Technical University of Vienna
Information Systems Institute
Distributed Systems Group

ICSoC 2005 Demonstration Session

Proceedings

Schahram Dustdar (ed.)
dustdar@infosys.tuwien.ac.at

TUV-1841-2005-34 December 7, 2005

This Technical Report contains the demonstration papers for the International Conference on Service Oriented Computing - ICSOC 2005, 13 - 15 December, 2005, Amsterdam. We received in total 20 demo submissions. We selected seven to be presented in the demonstration session, which are contained in this Technical Report and five demonstrations were selected to be included in the Springer LNCS conference proceedings (B. Benatallah, F. Casati, and P. Traverso (Eds.) ICSOC 2005, LNCS 3826, pp. 478 - 507). The presented systems demonstrate the blend of scientific research and practical value Service-oriented Computing (SoC) delivers. Furthermore, it increasingly becomes evident that SoC research can benefit from experiences gained from design, development, and large-scale deployment of service-oriented systems.

Keywords: ICSoC 2005, demo, proceedings

Argentinierstr. 8/184-1
A-1040 Vienna, Austria
phone: +43 1 58801-18402
fax: +43 1 58801-18491
URL: http://www.infosys.tuwien.ac.at/

Web Services based interorganisational architectural framework for B2B marketplaces

Dr. Jai Ganesh
Akash Saurav Das
Abhishek Malay Chatterjee
Ambar Verma
Mohit Chawla
Akhil Marwah
Infosys Technologies Ltd.
{jai_ganesh01; akash_das; Abhishek_Chatterjee; Ambar_Verma; Mohit_Chawla; Akhil_Marwah}@infosys.com

1. Introduction

This demo is a solution based on Web services and Service Oriented Architecture involving interorganizational information systems in the context of B2B marketplaces. B2B marketplaces involve disparate interorganisational interconnections with multiple customers as well as partners. As the number of customers and partners increases, managing the dynamics of interorganisational commerce becomes complex. The key requirements of the IT architecture supporting a B2B marketplace are:

- Ability to integrate with buyer and seller systems
- Ability to integrate smoothly with other partners (e.g. credit card firms, courier companies etc.)
- Ability to configure/reconfigure services
- Ability to match the information of the participants in a transaction

Given the nature of the scenario and the number of stakeholders involved (e.g. the B2B marketplace, sellers, buyers, credit card companies, courier companies etc.), a Web services based service-oriented architecture would be the ideal solution to the problem. Due to the loosely coupled-nature of Web services, the B2B marketplace does not need to have hardwired connections with the buyers, sellers, credit card companies or logistics service providers. This allows the B2B marketplace to have access to more services, offering more options to its customers. This would not only address the current needs but would also address the future needs when the B2B marketplace may be needed to make fast business connections with partners without going through the conventional pattern of making large scale changes to the system. Web services would enable the B2B marketplace to isolate the business logic from integration. Most conventional integration solutions embed part of the business logic in the integration layer thereby requiring considerable efforts in making modifications. Web services address the key requirements of the scenario listed above. Based on open standards like XML and SOAP, they define a means by which the services of the B2B marketplaces and their partners can be published, discovered and invoked.

1.1 Scope

In this demo, we demonstrate the workings of a private marketplace, which interacts with buyers and sellers as well as different third party service providers like payment services, authentication services, and logistic services. The following lists the high level overview of the B2B marketplace:

1. The implementation is purely for private marketplace for business to business transactions (includes auction engine, which conducts forward as well as reverse auctions, e-Catalog, which allows users to search for items on sale)
2. The Web services such as Authentication service (third party authentication mechanisms come into play to send to authenticate buyers and sellers as well as rating their previous transactions), Payment service (third party payment mechanisms come into play to send, execute and settle orders that have been agreed to in the marketplace), Logistics service (includes services such as logistics services offered in association with logistics service providers aimed to facilitate easy movement of equipment and other assets transacted through the marketplace).
3. We demonstrate a Web services based interoperability between the systems of the B2B marketplace with those of the payment service providers
4. The third party services and are implemented in different technologies to showcase interoperability.

2. Use Cases

The Actor Catalog as well as the Use cases are given below

2.1 Actor Catalog

No.	Actor	Activities performed
1.	Buyer	• Enrolls in to the system • Enquires his/her account history • Modify personal data • Does buy operations • Receives reports from system
2.	Supplier	• Enrolls in to the system • Enquires his/her account history • Modify personal data • Does sell operations • Receives reports from system
3.	Disposer	• Enrolls in to the system • Enquires his/her account history • Modify personal data • Does sell operations • Receives reports from system
4.	Administrator	• Accounts maintenance • User Management • Receives reports • Performs backup

5.	Registry	• Lists buyer and seller services • Modifies listings • Allows for search
6.	3rd Party service provider (Banks)	• Enrolls in to the system • Offers payment services
7.	3rd Party service provider (Logistics)	• Enrolls in to the system • Offers logistics services
8.	3rd Party service provider (Verification and Validation)	• Enrolls in to the system • Offers verification and validation services
9.	3rd Party service provider (Content)	• Enrolls in to the system • Offers content services
10.	Forward auction system	• Allows the Seller to create an online auction. • Buyers can then bid on the item in the auction • Buyers can search for items • Seller can close the auction
11.	Reverse auction system	• Allows the Buyer to create an online auction. • Sellers can then bid on the item in the auction • Sellers can search for items • Buyer can close the auction
12.	Item maintenance	• Adds items • Removes items • Modifies items

2.2 Use Cases

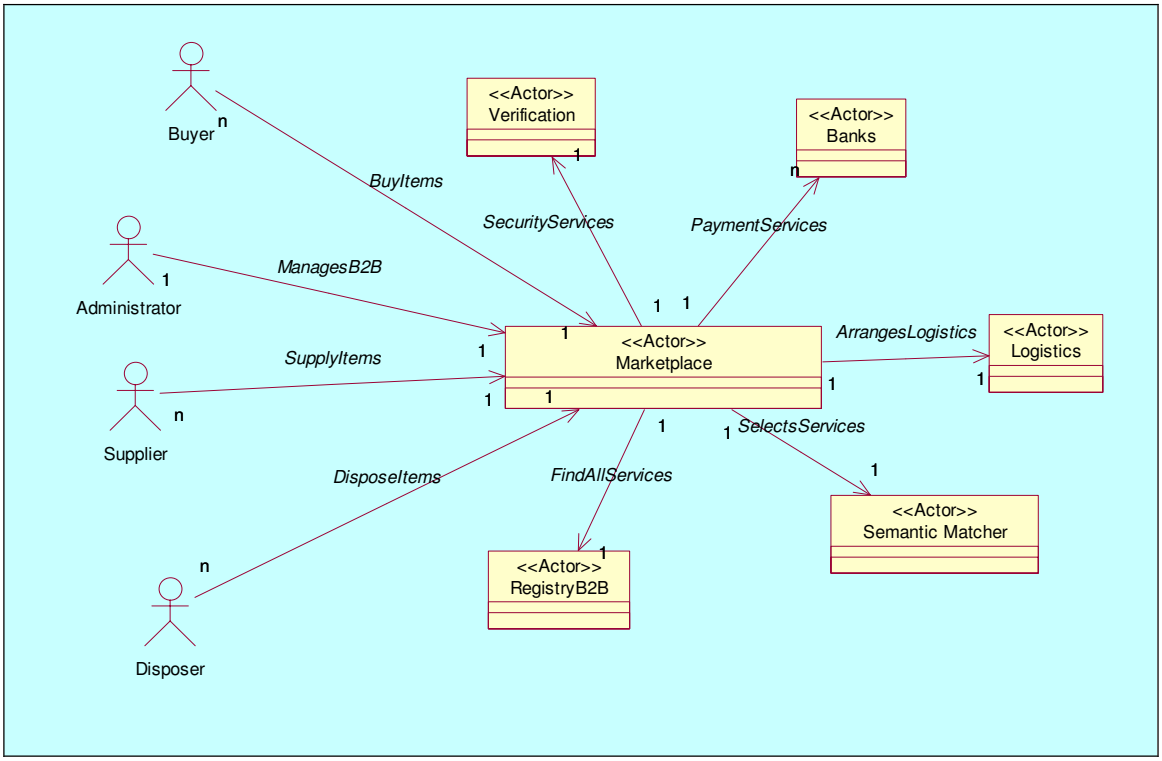
Given below is the list of Use cases being demonstrated in the demo:

1. Use Case: Create Forward Auction
2. Use Case: Create Reverse Auction
3. Use Case: Close Forward Auction
4. Use Case: Close Reverse Auction
5. Use Case: Buyer search for auction Item
6. Use Case: Seller search for prospective buyers
7. Use Case: Sell item through a forward auction
8. Use Case: Buy item through a reverse auction
9. Use Case: Update Participant Information
10. Use Case: Rollback Transactions

2.3 Use-Case Realization

Figure 1 given below show how the software actually works by giving a few selected use-case realizations.

Figure 1: Sample Use case realisations



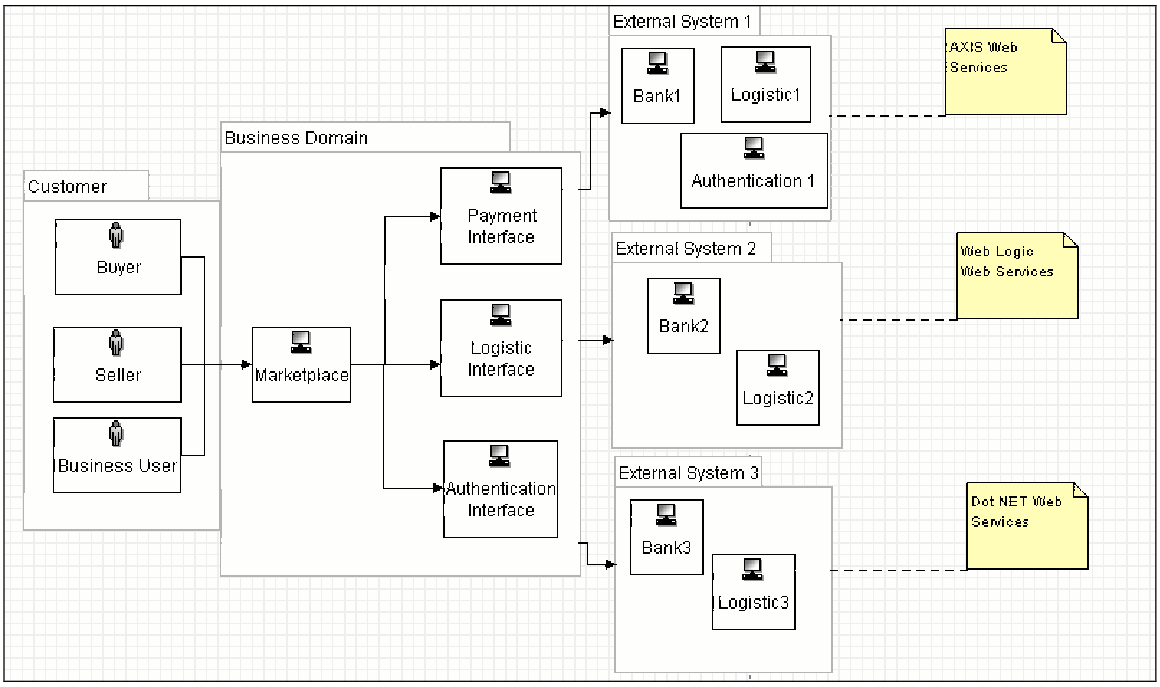
3. Architectural Goals and Constraints

The objective of this PoC is to showcase the usability of web services technology in B2B Marketplace scenarios. The key feature which had an impact on architecture is the interoperability across multiple technologies. Service-oriented development is used throughout this PoC and hence this PoC has Reusable Assets (components, web services and design patterns).

3.1 Logical View of the Architecture

In *Figure 2*, we describe the functional decomposition of the entire application based on a logical ordering of the application's requirements. The aspects of the application with similar functionality are aggregated into a subsystem and then subsystems organized to depict the dependencies between subsystems. We also depict the decomposition into the key classes and interfaces. The notation used for the Logical View is UML2.0.

Figure 2: Logical view of the Architecture



Figures 3 & 4 depict the Package hierarchy as well as the Sub-system overviews.

Figure 3: Package Hierarchy

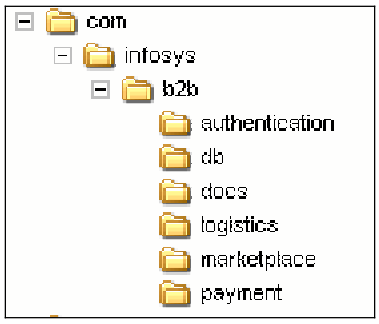
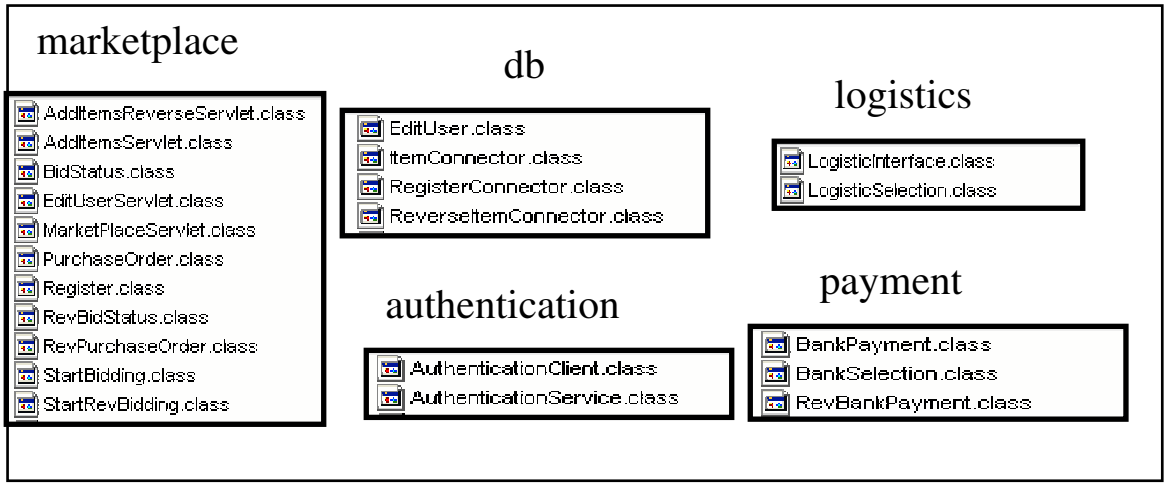


Figure 4: Marketplace Subsystem Overview


marketplace
AddItemsReverseServlet.class
AddItemsServlet.class
BidStatus.class
EditUserServlet.class
MarketPlaceServlet.class
PurchaseOrder.class
Register.class
RevBidStatus.class
RevPurchaseOrder.class
StartBidding.class
StartRevBidding.class
db
EditUser.class
ItemConnector.class
RegisterConnector.class
ReverseItemConnector.class
logistics
LogisticInterface.class
LogisticSelection.class
authentication
AuthenticationClient.class
AuthenticationService.class
payment
BankPayment.class
BankSelection.class
RevBankPayment.class

Executable Choreography Framework

Thomas Cottenier, Tzilla Elrad
Illinois Institute of Technology
{cotttho, elrad}@iit.edu

Abstract

The Executable Choreography Framework (ECF) introduces a language to specify executable choreographies and a platform extension to enable the deployment of executable choreographies on application servers.
The Executable Choreography Language (ECL) is a XML-based language to define refinements on the default control flow of service invocation and execution. ECL rules specify a set of actions to be performed when a message of interest is intercepted in the container. The ECF platform extension is a container level component that intercepts incoming SOAP messages before they are dispatched to a service provider. Likewise, outgoing messages are intercepted after serialization, before they flow out of the container.
The demonstration will introduce the languages constructs of the ECL, and illustrate the capabilities of the platform extension through an adaptive choreography application, a mobile agent application, and a distributed aspect application.
First, the ECF enables choreographies to be deployed in a competitive time frame, and dynamically adapted to meet changing requirements. Second, the ECF enables the implementation of mobile agents in Web Service environments, because ECL rules are platform independent and deployable on demand. Finally, the ECF enables distributed Aspect-Oriented Programming (AOP). A distributed aspect captures events that occur at the interface of remote services, and has the ability to inject behavior at those locations. In Web Service environments, distributed AOP is ideally suited to implement middleware-level services such as those provided by the Composite Application Framework (WS-CAF) specification.
The ECF is available for download at http://ww.iit.edu/~concur/ecf

1. Introduction

A service Choreography specification describes the global message exchanges between the services that participate in a composite service application. On the other hand, a service Orchestration specification defines the control flow of a local business process and the message it exchanges with its partners and the local services.
A composite service application that does not have a single center of control cannot be directly implemented with an orchestration language such as BPEL [12]. Several orchestration specifications need to be defined. The role of a Choreography specification is to coordinate the actions of the orchestration engines. Typically, a global contract is agreed on between the partners of the service composition. The contract is specified in a language such as the Choreography Description Language (WS-CDL) [11]. Second, BPEL orchestration specifications can be generated from the CDL contract, and executed.
Figures A,1 and A.2 of the annex illustrate a composite web service application for finding the directions between two locations (this example is derived from an application described in [1]). Fig. A.1 describes 4 business processes that are executed on BPEL engines of different domains. A1 and A2 are the endpoints of AddressBook services that return the address of a location based on a location name; A3 is the endpoint of a RoadMap service that returns directions from one location to another, given their addresses.
In the example of Fig A.1., BPEL engines act as wrappers around the services exposed by the domain. Control flow logic and data mappings are executed before and after the local services are invoked. Each business process is exposed with a new composite service endpoint, BPEL0, BPEL1, BPEL2 and BPEL3. The identity of the services is thereby modified. While BPEL0 exposes a semantically meaningful composite service, the composite services exposed by BPEL1, BPEL2 and BPEL3 have no meaningful semantics, outside the context of the FindRoute composite activity. These endpoints are therefore very unlikely to be published and discovered.
Choreographies implemented with orchestration engines introduce a service identity problem. Services that participate in composite applications will be attributed several identities (URI's), each of them coupled to a very specific activity context.
The Executable Choreography Framework (ECF) [5][6][7] aims at providing a more flexible way to specify, implement and deploy distributed workflows.

2. Choreographies with the Executable Choreography Framework

2.1. Executable Choreography Language

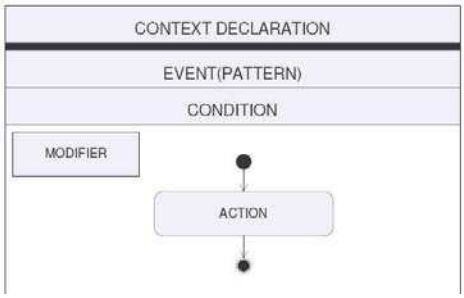


Fig. 1. ECL Rule

The ECF enables the default flow of control of a service invocation to be refined in a context-sensitive way. Refinements are applied non-invasively – without having to manually modify the workflow of the target service.

Given a choreography description, either provided as an activity diagram or a CDL specification, the ECF partitions the distributed workflow into ECL (Executable Choreography Language) rules. An ECL rule is similar to Event-Condition-Action (ECA) rule used in active databases [2]. It is composed of an activity context declaration, an event pattern, a set of conditions and an action:

1. CONTEXT: An Activity Context declaration uniquely identifies the rule. At runtime, the context encapsulates information that relates to a distributed activity. Activity contexts are propagated transparently from node to node, along the interactions of an activity.
2. EVENT: An Event expression defines when the rule should be applied. The ECF recognizes two types of events: sending a message and receiving a message. An event expression defines a pattern on the signature (PortType, Operation, Parameter types) of the messages to be intercepted.
3. CONDITION: The condition clauses identify the activities for which the refinement of the control flow should be applied. Interactions of an activity are discriminated based on their activity context information. Condition clauses take the form: CONTEXT(<Activity-Context-Identifier>)
4. MODIFIER: A modifier specifies when the rule behavior should be applied. The rule actions can execute either Before, Around (Instead) or After the events captured by the event pattern.
5. ACTION: The ECF takes control over the thread of execution of the service invocation or execution. The actions authorized by the ECF are:
   - Compound Action: Actions can be composed into compound actions, using the sequence, fork, join, decision and merge control flow operators.
   - Accept Request/Response Action: wait for a request or a response message from a remote service.
   - Send Request/Response Message Action: send a service request or response.
   - Invoke Service Action: invoke the functionality of a local service
   - Set Timer Action, Reset Timer Action and Accept Time Event Action: timing actions allow ECL rules to define timeouts and handle faults.
   - Data Mapping Action: The SOAP messages intercepted by the ECF can be transformed according to a XSL specification. SOAP messages can be aggregated and data consolidated.

2.2. Executable Choreography Platform Extension

ECF actions have unambiguous implementations in standard Application Servers. ECF-enabled platforms can interpret ECL rules, and deploy the corresponding control flow logic accordingly, in the language of the platform. The ECF platform extension implements 3 distinct functionalities: message interception, transparent activity context propagation and dynamic rule deployment.

2.2.1. Message Interception

Incoming SOAP messages are intercepted before they are dispatched to a service provider. Likewise, outgoing messages are intercepted after serialization, before they flow out of the container. When a message matches the event pattern of an ECL rule within its activity context, the ECF platform extension takes control over the thread of the service request or response, and injects the rule behavior, before, after or instead of the intercepted event.

2.2.2. Activity Context Propagation

The ECF provides transparent context propagation within a distributed activity. Activity contexts are piggybacked in the headers of the intercepted messages. The ECF platform extension ensures that contexts are propagated from node to node, along the interactions of a same distributed activity. Context propagation helps managing the life-cycle of distributed activities. The ECF concept of activity context is derived from

the Composite Application Framework (WS-CAF) specification [13]. WS-CAF is a standard to implement context-passing, coordination and transaction management in web-service based composite applications. As opposed the WS-CAF, the ECF propagates context transparently.

2.2.3. Dynamic Deployment

Given a choreography description, the ECF partitions the distributed workflow into ECL rules. These rules can be deployed on remote containers on-demand. ECF-enabled platforms expose a choreography deployment Web Service, whose endpoint is published into a choreography repository

ECF platform extensions for Axis [14] and the Globus toolkit [16] are available for download at http://ww.iit.edu/~concur/ecf. Their implementation is java based, and uses the Aspectwerkz [15] Aspect-Oriented framework to non-invasively integrate the extension with the target containers.

2.3. Executable Choreographies

Fig. 1.B. of the annex illustrates an ECF specification for the FindRoute activity. First, the client defines a context identifier for the FindRoute activity, 'client.FindRouteActivity'. At runtime, this context carries the callback endpoint of the client. The ECL rule deployed on the domain of A0 exposes the endpoint of the composite service. The rule reacts on reception of a 'FindRouteRequest' message, within the context of a 'client.FindRouteActivity' activity. On the domains of A1 and A2, the rules intercept the response messages of the AdressBook services, perform data mappings, and redirect the response to the RoadMap web service. The RoadMap activity aggregates those responses, invokes the local RoadMap service, and sends the reply back, directly to the client. The callback endpoint of the client is fetched from the activity context, which propagated from the client request, to the domains of A0, A1, A2 and A3.

ECF rules do not modify the identity of the services involved in the choreography. ECL rules expose new endpoints implicitly, through the activity context. By comparing Fig A.1 and Fig 2.A, we can tell that some of the complexity of the orchestration logic has migrated from the endpoint of the composite service on the domain of A0, to the domain of A3, where responses are aggregated and are routed back to the client. The ECF choreography implementation is more decentralized.

3. Mobile Agents with ECF

Mobile agents are straightforward to implement with ECL rules. ECL rules conform to a XML Schema and can be interpreted by ECF-enabled containers. Agent behavior can therefore be specified in a platform independent way, which is a fundamental requirement for implementing agents in web service environments.

A mobile agent typically performs a series of invocations of local services according to some control flow logic and aggregates the result data. Once it performed its local operations, it migrates to another host, along with its control flow logic and data.

In the ECF, an agent is composed of one or more ECL rules, and an itinerary description. A mobile ECL rule deploys itself on a target host by invoking the target Dynamic Deployment service. The data of the agent is propagated automatically through its activity context.

The ECF is the only platform the authors are aware of, that enables mobile agent on web services (other proposals [8][9][10] are platform dependent).

4. Distributed Aspect-Oriented Programming with ECF

The ECF provides the basic building blocks for a distributed Aspect-Oriented Programming platform for Web-Service environments. Aspect-Oriented Software Development [3][4] is a new software development paradigm that targets the encapsulation of crosscutting concerns. Crosscutting concerns are concerns that can not be cleanly encapsulated in the modularity units of the language, because they follow different composition rules. In the Web Service context, the implementations of many of the middleware-level services such as transaction management or security are tightly coupled to the implementation of specific applications. These concerns are hard to cleanly modularize into separate Web Services, because they affect composite applications at many locations of their workflow specification.

Within the ECF, an aspect is composed of ECL rules that intercept distributed Activities, as opposed to Send or Receive actions. An activity is spawned on one host, and may terminate on another host. An ECF choreography Aspect can inject behavior before, around or after an Activity. Distributed AOP is ideally

suited to implement Middleware-level services such as those provided by the Composite Application Framework (WS-CAF) specification.

5. Conclusions

The Executable Choreography Framework (ECF) introduces a language to specify executable choreographies and a platform extension to enable the deployment of Executable Choreographies on Application Servers. The Executable choreography language was presented, and the architecture of the ECF platform extension discussed. The demonstration will introduce the languages constructs of the ECL, and illustrate the capabilities of the framework through an adaptive choreography, a mobile agent application, and a distribute aspect application.

Acknowledgement

This work is partially supported by CISE NSF grant No. 0137743.

References

[1] Chafle, G., Chandra, S., Mann, V., Nanda, M. G.: Decentralized Orchestration of Composite Web Services. Proceedings of the Thirteenth International World Wide Web Conference, New York, NY, USA, ACM Press (2004)

[2] Norman W. Paton, Oscar Diaz, Active Database Systems, ACM Computing Surveys, New York, NY, USA, ACM Press (1999)

[3] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M., Irwin, J.:Aspect-oriented programming. Proceedings of the European Conference on Object-Oriented Programming, Springer-Verlag (1997)

[4] Filman, R., Friedman, D.: Aspect-oriented Programming is Quantification and Obliviousness. Workshop on Advanced Separation of Concerns, OOPSLA 2000 (2000)

[5] Cottenier T., Elrad T, Prunicki, A.: Contextual Aspect-Sensitive Services, formal demonstration presented at the 4th International conference on Aspect-Oriented Software Development (AOSD'05), Chicago, USA (2005)

[6] Cottenier, T., Elrad, T.: Dynamic and Decentralized Service Composition with Contextual Aspect-Sensitive Services, First International Conference on Web Information Systems and Technologies, Miami, USA (2005)

[7] Cottenier, T., Elrad, T.: Validation of Aspect-Oriented Adaptations to Components. Ninth International Workshop on Component-Oriented Programming as part of ECOOP'04, Oslo, Norway (2004)

[8] Zakaria Maamar, Quan Z. Sheng, and Boualem Benatallah. Interleaving web services composition and execution using software agents and delegation. In AAMAS'2003 Workshop on Web Services and Agent-based Engineering (2003)

[9] Amir Padovitz, Shonali Krishnaswamy, and Seng Wai Loke. Toward efficient and smart selection of web service. In AAMAS'2003 Workshop on Web Services and Agent-based Engineering (2003).

[10] Fuyuki Ishikawa, Nobukazu Yoshioka, Yasuyuki Tahara, and Shinichi Honiden, Toward Synthesis of Web Services and Mobile Agents, AAMAS'2004 Workshop on Web Services and Agent-based Engineering (2004)

[11] Web Services Choreography Description Language (WS-CDL) Version 1.0, W3C Working Draft 17, http://www.w3.org/TR/ws-cdl-10/ (2004)

[12] Business Process Execution Language for Web Services, BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems http://www-128.ibm.com/developerworks/library/ws-bpel

[13] Web Services Composite Application Framework (WS-CAF). Arjuna Technologies Ltd., Fujitsu Limited, IONA Technologies Ltd., Oracle Corporation, and Sun Microsystems, Inc, http://developers.sun.com/techtopics/webservices/wscaf/primer.pdf (2003)

[14] Apache, Axis homepage http://ws.apache.org/axis (2000)

[15] Aspectwerkz homepage http://aspectwerkz.codehaus.org/ (2004)

[16] The Globus Toolkit homepage, http://www.globus.org/toolkit/ (2002)

Annex

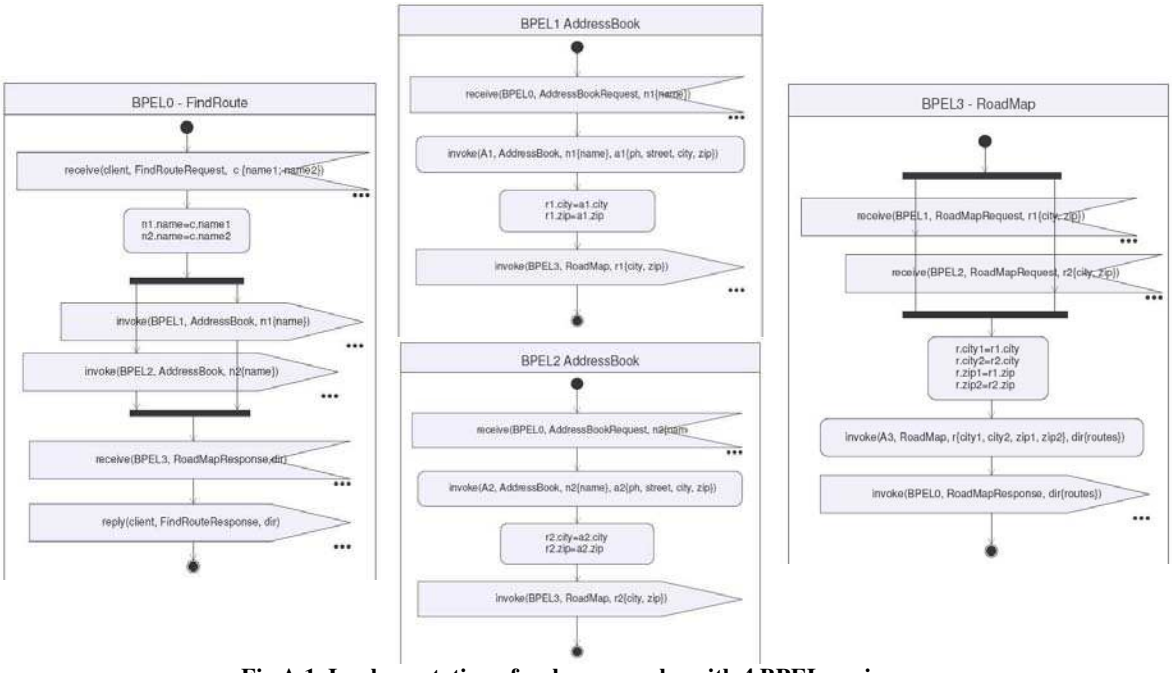


Fig.A.1. Implementation of a choreography with 4 BPEL engines

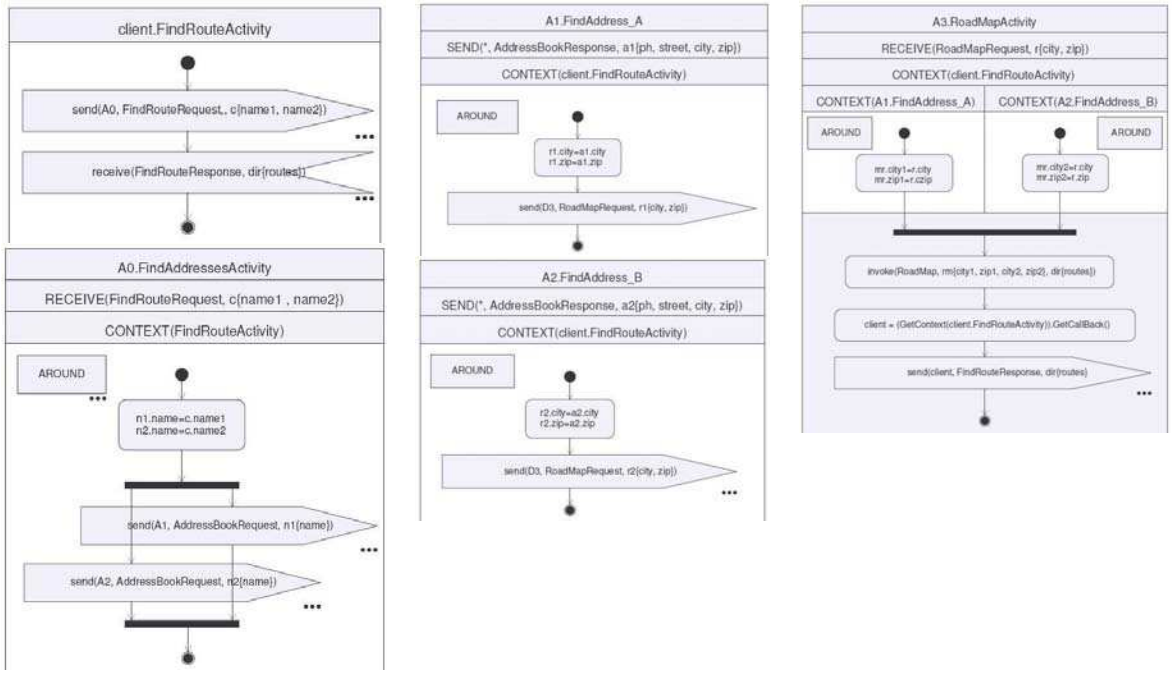


Fig.A.2. Implementation of a choreography with the ECF

Demonstrating FLAVOUR: Friendly Location-aware conference Assistant with priVacy Observant architectURe

Kavitha Muthukrishnan,Nirvana Meratnia,Georgi Koprinkov,Maria Lijding and Paul Havinga

University of Twente, Fac. of Computer Science,
P.O.Box 217, 7500AE Enschede, The Netherlands
{k.muthukrishnan,n.meratnia,g.t.koprinkov,m.e.m.lijding,p.j.m.havinga}@ewi.utwente.nl

Abstract. In this paper, we describe an implementation of FLAVOUR (Friendly Location-aware conference Assistant with priVacy Observant architectURe), in which people/infrastructure resources act as individual service providers offering their location as a service. By subscribing to this service, in the one hand, conference participants can be aware of each others whereabouts as well as being able to chat. On the other hand, conference organizers can notify interested attendants about special events such as cancellation of a track or change in the presentation rooms. The presented architecture uses existing WLAN infrastructure for cost efficiency, and uniquely incorporates the location information as a service into Jini service discovery platform. Location itself is determined with high accuracy by using a calibration free technique.

1 Motivation

We all have occasionally experienced being alone in a foreign territory. Naturally, it had come to our mind it would have been nice if we were accompanied by a trustworthy native person who knows a great deal about the area, places worth visiting, and how to find our way and our interest points, etc. As unrealistic as it may sound, that is exactly what this research aims at, i.e., building a mobile guide to (temporarily) be your best friend when you are attending a conference.

Equipped with 650 individual wireless network access points, with each point having a range of about 100 meters, in June 2003, University of Twente (UT) announced the launch of its wireless campus. In short, spread over 140-hectare campus, UT offers its staff, students, as well as its visitors, i.e., anyone with a desktop, laptop, handheld or wireless fidelity (Wi-Fi) devices to wirelessly access the university's network and the internet from everywhere on the campus [5]. Availability of such an infrastructure and the fact that SVGOpen 2005 conference was scheduled to be held at UT, were two strong driving forces towards building a user-friendly conference assistant, in which *location* proves to be one of the key components.

There are numerous location-aware applications, being employed in various environments and used by diverse user groups, which are developed using WLAN infrastructure. There are several great challenges that these applications face. Four of them are as follows:

- **Localization:** Despite of offering many advantages such as providing economical solution, higher coverage, and scalability, Wi-Fi based localization techniques in general suffer from high calibration effort. The calibration needs enormous amount of manual labor and should be performed repeatedly. As a rule of thumb, there is a trade-off between the amount of effort put on reducing the calibration and the accuracy obtained. For instance, Ekahau positioning system [6] offers an accuracy of about $1\ m$, while it requires quite a lot of calibration effort. On the other hand Place lab [7] does not involve much calibration, and the reported accuracy ranges from 13 to $20\ m$.
- **Heterogenity:** End-users may utilize various personal devices on which different platforms run. Hence supporting interoperability between various devices and platforms is mandatory and by no means is an easy task.
- **Privacy:** It is very easy to create and implement big-brother scenarios that track users movements and allow to deduce patterns of behavior. Careful definition of privacy policies and a proper architecture design can reduce or even eliminate this risk.
- **Infrastructure**: Since applications are often built on top of already existing infrastructures, requirements of the applications may not always be met. In other words, to be (better) operational, applications may require add-on to the existing infrastructure.

To address the above challenges, we developed a research prototype called FLAVOUR (Friendly Location-aware conference Assistant with priVacy Observant architectURe).

2 Features of FLAVOUR

The important features of FLAVOUR are:

- **Offering location as a service:** In FLAVOUR each conference participant/infrastructure resource acts as an individual service provider. This means that, location of each individual is published as a service to which interested participants may subscribe.
- **Providing multiple services:** Both pull and push services are provided by FLAVOUR. Examples of the former include *(i)* finding fellow attendants, and *(ii)* locating and using resources available in the infrastructure such as printers, copiers, coffee machines etc. Being notified about important events by conference organizers *(iii)*, and communication with other contacts, i.e., colleagues, friends etc.*(iv)* are examples of the latter.
- **Accuracy:** Compared to existing WLAN-based localization techniques, which rely on huge calibration phase, localization method used in FLAVOUR offers an accuracy of 6 m in average with zero calibration effort.
- **Highly dynamic interface:** Since FLAVOUR utilizes Scalable Vector Graphics(SVG), panning, zooming and other functionalities at the user interface are performed very fast. Also due to having SVG viewer on the client side, rendering of the SVG map can be done quickly .
- **Privacy:** FLAVOUR does not have tracking functionality. Users can be aware of each others location only if they have proper privileges. FLAVOUR gives users the freedom to choose whom and for how long can access their location information.
- **Availability of services in off-line mode:** Due to the limitation in battery, user devices are not always on. An important feature of FLAVOUR is to be able to provide its services even if the user device is switched off. In this case people can be aware of the location that the off-line user was last seen and the user himself can receive off-line messages once he is online.
- **Platform support:** At the moment, FLAVOUR runs on both Windows and Linux platform.

3 Service-oriented Architecture

Figure 1 represents 3- tier FLAVOUR architecture, which consists of client side, surrogate host and server side. Both surrogate host and server side reside on Jini infrastructure. The following building blocks are the key components in the implementation of FLAVOUR:

- **Device Location Service (DLS):** It consists of spotting functionality, which scans for near-by access points whenever the client requests its location.
- **Device Agent Service (DAS):** The copy of DLS instantiated in the surrogate host together with the mapping service form the DAS. Mapping service itself is responsible for overlaying location coordinates on the footprint map.
- **Reggie:** It registers all the available services so that they can be used through Jini infrastructure.
- **User Device Manager Service (UDMS):** When the DAS needs to know the location of other buddies, a request is sent from the client to UDMS which is residing on the server side. To offer/take back its own location, DAS registers/de-registers its location information to/from this service.
- **Access Point Information Service (APIS):** It is a database providing the location of the access points in the area of interests.
- **Location Manager Service (LMS):** The estimation of the actual location is accomplished by LMS. Although LMS resides on the server side, it does not keep any record of the estimated location. In this way user privacy is maintained. Only in a special case, i.e., when the user is off-line, a record of the location where the user was last seen is kept.
- **Buddy List Service (BLS):** It contains the list of buddies subscribed to a location service provided by a particular user. In addition it is responsible for re-directing all the authorization requests as well as storing messages when the user is off-line.
- **User Announcement Service (UAS):** It is the Jini representation of a client in the surrogate host, which only exists when the device is on. UAS basically facilitates communication between BLS and the client.

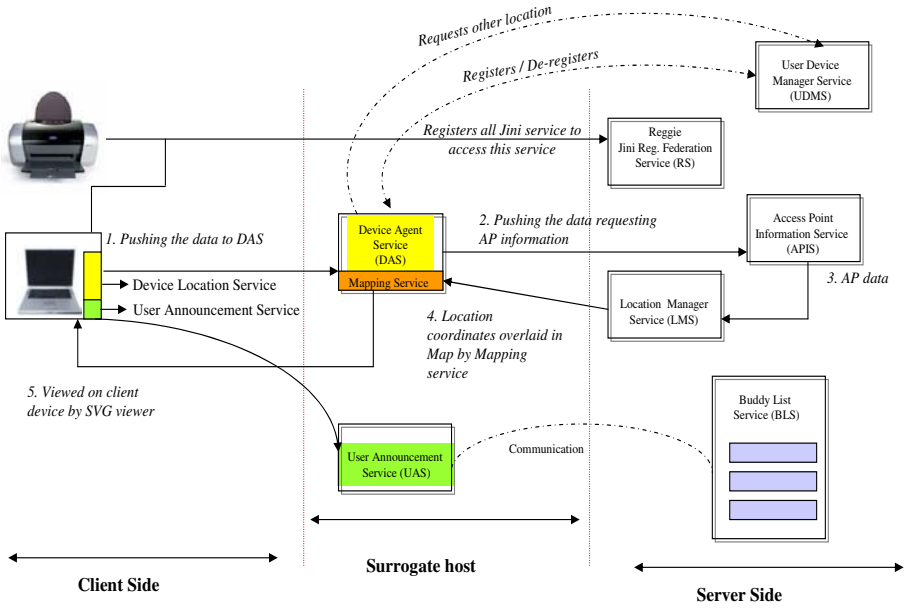


Fig. 1. Architecture of FLAVOUR

4 Demonstration

Our demonstration consists of three major parts. First, we present locating individual conference participants as well as available resources in the infrastructure. Second, we show how individual users can be aware of each others location. Last, we illustrate the massaging capability of the system both from the organizers point of view and participants. Various snapshots of FLAVOUR can be seen in Figure 2.

5 Conclusion

In this paper, we presented FLAVOUR, a privacy-sensitive, location-aware service architecture for conference environment. FLAVOUR uniquely incorporates location information into the Jini service discovery platform to provide conference participants with service sharing based on their location. It also facilitates the availability of location information even when the user is off-line. The location is determined with high accuracy by using a *calibration-free* localization technique. Another advantage of the presented architecture is *cost-efficiency* because it uses existing WLAN infrastructure.

On-going work includes enhancing the accuracy of localization technique. Not to end up with a big-brother scenario, we plan to incorporate more privacy policies in the future. Last but not the least, we also aim at extending FLAVOUR to encompass campus-wide services.

6 Acknowledgements

This work is part of the *Smart Surroundings* project, funded by the Ministry of Economic Affairs of the Netherlands under the contract no. 03060.

Authors would like to thank Department of Information Technology, Library & Education (ITBE) of University of Twente and Drs. Barend Köbben from ITC, respectively, for manual mapping of the access points and providing enormous help in respect to geo-database.

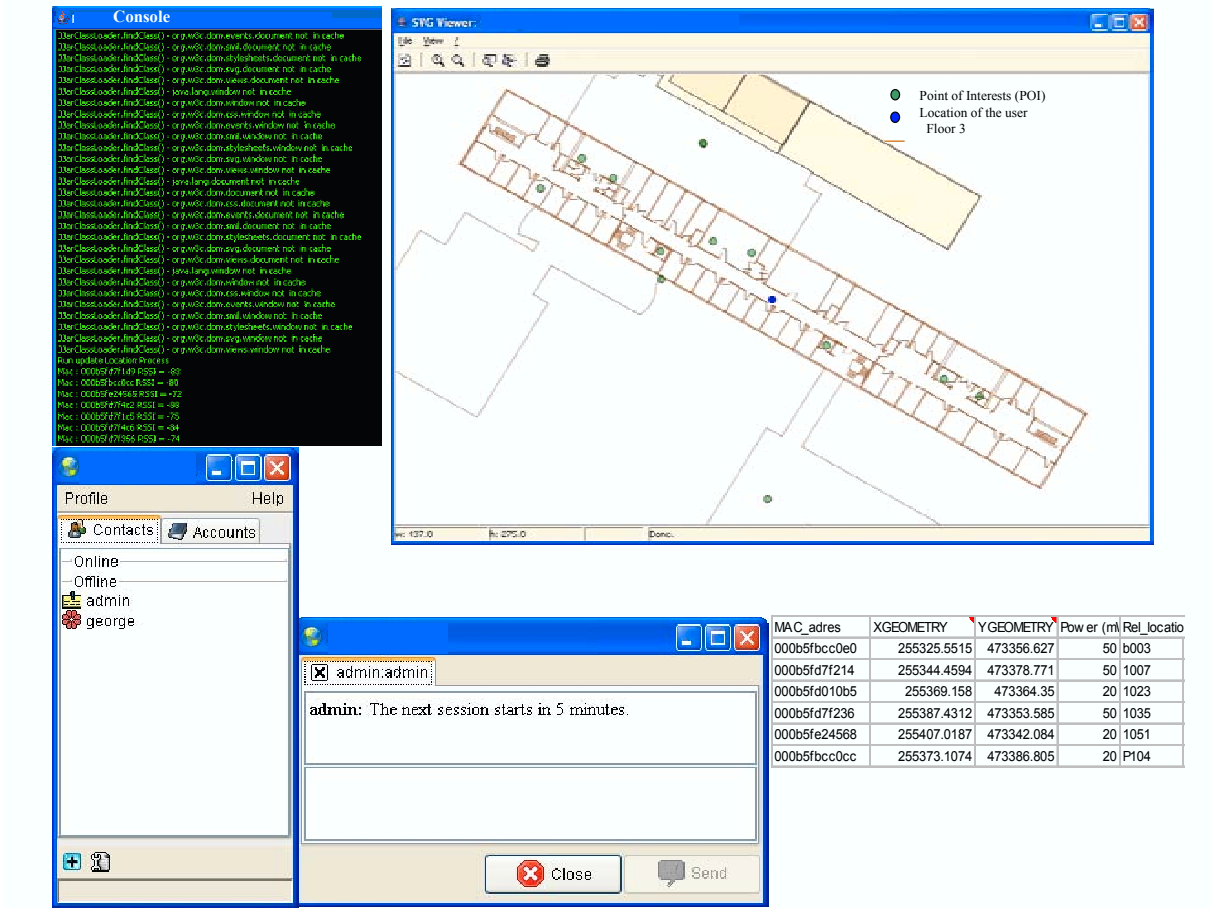


Fig. 2. Snap shots of FLAVOUR prototype

References

1. Hegde, S.: Potential of SVG for a cartographic interface to a route optimization model for the transport of hazardous material. Master thesis. ITC. 2004
2. http://www.w3.org/TR/SVG/intro.html
3. Kavitha Muthukrishnan, Nirvana Meratnia and Maria Lijding: FLAVOUR- Friendly Location- aware conference Aid with pri-Vacy Observant architectURe, CTIT Technical report TR-CTIT-05-28, June 2005, 16 pp.
4. Sun Microsystems, http://www.sun.com/jini, Technical White Paper, December ,1999.
5. http://www.newscientist.com/article.ns?id=dn3834
6. http://www.ekahau.com
7. Anthony LaMarca and Yatin Chawathe and Sunny Consolvo and Jeffrey Hightower and Ian Smith and James Scott and Tim Sohn and James Howard and Jeff Hughes and Fred Potter and Jason Tabert and Pauline Powledge and Gaetano Borriello and Bill Schilit, Place Lab: Device Positioning Using Radio Beacons in the Wild, Proceedings of Pervasive'05,Munich, Germany.

A Service Architecture for Intellectual Work on Mobile Devices

Nadya Belov and Ilya Braude and Werner Krandick

Department of Computer Science
Drexel University, Philadelphia PA 19104, USA

1 Introduction

Recent proliferation of cell phone and mobile computing technologies has yielded an opportunity to design software systems intended to facilitate collaboration and teamwork among geographically dispersed users. On-the-go collaboration on portable computers, however, cannot be facilitated with traditional *Computer Mediated Collaborative Systems* (CMCS)s [1] typically deployed on desktop computers. In the last couple of years, several architectures and software systems have been designed with the mobile platform in mind. The *Wireless Internet Collaborative System* (WICS) is one such CMCS. The design of WICS depicts an example of an architecture tailored especially to the mobile platform [2]. The WICS is furthermore designed especially for intellectual teamwork on-the-go. The architecture of the WICS integrates a dynamic services component intended as an on-the-go, plug-and-play feature where each user will be able to choose any number of available services depending on their needs. The application of services in the mobile computing domain has been previously outlined by Dustdar and Gall [3]. Dustdar and Gall introduce a peer-to-peer architecture with similar goals to the client/server architecture of WICS, namely to allow participants to collaborate with each other on multiple platforms, including the mobile platform. This paper sets to outline the service-oriented architecture of WICS and presents its unique functionalities and benefits in the on-the-go collaboration on the mobile platform.

2 WICS Architecture

2.1 Client & Server

The WICSis an ad-hoc, dynamic, and portable collaborative system designed to provide a means of collaboration in the intellectual domain between two or more users. The WICS is cross-platform and lightweight in its footprint; its small bandwidth requirements allow for the utilization of networking services of cell phone providers.

The WICS application suite is built on top of the Java platform. It utilizes both the Java 2 Standard Edition (J2SE) and the Java 2 Micro Edition (J2ME) platforms. J2ME is build to run on mobile devices such as mobile phones and PDAs. The Java abstraction layer allows us to port the system to various architectures easily.

The WICS network design mostly resembles a traditional client-server architecture. However, it integrates a service architecture that provides users with access to functionalities not available in the core set of system features. WICS clients are able to utilize

these services using a *ServiceManager*. Figure 1 shows a high level diagram of the WICS architecture.

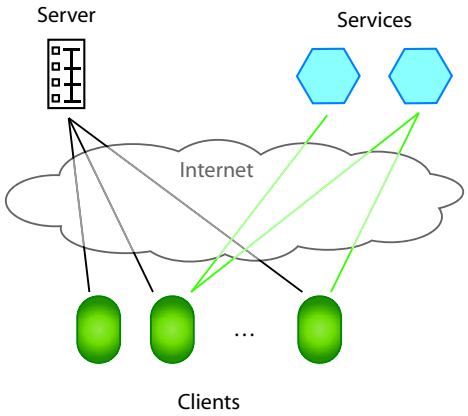


Fig. 1. WICS Network Architecture

2.2 Services

The WICS furthermore supports interaction with outside services via its service architecture. Because the message protocol utilized in WICS is easily parsed and readable, many outside applications can be extended with a façade to work with WICS as services. In turn, these services can also greatly extend the functionality of the system.

Services that make up the dynamic service architecture of WICS are run completely independent of the WICS system. They are published to the clients through a directory on the WICS server. When a service starts, it notifies the WICS server of its name and capabilities. Clients are then able to take advantage of the services that are advertised by the server to augment their own capabilities. However, as shown in Figure 2, once a client is aware of a service, the communication is performed directly between the client and the service, not through the WICS server.

Services can greatly accelerate the user's task [4]. For example, a LaTeXservice can assist a user by rendering high quality math formulas. Other services can provide ready-made diagrams, factor formulas, solve equations, graph functions, and provide other useful tools which increase a user's productivity while on-the-go. Each client keeps an individual service manager, as some services may be available to some clients and not to others. For example, if a client is able to render its own LaTeX formula or plot its own graph, it will not need the functionality of those services.

3 Future Directions

Future developments of the WICS include the development of services designed to monitor and maintain user preferences which will facilitate *Affective Computing* [5].

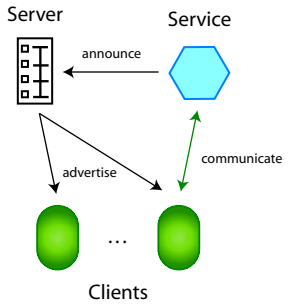


Fig. 2. WICS Services Architecture

Furthermore, an *allocation* service responsible for discovery and coordination of functional services (i.e. graph rendering service) based on user requests and need specification will be integrated into the WICSservice architecture.

4 Demo Proposal

The authors propose to demonstrate the WICS and the benefits it creates in integrating a service architecture with a traditional client-server application for the mobile domain. The demonstration will consist of a collaborative session between two or more participants. They will be provided with a mathematical problem and asked to solve it using the system running on the *Treo 600* Smartphones. All of the participants will have to start the system on their own device, log in and join or create a session. After the completion of the login process, each of the participants will use the service coordinator to request access to a set of services based on the nature of their assignment. Once all of the participants have been granted access to a set of available services which meet the criteria they specified, they will work as a team to solve the assigned problem. The participants will have available to them all of the features of the system as described in § 2.

The authors chose to demonstrate the WICS capabilities in the domain of mathematics. However, it is important to mention that system is designed to facilitate intellectual teamwork in any domain. Mathematics is a universal language, rich in its expressive power. That very same richness tests the usability of a system such as WICS where bandwidth and screen real-estate are limited.

4.1 Hardware Requirements

In order to successfully demonstrate the WICS, the authors will require access to a projector, a power supply and an Internet connection.

References

1. Belov, N.: Facilitating intellectual teamwork on mobile devices. Master's thesis, Drexel University, 3141 Chestnut Street, Philadelphia PA 19104 (2005)

2. Belov, N., Braude, I., Krandick, W., Shaffer, J.: Wireless internet collaboration system on smartphones. In Castro, J., Teniente, E., eds.: Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS), 17th International Conference on Advanced Information Systems Engineering (CAiSE 2005). Volume II. Faculdade de Engenharia da Universidade do Porto (2005) ISBN 972-752-077-4.
3. Dustdar, S., Gall, H.: Architectural concerns in distributed and mobile collaborative systems. J. Syst. Archit. **49** (2003) 457–473
4. Belov, N., Shaffer, J.: Mixed-initiative approach to collaboration in the mathematical domain. In: Proceedings of the Twentieth National Conference on Artificial Intelligence, AAAI Press (2005)
5. Picard, R.W.: Affective Computing. MIT Press (1997)

Achieving Flexibility in Securities Processing & Settlement using Service Oriented Architecture

Dr. Sriram Anand & Naveen Kulkarni,
Web Services Center of Excellence
Infosys Technologies, Ltd
Electronics City
Bangalore, India 560 100
Sriram_anand@infosys.com

Abstract

This paper discusses the applicability of SOA in the area of equity trading and settlement. The lifecycle of a security is quite complex and there are multiple complex steps that are necessary. In most cases, the end user is completely unaware of the number of activities that take place downstream of a trade being placed. Most brokerages and financial service providers have a variety of technologies in place for handling and settling an equity trade. A large chunk of functionality usually resides on legacy systems. These systems are responsible for trade settlement, billing, confirmations, accounting etc. Changes predicated by business model changes or new rules and regulations such as compliance laws require significant changes to legacy systems. Because of the nature of the technologies associated with legacy systems, it is usually a fairly involved activity to make changes to these systems. Apart from this, large financial service organizations have a variety of technologies. Integrating a number of technologies with legacy systems requires tightly coupled, point to point integration that is brittle and difficult to change as well as maintain. In this paper, we present a demo that illustrates the usage of service oriented architecture to increase flexibility with legacy systems. We have demonstrated an approach of achieving flexibility by integrating legacy systems to other web applications using web services.

Example Overview

As enterprises reach architectural maturity with complex IT portfolios to support business, the usage of SOA, leads to the partitioning of such functionalities as a set of services. Business agility is associated with the responsiveness of the IT organization which primarily is based on the reusability of the services.

The example system under consideration is an equity order recording and processing system that places the order recorded on to a particular exchange during the trading time period. In this case, an organization that has a brokerage line of business has been considered along with a basic banking business unit. This system has constituent systems that have been developed in various technologies as illustrated below.

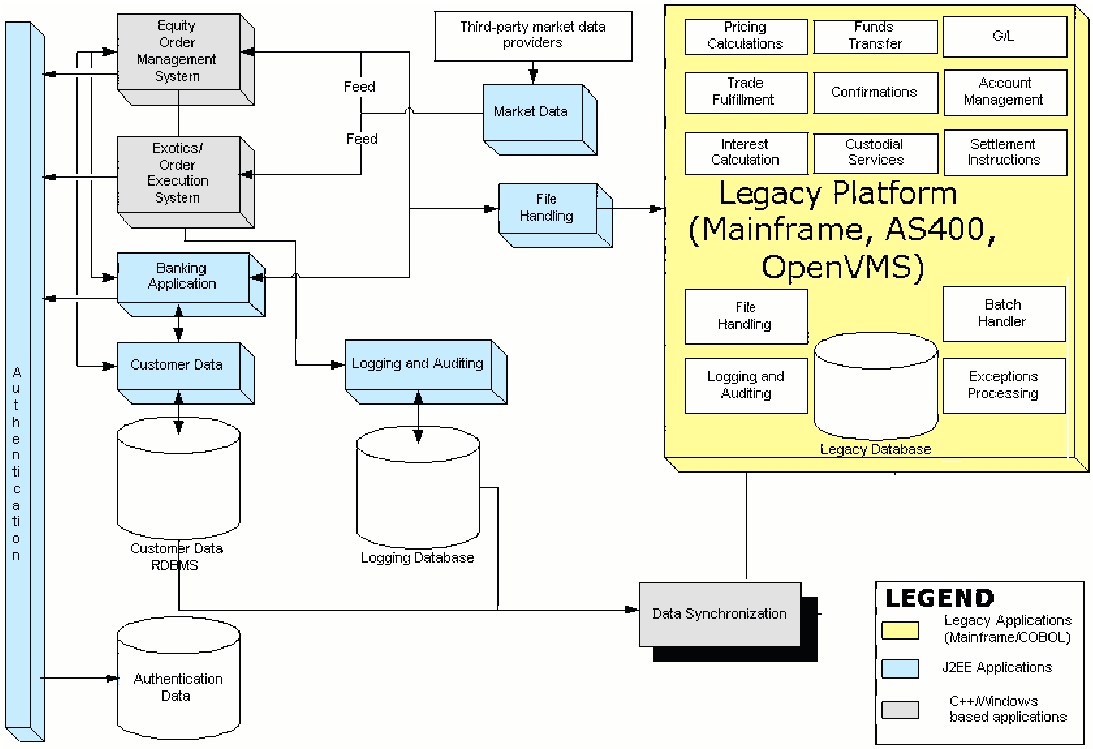


Figure 1 : High level IT landscape

The components of this system are as follows:

Web Applications, these could be built using J2EE or Microsoft based technologies

- Equity order management system: Used to capture orders from users and traders
- Exotics order management system: Used to capture orders from users and traders
- Banking system: Suite of applications for handling basic banking functions.

Legacy Applications:

These applications run on a platform such as the IBM Mainframe z/series platform and typically contain the following modules:

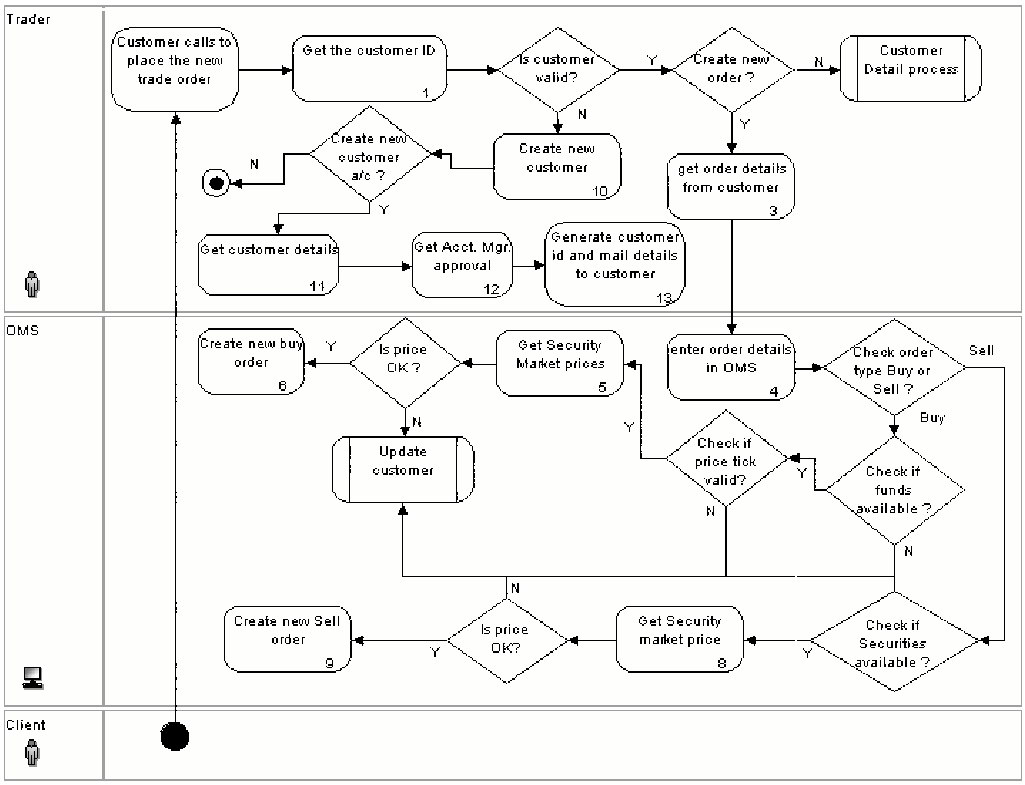
- Pricing Calculations
- Funds Transfer
- Trade Fulfillment
- Custodial Services
- Billing
- G/L
- Confirmations and Statements
- File Handling
- Batch Jobs Execution

It is assumed that the communication between the web applications and the legacy systems is achieved through the usage of files transferred using a proprietary format.

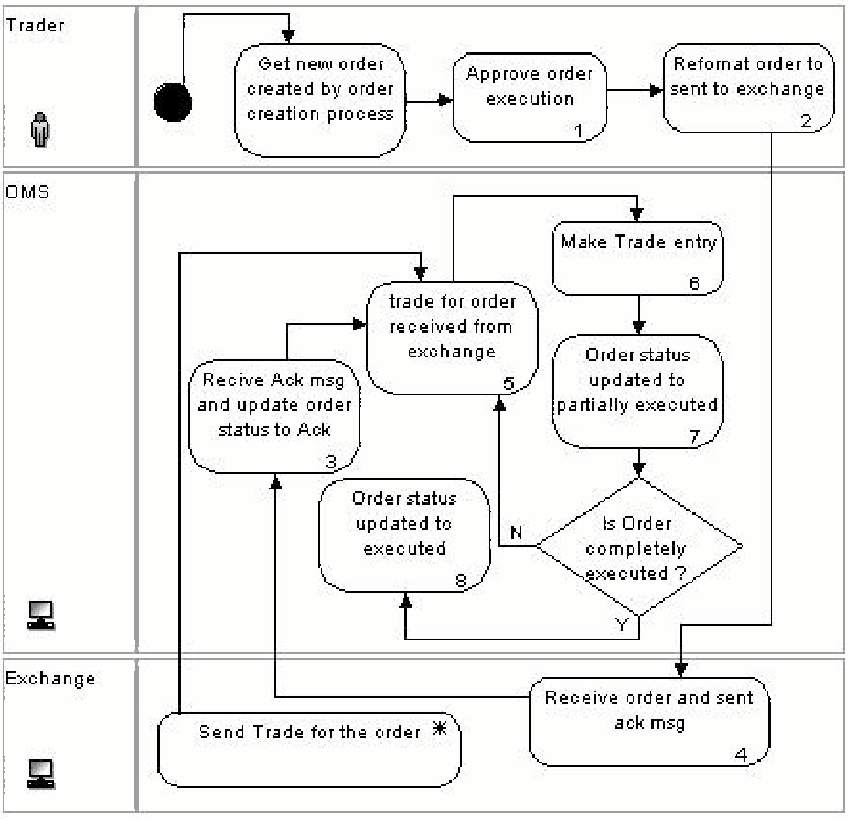
The use cases that are used for the demo have been illustrated below.

In the example, the high level IT system supporting the Order management and execution has been shown in the figure 1. It could be understood that the business applications has been scattered across various applications built or acquired over time. Some of the applications maintained by Settlement department run on different technologies like Pricing, Account management run on host machines where as the order management on the open systems. The following are workflows captured through the interactions with the business analyst while interviewing and also through the documents supplied

Order Creation



Order Execution



Pain Points

The issues associated with legacy systems along with a heterogeneous IT portfolio are as follows:

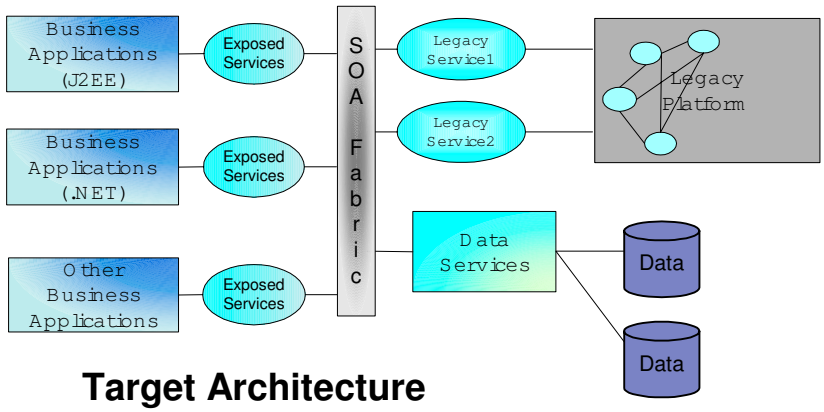
- Unstructured Legacy Programs: Typically legacy programs have evolved without any coherent strategy resulting in redundancy and lack of clarity
- Proprietary Integration Techniques: Typically, it is difficult to integrate legacy programs with other applications. Proprietary techniques are developed, but they are brittle and require specialized knowledge.
- Batch Cycle: Legacy programs usually run on a batch cycle and do not provide online functionality

Thus, these systems lead to difficulty in achieving agility due to the tight coupling and closed architecture.

Advantages of the SOA based approach

The demo attempts to showcase the benefits of SOA for improved integration of legacy systems. SOA will also be used to integrate heterogeneous systems so as to minimize maintenance cost and provide a consistent interface to all collaborating systems.

The target architecture consists of service interfaces to legacy systems that are enabled using tools such as Neonsys Shadow. These tools will perform the mapping between web services and mainframe technologies. A web service based interface will be presented to collaborative systems that will be insulated from the complexities of the legacy system. This provides a non intrusive mechanism for the integration of legacy systems. The target architecture that will be used for the demo is illustrated below:



Target Architecture

Technology Landscape

The table below illustrates the typical technology choices that may be present in financial services organizations for securities processing and settlement processes.

Application	Modules	Platform	Architectu	Communication	Networ	Dependencies

			re	/ Protocol	k	
OMS (Order Management System)	Trader UI Console, OrderManager, TraderManager, SecurityManager, RulesValidator, Authenticator	Windows 2000, JRE	Java, Swing, EJB, JNI, Oracle	RMI, JDBC, Customer fixed length messaging, FIX	TCPIP, VPN	OES, Linked tightly with Market Data system using bridges
OES (Order Execution System)	OrderExecution, LineTranslator, FIXGateway	Windows 2000, JRE	Java, EJB, Oracle	RMI, Custom Fixed length Message, FIX	TCPIP	External link to the FIX Session on Exchange, Linked to the TPS through queue and to DTC through FTP. Messaging is custom define.
MarketData	RTLQuote (Realtime quote), DLDQuote (Delayed quote), MKTNews (Market News), MKTAnalysis (Market Analysis)	Windows 2000 Advance Server	C++, Packaged solution	DCOM	TCPIP, VPN	External link to the Market data provider
TPS (Trade Processing System)	TRDRVW (Trade Review), TRDALLC (Trade Allocation), TRDDTLS (Trade Details), RVCCONF (Receive Confirmation)	IBM z/Series	CICS, JCL, Cobol, DB2, RACF	FTP, MQ, SQL	LU 6.2	
DTC (Depository Trust Clearance)	DTCTRDR (Trade Recorder), GENCONF (Generate Confirmations), GENSETI (Generate Settlement instructions)	IBM z/Series	JCL, Cobol, DB2	FTP, SQL	LU6.2	
Custodian	CSTRTRD (Receive Trade Details), INTMTCH (Matching of Trade details and confirmations), RSKANLY (Risk Analysis)	IBM z/Series	JCL, Cobol, DB2	FTP, MQ, SQL	LU 6.2	Risk analysis is linked to the market data system through MQ

The figure shown below is a shown below gives a brief snapshot on different protocols used for communicating between the applications/system. Such an analysis of the identified resources is very essential during the planning for services as their might have been a significant investment towards setting up such medium, where the services are expected to reuse these medium of communication, for instance, the usage of IBM MQSeries between Market-data and Risk Analysis. Various other analysis can be done during this stage such as understanding the cluttering of the business functionalities and its re-factoring possibilities for instance order status update being done in both OMS and TPS, comparisons on moving from older technologies to newer ones due to lack of availability of skilled engineers, for instance using CICS based transactions for Allocation of order in TPS. This also helps in identifying the liaison between the external partners which the services have to continue using, for instance use of DCOM for market feed and FTP for the core banking.

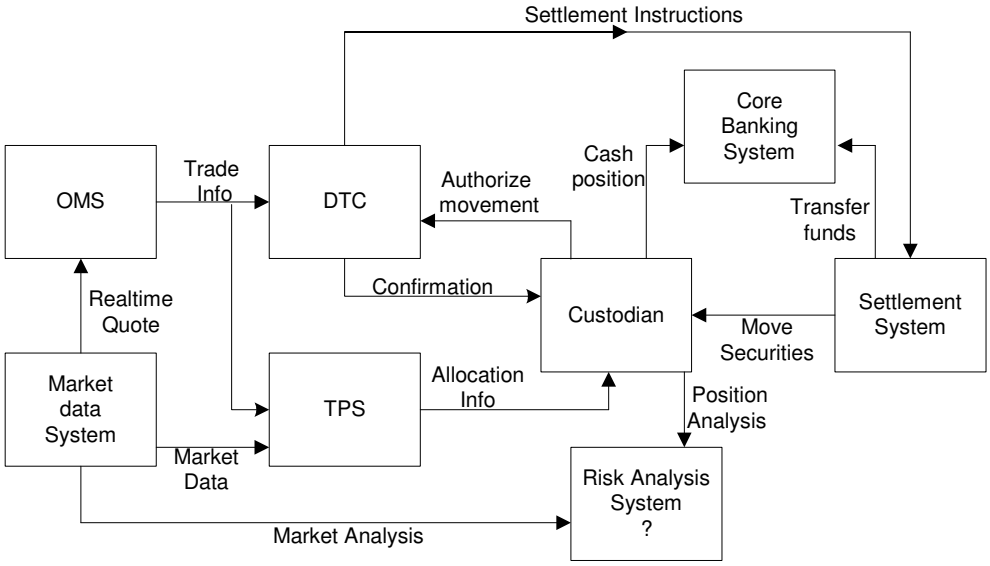
Demo: Sub System view

The Equity order trading system consists of the following sub system:

- Order management system (OMS)
- Market data system
- Transaction processing system (TPS)
- Depository Trust clearance (DTC)
- Custodian
- Risk Analysis system
- Settlement system

- Core banking system

The following diagram shows the collaboration of the various sub systems in order to achieve the pre and post trading of equity orders.

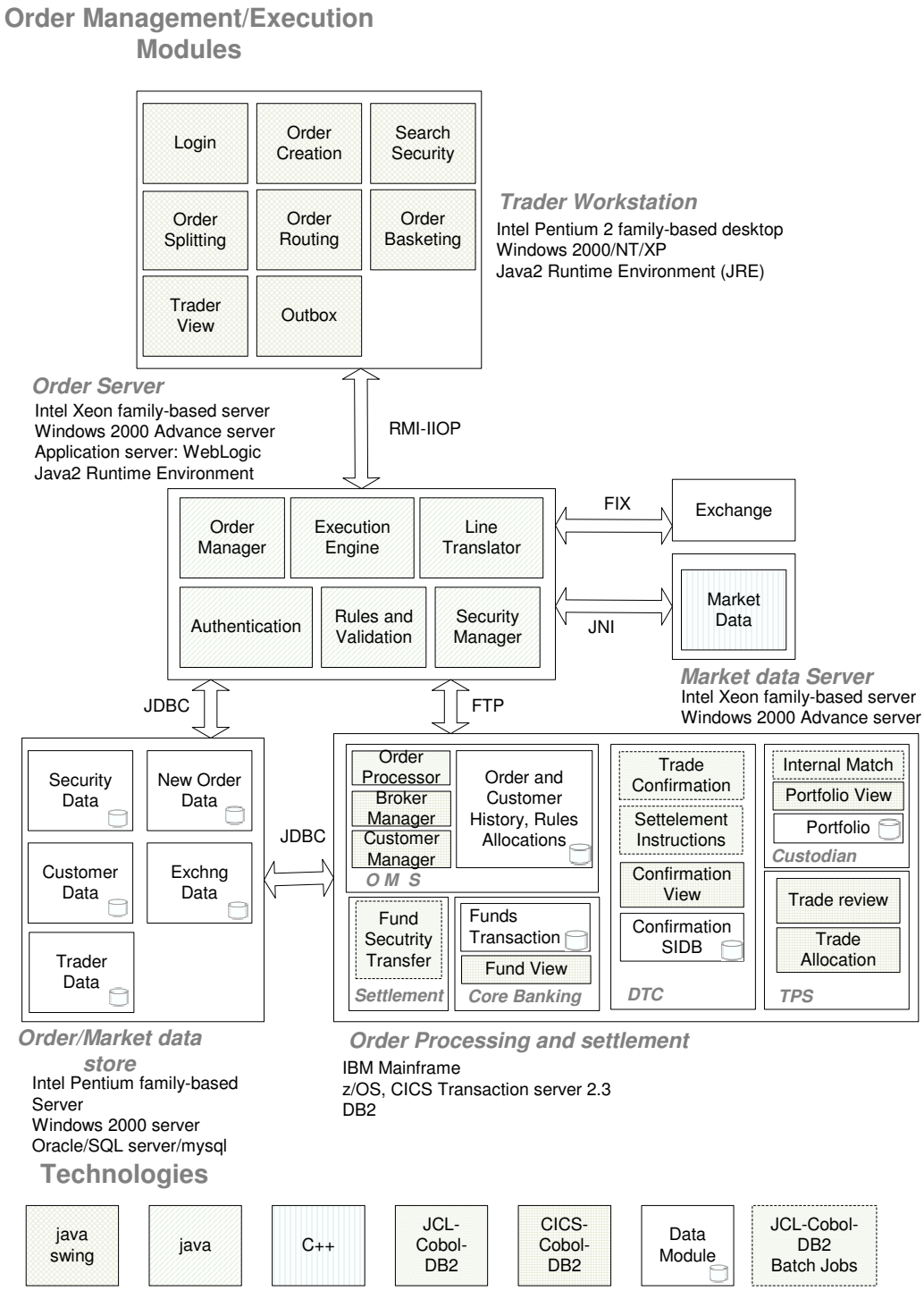


The above diagram indicates the various sub-systems that take part in the equity order execution workflow. The figure also provides the information exchange between them.

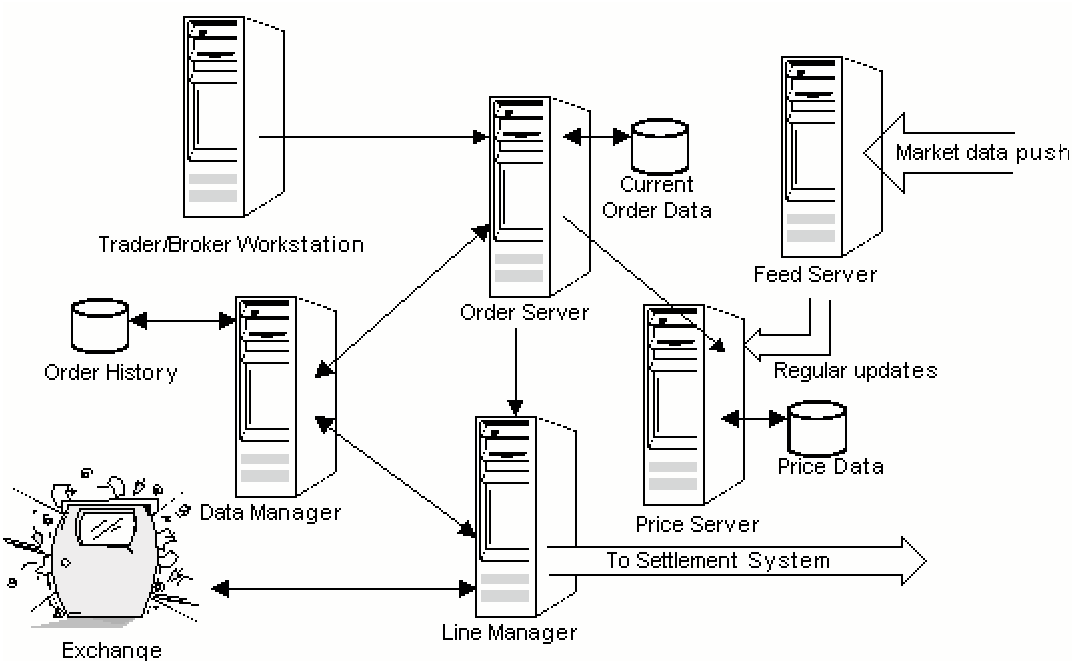
The Order management system (OMS) being one of the sub systems of the equity order trading system is responsible for providing the trade information to the other sub systems as shown in the figure above. The OMS is used by the sales traders and execution traders. The OMS provides modules that will allow the sales traders to create orders and capture the same. It validates the orders captured to check for the few basic rules like order size, tick size, price etc using security and limits management modules. Orders created are broken or combined by the execution trader based on the convenience or on the basis of rules defined. It handles all the working orders and provides the facility for the execution trader to route such orders to any of the other active execution traders. OMS also handles the interactions with the exchange and eventually books the trade.

Implementation Model View

In this section, the overall architectural components used for the model have been illustrated. The following components have been developed for the purpose of the demo.



Based on this technology landscape for the demo, the actual deployment model with the servers used is as illustrated below.



Demo Deliverables

The demo will showcase the following to illustrate the benefits of using SOA to integrate legacy systems in the securities settlement process. First, the demo will illustrate the as-is situation where modification of a business process requires significant change to systems in a variety of technologies, apart from legacy systems. Subsequently, the demo will showcase the target system using SOA. The demo will illustrate the ease of making changes to legacy systems and the mitigation of risk to collaborating systems.

Conclusions

In this demo, we have demonstrated a typical equity order processing and settlement environment. We have made the case for SOA based integration of various systems along with web service based enablement of legacy systems. We have illustrated the flexibility provided by web service based enablement that will lead to enterprise agility and reduced ongoing costs.

ConWeSc$_{prototype}$ - Context-based Semantic Web Services Composition

S. Sattanathan[1], N. C. Narendra[2], and Z. Maamar[3]

[1]National Institute of Technology Karnataka Surathkal, India, Ss_nitk@yahoo.co.in
[2]IBM Software Labs India Bangalore, India, narendra@in.ibm.com
[3]Zayed University, U.A.E, zakaria.maamar@zu.ac.ae

1 Background

A Web service is an accessible application that other applications and humans can discover and trigger to satisfy multiple needs (e.g., travel booking). One of the strengths of Web services is their capacity to be composed into high-level business processes known as composite services [1]. Currently, Web services composition is only achieved at the level of message interactions. This is by far not sufficient, as composition requires to be achieved too at the level of message semantics. The need for a common semantics is intensified when Web services, which originate from different providers, take part in the same composition. To tackle the information disparity challenge, Web services have to agree on the information they will exchange by binding to the appropriate ontology.

Besides the information disparity challenge, further challenges still hinder Web services composition like which businesses have the capacity to provision Web services, when and where the provisioning of Web services occurs, and how Web services from independent providers coordinate their activities, so conflicts are avoided. To face some of these challenges, we recommended considering the context of the composition and execution of Web services [3]. From a Web services perspective, we defined context as a set of common meta-data about the current execution status of a Web service and its capability of collaborating with peers, possibly enacted by distinct providers.

ConWeSc, standing for **Con**text-based semantic **We**b **S**ervices **c**omposition, is a research prototype developed as a proof-of-concept of the feasibility of the outcomes of the project on ontologies for contexts of Web services [3]. The objective of this project is to develop a language similar to **OWL-S** for managing contexts of Web services. This language known as **OWL-C**, standing for **Ontology Web Language-based Context Ontology**, is built upon the following aspects:

- Web services of type instance are obtained out of Web services [2].
- Web services are subject to multiple constraints like maximum number of Web service instances to make available for concurrent use, and strategy for selecting the ontology.
- Prior to participate in a composite service, a Web service assesses its ongoing participations. To perform this assessment, the three types of services, composite, Web, and instance, are each associated with a context of type $\mathcal{C}$-context, $\mathcal{W}$-context, and $\mathcal{I}$-context, respectively. Each type of context is specially geared towards fulfilling the requirements of each type of service [2].
- Values of security contexts ($\mathcal{ISec}$/$\mathcal{WS}$ec/$\mathcal{CS}ec$-contexts) are identified based on the information of service contexts ($\mathcal{I}$/$\mathcal{W}$/$\mathcal{C}$-contexts) [2].

ConWeSc focuses more on context definition, context consolidation at the Web service level, and context reconciliation at the composite service level [3]. *ConWeSc* supports processing the three types of service context ($\mathcal{I}$/$\mathcal{W}$/$\mathcal{C}$-contexts) and three types of security context ($\mathcal{ISec}$/$\mathcal{WS}$ec/$\mathcal{CS}ec$-contexts). $\mathcal{I}/\mathcal{ISec}$-

context has the fine-grained content, whereas $\mathcal{C}/\mathcal{CS}ec$-context has the least grained-content. $\mathcal{W}/\mathcal{WS}ec$-context is in between the two contexts. Details on $\mathcal{I}/\mathcal{IS}ec$-context update $\mathcal{W}/\mathcal{WS}ec$-context, and details on $\mathcal{W}/\mathcal{WS}ec$-context update $\mathcal{C}/\mathcal{CS}ec$-context. **OWL-C** statements are represented as a triple structure consisting of *subject*, *predicate*, and *object*. Fig. 1 presents a typical **OWL-C** expression of $\mathcal{W}$-context. *Subject* is the resource from which the arc leaves. *Predicate* is the property that labels the arc. Finally, *object* is the resource or literal pointed by the arc. In Fig. 1, $\mathcal{W}$-context is the subject (i.e., resource), {status, time_av, running, allowed, ID} are the predicates, and {Active, T=5, 1, 4, WS_1} are the objects.

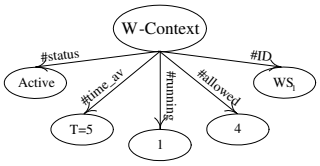


Figure 1: Triple structure of $\mathcal{W}$-context

2 Implementation of *ConWeSc*

ConWeSc comprises a set of plug-ins that runs on top of Eclipse (www.eclipse.org). Eclipse is a platform-independent, open, and extensible workbench, and provides well-designed and well-documented extension points for developers to build domain-specific applications. These plug-ins can be integrated into the workbench without the technical limitations imposed by most proprietary development environments.

The development of *ConWeSc* has called for four plug-ins: *user interface*, *WS-execution platform*, *ontology repository*, and *help repository*. Fig. 2 illustrates these plug-ins binding to Eclipse. The user-interface plug-in extends the workbench in terms of perspective, wizards, views, and editor. The WS-execution platform plug-in extends the workspace in terms of project nature (i.e., context based semantic Web services) and builder (i.e., context assessment, validation, and reasoning). The ontology repository plug-in stores and retrieves the context ontologies by extending them. Finally, the help repository plug-in provides the necessary documentation for using *ConWeSc* (Section 3). We use the following software for the development of *ConWeSc*: Eclipse PDE (Plug-in Development Environment) for developing the aforementioned plug-ins, SWT (Standard Widget Toolkit), JFACE (User Interface tool kits) classes for developing user interfaces, Jena (jena.sourceforge.net) for defining and processing context ontologies, and Mindswap's OWL-S API (www.mindswap.org/2004/owl-s/api) for processing OWL-S based ontologies.

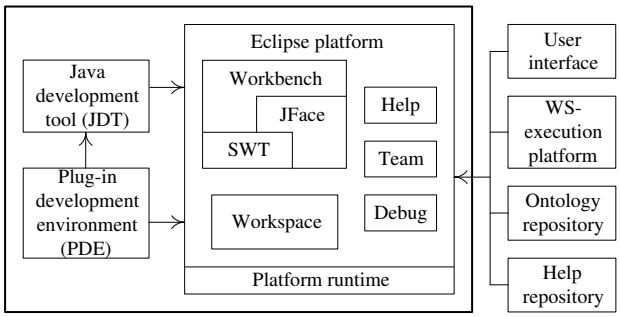


Figure 2: Architecture of *ConWeSc*

3 Steps of demonstrating *ConWeSc*

During the demonstration we aim at showing how context ontologies back Web services composition and execution. We have created *Book Finder* and *Book Payment* Web services, as well as *Book Purchase*

composite service. *Book Finder* Web service identifies the details about a given book, and *Book Payment* Web service performs payment related operations (e.g., credit card verification, account debiting). Finally *Book Purchase* (noted *Book Service* in the various interfaces) composite service sequentially integrates these two Web services.

For illustration needs, we have set the limit of Web service instances of *Book Finder* and *Book Payment* to 3 and 1, respectively. When an instantiation request for either service is received after its respective limit has been reached, *ConWeSc* displays an error message, and the Web service will not be instantiated. *ConWeSc* also supports the ontology representation of $\mathcal{W}/\mathcal{WS}ec$-, $\mathcal{C}/\mathcal{CS}ec$-, and $\mathcal{I}/\mathcal{IS}ec$-context using the **OWL-C** formalism [3]. *ConWeSc* processes **OWL-C** files in the form of triples and displays results after running consolidation and reconciliation operations.

3.1 Part A - Contexts of Services

How to consolidate contexts? Consolidation happens at the level of Web services [3]. Consolidation means the combination of details that stem from a lower level (Web service instances) to a higher level (Web services). Once the consolidation is completed, a Web service can determine for each of its Web service instances the following: execution status, the actions it has performed, and the expected completion execution-time, so the Web service can commit additional Web service instances as per other composite services' requests. Fig. 3-(a) presents the initial values of the $\mathcal{W}$-context parameters of *Book Finder*. In this figure, the focus is on *InstanceRunning* parameter (highlighted in green). After the acceptance of two instantiation requests, the consolidated version of $\mathcal{W}$-context shows two Web service instances under execution (Fig. 3-(b)). When one of the instances successfully completes its execution, the number of running instances drops to 1 (Fig. 3-(c)). *InstanceAllowed* parameter corresponds to the maximum number of service instances that a Web service can concurrently deploy.

(a)

W-Context Parameters	Values
Label	Book_Finder
InstanceAllowed	3
InstanceRunning	0
NextInstanceAvilability	true
Status	Active

(b)

W-Context Parameters	Values
Label	Book_Finder
InstanceAllowed	3
InstanceRunning	2
NextInstanceAvilability	true
Status	Active

(c)

W-Context Parameters	Values
Label	Book_Finder
InstanceAllowed	3
InstanceRunning	1
NextInstanceAvilability	true
Status	Active

Figure 3: Consolidation of contexts

How to conciliate contexts? Reconciliation happens at the level of composite services, since the component Web services of a composite service have multiple providers, and the definition of their respective $\mathcal{W}$-contexts (and obviously the definition of the $\mathcal{I}$-contexts of their Web service instances) varies in terms of structure and content (e.g., different numbers of arguments, different names of arguments) [3]. The transfer of details from the $\mathcal{I}$-contexts of the Web service instances to the $\mathcal{C}$-context of a composite service is featured by a reconciliation of these details before the $\mathcal{C}$-context is updated

A part of the reconciliation as supported by *ConWeSc* is shown in Fig. 4. Fig. 4-(a,b) shows the initial status of both the $\mathcal{W}$-context of *Book Finder* and the $\mathcal{C}$-context of *Book Purchase* after *Book Finder* accepts the request of *Book Purchase*. *PreviousWebService*, *CurrentWebService*, and *NextWebService* parameters are significant for the demonstration. It can be seen for instance that *Book Purchase* will sequentially execute *Book Finder* and *Book Payment*. In Fig. 4-(b), *Book Purchase* is under execution whereas *Book Payment* is expected to be initiated upon completion of this execution. Fig. 4-(d) presents the $\mathcal{I}$-context of *Book Payment* service instance (highlighted in green). This instance has a waiting status, i.e., waiting for the completion of *Book_Finder_Service_Instance_1*. Once the execution of this instance is over (Fig. 4-(c)), *Book_Payment_Instance_1* will be changed to active (Fig. 4-(e)). The appropriate parameters of the $\mathcal{C}$-context of *Book Purchase* are also updated based on the execution success of its component (Fig. 4-(f)).

How to relate services to contexts? *ConWeSc* provides three different form-based editors for creating $\mathcal{C}$-, $\mathcal{W}$-, and $\mathcal{I}$-contexts using **OWL-C**. The generation of a $\mathcal{C}$-context requires that the user inputs the re-

(a)

W-Context Parameters	Values
Label	Book_Finder
InstanceAllowed	3
InstanceRunning	0
NextInstanceAvilability	true
Status	Active

(b)

C-Context Parameters	Values
Label	Book Service
Status	Active
PreviousWebService	Nil
CurrentWebService	Book Finder
NextWebService	Book Payment

(c)

I-Context Parameters	Values
Label	Book_Finder_Service_Instance_1
Status	Active
PreviousServiceInstance	Nil
NextServiceInstance	Book_Payment_Instance_1
RegularAction	Finding a Book
ReasonsOfFailure	Nil
CorrectiveAction	Nil

(d)

I-Context Parameters	Values
Label	Book_Payment_Instance_1
Status	Waiting
PreviousServiceInstance	Book_Finder_Instance1
NextServiceInstance	Nil
RegularAction	Getting Payment for Book
ReasonsOfFailure	Nil
CorrectiveAction	Nil

(e)

I-Context Parameters	Values
Label	Book_Payment_Instance_1
Status	Active
PreviousServiceInstance	Book_Finder_Instance1
NextServiceInstance	Nil
RegularAction	Getting Payment for Book
ReasonsOfFailure	Nil
CorrectiveAction	Nil

(f)

C-Context Parameters	Values
Label	Book Service
Status	Active
PreviousWebService	Book Finder
CurrentWebService	Book Payment
NextWebService	Nil

Figure 4: Reconciliation of contexts

quired context-ontology parameter values in the appropriate text fields. If any context parameter is missing then *ConWeSc* will raise an error message, so the user is alerted.

For example, if the input values {Book Purchase, Nil, Book Finder, Book Payment, Active, 20:30:45, 17/12/2004} are respectively associated with the parameters of $\mathcal{C}$-contexts {Label, Previous Web Service, Current Web Service, Next Web Service, Status, Begin Time, Date}, then the generated **OWL-C** description of $\mathcal{C}$-contexts is given in Fig. 5. The generated context file will have "**owlc**" as extension. Similarly, a user can generate $\mathcal{I}$-context and $\mathcal{W}$-context descriptions based on **OWL-C** using the appropriate forms.

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<rdf:RDF
  xmlns:CContext="http://www.nitk.ac.in/ sattanathan/OWLC/Context/CContext#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://defaultURI/CContext#">
  <CContext:Date>17/12/2004</CContext:Date>
  <CContext:NextWebService>Book Payment</CContext:NextWebService>
  <CContext:Status>Active</CContext:Status>
  <CContext:BeginTime>20:30:45</CContext:BeginTime>
  <CContext:Label>Book Purchase</CContext:Label>
  <CContext:CurrentWebService>Book Finder</CContext:CurrentWebService>
  <CContext:PreviousWebService>Nil</CContext:PreviousWebService>
  </rdf:Description>
</rdf:RDF>
```

Figure 5: OWL-C representation of $\mathcal{C}$-context of Book Service

3.2 Part B - Security of Contexts of Services

How to secure contexts? Our model for securing the interactions between Web services features three security contexts, i.e., $\mathcal{ISec}$-context for Web service instance, $\mathcal{WSec}$-context for Web service, and $\mathcal{CSec}$-context for composite service. These security contexts are defined along with the regular service contexts (i.e., $\mathcal{I}/\mathcal{W}/\mathcal{C}$-context). A security context exposes the security strategy that a service adopts. A security context has a major role in highlighting the security strategy that a service adopts. Any change in this

strategy is automatically reflected in the security context so other peers are aware of the change and might have to comply with it. The **OWL-C** representation of $\mathcal{CS}ec$-context is shown in Fig. 6.

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<rdf:RDF
  xmlns:CsecContext="http://www.nitk.ac.in/ sattanathan/OWLC/Context/CsecContext#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://defaultURI/CContext#">
  <rdf:Description rdf:about="http://www.csp.com/CsecContext#">
  <CsecContext:Label>Book_Service</CsecContext:Label>
  <CsecContext:Signature>Composite</CsecContext:Signature>
  <CsecContext:SecurityMechanism>BlowFish</CsecContext:SecurityMechanism>
  <CsecContext:SecurityOfPreviousWebServiceInstances>DES</CsecContext:SecurityOfPreviousWebServiceInstances>
  <CsecContext:SecurityOfCurrentWebServiceInstances>AES</CsecContext:SecurityOfCurrentWebServiceInstances>
  <CsecContext:SecurityOfNextWebServiceInstances>AES</CsecContext:SecurityOfNextWebServiceInstances>
  <CsecContext:CorrectiveActions>None</CsecContext:CorrectiveActions>
  </rdf:Description>
</rdf:RDF>
```

Figure 6: OWL-C representation of $\mathcal{CS}ec$-context of Book Service

How to consolidate security contexts? The following Fig. 7 shows the security context consolidation of Book Finder Web Service Provider. In Fig. 7-(a) shows the initial security context argument values. This context values are determined according to the values of service context (i.e., in this case $\mathcal{W}$-Context), Fig. 7-(b) shows the security context argument values after the successful completion of two service instances (shown in *SecurityStatusPerServiceInstance* field). As per *ConWeSc*, *SecurityStatusPerServiceInstance* can have either success or failure value. Fig. 7-(b) shows the successful case.

How to conciliate security contexts? Security context reconciliation happens at the level of composite service, since the component Web services of a composite service have multiple providers, and the definition of their respective WSec-contexts varies in terms of structure and content. The transfer of details from the $\mathcal{IS}ec$-contexts of the Web service instances to the CSec-context of a composite service is featured by a reconciliation of these details before the CSec-context is updated. Fig. 8-(a) shows the initial security context values of Book Service Composite Service. Fig. 8-(b) shows the security context values of *Book_Finder_Service_Instance*. Fig. 8-(c) shows the security context values of Book Payment Service Instance. Fig. 8-(d) shows the results of security context after the successful completion of *Book_Finder_Service_Instance_1*.

4 Summary

In this paper, we overviewed *ConWeSc*, a research prototype that demonstrates Web services composition and execution using a dedicated ontology for contexts of Web services. To back this demonstration, we adopted a simple yet realistic example of online book purchase. Context formalization has several clear advantages. For instance, this allows storing the context for further use since its meaning remains the same. In addition, this enables communicating context with other systems. Ignoring the problem of context het-

(a)

Wsec-Context Parameters	Values
Label	Book_Finder
Signature	WebService
SecurityMechanism	DES
SecurityStatus	success
SecurityViolation	none
CorrectiveActions	none
SecurityStatusPerServiceInstance	<no instances available>

(b)

Wsec-Context Parameters	Values
Label	Book_Finder
Signature	WebService
SecurityMechanism	DES
SecurityStatus	success
SecurityViolation	none
CorrectiveActions	none
SecurityStatusPerServiceInstance	1<success>;2<success>;

Figure 7: Security context consolidation of Book Finder (at the Web service-provider level)

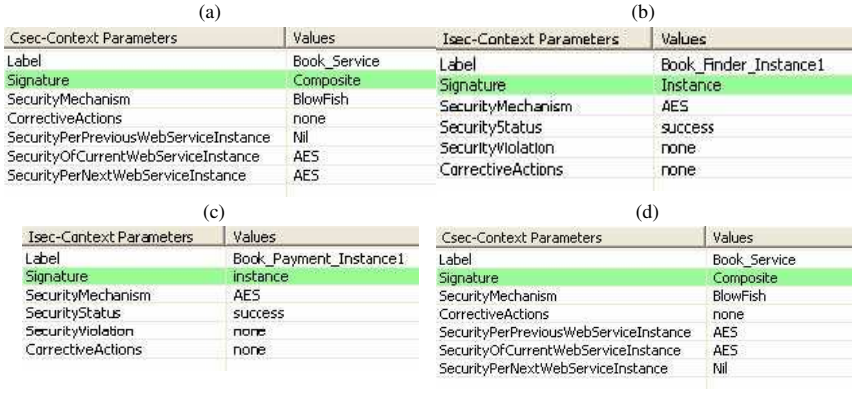


Figure 8: Security context reconciliation of Book Service (at the composite service-provider level))

erogeneity of Web services has side-effects on the progress of their composition. Indeed, formal modeling of context has been stressed by several works [4]. These side-effects are multiple like adopting the wrong strategy for selecting a component Web service (e.g., favoring execution-cost criterion over reliability criterion, instead of the opposite), delaying the triggering of some urgent component Web services, or poorly assessing the exact execution status of a Web service.

Acknowledgments

The first author is supported by the Center for Advanced Studies (CAS) program of IBM Software Labs India. The first author would also like to thank Prof. K. C. Shet of NITK, for supporting his doctoral work. The second author wishes to thank his manager, K. Muralidharan, for his support. Other company (i.e., non-IBM), product and service names may be trademarks or service marks of others.

References

[1] I. Budak Arpinar, B. Aleman-Meza, R. Zhang, and A. Maduko. Ontology-Driven Web Services Composition Platform. In *Proceedings of The IEEE International Conference on E-Commerce Technology (CEC'2004)*, San-Diego, USA, 2004.

[2] Z. Maamar, S. Kouadri Mostéfaoui, and H. Yahyaoui. Towards an Agent-based and Context-oriented Approach for Web Services Composition. *IEEE Transactions on Knowledge and Data Engineering*, 17(5), May 2005.

[3] Z. Maamar, N. C. Narendra, and S. Sattanathan. Towards an Ontology-based Approach for Specifying and Securing Web Services. *Information and SoftwareTechnology, Elsevier Science Publisher*, 2005 (forthcoming).

[4] A. Shehzad, H. Q. Ngo, K. Anh Pham, and S. Y. Lee. Formal Modeling in Context Aware Systems. In *Proceedings of The 1st International Workshop on Modeling and Retrieval of Context (MRC'2004)*, Ulm, Germany, 2004.

The Gardens Point Service Language: Overview and Implementation

Dominic Cooney, Marlon Dumas, and Paul Roe

Queensland University of Technology, Australia
{d.cooney, m.dumas, p.roe}@qut.edu.au

Abstract. Implementing web services that participate in long-running, multi-lateral conversations is difficult because mainstream programming languages are poor when it comes to manipulating XML data and handling concurrent and interrelated interactions. We have designed a programming language to deliberately address these problems. In this paper we describe how to use this language to consume a popular web service, and discuss the compiler and runtime system. We demonstrate the compiler, including the kinds of semantic checks it performs, the running program, and the SOAP messages produced at runtime. The compiler and sample program are available at http://www.serviceorientation.com.

1 Introduction

The need to integrate applications within and across organizations is increasingly met with web services. Implementing simple request-response interactions between statically known participants using traditional middleware and programming languages is reasonably straightforward, but implementing long-running conversations amongst large and changing sets of participants is difficult. Aspects of web services that provide serious implementation challenges include: prevalent XML data, explicit boundaries, concurrent messages, and process awareness.

XML data: The data model of web services is XML InfoSet. InfoSet is an open data representation with no notion of behavior. Object-oriented (OO) programming prizes data encapsulation by marrying data and behavior. To address the mismatch OO programming languages variously model InfoSet with objects, map between objects and InfoSet, or support InfoSet directly via language extensions. In these solutions object models are indirect, mappings are incomplete, and language extensions are redundant in their OO data model.

Explicit boundaries: Unlike components in a virtual machine, or processes in an operating system, there is no supervising infrastructure between services. Since implementation technologies vary, or because organizational boundaries entail secrecy, the internal logic of other services may be completely opaque. Programming languages with global models of interacting services, typically distributed object models, are useful for abstractly modeling service oriented architectures, but implementers are limited to purely local phenomena, such as messages, and can not rely on a global view.

Concurrent messages: Messages link distributed nodes, all processing concurrently. For basic scalability web services must handle concurrent messages. Implementers must be cautious of race conditions, deadlocks, and live-locks—all problems that mainstream object-oriented languages make difficult to solve.

Process awareness: Web services often correspond to business functionality, and so are likely to be part of long-running interactions driven by explicit process models. They may engage in conversations with a dynamically changing set of partners and a large number of events that may occur in many orders.

BPEL [1] addresses some of these problems, but it turns out that coding complex multi-lateral interactions in BPEL, especially those that require partial synchronization and one-to-many correlation can be cumbersome [2].

We addressed the above issues in the design of Gardens Point Service Language (GPSL) [3, 4] with the following features:

- **Embedded XQuery.** XQuery is a functional language for querying and synthesizing XML data [5], with a data model close to XML InfoSet. GPSL supports the manipulation of XML data via embedded XQuery expressions.
- **Services, contracts, and explicit message sending.** GPSL has explicit service and contract language elements. Lexical scoping ensures services rely on purely local data. Services exchange data by explicitly sending messages.
- **Join calculus-style concurrency [6].** GPSL simplifies forking, joining, and concurrent operations with declarative rules. At a low level of abstraction these rules facilitate the manipulation of concurrent messages; at a higher level they support the modeling of complex processes with state machines.

The feature set of GPSL is unique, yet GPSL belongs to a small set of service-oriented programming languages [7–10]. In this paper we focus on writing service consumers, which is important for implementing services that aggregate other services. We also present the compiler and runtime system.

2 GPSL by Example

GPSL is primarily for developing services, and an important aspect of implementing a service is interacting with other services. In this example we describe how to use GPSL to consume the Amazon queue service.[1] The Amazon queue service is a SOAP document/literal style service that supports inserting XML data into a queue; reading from a queue, with time-outs; and managing queues.

First we declare an XQuery XML namespace for data used by the service:

```
declare namespace sqs =
    'http://webservices.amazon.com/AWSSimpleQueueService/2005-01-01';
```

Next we write the service contract. The Amazon queue service uses a pattern where all operations have the same SOAP action and the behavior is controlled by the data in the body of the message, so the contract declaration is simply:

[1] `http://webservices.amazon.com/AWSSimpleQueueService/AWSSimpleQueueService.wsdl`

```
declare interface SimpleQueueService {
  declare operation SQSOp webmethod action = 'http://soap.amazon.com'
}
```

SimpleQueueService and *SQSOp* are identifiers we use to refer to the operation. *webmethod* declares this operation as synchronous SOAP-over-HTTP. This piece of metadata governs the behavior of the runtime system, but to the programmer *in-out* SOAP operations via a pair of asynchronous messages and synchronous *webmethod* operations appear uniformly as asynchronous operations.

Now we bind some constant values: the endpoint of the Amazon queue service, and our subscriber ID, which we have to include in every message. We could, of course, vary these with parameters if desired.

```
(: URI of the Amazon Simple Queue Service :)
let $sqs :=
'http://webservices.amazon.com/onca/soap?Service=AWSSimpleQueueService' in
(: Amazon Web Services subscription ID :)
let $subscriptionID := '...' in
```

Performing an interaction, e.g. to create a queue, involves constructing a request, sending it, and processing the response:

```
let $request := element sqs:CreateQueue {
  element sqs:SubscriptionId { $subscriptionID },
  element sqs:Request {
    element sqs:CreateQueueRequest {
      element sqs:QueueName { 'My queue' },
      element sqs:ReadLockTimeoutSeconds { 10 }
    }
  }
} in
def Ignore($response) { } in
$sqs: SQSOp($request, Ignore)
```

This sequence of element constructors produces XML like the following:

```
<sqs:CreateQueue xmlns:sqs=
  "http://webservices.amazon.com/AWSSimpleQueueService/2005-01-01">
  <sqs:SubscriptionId>...</sqs:SubscriptionId>
  <sqs:Request>
    <sqs:CreateQueueRequest>
      <sqs:QueueName>My queue</sqs:QueueName>
      ...
```

The *def* construct is used to introduce a new internal label, *Ignore,* and an associated block to execute when a message is produced on that label. In the above example, sending a message to *Ignore* would do nothing—the block labelled with *Ignore* is empty.

The line *$sqs: SQSOp($request, Ignore)* actually sends the message. The prefix argument *$sqs* is the endpoint to send to. In this case *$sqs* is bound to the

endpoint of the Amazon queue service, but we could have bound *$sqs* to data in a request received at runtime! This is how GPSL supports invoking services dynamically.

SQSOp is the operation declared in the *SimpleQueueService* contract, which provides the SOAP action and *webmethod* operation style. The fragment of XML we just constructed in *$request* supplies the body of the SOAP message. Finally, the *Ignore* argument supplies the label to process SOAP replies with. Because *SQSOp* is declared as a *webmethod,* we must provide some way to handle replies.

This approach to message sending, though direct, is inconvenient if we need to create more than one queue. The *def* construct is very convenient for small-scale abstraction building:

```
def CreateQueue($queue-name, $timeout, create-reply) {
  let $request := element sqs:CreateQueue {
    element sqs:SubscriptionId { $subscriptionID },
    element sqs:Request {
      element sqs:CreateQueueRequest {
        element sqs:QueueName { $queue-name },
        element sqs:ReadLockTimeoutSeconds { $timeout }
      }
    }
  } in
  $sqs: SQSOp($request, create-reply)
} in
def Ignore($response) { } in
CreateQueue('My queue', 10, Ignore)
```

Usually the response contains useful data, and these *defs* can introduce nested *defs* that extract data from the response and forward the distilled result to *create-reply:*

```
def CreateQueue($queue-name, $timeout, create-reply) {
  let $request := (: same as above :) in
  def Unpack($response) {
    let $queue-id := $response//QueueId/text() in
    create-reply($queue-id)
  } in
  $sqs: SQSOp($request, Unpack)
} in
...
```

3 The GPSL Compiler

The GPSL compiler has a traditional parse, analyse, emit structure. The parser must handle XQuery for expressions. For our prototype we found ignoring XQuery direct constructors—the angle-brackets syntax for synthesizing XML which require special handling of whitespace—greatly simplifies parser development. Because syntactically simpler computed constructors can do the job of direct constructors, the expressive power of XQuery is unimpeded.

The analysis phase of the compiler is dominated by resolving identifiers and reporting undeclared variables or on passing too few or too many parameters. This phase includes a Hindley-Milner style type inferencer for labels. This is because we must prevent labels leaking into XML values. Syntax trivially prevents labels appearing in XQuery expressions, because variables bound to XML values are always prefixed with a *$,* whereas labels and variables bound to labels are not. However sending a message on a label could pass a label where an XML value was expected. The types from our inference let the compiler guarantee statically that this does not happen.

If a label could escape into a larger XML value, we would have to track the reference to that label in order to keep the closure it refers to alive; in the worst case if a label escapes from the service we must set up the SOAP messaging machinery to marshal messages into the closure. Of course, when the programmer supplies a label as the *reply-to* parameter of an operation, that label *must* be reified as an XML value.

The last semantic check of the analysis phase is that *in, in-out* and *webmethod* operations obey the convention of a parameter for the SOAP body and a parameter for the reply channel (in the *in-out* and *webmethod* case.) These lead to a limited set of types that are used to generate code that marshal between SOAP messages and messages on internal labels. In principle, schema validation of messages could be incorporated.

The code generator produces Microsoft Intermediate Language (MSIL), which is similar to Java byte code although differs in many details. Most of the complexity in the code generator is in creating closures and delivering messages on internal labels. For each *def* we create a class with a method for each concurrency rule, a field for each captured variable, and a method and field for each label. This field holds a queue of pending messages; the method takes a message to that label, tests whether any rules are satisfied, and if so, calls the method for the rule. We perform the rule testing on the caller thread and only spawn a thread when a rule is satisfied, which avoids spawning many threads.

We do not compile XQuery expressions because implementing an XQuery compiler is a daunting task. Instead we generate code to call an external XQuery library at runtime. One critical criterion for the programming language implementer integrating an XQuery implementation is how that XQuery implementation accepts external variables and provides results. GPSL requires access to expression results as a sequence of XQuery data model values—which is distinctly different from an XML document—to behave consistently with XQuery when those values that are used later in subsequent expressions. We use an interoperability layer over the C API of Galax[2], which has exactly the kind of interface for providing external values and examining results that we want. Our biggest complaint about Galax is that evaluating expressions must be serialized because Galax is non-reentrant.

[2] `http://www.galaxquery.org`

GPSL programs also depend on the Microsoft Web Services Extensions[3] (WSE) for SOAP messaging. WSE has a low-level messaging interface which is sufficient for GPSL's needs, but it has major shortcomings too: WSE does not support SOAP RPC/encoded, and we have to include some bookkeeping to make SOAP over synchronous-HTTP work using this low-level messaging interface.

About the Demonstration We demonstrate an extended version of the program outlined in Section 2, including the SOAP messages produced at runtime; and the compiler, including the kinds of semantic checks described in Section 3. The demonstration compiler and sample program are available online.[4]

References

1. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution language for web services. Technical Report Version 1.1, BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems (2003)
2. Barros, A., Dumas, M., Hofstede, A.: Service interaction patterns. In: Proceedings of the 3rd International Conference on Business Process Management, Nancy, France, Springer Verlag (2005) Extended version available at: http://www.serviceinteraction.com.
3. Cooney, D., Dumas, M., Roe, P.: A programming language for web service development. In Estivill-Castro, V., ed.: Proceedings of the 28th Australasian Computer Science Conference, Newcastle, Australia, Australian Computer Society (2005)
4. Cooney, D., Dumas, M., Roe, P.: GPSL: A programming language for service implementation. Submitted to ICSOC (2005)
5. Boag, S., Chamberlin, D., Fernández, M.F., Florescu, D., Robie, J., Siméon, J.: XQuery 1.0: An XML query language. W3C Working Draft (2005)
6. Fournet, C., Gonthier, G.: The reflexive chemical abstract machine and the join calculus. In: Proceedings of the 23rd ACM Symposium on Principles of Programming Languages (POPL). (1996) 372–385
7. Onose, N., Siméon, J.: XQuery at your web service. In: Proceedings of the 13th International Conference on World Wide Web, New York, NY, USA, ACM Press (2004) 603–611
8. Florescu, D., Grünhagen, A., Kossmann, D.: XL: A platform for Web services. In: Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA (2003)
9. Cardelli, L., Davies, R.: Service combinators for web computing. Software Engineering **25** (1999) 309–316
10. Kistler, T., Marais, H.: WebL – A programming language for the web. In: Proceedings of the 7th International Conference on World Wide Web, Amsterdam, The Netherlands, The Netherlands, Elsevier Science Publishers B. V. (1998) 259–270

[3] `http://msdn.microsoft.com/webservices/building/wse`
[4] `http://www.serviceorientation.com`