JISC DEVELOPMENT PROGRAMMES

Project Document Cover Sheet

An Overview of Service-Oriented Architecture

Project

Project Acronym	CORE	**Project ID**	
Project Title	Collaborative Orthopaedic Research Environment		
Start Date	01 November 2004	**End Date**	31 October 2006
Lead Institution	University of Southampton		
Project Manager & contact details	Dr Gary Wills Intelligence, Agents, Multimedia Group School of Electronics and Computer Science University of Southampton Southampton, SO17 1BJ		
Project Web URL	www.core.ecs.soton.ac.uk		
Programme Name (and number)	*Virtual Research Environments (05/04) Strand III*		
Programme Manager	Dr Maia Dimitrova		

Document

Document Title	An Overview of Service-Oriented Architecture		
Reporting Period			
Author(s) & Project Role	Yee Wai Sim (Co-Investigator), Chu Wang (Co-Investigator), Lester Gilbert (Technical Manager), Gary Wills (Project Manager).		
Date	01/07/2005	**Filename**	ecstr_iam05_004.doc
ISBN	0854328262		
Technical Report Number	ECSTR-IAM05-004		
Access	☐ Project and JISC internal	■ General dissemination	

Document History

Version	Date	Comments
0a	17 June 2005	Initial draft of report
1a	1 July 2005	Final version

An Overview of Service-Oriented Architecture

Y. W. Sim, C. Wang, L. Gilbert, G. B. Wills
School of Electronics and Computer Science
University of Southampton
E-mail: {yws01, cw2, lg3, gbw}@ecs.soton.ac.uk
Technical report Number: ECSTR-IAM05-004
ISBN: 0854328262

Abstract

This report is a literature review of Service-Oriented Architectures (SOA). An SOA is a loosely couple network of communicating services. The key elements and characteristics of SOAs are outlined to guide the CORE project in creating a Virtual Research Environment (VRE). The benefits of using SOA concepts in implementing the VRE are presented and justifications of their use in the CORE project are also discussed.

1 Introduction

The *Collaborative Orthopaedics Research Environment* (CORE) [7] is a JISC funded project, which aims to provide an infrastructure that combines clinical, educational and research in one working environment. The current paradigms of information sharing and resource use in biology and medicine are being challenged at several fronts. Firstly, as the number of investigators, organisations and institutions conducting biomedical research increases, it becomes difficult to track the work and provide infrastructure to support this expansion. Although current information technology supports ready access, it does not address abstraction, integration, and interpretation of information. The diverse bio-informatics tools generated to consume and evaluate the data rarely interoperate [5]. Secondly, the very large volume of data generated in modern biomedicine presents a primary challenge to the researcher. To integrate biological data one would want to move seamlessly between biologic and chemical process, between organelle, cell, organ, and organ systems, and between individuals, family, community, and populations. Such integration generates challenges to information structure as each research community tends to speak its own scientific dialect [8]. Finally, biomedicine's culture is at the nexus of the challenge faced within many other scientific fields: the need for collaborative research. The collaborative researchers recognise that many of the technology approaches required in biology and medicine are expensive, beyond the reach of individual investigators, and increasingly challenging the resource reserves of all but a few institutions. New paradigms are required to support such researchers.

The CORE project intends to address the challenges discussed above by implementing a *Virtual Research Environment* (VRE) demonstrator using *Service-Oriented Architecture* (SOA) concepts. This report is a literature review of SOA, which refers to systems structured as networks of loosely coupled, communicating services [4]. The purpose of this review is present the SOA concepts, which will be used as guidelines in designing and building the CORE VRE using Grid/Web services technology.

A SOA is a style of design that guides all aspects of creating and using services through their lifecycle (from conception to retirement), as well as defining and providing the information infrastructure that allows different applications to exchange data regardless of the operating systems or programming languages underlying those applications. The report presents the definition of services and then continuing with a discussion of the SOA concepts, namely the goals, key elements and characteristics of a SOA. The authors conclude the report with discussion of benefits in using SOA to underpin an information infrastructure, and finally describe the work of standard bodies in producing SOA specifications.

2 Definition of Services

From an operational perspective, services are *Information Technology* (IT) assets that correspond to real-world activities that can be accessed according to the service policies that have been established for the services. The service policies define, for example, who or what is authorised to access the service, the performance and reliability levels of the service, and the security levels of the service.

Viewed from a technical perspective, services are coarse-grained, reusable IT assets that have well-defined interfaces that clearly separate the services' externally accessible interface from the services' technical implementation. This separation of interface and implementation decouples service requesters from service providers, enabling both requesters and providers to evolve independently as long as the interfaces remain unchanged.

In a SOA, a system operates as a collection of services. Each service may interact with various other services to accomplish a certain task. The operation of one service might be a combination of several low level functions, e.g. functions that converts objects to basic data types, and in this case, these low level functions are not considered as services.

3 The Goals of SOA

An important goal of using SOA is to align the information infrastructure with real-world activities. A SOA reduces project costs and improves project success rates by adapting technology more naturally to the people who need to use it, rather that focusing on the technology itself. The major difference between a SOA and other approaches, i.e. object orientation and procedure orientation, is that it concentrates on the description of the real-world problem, whereas other approaches require developers to focus more on the use of specific execution environment technologies such as COM/DCOM, J2EE or the .NET framework.

A second goal of the SOA approach is to provide an agile technical infrastructure that can be quickly and easily reconfigured as user requirements change. The promise of the SOA approach is that it breaks down the barriers between technical implementation and real-world processes by combining the advantages of custom solutions and packaged applications while reducing lock-in to any single IT vendor. This is achieved by separating the service interfaces from their implementation. The operations underlying each service can be coded using technologies, such as J2EE and CORBA objects, but none of the details of any of these technologies are visible to the service requesters. Hence, the implementation underlying the services can change (i.e.

for better performance) independently of their requesters provided the interfaces remain the same.

4 Key Elements of SOA

A SOA is an evolving entity that changes over time; therefore, processes, principles, and tools need to be put in place to facilitate its evolution and growth [9]. There are three key components of a SOA, illustrated in Figure 1 .

The *SOA Governance Policies & Processes* component in Figure 1 represents the high-level processes for governing the SOA, including the SOA decision making and issue resolution processes, roles and responsibilities of teams, development processes, testing processes, quality assurance processes, registering services and so on.

Figure 1: Key components of a service-oriented architecture

The *SOA Principles & Guidelines* component depicted in Figure 1 describes the principles that guide architects and developers when defining services, such as the principles of reusability that needs to be taken into account whenever designing or developing a service.

The *SOA Methods & Tools* component in Figure 1 defines the methods (analysis, design, testing, etc.) and tools (design tools, development tools, test tools, etc.) that have been approved for use in a given SOA. In general, a SOA should be based on standards that are independent of any single product or vendors and so that different technologies can be used as part of the SOA as necessary.

5 Characteristics of SOA

This section presents the key characteristics that should go into the design and implementation of services in order to deliver the goals of SOA discussed earlier in this report. However, sometimes the cost of including a particular service characteristic (e.g. making the service stateless) is prohibitive when compared to a specific organisation's goals.

5.1 Loose Coupling

A SOA is an architectural style whose goal is to achieve loose coupling between the service requesters and service providers. This means that the service requester has no knowledge of the implantation details of the service provider, such as the programming language used, the deployment platform, etc. The service requester should be able to invoke a service by way of messages through a published interface (service contract), rather than through the use of APIs. For example, under no circumstances should the service requester be asked to provide one of the input parameters as a SQL command even though the service provider uses a SQL database. In other words, the service interface should encapsulate all implementation details and make them opaque to service requesters.

5.2 Well-Defined Interface

Every service should have a well-defined interface that defines the service's capabilities and how to invoke the service in an interoperable fashion, clearly separating the service's externally accessible interface from its technical implementation. Although it is not widely recognised, service contracts are generally more valuable than the service implementations. The service interface is the basis for service sharing and reuse and is the primary mechanism for reducing interface coupling. Furthermore, changing a service interface could be more expensive than modifying the implementation of a service. This is due to the fact that changing a service interface might require modification to be made at the service requesters' side, while changing the implementation of a service does not usually have such far-reaching effects. Hence, it is important to have a formal mechanism for extending and versioning service interfaces to manage these dependencies.

5.3 Base on Open Standards

Services should be designed and implemented based on open standards. Using such an approach provides a number of advantages such as minimising vendor lock-in and increasing the opportunities for the service provider to support a wider base of service requesters. Web services technologies are one of the open standards that have been adopted by SOA developers. Its open, standards-based technologies allow service requesters and service providers to be isolated from proprietary, vendor-specific technologies, e.g. J2EE and .Net framework. The open standards-based approach also increases the opportunities to take advantage of open source implementations of the standards, and of the communities that have grown up around these open source implementations.

5.4 Discoverable

Publication of services should be in a manner which permits discovery and consumption with minimum intervention of the provider. A service interface should use metadata to define the service capabilities and constraints. The service interface should be machine-readable, i.e. an XML-based text file, so that it can be dynamically registered and discovered. This lowers the cost of locating and using services, reduces errors associated with such use, and improves the management of services.

5.5 Stateless Service

Dependencies among services should be minimised. Most importantly, services should be self-sustaining so that they can interoperate with other services without

unnecessary internal dependencies and without sharing state. In particular, they should be implemented so that each invocation is independent and does not depend on the service maintaining persistent state between invocations.

Stateless interactions scale more efficiently because any service requester can be routed to any service instance. The requirement of stateless service makes a service provider more scalable because it does not have to store state information between requests. In addition, the lack of intermediate states makes recovery from partial failure relatively easy. This makes a service more reliable.

When dependencies among services are required, they are best defined in common terms of common application processes, functions and data models, not implementation artefacts (e.g. a session key). Nevertheless, in certain situations, the requirement of persistent state between service invocations is unavoidable, but his should be separate from the service provider.

5.6 Service Granularity

The use of coarse-grained interfaces for external consumption is recommended, whereas fine-grained interfaces might be used internally. Although fine-grained interfaces offer more flexibility to the consumer application, it also means that patterns of interaction may vary between difference service requesters. This can make support more difficult for the service provider. A coarse-grained interface ensures that the service requesters will use the service in a consistent manner.

5.7 Quality of Services

Service developers need to consider the security capabilities and requirements when using the Internet and linking across partners' security domains. Further, Internet protocols are not designed for reliability (guaranteed delivery and order of delivery). It is therefore up to the service developers to ensure that a message is delivered and processed once and on time (alternatively, the developers can permit duplicate messages provided this has the same effect as receiving a unique message).

6 Benefits of SOA

Services that posses the characteristics discussed earlier deliver the benefits presented in the following sections.

6.1 Efficiency

A SOA promotes modularity because services are loosely coupled. This modularity has positive implications for the development of composite applications because after the service interfaces have been defined, each service can be designed and implemented separately by the developers who best understand the particular functionality that is required. As for the service requesters, they can design and implement applications based solely on the published service interfaces and without reference to the source code that implements the service consumed.

At the application level, there are two distinct, well defined tasks. The first task is to model the application in terms of the data it produces and consumes. After the model has been defined, the application can be created by composing or orchestrating the available services. For complex applications where the service composition logic is

likely to change, service composition or orchestration is best handled using a product designed for that purpose (such as one that supports WS-BPEL [11]).

6.2 Reusability

One of the benefits of using SOA is that service reuse will lower development costs and speed time to market. Services that have well-defined interfaces make it easier for the developer to locate the appropriate service. Proper registration policies and standardised taxonomies enable easy discovery. Furthermore, the metadata-driven interfaces can be used to fully or partially generate artefacts for using the service and for run-time code to dynamically adapt to changing conditions.

Another key characteristic of a SOA is loose coupling among services. This characteristic facilitates reuse across different applications since services are decoupled from a single real-world process. In addition, the encapsulated implementation of the SOA also simplifies the developers' life as they do not have to worry about compiler version, platforms, and other incompatibilities that typically make code reuse difficult.

6.3 Simplified Maintenance

The concepts of a SOA simplify maintenance and reduce costs because of the fact that SOA applications are modular and loosely coupled. A service and its associated interface encapsulate process logic in such a way the other services can be agnostic about its implementation and focuses on inputs and responses of the service. This means the developers can modify the services (including major modification) without affecting those who maintain other parts of the system, as long as the service interfaces remain unchanged. For instance, a service can be rewritten and hosted on a lower-cost platform without having any necessary impact on its requesters.

6.4 Incremental Adoption

Due to the nature of modularity and loose coupling of the SOA, applications can be developed and deployed incrementally. Often, a reasonable subset of the full functionality can be developed quickly, which has obvious time-to-deployment advantages. Additional functionality can readily be added in planned stages until the full feature set has been realised.

7 Conclusion

Two common methods of integrating systems are integration at the user interface level using portals, or at the data level by creating large combined datasets or data warehouses. The SOA approach does not preclude using portals or data warehouses, and is in fact agnostic about how the rest of the enterprise is configured. Thus, the use of SOAs can be regard as a good approach for constructing frameworks.

In addition, adopting SOA is essential to delivering agility and flexibility in technical terms. The SOA theme enables reuse via shared services, where flexible granular functional components expose service behaviours accessible to other applications via loosely coupled standards-based interfaces. The benefits of using SOA can only be fully realised when the key principles articulated in this report are followed closely in creating a service oriented application.

SOA specifications are progressing toward standardisation through a variety of ways, including small groups of vendors and formally chartered technical committees. For example, a SOA Reference Model Technical Committee [10] has been formed by OASIS members to encourage the continued growth of different and specialised SOA implementations whilst preserving a common layer of understanding about what SOA is. Another function of these committees is to help architects and software vendors make consistent logical divisions in their architectures and products. JISC SOFER is another working group starting to define a Service-Oriented Framework for Education and Research [1]. The working group's aim is to define a classification of services and related specifications, standards and protocols that are of relevance to a VRE, based on the concepts of SOA. It is intended that, in fullness of time, JISC funded projects should follow the recommendations of this working group in order to ensure interoperability of the tools and services being deployed. However, there are still difficulties in standardising SOA specifications since no single standards body is clearly in a leadership position.

References

1. Allan, R. (2005) SOFER: The Service Oriented Framework for Education and Research, *CCLRC e-Science Centre, Daresbury Laboratory*, articles available from http://www.grids.ac.uk/Papers/SOFER/sofer.pdf
2. Bairoch, A. and Apweiler, R. (2000) The SWISS-PORT Protein Sequence Database and its Supplement TrEMBL in 2000. *Nucleic Acids Research*, 28(1), pp. 45-48.
3. Bernholdt, B., Bharathi, S., Brown, D., Chanchio, K., Chen, M., Chervenak, A., Cinquini, L., Brach, B., Foster, I., Fox, P., Garcia, J., Kesselman, C., Middleton, D., Nefedova, V., Pouchard, L., Shoshani, A., Sim, A., Strand, G. and Williams, D. (2005) The Earth System Grid: Supporting the Next Generation of Climate Modeling Research. *Proceedings of the IEEE*, 93, pp. 485-495.
4. Booth, D., Champion, M., Ferris, C., McCabe, F., Newcomer, E. and Orchard, D. (2004) Web Services Architecture (W3C Working Group Note 11 February 2004). Available from: http://www.w3.org/TR/ws-arch/
5. Buetow, K. (2005) Cyberinfrastructure: Empowering a "Third Way" in Biomedical Research. *Science*, 308(5723), pp. 821-824.
6. Castells, M. (2001) *The Internet Galaxy: Reflections on the Internet, Business, and Society*. Oxford University Press, Inc., New York.
7. Collaboration Orthopaedics Research Environment. *University of Southampton*, Web site available from: http://www.core.ecs.soton.ac.uk
8. Hey, T. and Trefethen, A. (2005) Cyberinfrastructure for e-Science. *Science*, 308(5723), pp. 817-821.
9. Newcomer, E. and Lomow, G. (2005) *Understanding SOA with Web Services*. Addison-Wesley Professional.
10. OASIS SOA Reference Model TC. *OASIS*, Web site available from: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

11. OASIS Web Services Business Process Execution Language (WSBPEL) TC. *OASIS*, Web site available from: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
12. Szalay, A. and Gray, J. (2001) The World-Wide Telescope. *Science*, 293(5537), pp. 2037-2040.