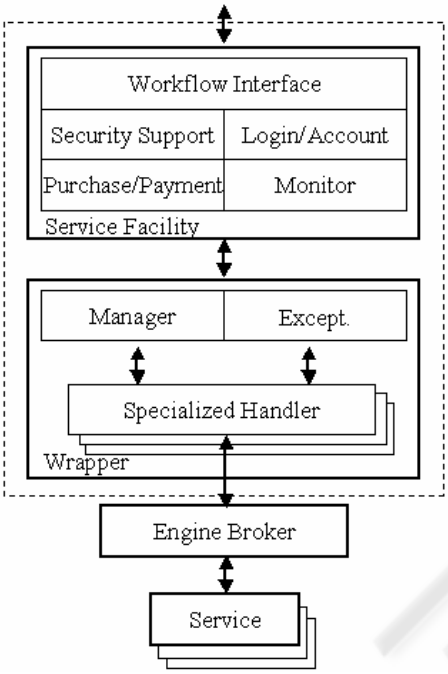

Figure 4: Adapter

The Service Facility is structured as following:

- Workflow Interface – it enables a service to participate in a workflow and allows the workflow engine to inquire about a service execution;
- Security Support – it complements the authentication and authorization management that are made for the stack of specialized functions. In Web Services, a high granularity of protection is desirable. Because of this, in our framework, each service has a suitable set of entities and a list of restrictions to provide security;
- Login/Account – it controls all accounts and limits the access to a registered service. Moreover, it is completed by the specialized stack. Furthermore, each service provider must limit the use of its service;
- Purchase/Payment – it defines and stores the constraints to sell a service, negotiate it and enable its payment. It has access to functionalities in the stack of specialized functions (Purchase/Payment) in order to realize the sale of a service and receive the benefits of this business;
- Monitor – this is a tool to facilitate the load monitoring in WEWS;

The Wrapper is structured in:

- Manager – it realizes a local schedule of service and provides a mechanism control for the Service Handler. The workflow engine executes a process, but the local manager controls each service. It allows the management of concurrent access and keeps the coherent execution of services;
- Exception Support – it detects the occurrence of exceptions in a service, gets information (e.g. messages about errors and faults) and notifies the workflow engine;
- Specialized Handler – each service has a specialized handler to accomplish its direct control and hide all the complexities of WEWS.

6 RELATED WORKS

In the next subsections, we present some approaches that have been proposed as a solution to support the development of Web Services in small, medium and larger virtual enterprises. Moreover, we will state a comparison with our proposal (i.e. WEWS) and these approaches, showing similarities and differences.

The Web Service Enablement Framework (Akkiraju, 2001) is a software layer between service providers and service requesters that facilitates several steps of service life-cycle. It allows business partners to discover, select and automatically invoke services. It provides authentication, logging, monitoring and data protection. In other words, it assists the management of dynamic e-business. This framework has been developed using IBM's WebSphere Application Service and Web Service Toolkit (Akkiraju, 2001).

In comparison with the Web Service Enablement Framework, WEWS aims to support the business process composition, which is based in Workflow Technology and has a stack of specialized functions.

In (Helal, 2002), an infrastructure is presented to manage Web Services. It uses services as participants in an Internet-wide workflow engine. It is composed by: BizBuilder, Sangam and Dynamic Workflow Server. This framework allows several steps of service life-cycle: publish, discover and binding. Hence, it supports service composition, negotiation and contracts, process modeling and legacy systems integration.

This framework (Helal, 2002) and WEWS allows the services composition based on Workflow

Technology. They have a coherent and well-defined infrastructure to publish, discover, bind and compose services.

However, in contrast with all presented frameworks in this paper, WEWS introduces an extension of WSFL as workflow language, a compensation transaction model and CORBA objects wrapped as Web Service.

7 CONCLUSION

In this paper, we have discussed the features of CORBA and Web Services. On one hand, we have shown that CORBA was not yet completely explored; and the Web protocols have not evolved enough to support CORBA. On the other hand, Web Services use the protocols of the Web, thus it is more adapted for the Web than CORBA. The Web object was not a very successful solution to integrate CORBA on the Web, but with the sprouting of Web Services new opportunities can be envisaged. We propose the externalization of CORBA objects inside an Intranet as Web Services.

Web Services is not a complete solution to create complex systems. We have proposed an integration of Web Services and Workflow to compose services in order to create real systems. Moreover, we have suggested an extension to WSFL and a transaction protocol.

An architecture (WEWS) that uses our approach for Web Services composition and transaction models was depicted and compared with other propositions in the same context. To validate this architecture, several tests are envisaged as implementation of a virtual enterprise in WEWS, performance monitoring and analysis.

REFERENCES

Akkiraju, R., et al., (2001). A Framework for Facilitating Dynamic e-Business Via Web Services, *In OOPSLA 2001, Workshop on Object-Oriented Web Services*, Tampa Bay, USA.

Dayal, U., Hsu, M. and Ladin, R., (2001). Business Process Coordination: State of the Art, Trends, and Open Issues, *In VLDB 2001, 27th International Conference on Very Large Data Base*, Roma, Italy.

Gokhale, A., and et al., (2002). Reinventing the Wheel? CORBA vs. Web Services. *In WWW2002, the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, USA.

Harkey, D., and Orfali, R., (1998). *Client/Server Programming with Java and CORBA*, 2nd Edition, John Wiley & Sons.

Helal, S., et al., (2002). The Internet Enterprise, *In SAINT'02, Second IEEE/IPSJ Symposium on Applications and the Internet*, Nara, Japan.

Leymann, F., (2001). Web Services Flow Language - (WSFL 1.0)".

Northrop, L., et al., (1997). Distributed Object Technology with CORBA and Java: Key Concepts and Implications, Technical Report CMU/SEI-97-TR-004, Printed in the USA.

OMG, (2000). Trading Object Service Specification, Version 1.0.

OMG, (2001). The Common Object Request Broker: Architecture and Specification, Revision 2.4.2.

Ouyang, J., and et al., (2001). An Approach to Optimistic Commit and Transparent Compensation for E-Service Transactions, *In PDCS 2001, 14th International Conference on Parallel and Distributed Computing Systems*, Dallas TX, USA.

Tsur, S., (2001). Are Web services the next revolution in e-commerce?, *In VLDB 2001, 27th International Conference on Very Large Data Bases*, Roma, Italy.

UDDI Project, (2000). UDDI Technical White Paper.

van der Aalst, W.M.P., (2003). Don't go with the flow: Web Services composition standards exposed. Web Services – Been there done that? *Trends & Controversies, issue of IEEE Intelligent Systems*, IEEE Press.

W3C, (2001a). Simple Object Access Protocol (SOAP) 1.1, W3C Submission.

W3C, (2001b). Web Services Description Language.

WfMC, (2002a). *The Workflow Handbook 2002*, Edited by Layna Fischer, 428 pages.

WfMC, (2002b). Workflow Process Definition Interface – XML Process Definition Language, Specification (Draft), version 0.04.

Zhang, L., Yadav, P., Chang, H., et al., (2001). ELPIF: An E-Logistics Processes Integration Framework Based on Web Services, *In OOPSLA 2001, Workshop on Object-Oriented Web Services*, Tampa Bay, USA.