An Architecture for IMS Services Using Open Source Infrastructure

Robert Mullins, Shane Dempsey, Tom Pfeifer

Telecommunications Software & Systems Group [TSSG]
Waterford Institute of Technology [WIT]
Cork Road, Waterford, Ireland
rmullins@tssg.org, sdempsey@tssg.org, t.pfeifer@computer.org

Abstract. IMS ARCS is an Industrial and Academic cooperative program conducting research in the area of IMS technology with a view to creating a body of intellectual property for the use of project partners, and serving as a case study on how to develop IMS end user and enabling services. The project employs the OpenIMS testbed from Fraunhofer Fokus, as the basis for the development and execution of services, and has built upon the facilities available in the OpenIMS testbed, while adding value to these through the development of enabling services. This paper describes the general architecture for both enabling and end user services developed during the project and shows how these have been used to create a useful location aware service which can help a user maximise use of public transport facilities in a city.

Keywords: IMS, OpenIMS, services, enablers, architecture

1 Introduction

The IMS ARCS research project [1] was conceived as a means of bringing together a group of academic institutions to work with a number of companies in the telecommunications field. The companies bring their commercial experience and market knowledge; the academic group bring their research and development skills and knowledge of leading edge technology and trends. Together, through discussions and the application of their respective expertise, they create innovative new product concepts, prototypes and processes for developing future applications. The output from the project, such as prototype code and documents, is freely available to all members of the project to develop and commercialise, and after a period of time will become open source and freely available to the general public.

The initial objective of the project focused on end-user and enabling services for IMS, by developing a number of end-user service concepts and analysing these under a number of headings, including value proposition, service functionality and implementation requirements. This was followed by the development of a number of working service prototypes which had been deployed on the OpenIMS [2] testbed. As the project evolved it became apparent that a further objective would be to create a general architecture for both enabling and end-user services.

2 Prior Work and State of the Art

The participants in the project had previously lead the Pervasive Services Management aspect of the IST Daidalos project in FP6 [3] [4]. This project had concentrated on developing a platform to support a set of context aware and personalised end-user services across a range of connectivity technologies. The IMS ARCS project aimed to build upon the experiences in this area, but focused on IMS as the platform of choice on which to create the services. The project also chose to aim for service development which was commercialisable in the near to medium term, so reliance on sophisticated network capabilities or demonstrating concepts that could not be reliably implemented was avoided. Instead, the emphasis is on value proposition, and practical use of technologies provided by IMS such as location awareness, presence, call control and rich internet.

As a key objective of the project was to develop a light weight service oriented architectural model for services and to achieve this using open source and freely available tools, attention was paid to the existing work of Fraunhofer Fokus in the area of Service Oriented Architecture and IMS, and to their portfolio of products in this area [5] [6]. However the IMS ARCS project aims to be of more general availability and for this reason has avoided the use of proprietary software. Instead it was focussing on SOA tools.

3 Technology Description

The IMS ARCS team analysed the features of a number of services, how the user would interact with them and how the services would interact with the underlying IMS network. The issue of what development tools and infrastructure could be used was also considered, based on considerable experience in the area of Web2.0 development. These included mobility, location awareness, user profile and authentication, authorisation and accounting (AAA). The services needed to provide a general "context awareness", where the service could react to changes in location, presence and profile.

As such the general approach adopted for the services was to follow a Web pattern where core functionality (backend) resides on an application server and the service is accessed via a web browser on the client. A rich interface for the service is enabled through the use of a Mashup AJAX framework. The application server backend interfaces with the IMS network and the various service enablers to deliver the functionality characterised by IMS.

A further focus of the project was to use open source and freely available software throughout development. This helps the output of the project to be easily distributed and reusable without users needing to incur commercial license costs. Additionally, the output of the project will itself eventually become open source in which case dependencies on commercial licenses would not be feasible.

Figure 1 provides a overview of the general architecture, as discussed in the following sections.

3.1 Enabling Services

An analysis of the functionality of the end-user service prototypes showed that many of the features and functionality were common across the various services. These included:

- User's location and presence,
- Other users' locations and presence,
- Rich presence (including device capabilities information),
- Service personalisation to a user's preferences,
- One to One SIP Call setup and Messaging,
- User Identity features and Anonymisation,
- Service login.

For the purpose of reuse, modularity and consistency, it made sense to isolate common functionality as *Enabling Services* which could then be used as the building blocks for the end-user services. The enabling services would also function as an abstraction of the underlying IMS network so that from an application developer perspective, they could interact with enablers without needing explicit knowledge of the IMS specific protocols such as SIP [7] or DIAMETER [8]. To provide such an abstraction, the enablers would need to support interfaces that would be familiar to a typical web developer, so it was decided that a web service interface based on the use of WSDL/SOAP would be supported by enabling services.

To have a cohesive and consistent architecture, both the end-user services and the enablers would all follow a similar architectural pattern, would communicate with each other in a similar manner, and – as far as possible – would use

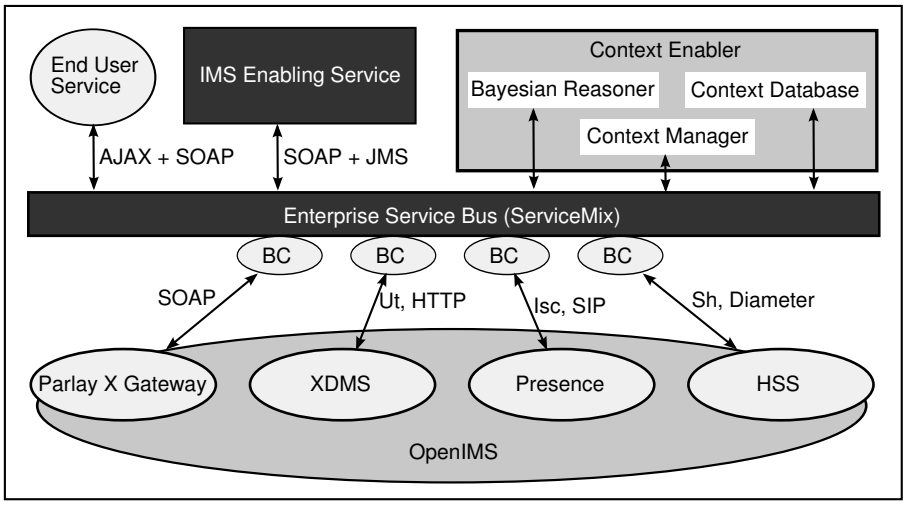

Fig. 1. Components of the general architecture

similar concepts and identifiers to interoperate. As such one of the first steps was to decide on a communications framework. As noted above the end-user service would communicate with the enablers via a web services framework. However the enablers would also need to communicate with each other and various communication patterns such as Query/Response and Publish/Subscribe needed to be allowed. In keeping with the use of enablers as service building blocks, a further requirement was to support a means of service composition and orchestration, using a language such as BPEL. To facilitate this, an Enterprise Service Bus was adopted as the middleware for the architecture.

The appropriate language for enabler development was judged to be Java, because of its ubiquity among developers, its rich development environment with a huge choice of tools, runtime libraries and infrastructure which would be suitable for the type of development and functionality. As such the ESB chosen was one that was based around the Java Business Integration (JBI) standard [9]. One of the most popular implementations of this standard is Apache ServiceMix [10]. This had the advantage of being widely used and supported across a number of different application servers including a number of open source ones. It also had the advantage of having a commercially supported variant, called FUSE ESB [11], which was an important consideration if an enabling or end-user service was to be commercialised. It is also closely integrated with other related tools such as Apache ActiveMQ and an Orchestration engine which plugs into ServiceMix, called Apache ODE (Orchestration Director Engine) and uses BPEL.

The project needed to select an application server to host the enabler components and also the end-user services. Because many of the enablers and components would use the SIP protocol for call setup, the application server needed to be SIP enabled. Glassfish/Sailfin [12] was chosen because of our preference for an open source and freely available option, without restrictions. An enabler can comprise of a number of sub-components that link together via the service bus. Some of these are written as service assemblies which run in the context of ServiceMix. Typically the binding components (BC) which implement the protocol translation follow this pattern. They are specific to the OpenIMS component that they interface with. Other sub-components within the enabler which encapsulate business logic may run within an application server and communicate via either SOAP or JMS with the other sub-components. However, where an enabler communicates with either the end-user services or with another enabling service, this communication takes place via the service bus using SOAP.

To allow the end-user services to avail of web technologies such as AJAX, HTTP requests can also be routed through the service bus to enabling services. The ESB also incorporates reusable Enterprise Integration Patterns (EIPs) which facilitate scalable message routing, for example Publish/Subscribe.

Information can typically be retrieved from the underlying IMS network using two different methods and a combination of these is typically implemented by the binding component. The first is where the components within the IMS network can be queried directly for information. The second approach is where asynchronous notifications are received when new information is created or changes

in existing information occurs. If the underlying component is capable of delivering asynchronous notifications then the binding component subscribes to these, however if such is not possible or unavailable then this can be achieved by polling the underlying components regularly, however this is less efficient.

3.2 Context Management Enabler

Much of the information being gathered from the IMS network refers to what is generally classed as "Context" information. This refers to current and historical information about users in the network, where they are, what they are doing, what types of devices they are using and their capabilities, in addition to any information that can be inferred from this [13]. As such it acts as an abstraction of the underlying IMS network as a source of information, while not changing the state of the network through proactive functionality such as management or call setup.

The enabler architecture described above has been used to implement a Context Enabler as part of the IMS ARCS project. The Context Enabler interfaces with the end-user service via a publish/subscribe interface. The end-user service subscribes for context information relating to a specific user or set of users. The criteria used are a user identifier(s), a filter expression and an identifier that will allow the Context Enabler send notifications to the service. The user identifier identifies the user whose context information (presence, location etc.) the service wishes to be informed of. The filter criterion is an XQuery statement, with an additional reference to the URI of an XML Schema file that describes the schema of the data on which the condition acts. This can be used to exclude data and events that the service is not interested in, or to place thresholds for changes. An example of this may be that the end-user service only wishes to know about changes in a user's location of more than 100 metres (rather than presence changes etc.). The notification that the Context Enabler sends to the end-user service will contain a URI reference to an XML document whose contents will contain the context information that the service subscribed for.

As shown in the above, the context enabler avails of the data collected from the OpenIMS and this is delivered via the service bus. The Context Manager subcomponent specifically manages the subscriptions from end-user services through monitoring which services have registered for information on users and the associated filter criteria. This component is then responsible for publishing the appropriate events to be delivered on the service bus. Where security considerations exist, these will also be managed by this component.

The context database manages the storage of the context events in their XML representation. The context database leverages prior work from the Daidalos project on the development of a spatially aware database [14]. It provides both the URIs for such events, acts as an event cache to store the stream of events coming from the network and allows queries to be created so that sets of events relating a user can be retrieved. This allows value added functionality, such as inferences, to be made.

The Bayesian Reasoner observes the user information in the database, and where an end-user service has registered an interest in a user's context, the reasoner can make inferences based on the context information. These inferences are stored as a reusable XML document in the database and an event is generated and published to the subscribed services. Because of the nature of Bayesian reasoning, all inferences will have an associated probability which qualifies the reliability of the inference. The services are free to use these inferences or discard them. An example of the type of inference that may be created may be based around the changes in a user location. If a user's location is changing frequently and such changes correspond with the coordinates of a road, then it is reasonable to infer that the user is travelling along that road in a particular direction. If the rate the user is travelling is greater than 40km/hour then it is reasonable to infer the person is travelling in a motorised vehicle. Depending on time and presence information, various other inferences may be made such as whether the person is driving or caught in traffic.

3.3 End-user Services

To demonstrate both enabling and end-user services, a number of end-user services have been developed and these services are integrated with the described context enabler and can be used to demonstrate the functionality that is offered by it. Some of these services have been demonstrated at public events such the Mobile World Congress 2008 in Barcelona. The end-user services have been designed in consultation with a project partner whose specialisation is Software Usability and Human Computer interactivity.

The Public Transport Advisor service is a typical example of a service offering a value proposition, using context information on the user and delivering a Web 2.0 type experience over an IMS network. The objective of the public transport advisor is to help a user to maximise their use of public transport infrastructure in a metropolitan area, particularly one which may be unfamiliar to them, where they are not aware of what services exist or what routes are available.

When a user invokes the service, the service backend subscribes to the context enabler for information on the users location. The user can select the destination. The service can then calculate optimal routes, plot a map, monitor the changing user location and notify the user about points to change transport or approaching the destination. This can be done via a number of methods, such as instant message or a voice call with instructions.

4 Conclusions

The project has been of significant benefit to a number of small and medium enterprises engaged in the telecommunications software industry in Europe and specifically in Ireland, in addition to a number of operators who are interested in rolling out sophisticated services over their next generation networks. The general architecture, toolset and exemplar services provide an ideal proof of

concept and starting point for companies interested in developing and deploying such services.

To support this, the project team has developed a "Jumpbox" DVD, which contains a VMWare image of a Linux machine with the OpenIMS, ServiceMix, Glassfish/Sailfin, Java, Maven and Subversion installation, together with the codebase of the various services and enablers that are used in the project.

5 Acknowledgement

The work described in this paper has been funded by Enterprise Ireland as an Industry Lead Research project, IMS ARCS, consisting of an academic consortium lead by TSSG and a number of companies.

References

1. IMS ARCS project. http://www.ims-arcs.com/ (2008)
2. Fraunhofer FOKUS OpenIMS Testbed. http://www.open-ims.org (2008)
3. Daidalos. http://www.ist-daidalos.org/ (2008)
4. Mullins, R., Mahon, F., Kuhmuench, C., Crotty, M., Mitic, J., Pfeifer, T.: Daidalos: A platform for facilitating pervasive services. In Pfeifer, T., et al., eds.: Pervasive 2006: 4th International Conference on Pervasive Computing, Advances in Pervasive Computing 2006. Adjunct Proceedings (Dublin, Ireland), Vienna, Austria, Austrian Computer Society (May 2006) 167–172
5. Blum, N., Magedanz, T., Schreiner, F.: Definition of a service delivery platform for service exposure and service orchestration in next generation networks. UbiCC Journal **3**(3)
6. N. Blum, T. Magedanz, P.W. In: The Integration of IMS into Service Delivery Platforms based on ServiceOriented Architectures. Taylor & Francis, New York (Nov 2008)
7. SIP: Session Initiation Protocol (RFC 3261). http://www.ietf.org/rfc/rfc3261.txt (2002)
8. Diameter Base Protocol (RFC 3588). http://www.ietf.org/rfc/rfc3588.txt (2003)
9. JSR 208 Java Business Integration. http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html (2008)
10. Apache ServiceMix. http://servicemix.apache.org (2008)
11. FUSE ESB. http://fusesource.com/products/enterprise-servicemix (2008)
12. SailFin Application Server. https://sailfin.dev.java.net (2008)
13. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In: HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing (Karlsruhe, Germany), Heidelberg, Germany, Springer (1999) 304–307
14. Pils, C., Roussaki, I., Strimpakou, M.: Distributed spatial database management for context aware computing systems. In: 16th ICT Mobile & Wireless Communications Summit (Budapest, Hungary), ICT (July 2-4 2007)