Using a Transaction Definition Language for the Automated ARMing of Web Services

J.D. Turner, D. Bacigalupo, S.A. Jarvis, G.R. Nudd
High Performance Systems Group, University of Warwick, Coventry, UK
{jdt,daveb,saj,grn}@dcs.warwick.ac.uk

Abstract

Web service technology will provide a platform for next generation dynamic e-business applications. This paper describes a framework for identifying, monitoring and reporting performance data of critical transactions within a web service using the Java ARM standard, a Transaction Definition Language (TDL) and an Automated Instrumentation technique. Gourmet2Go, a demonstrator provided with the IBM web services toolkit, is used as a case study.

Keywords: *Web Services, ARM, Performance Measurement, Service Routing, Java*

1 Introduction

While high performance computing has been based on the culmination of massively parallel multi-node MIMD machines for a number of years, the vast increase in network bandwidth and desktop computer performance has meant that disparate heterogeneous resources executing highly scalable distributed applications have become the preferred infrastructure for high performance computing. This interest in geographically dispersed resource-sharing networks spread across multiple administrative domains has stimulated the massive demand for both academic and industrial research into a number of standards for the creation of a Grid[10, 17] infrastructure (for example Globus[11]), and for high performance resource allocation services with the ability to adapt to the continuous variations in user demands and resource availability.

More recently it has been proposed that Grids should provide an infrastructure that would provide the resources necessary for both e-science *and* e-business applications[12] to execute efficiently whilst meeting quality of service (QoS) demands and requirements. However, the different characteristics of e-science applications (such as large 'run-once' jobs) and e-business applications (such as high-frequency request-driven services) necessitate different methodologies for each; for example the use of resource allocation for large jobs, as opposed to a service routing approach for high frequency requests.

1.1 Performance-Driven Grid Scheduling

TITAN[22], a distributed resource scheduler, is being developed to provide efficient resource allocation of distributed, scientific applications on Grid architectures. An overview of TITAN can be seen in Figure 1. TITAN currently uses iterative heuristic algorithms to minimize a number of scheduling metrics including makespan and idle time, while aiming to meet a number of QoS requirements; dead-line time for example. An agent-based resource discovery and advertisement framework A4[4, 5] is used by TITAN to gain access to and maintain a knowledge of the available resources within its administrative domain. This work is differentiated from other Grid scheduling research by the use of PACE[3, 18], a performance prediction environment that allows TITAN to predict the communication and computational needs of scheduled applications on the available resources. PACE abstracts an application into a single performance model that provides accurate predictions based on input parameters (such as the number of processors) and a hardware model (provided from a benchmark of the resource upon which the application will be executed). This allows the current schedule to be modified continuously according to these predictions of scheduled applications and therefore allowing the scheduling metrics to be minimised while achieving the highest efficiency possible.

1.2 Performance-Driven Web Service Routing

While TITAN is well-suited to the resource allocation of large scientific applications, a different methodology is required for

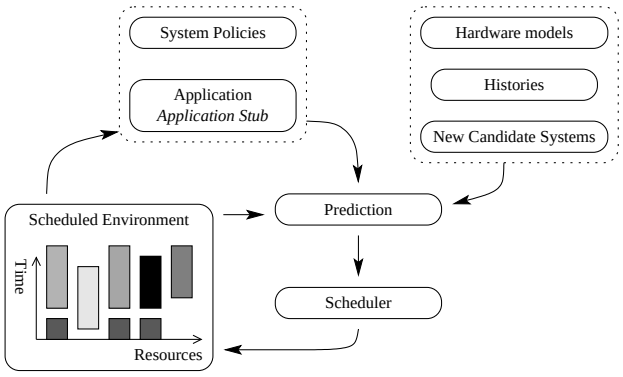

Figure 1: TITAN: a metric-based scheduling environment incorporating the performance prediction of applications, resource allocation, system policies, performance histories and QoS. Feedback of information during the application's execution aids the prediction and scheduling of similar applications in the future.

e-business applications. Business to business (B2B) and business to consumer (B2C) applications are often characterised as long-running applications that service client requests. These connections tend to have a short execution time but have the potential to be very high-frequency. The web services framework, which is currently being standardised with a strong but not exclusive focus on e-business applications, will provide a platform for these dynamic, distributed applications.

In a similar way that Grid applications require resource allocation, it is necessary to make web service selection decisions. This normally involves routing the communication from a service requester through several intermediaries to the ultimate service provider. To achieve this, a service routing mechanism is necessary.

This paper describes a framework for the monitoring of performance-critical transactions within a web services environment with a focus on providing performance data that can be used in making service routing decisions. As such, the average end-to-end response time for a web service is the primary QoS metric. The Application Response Measurement[15, 20] (ARM) standard is used for the measurement of defined performance-critical transactions, and a Transaction Definition Language[24] (TDL) is used to instrument application bytecode so as to appropriately process these transactions. Results including a set of end-to-end response times of performance-critical transactions within Gourmet2Go, a demonstrator from the IBM Web Services Toolkit[14], are shown, with emphasis on how these results may be used by a service routing algorithm.

The remaining sections of this paper describe:

- An introduction to the Web Services framework (see Section 2).
- An overview of ARM and an ARM compliant XML-based Transaction Definition Language, allowing the efficient 'ARMing' of Java applications (see Section 3).
- The demonstration of using a TDL as part of a performance monitoring framework within an example Web Services application (see Section 4).

2 Web Services

The recent interest in web service technology parallels the increase in distributed infrastructures and their associated application domains. The Web, for example, is known to facilitate human-to-application interaction, but currently provides limited support for fully automated and dynamic application-to-application interactions. Distributed enterprise computing platforms and their associated component technologies, including Enterprise Java Beans (EJB), COM+ and CORBA, facilitate application-to-application integration, but are hard to apply across organisational boundaries because of the lack of a standardised infrastructure.

One approach to facilitating application-to-application interactions across distributed infrastructures is the creation of a standard framework that could be used to create, describe and publish network accessible services (including programs, databases, networks, computational or storage resources etc. [12]). It has been the emergence of e-business however that has led to the increase in the industry-wide effort to create a standard web services framework, the primary goals of which are described as [7]:

- systematic application-to-application interactions on the Web;
- the smooth integration of existing infrastructure;
- the support for efficient application integration.

While the above goals are based on a web services framework that is based on existing existing web protocols (including HTTP, for example), web services are likely to function in other environments including private wide-area and local-area networks. The open, heterogeneous and dynamic nature of the Web leads to a set of key requirements for a web-based distributed computing model [8]:

- *A small set of supported standard protocols and data formats for each computing node.* Eg., a web services communications protocol and a web services description language are all that is required for basic interactions.

- *A loose coupling between web services.* Eg., allowing web services to switch between sub-services which provide the same functionality.
- *Interfaces defined in terms of the messages that they process.* Involving a shift in focus from the APIs supported by each platform to what goes 'on the wire'.
- *Web services which are able to integrate with each other on an ad-hoc basis.* Eg., e-business web services that are able to establish short-term goal-directed relationships.
- *Web services located 'by-what' and 'by-how'.* Eg., searching for web services that provide particular functionality (from a given community) with a predefined QoS.

These requirements could be realised by a framework in which a wide variety of functionalities are encapsulated as web services and run on any computing node that meets a minimal set of requirements. These web services will be able to 'dynamically bind' to each other at runtime, which might involve a web service searching for other web services based on their functional and non-functional criteria. These services would interact through the exchange of 'messages on the wire' and would enable future applications to be constructed by dynamically orchestrating collections of contributing services.

2.1 Standardising the Web Services Framework

One of the early attempts at web services standardisation began with the release of IBM and Microsoft's web service specifications for B2B e-business. The specifications, built on existing industry standards, were based on XML and a simple web services model (see Figure 2 [16]) and contained two core initiatives [7]:

1. *A data exchange protocol* called SOAP [2], for sending XML messages wrapped in a standard 'header and body'-based XML envelope over an arbitrary transport protocol. SOAP includes a binding for HTTP and a convention for Remote Procedure Call (RPC). Potential extensions include a standard routing mechanism and a unified transaction model [19].
2. *A unified representation for services* called WSDL [6], providing an XML-based language for describing web services. An abstract description, for example the XML messages sent and received by the web services, is mapped onto one or more communication protocol bindings. An end-point address is added to the protocol bindings to define a service end-point or 'port', and an application wishing to use the web service chooses which of the available protocols to use. Despite a focus on e-business, WSDL can be seen as implicitly providing the basis of a distributed web services component interaction model [8]. It is likely that other specifications will build on WSDL to enable a more complete description of both web service components (for example non-functional aspects of the service such as QoS) and the different types of interaction that they support (for example synchronous/asynchronous and point-to-point/broadcast).

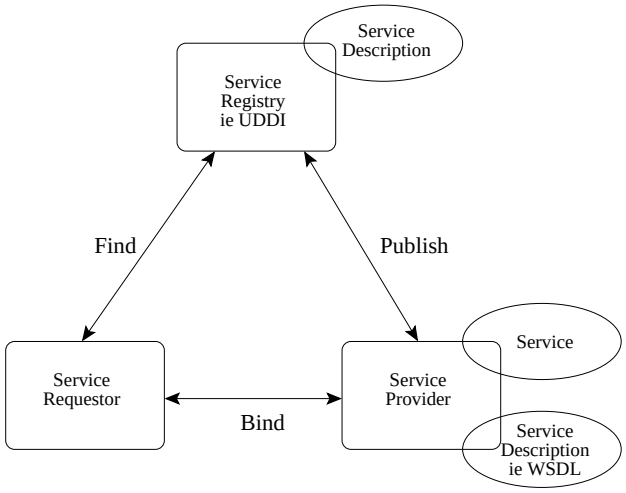

Figure 2: An overview of the web services model on which the web services framework is based

More recently web service invocation frameworks such as WSIF [9] have been proposed. These should simplify the interaction with WSDL-described services, for example, an application might only be required to specify the WSDL file of a web service to interact with it at the XML exchange level. Remaining interactions, such as selecting the protocol binding, could then be handled by the invocation framework.

SOAP and WSDL have now become de-facto industry standards[1]. Since their release, a number of other specifications have been proposed:

- *Service discovery:* Service registries such as UDDI [25] allow web service descriptions to be published. These registries can be searched at develop-time and at run-time; for example, web services can search and 'dynamically bind' to other services. Services can also be published using inspection languages such as WS-Inspection [1], where service discovery is performed through the inspection of service sites.

[1] and have been submitted to the W3C as part of the recent web services activity [26]

- *Composition:* A major advantage of web services is their connectivity potential and the ability to form complex 'compositions' at run-time. Web service composition languages include IBM's WSFL [13] and Microsoft's XLang [23].

2.2 Implementing Web Services

There are currently two main implementation platforms for web services, Microsoft's .NET and J2EE. Both platforms use SOAP, WSDL and UDDI to promote basic interoperability, but additional non-standard functionality is also provided. Supporting web-server application products are currently being implemented and include IBM's Websphere.

In addition to the WebSphere e-business platform, IBM has also published their J2EE-based Web Services Toolkit (WSTK) [14]. This provides a variety of APIs, a pre-installed run-time environment, an implementation of a private UDDI registry and a number of demonstrators. One of these demonstrators, Gourmet2Go, forms the basis of the case study found in Section 4.

2.3 Service Routing Intermediaries

The web services framework enables different versions of a conceptual service to be published. The communication between service requester and a service provider can traverse several service selection levels:

1. *Service provider selection:* A standard interface for providing a particular product or service which is implemented (or extended) by competing companies.
2. *Service offering selection:* The service provider offers a web service with various QoS characteristics and associated prices.
3. *Service replica selection:* The web service is replicated on different servers and a server that can provide the agreed QoS level is selected.

It is likely that the communication will pass through a number of intermediaries that will contribute toward service selection decisions and hence perform a service routing role; part of the work of the W3C XML protocol working group is to consider the specification of such intermediaries. Web service routing (or *message exchange*) is the subject of significant attention and has been identified as one of the key functional components of the web services framework. One proposed routing standard is Microsoft's SOAP-based WS-Routing.

An example of an intermediary might be a broker that helps a shopper select and then connect to a company to obtain a product or service based on criteria such as price, quality and the performance of the company's web service. Such brokers are likely to make their selection decisions based on the characteristics and capabilities of the potential services, using parameters set by the requester and/or provider.

An important non-functional criteria is the performance of the web services being considered for selection. This requires mechanisms for measuring, storing and utilising web service performance data. An approach to web service performance monitoring using the ARM standard is discussed in the next section.

3 Application Response Measurement

With the increasingly complex computer infrastructures present within the industry today, it has become more difficult to analyse the performance of modern e-business and e-science applications. Such applications are commonly distributed across multiple administrative domains, systems, processes and threads, and can be broken down into individual units of work defined as '*transactions*'. Transactions, or sub-transactions within a transaction, may include data-base access, application logic and data input and/or presentation.

Analyzing the performance of such a distributed architecture can be problematic and application administrators are faced with a number of difficult questions:

- Are transactions succeeding?
- If a transaction fails, what is the cause of the failure?
- What is the response time experienced by the end user?
- Which sub-transactions are taking too long?
- Where are the bottlenecks?
- How many of which transactions are being used?
- How can the application and environment be tuned to be more robust and to perform better?

The Application Response Measurement (ARM) standard allows these questions to be answered. ARM allows the application developer to define a number of transactions within an application, the performance of which are then measured during the application's execution by an ARM consumer interface, allowing the developer or user to analyse these results and provide answers to the questions above. Which transactions and the performance metrics that are monitored is the developer's decision, although they are normally the areas of an application that are considered performance critical.

3.1 Java 3.0 Standard

In October 2001, the ARM 3.0 specification was accepted by the Open Group[21]. This specification provides a Java binding for the response measurement of Java applications, defining a number of Java interfaces which are implemented by every ARM consumer interface. This standard interface allows 'ARMed' Java applications to utilize all consumer interfaces which conform to the appropriate version of the standard.

An example of an 'ARMed' Java application, and the necessary method calls to a consumer interface to define a single transaction can be seen in Figure 3. Each transaction is defined by creating a new instantiation of an 'ArmTransaction' object via the 'ArmTransactionFactory' implementation of the consumer interface, and then invoking 'start' and 'stop' calls on that instantiation at the beginning and end of the defined transaction respectively. The consumer interface measures the performance of this transaction and reports it to the data repository; shown in Figure 3 by the reporting classes.

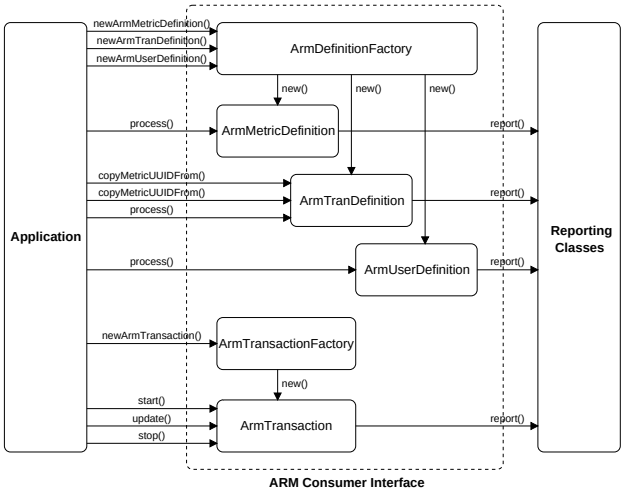

Figure 3: An example of an 'ARMed' application processing descriptive information and defining the location of a transaction via communication with an ARM consumer interface. All ARM transactions have (optional) definition information associated with them via 'ArmMetricDefinition', 'ArmTranDefinition', and 'ArmUserDefinition' objects. Each of these objects are populated with information describing the transaction; when the application invokes a process call, the consumer interface reports the data to the reporting classes. The application then creates a new instantiation of an ArmTransaction object, and defines the location of the transaction by invoking the start and stop transaction calls.

Optional descriptive information may also be associated with the transaction to aid analysis. This data comes in the form of definition objects such as 'ArmMetricDefinition', 'ArmTranDefinition' and 'ArmUserDefinition' as seen in the diagram. Instantiations of these objects are provided by the consumer interface's implementation of the 'ArmDefinitionFactory', which are then populated with descriptive information by the application and processed whenever possible. This descriptive information is associated with the transaction by their individual UUID numbers, allowing a more meaningful description each transaction within the repository.

3.2 An ARM Implementation

An ARM consumer and data repository implementation has been developed to allow distributed 'ARMed' applications to process and report transactions to a remote data repository service. The ARM consumer interface connects to this data repository service, and reports all processed transaction and descriptive measurement data. The service gathers this information and stores it in an efficient manner for future use. A number of standard clients have also been implemented that allow data to be extracted from the repository, including a GUI client so a developer can monitor currently executing and previously executed transactions. This ARM data repository implementation infrastructure is shown in Figure 4.

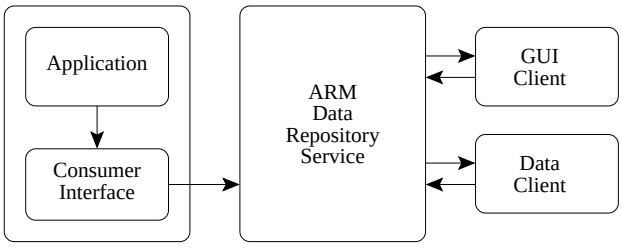

Figure 4: An ARM implementation model containing: an ARM consumer interface client residing in the same JVM as the original application; an ARM data repository server for all reported and processed transactions and their respective descriptive information; a number of server clients for data retrieval, including a GUI client for a continuous update of all previously reported and currently executing transactions; a data client for specific transaction queries.

In the current implementation one central ARM repository service is defined. Scalability limitations are likely to be addressed in the future by associating local data repository services with each 'group' of web services to be monitored; possibly connected in a hierarchy.

3.3 Transaction Definition Language

While the ARM standard provides a strict framework for providing application response measurement of distributed applications, there are still inherent disadvantages in this approach; these include the need to have a detailed knowledge of the ARM specification, the time necessary to instrument all

performance-critical transactions within classes of an application, and the fact that propriotry source code may not be available to instrument.

To overcome these issues, a Transaction Definition Language (TDL)[24] has been developed. Developers define where transactions are located within their application using an XML syntax, and the object code of the application is automatically instrumented with the use of a bytecode instrumentation tool to conform to both the ARM and Java Virtual Machine specifications. Within the TDL, descriptive information can be associated with each transaction using optional attributes, and groups of transactions can be easily defined within individual or groups of classes in an application's jarfile or classpath. The current version of the TDL specification (version 1.1), provided as an XML DTD, can be seen in Figure 5.

```
<!-- Transaction Definition Language DTD (v1.1) -->

<!ELEMENT tdl (transaction+)>

<!ELEMENT transaction (location, line_number?, metric*)>
<!ATTLIST transaction type (method_source | method_call |
                            line_number) #REQUIRED
                      appl_name CDATA #IMPLIED
                      tran_name CDATA #IMPLIED
                      user_name CDATA #IMPLIED
                      fail_on_exception (yes | no) "yes">

<!ELEMENT location EMPTY>
<!ATTLIST location class CDATA #REQUIRED
                   method CDATA #REQUIRED>

<!ELEMENT line_number EMPTY>
<!ATTLIST line_number begin CDATA #REQUIRED
                      end CDATA #REQUIRED>

<!ELEMENT metric EMPTY>
<!ATTLIST metric type (Counter32 | Counter64 |
                       CounterFloat32 | Guage32 |
                       Guage64 | GuageFloat32 |
                       NumericId32 | NumericId64 |
                       String8 | String32) #REQUIRED
                value CDATA #REQUIRED>
```

Figure 5: Transaction Definition Language DTD (v1.1).

The advantage of this approach is that developers can use the TDL to ARM performance-critical areas of code by editing the associated XML file. The process of code instrumentation is then automated (saving the developer time) and bytecode-based (eliminating the need for proprietary source code). Applications can then be monitored during the process of development, or alternatively after they have been deployed into a live environment.

4 Case Study

Service routing intermediaries, as discussed in Section 2, are likely to play an important role in future web service networks. Such intermediaries will need to consider QoS metrics when making service selection decisions; the end-to-end response time that a service can provide being one example. For this to be possible, there must be a mechanism for monitoring the performance of the web services, and using this data to make a decision.

This section provides a case study demonstrating the use of ARM and the TDL to monitor such QoS metrics within a web services environment. Once performance-critical transactions are defined within the environment, the TDL is used to 'ARM' each transaction and therefore monitor each and every transaction during the environment's life-cycle. This historical performance information can then be used for a number of reasons, including the basis for an intermediary's service selection algorithm.

Gourmet2Go, a demonstrator provided with the IBM Web Services Toolkit, is provided as an example of such a web services environment. Gourmet2Go's design, an overview of the performance-critical transactions within Gourmet2Go, and the response times of these transactions during a test-run as measured by ARM, are documented within the following text.

4.1 Gourmet2Go

Gourmet2Go is a Web services demonstrator from the IBM Web Services Toolkit. The demonstrator provides an example of how a Web service acts as a intermediary 'broker' that helps users select a 'back-end' Web service by obtaining bids from Web services published in a UDDI registry. In Gourmet2Go, the back-end Web services sell groceries and the broker presents itself as a value-added 'meal planning' service. However, the underlying architecture is generic and is used to demonstrate the brokering of any kind of service.

The architecture of the Gourmet2Go demonstrator is illustrated in Figure 6. A typical interaction is as follows:

1. The user interacts with the Gourmet2Go Web application via a Web browser with the intention of building a shopping list of groceries. This represents the user specifying the service that the back-end web service must perform.

2. The broker searches the registry for businesses with published web services that sell groceries. This represents the broker selecting a number of potential services using information provided in the registry.

3. The shopping list is passed by the broker to each of the back-end web services, as located from the registry, via a '**getBid**' request. This represents the broker contacting the 'shortlist' of candidates directly.

4. The broker summarises the bids for the user based on price, and the user then selects a supplier using the in-

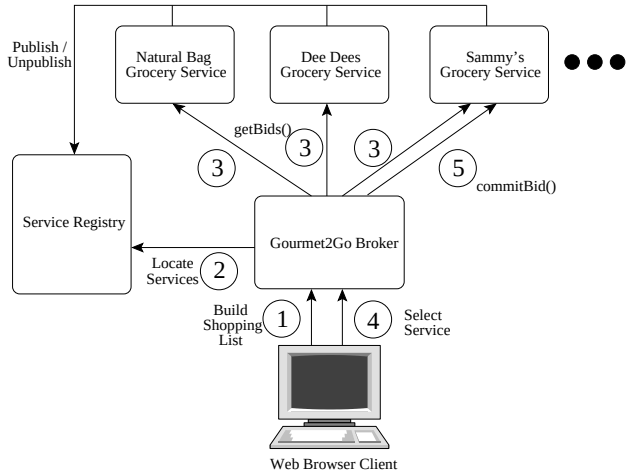

Figure 6: The design of the Gourmet2Go demonstrator from the IBM Web Services Toolkit, showing a typical interaction in which a user browses the Gourmet2Go broker to obtain information regarding the available services, and then select their preference.

formation presented to them. This stage represents the broker assisting the user in decision making.

5. The broker sends a '**commitBid**' request to the service that the users selects. This stage represents the broker continuing to act as an intermediary whilst the user interacts with the selected back-end service. This interaction has the potential to be significantly more complex than the confirmation message in the Gourmet2Go demonstrator. For example, specifying the details of what exactly is to be purchased and how it is to be paid for, through to delivery tracking and after-sales support.

It should be noted that the Gourmet2Go application also acts as a broker for delivery services and the broker itself is constructed from a number of dynamically registered services. For example, the meal planning 'offering' service provides the broker functionality that enables the user to build up a shopping list of groceries. However, we do not concern ourselves with these details except to note how Gourmet2Go is constructed by dynamically 'orchestrating' a set of services that are published and unpublished whilst it is running.

The Web services in Gourmet2Go are implemented in Java using the WSTK and associated standards.

4.2 ARMing Gourmet2Go

Within the Gourmet2Go design, the most performance-significant transactions are the requests from the broker to the back-end services, and in-particular the 'commitBid' request. It is the historical performance of these transactions that would be required by a service routing algorithm as the basis for its QoS decision making. To implement this, two methods within each grocery service are monitored using ARM so as to measure this performance-critical information during each user iteration.

To ARM these two methods within the available three back-

```
<?xml version="1.0"?>
<!DOCTYPE tdl SYSTEM "tdl.dtd">

<tdl>

  <transaction type="method_source"
               appl_name="Dee Dees Grocery Ordering Service"
               tran_name="getBid_backend"
               user_name="ARM">
    <location class="com/ibm/ews/g2gServices/
                     grocery/DeeDeesGroceryService"
              method="getBid"/>
  </transaction>

  <transaction type="method_source"
               appl_name="Dee Dees Grocery Ordering Service"
               tran_name="commitBid_backend"
               user_name="ARM">
    <location class="com/ibm/ews/g2gServices/
                     grocery/DeeDeesGroceryService"
              method="commitBid"/>
  </transaction>

  <transaction type="method_source"
               appl_name="Natural Bag Grocery Service"
               tran_name="getBid_backend"
               user_name="ARM">
    <location class="com/ibm/ews/g2gServices/
                     grocery/NaturalBagGroceryService"
              method="getBid"/>
  </transaction>

  <transaction type="method_source"
               appl_name="Natural Bag Grocery Service"
               tran_name="commitBid_backend"
               user_name="ARM">
    <location class="com/ibm/ews/g2gServices/
                     grocery/NaturalBagGroceryService"
              method="commitBid"/>
  </transaction>

  <transaction type="method_source"
               appl_name="Sammys Grocery Ordering Service"
               tran_name="getBid_backend"
               user_name="ARM">
    <location class="com/ibm/ews/g2gServices/
                     grocery/SammysGroceryService"
              method="getBid"/>
  </transaction>

  <transaction type="method_source"
               appl_name="Sammys Grocery Ordering Service"
               tran_name="commitBid_backend"
               user_name="ARM">
    <location class="com/ibm/ews/g2gServices/
                     grocery/SammysGroceryService"
              method="commitBid"/>
  </transaction>

</tdl>
```

Figure 7: The Gourmet2Go TDL used within this example defining six transactions associated with six individual methods; 'getBid' and 'commitBid' on each of the three back-end grocery services.

Application Name	Transaction Name	User Name	Status	Number Executed	Average Response Time	Last Response Time	Last Stop Date
Sammys Grocery Service	getBid	ARM	GOOD	40	958	828	Sun Mar 24 13:43
Sammys Grocery Service	commitBid	ARM	GOOD	3	988	1091	Sun Mar 24 13:39
Natural Bag Service	getBid	ARM	GOOD	40	906	1246	Sun Mar 24 13:43
Natural Bag Service	commitBid	ARM	GOOD	26	1000	773	Sun Mar 24 13:41
Dee Dees Servive	getBid	ARM	GOOD	20	541	314	Sun Mar 24 13:43
Dee Dees Service	commitBid	ARM	GOOD	11	533	502	Sun Mar 24 13:43

Table 1: Gourmet2Go Back-End Grocery Service Transaction Measurements. (All response times are measured in ms)

end grocery services provided with the Gourmet2Go demonstrator, the TDL shown in Figure 7 was used. Six transactions are defined, two for each service, selecting the 'getBid' and 'commitBid' methods within each service's class respectively. Optional descriptive information was associated with each transaction to provide clearer results within the ARM data repository: the application name is the name of the back-end service; the transaction name is the name of the method, either 'getBid' or 'commitBid' in this case; the user name remains constant at ARM.

4.3 Results

As well as 'ARMing' the back-end web services within Gourmet2Go, each transaction was also weighted to simulate a random element in response time that may occur during a service's life-time due to resource load, communication bottlenecks, etc.. This also provides more interesting response times, as without these loads these times were measured as being 0ms for a large number of iterations on the resource[2] used. The random elements chosen for this experiment were to set two of the services ('Sammys' and 'Natural Bag') to have an average response time of 1000ms, while 'Dee Dees' was set to 500ms. Dee Dee's was also published half-way through the experiment.

Once the TDL was used to 'ARM' the six defined transactions within the back-end web services, the Gourmet2Go demonstrator was brought online as per normal and standard user interactions were made via a web client to the broker for forty iterations. During each iteration, successfully completed transactions within the three grocery services were reported and could be seen updating within the ARM data repository.

The results reported by the ARM transactions defined within the back-end web services can be seen in Table 1. During each iteration of Gourmet2Go controlled by the user, the 'getBid' method of each published grocery service is executed, while the 'commitBid' method of only the user's selected service is executed. From the results, it can be seen that over the course of the experiment that the 'Sammys' service was selected three times, 'Natural Bag' was selected twenty-six times, and 'Dee Dees' was selected by the user eleven times, while only being published in the UDDI registry twenty out of the forty total iterations. It can also be seen that the 'Dee Dees' service performed with an average response time of roughly half the other two, due to the random load induced within the services for this experiment.

This case study, while providing a list of results obtained regarding the performance of Gourmet2Go, is mainly provided as an exercise into how the TDL can be used to monitor the performance of any transaction within any Java application; a web services environment implemented using J2EE such as Gourmet2Go being one example. This information can then be used by a developer to locate performance-critical areas of code or, as explained previously within the paper, as the QoS measurements used within a service routing decision algorithm.

5 Conclusion

This paper has demonstrated how the ARM standard and a novel transaction definition and instrumentation technique developed here at Warwick can be applied to assist web service intermediaries in making routing decisions by monitoring the web services under consideration. Our framework and experiences of applying our ARM tools and implementation to 'Gourmet2Go', a real web service sample from the IBM Web Services Toolkit has been discussed in detail in the hope that it will provide insight into the process of applying existing performance tools and standards to Web Service based applications.

It is envisaged that future work will include an implementation of a service routing algorithm that bases its decisions on the QoS metrics measured and reported by the ARM framework described in this paper. The TDL is also being modified so as to be able to concisely define how entire web service applications can be automatically instrumented and monitored.

6 Acknowledgments

The authors would like to express their gratitude to both IBM's T J Watson Research Center, NY. and IBM's Hursley Labora-

[2] All results were obtained using a Pentium III 450 Mhz with 256 Mb RAM, running Redhat Linux 7.1, kernel 2.4.17, IBM JVM 1.3.7, and the WSTK 2.4.2 run-time environment which includes Webshere Micro-Edition 4.

tories, UK. for the contribution towards this research.

The work is sponsored in part by a grant from the NASA AMES Research Center, and administered by USARDSG (contract no. N68171-01-C-9012), as well as the ESPRC (contract no. GR/R47424/01).

The authors would also like to convey their sincere thanks to the Computer Measurement Group for their support with travel and accommodation expenses during the 27th CMG Annual Conference, Anaheim, USA.

References

[1] K. Ballinger, P. Brittenham, A. Malhotra, W. Nagy, S. Pharies, "Web Services Inspection Language (WS-Inspection) 1.0", Available at www.ibm.com/developerworks

[2] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, D. Winer, "Simple Object Access Protocol (SOAP) 1.1", May 2000. Available at www.w3.org/TR/SOAP

[3] J. Cao, D.J. Kerbyson, E. Papaefstathiou and G.R. Nudd, "Modeling of ASCI High Performance Applications using PACE", 19th IEEE International Performance, Computing and Communication Conference, Phoenix USA. 485-492 (2000).

[4] J. Cao, D.J. Kerbyson, G.R. Nudd, "Performance Evaluation of an Agent-Based Resource Management Infrastructure for GRID Computing", Proceedings of 1st IEEE/ACM Int. Symposium on Cluster Computing and the Grid, Brisbane Australia. 311-318 (2001).

[5] J. Cao, D.P. Spooner, J.D. Turner, S.A. Jarvis, D.J. Kerbyson, S. Saini and G.R. Nudd, "Agent-based Resource Management for Grid Computing", Invited paper at the 2nd IEEE/ACM Int. Symposium on Cluster Computing and the Grid, Berlin. 21-24 May (2002).

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1", W3C Note, March 2001, Available at www.w3.org/TR/wsdl

[7] F. Curbera, W. Nagy, S. Weerawarana, "Web Services: Why and How", OOPSLA 2001 Workshop on Object-Oriented Web Services, Florida, USA, 2001

[8] F. Curbera, N. Mukhi, S. Weerawarana, "On the Emergence of a Web Services Component Model", 6th International Workshop on Component-Oriented Programming (ECOOP), Budapest, Hungary, 2001

[9] M. Duftler, N. Mukhi, A. Slominski,S. Weerawarana, "Web Services Invokation Framework", OOPSLA 2001 Workshop on Object-Oriented Web Services, Florida, USA, 2001

[10] I. Foster, C. Kesselman, "The Grid : Blueprint for a New Computing Infrastructure", Morgan Kaufmann. 279–290 (1998).

[11] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", To be published in Intl. J. Supercomputer Applications, (2001).

[12] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", www.globus.org/research/papers/ogsa.pdf

[13] IBM Corporation, "Web Services Flow Language (WSFL 1.0)", May 2001, Available at www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[14] IBM Web Services Toolkit, Available from www.alphaworks.ibm.com/tech/webservices toolkit

[15] M.W. Johnson, J. Crowe, "Measuring the Performance of ARM 3.0 for Java", Proceedings of CMG2000 International Conference, Orlando USA. (2000).

[16] H. Kreger, "IBM Web Services Conceptual Architecture (WSCA 1.0), May 2001, Available at www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

[17] W. Leinberger, V. Kumar, "Information Power Grid : The New Frontier in Parallel Computing?", IEEE Concurrency **7(4)**, (1999).

[18] G.R. Nudd, D.J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper, D.V. Wilcox, "PACE - A Toolset for the Performance Prediction of Parallel and Distributed Systems", International Journal of High Performance Computing Applications, Special Issues on Performance Modelling. **14(3)**, 228-251 (2000).

[19] OASIS, Business Transaction Protocol (0.9), OASIS prefinal draft 2001, Available from www.oasis-open.org

[20] The Open Group, "Application Response Measurement (Issue 3.0 - Java Binding)", Open Group Technical Specification, (2001). Available from regions.cmg.org/regions/cmgarmw/index.html

[21] The Open Group. www.opengroup.org

[22] D.P. Spooner, J. Cao, H.N. Lin Choi Keung, S.A. Jarvis, G.R. Nudd, “Titan: A Performance-Driven Scheduler”, Submitted to 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, July 2002

[23] S. Thatte, “XLANG: Web Services for Business Process Design”, May 2001, Available at www.gotdotnet.com/team/xmlwsspecs/xlang-c/default.htm

[24] J.D. Turner, D.P. Spooner, J. Cao, S.A. Jarvis, D.N. Dillenberger, G.R. Nudd, “A Transaction Definition Language for Application Response Measurement”, International Journal of Computer Resource Measurement, **105**, 55-65 (2001).

[25] UDDI Project, “UDDI Technical White Paper”, September 2000, Available at www.uddi.org

[26] World Wide Web Consortium, Web Services Activity, Available at www.w3.org/2002/ws/