18th World IMACS / MODSIM Congress, Cairns, Australia 13-17 July 2009
http://mssanz.org.au/modsim09

Delivering Heterogeneous Hydrologic Data services with an Enterprise Service Bus Application

Bai, Q.F and Peter Fitch

CSIRO Land and Water, Black Mountain, Canberra, ACT, Australia

Qifeng.Bai@csiro.au

Abstract:

There is much interest in using emerging technologies such as web services for the discovery and delivery of hydrological data. In Australia currently there are two different approaches being trialed each with different content structure and semantics. These services use different schemas to represent the variety of hydrological data and also they use different queries to access the data. These complexities of data structure and semantics increase the difficulty of using these services. It is important for end users to have a middleware application that provides a standard interface to access these data services. However, traditional software architecture bundles data approaches and data format translators together to implements its business logic. Under this design, it is inevitable that software developers have to modify the source code if users need new data approaches or data format translators. This requires software developers to run an additional software development circle and it is also impossible to provide updates at runtime. Enterprise Service Bus (ESB) is a software architecture pattern that is commonly used to allow a client to easily use heterogeneous data sources. It uses a scalable infrastructure that connects disparate applications and resources, mediates their incompatibilities and connects their interactions. ESB does not provide any business logic itself but rely on the IT resources it is connecting.

This paper proposes a lightweight Enterprise Service Bus middleware framework that provides a stable, extendable and transparent data delivery service to readily access data services. The business logic is implemented by a data approach that accesses a certain hydrological data service and a data translator that converts the response to the desired format. The framework regards data approaches and data translators as discrete resources. The core function of the framework is to provide an interface to allow users to select these data approaches and data translators and intelligently route messages between the selected data approaches and data translators. In other words, the framework does not implement any business logic and it relies on injecting data approaches and data translators to realize business logic according to users' requirement. These data approaches and data translator are well encapsulated and they can be designed, coded, tested and deployed in an independent software development circle without interfering with the framework.

Keywords: *OGC, WFS, HIS, Heterogeneous, ESB, AOP, IoC, XML, Hydrological Information Systems.*

1. Introduction

With the use of enabling technologies such as web services to discover and deliver data to users and the multitude of possible standards and approaches for those services, a heterogeneous data service environment is emerging. In Australia, at least two hydrological web services are currently being evaluated: Australian Water Data Infrastructure (AWDI)[1] services which are based on Open Geospatial Consortium (OGC)[2] Web Feature Service (WFS)[3] and the Consortium for the advancement of Hydrological Science Initiative's (CUAHSI)[4] Hydrological Information System (HIS). Conceptually both services are similar in function and information to be transmitted, but differ substantially in their implementation. There are differences in the type of web service: Representational State Transfer (REST) for AWDI and Simple Object Access Protocol (SOAP) for HIS, as well as differences in the interface specification, query syntax and the data schemas used. In addition, the semantics of the data is also different and needs to be reconciled.

The data complexity and differences introduces difficulties to users to access and use these services. There are a number of traditional solutions to this problem which typically utilize the concept of data adapters in which a new software interface module is developed to take care of the differences. Whilst data adapters used widely suffer from the limitations such as the inability to add in new adapters without code recompilation and as well as propagation of errors to other parts of core system if the adapter is not designed properly.

This paper presents a solution that uses Service Oriented Architecture (SOA), Aspect-Oriented Programming (AOP) and the Inverse of Control (IOC) design pattern to build an Enterprise Service Bus (ESB) architecture middleware framework to access and homogenize heterogeneous data sources with standard SOAP interfaces. This platform is able to dynamically install, load and execute Service Agents (SA) that communicate with different data providers allowing users to flexibly configure the application to support their need to access from a variety of service types.

The remainder of the paper is structured as follows: Firstly an overview of ESB is presented, and followed by the introduction to our light weight ESB middle ware. We then explain the features and the key workflow with a case study using this application being presented. Lastly we make comments on future work, which is still required.

2. Overview of Solutions

Software applications typically support different data protocols and different outputs by designing data adapters for each service. A data adapter can further be thought of in two parts, a Service Client responsible for interacting with the application and a Data Translator to interpret and convert the messages to a format our application can understand and use. For example, if we need our application to access and use a HIS service the application will need a Service Client to access the HIS servers and a HIS Data Translator to convert the responses to a format the application can use.

2.1 Traditional Solution

Traditional software design (adapter pattern) uses "switch" statements to select the required Service Clients and Data Translators (or data adapter). In this case, adding new adapters requires the software to be rewritten. So, although, these modules have a certain level of abstraction and encapsulation, they cannot be easily changed during runtime. They cannot be designed and tested without modifying the application´s core code, which raises security and efficiency issues.

2.2 ESB Solution

An ESB is a scalable service based infrastructure that connects disparate applications and IT resources, mediates their incompatibilities, orchestrates their interactions, and makes them broadly available as services on a "service bus" (Chappel, 2004). The basic function of an ESB is to broke communication

[1] The Australian Water Data Infrastructure Project is to facilitate Australia-wide assessments of water resources through ongoing development of a comprehensive and accessible water information framework.
[2] Open Geospatial Consortium is to develop standards for geospatial and location based services.
[3] The OpenGIS Web Feature Service Interface Standard (WFS) defines an interface for specifying requests for retrieving geographic features across the Web using platform-independent calls
[4] The CUAHSI HIS is a geographically distributed network of hydrologic data sources and functions that are integrated using web services

and send data between processes on the same or different computers. The ESB is capable of locating and assembling the connection and intermediating between the sender and the receivers (Louis, May 23, 2008)

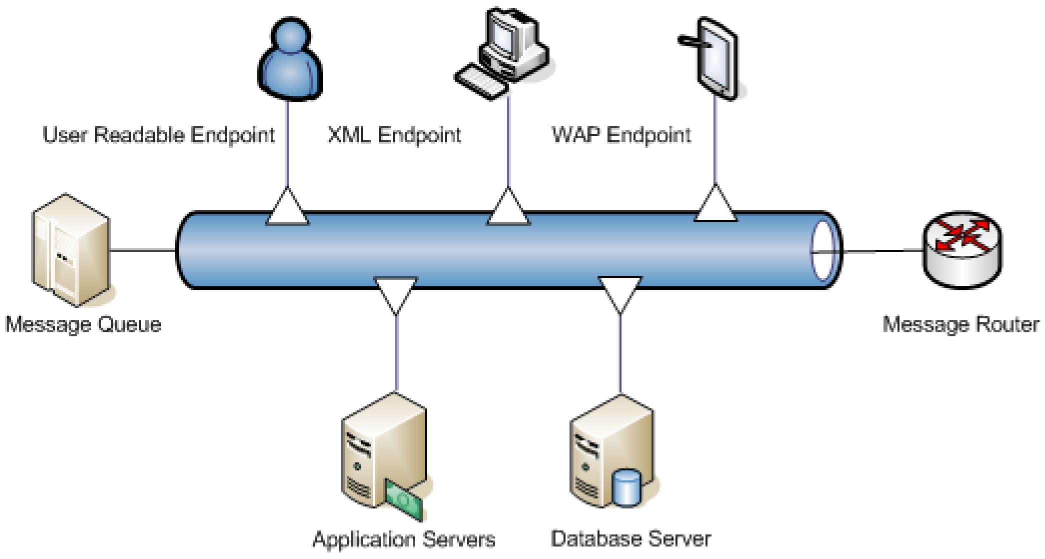

Figure 1. Enterprise Service Bus Architecture Pattern

With reference to Figure 1, in an ESB, end point applications (clients) communicate with each other via the ”bus”, which acts as a message broker between services. The bus also has a message queue to manage the message flow, and a message router, which can be configured, based on a workflow configuration. The primary advantage of this approach is reduction of the number of different interactions between endpoints. That is, a client only needs to be able to communicate with the “bus” to use all the services available on the bus. Furthermore the client can compose a workflow as a sequence of services by using the massage router to pass the messages from one service to another as a sequence of services. By reducing the number of points-of-contact to a particular application, the process of adapting the system to changes becomes easier.

3. Light-Weight ESB Data Middle Ware

In this paper, we present a solution that is a “light-weight ESB” platform using Service-Oriented Architecture, Aspect-Oriented Programming (AOP), and Inverse of Control (IoC) conceptions. By light-weight we mean that this platform focuses on dealing with hydrologic information and uses an internal messaging strategy. This implementation uses Spring.Net (Craig & Breidenbach, 2007) to provide AOP and inversion of control. The solution provides intelligent routing management to enable end users to assemble service agents with different data translators at runtime. Also it makes it possible to decouple the development and deployment of data modules from the logging, authentication and management functionality of the architecture, which are essential for maintaining system robustness.

3.1 Software Architecture

Our ESB services contains following components: Data Module which contains a Service Agent and Translator, Content-Based Routing and User Defined Mediation.

3.1.1 Data modules

A data module contains a Service Agent that is responsible for communicating with the data service provider and multiple data translators, which convert the data to the different message types.

The functionality that an ESB system implements depends on the endpoints of data modules injected through Inversion of Control entity. The distinguishing feature of IoC is in the difference in which dynamically loaded software modules interact with the application. In traditional software the client is responsible for calling the loaded software modules (dynamically linked library for example), whereas in an IoC framework the loaded module is the one which calls the application-specific client code (Fowler, 2005). IoC makes it possible to link new Service Agents without modifying source code. In other words the system can be dynamically configured at runtime depending on what functionality is required (i.e. what data services need to be accessed).

3.1.2 Content-Based Routing Service.

A Content-Based Routing Service directs messages to various other service endpoints according to configured routing rules. The routing service is XML based and usually there are starting and finishing endpoints defined.

The following XML configuration and source code demonstrates how to link a service agent to a translator by modifying route table instead of changing code.

In Figure 3, the "Service Agent" element in the XML configuration points to a HIS server and tells the router to initiate the class WQSAR.Mediator.HIS.Client with the URL http://wron.net.au/CsiroHIS/cuahsi_1_0.asmx. The "Parser" element defines the output endpoint that transforms the data from HIS. Here, HIS.WQSARParser is used to translate HIS responses to the format required by Water Quality Statistical Analysis and Reporting Tool project (WQSAR)[5].

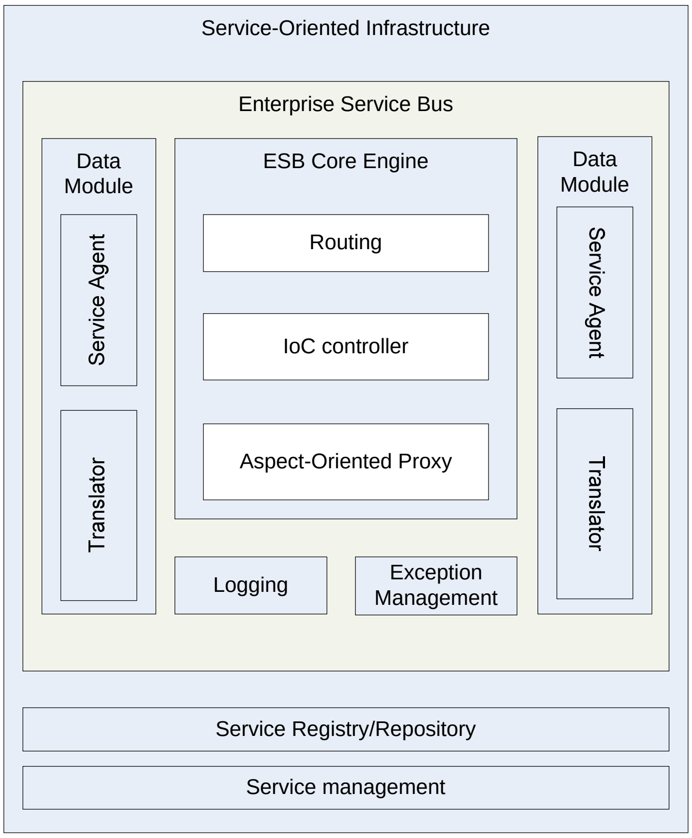

Figure 2. Structure of Light-Weight ESB Date Middle Ware

3.1.3 User-Defined Mediation

A User-Defined Mediation Service implements functions such as logging and exception management. Current Object Oriented Programming (OOP) approaches support some level of abstraction and encapsulation of concerns into independent entities like Service Agent and Content-Based Routing Agent. However, some functionality like logging, authentication and exception management need to be capable of penetrating other encapsulated functionality. Aspect-oriented programming (AOP) aims to increase modularity of a software application by allowing the separation of cross-cutting concerns (Kiczales & John Lamping, 1997). By using AOP architecture, these logging and exception management components are able to traverse all business logic (data modules) providing an elegant solution to these cross cutting concerns.

3.2 Work Flow

The following diagram shows the internal work flow of ESB:

[5] The Water Quality Statistical Analysis and Reporting Tool project aims to track water quality trends and get reports on these for freshwater and estuarine systems all over Australia

In Figure 13, the Service Registry (SR) manages the information about the servers and their types; it manages Data Modules, e.g. HIS and WFS Service Client and their Translators. Once the user chooses a server, the system can initiate a corresponding Service Client by searching SR. Also, SR lists all possible Translators that this Service Client owns and allows the user to select the one producing the required output format. This information is passed to the Connection Factory to generate an XML routing table. The routing table consists mainly of three parts:

1. **Proxy Interface**: it defines what types of classes could be used, and more important, it tells the mediator only these types defined in this area can be traversed.
2. **Target Objects**: it defines which Service Client and Translator are used in routing agent
3. **Interceptor**: it defines which classes are used to traverse the classes defined in Proxy Interface.

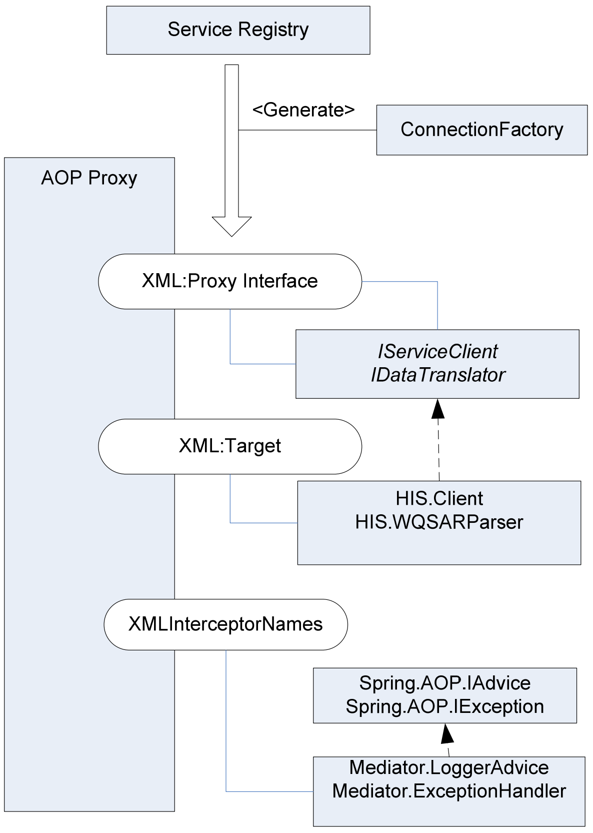

Figure 3. Core working flow

In Figure 3, we have shown how a routing agent assembled starting (Service Agent) and ending (Translator) endpoints. Figure 4 shows that IServiceClient and IDataTranslator are defined in the Proxy Interface section, which means instances of these interfaces can be traversed by interceptors in InterceptorNames section, like LoggerAdvice and ExceptionHandler etc. Now, we have completed a whole ESB lifecycle that shows the process that how data is passed from a starting point to an ending point and how ESB monitors and manages this process.

The pseudo code in Figure 4 shows how to execute a Service Agent. First step creates an instance of IServiceClient based on the method we discussed in Figure 3. Then, interceptors including LoggerAdvice and ExceptionHandlerChain are injected into the ProxyFactory. Before a method of ServiceClient is called, LoggerAdvice.Before will be executed first, and then command.Execute will call the real method in the ServiceClient by reflection. Exceptions will be captured and dealt with by ExceptionHandler.

```
IServiceClient client =
(IServiceClient)SpringContext.Context.GetObject("ServiceAgent");
factory = new ProxyFactory(client);
factory.AddAdvice(new LoggerAdvice());
factory.AddAdvice(new ExceptionHandlerChain());
ICommand command = (ICommand)factory.GetProxy();

command.Execute(LoggerAdvice.Before());
try{
    command.Execute(); // Call the real methods by reflection.
}catch
   { command.Execute(ExceptionHandlerChain.AfterThrowing());}
command.Execute(Logger.Advice.After());
```

Figure 4. Pseudo code of executing process

From the above it can be seen how the framework easily, dynamically configures the ServiceAgents for use as well as for exception handling and management. And these functionalities do not require any code modification, but only different configuration files

4. Case Study

In this case study we are applying our framework to The Water Quality Statistical Analysis and Reporting Tool project (WQSAR) which requires data from two different data service types to feed a statistical web service which then presents results to the users. WQSAR project aims to provide a web based tool, which allows users to access data made available on the web as part of the AWDI and apply best practice statistical methods to that data to assess long term trends in water quality. In addition to OGC WFS servers which are being deployed nationally, WQSAR is interested in testing CUAHSI HIS which is being trialed by the Bureau of Meteorology.

The approach outlined in this paper has been configured and deployed for the WQSAR project. The ESB middleware framework was deployed as GeoDataService server, which currently has HIS and WFS data modules. GeoDataService Server provides standard queries for GetSites (return a list of water quality monitoring stations), GetVariables (return a list of water quality variables at that station), GetValues (return data for a station, for a water quality phenomenon). Users can retrieve data from any of the data sources the system supports and route that data to the statistical services. Also users can register new servers for configured data modules or choose any of the registered data sources completely oblivious to the underlying service type. The framework was found to provide a stable and compatible data service for the case study WQSAR project with the benefits of easy reconfiguration for ne[6]w data sources as they become available.

5. Conclusion

The ESB architecture middleware framework overcomes the shortcoming of traditional software architecture that cannot adopt new data modules without recompilations due to entire couple of data modules and core system. It has proved good at dealing with heterogeneous data schemas; especially through its ability to add new data modules without modifying the framework application. Moreover, this framework has a strong error controlling ability and management that prevents deficiency of data modules to affect the system.

The ESB architecture used in the framework has three key features that successfully help to solve these problems that traditional software architecture brings.

1. This core framework does not provide any business logic functionality and relies on injected data modules.
2. An intelligent Routing Manager is in charge of assembling and delivering messages between Service Client and Data Translators in the Data Module, which allows data consumers to get different output formats easily.
3. More importantly, the framework provides ability for installing, running, stopping and updating - the whole life cycle of data modules.

6. Future Works

At the current stage, there is only one output format designed for WQSAR project. We have a Data translator (parser) for each input source (IServiceClient) and messages from IServcieClient which are passed to the corresponding parser with a static route. In future, additional Data Translators can be developed for each Service Client to support others requirements. And later on, this framework can be extended to be used with Microsoft Message Queue Server (MSMQ) or similar to build a Subscribe/Publish system to publish raw messages to multiple output endpoints.

ACKNOWLEDGEMENTS

Thanks to Brent Henderson, Mengxiang Zhen, Ian Szarka and Robert Bridgart who generously provided extensive comments on the paper. The Water Quality Statistical Analysis and Reporting Tool project is funded by the Australian Government through the National Heritage Trust / Caring for our Country, the Bureau of Meteorology, and CSIRO's Water for a Healthy Country Flagship.

REFERENCE

Chappel, D. (2004). *Enterprise Service Bus.* O'Reilly.

Craig, W., & Breidenbach, R. (2007). *Spring in action.* Greenwich, CT. USA: Manning Publications Co.

Fowler, M. (2005). *Inversion of Control Containers and the Dependency Injection pattern.* http://www.itu.dk/courses/VOP/E2006/8_injection.pdf: IT University of Copenhagen.

Kiczales, G., & John Lamping, A. M.-M. (1997). Aspect-Oriented Programming. *Proceedings of the European Conference on Object-Oriented Programming*, (pp. 220-242).

Louis, A. (May 23, 2008). ESB Topology Alternatives. *InfoQ* .