Pragmatic Web Service Design: An Agile Approach with The Service Responsibility and Interaction Design Method

David E. Millard , Yvonne Howard, Noura Abbas, Hugh C. Davis, Lester Gilbert, Gary B. Wills, Robert J. Walters

School of Electronics and Computer Science

University of Southampton, Southampton, UK

+44 (0)23 8059 5749

{dem, ymh, na06r, hcd, lg3, gbw, rjw1} @ecs.soton.ac.uk

Abstract

Service-Oriented Architectures (SOAs) are increasingly deployed to achieve distributed systems that are modular, flexible and extensible. However, designing for SOA can be challenging; there are issues involving the granularity of the cooperating services, problems with proprietary data models being exchanged, and there are no currently accepted conventions for describing a service or its interactions at an abstract level. This paper gives an overview of the Service Responsibility and Interaction Design Method (SRI-DM), an agile approach for engineering a Web Service design based on capturing a scenario as a use-case, factoring this into a set of Service Responsibility and Collaboration Cards, and constructing a Sequence diagram illustrating their interactions in fulfilling the scenario. Through two case studies the paper shows how using SRI-DM can expose many of the problems of over-engineering SOA and help to create simpler, more pragmatic web service designs.

Keywords: Web service design, Agile software development
Subject Classification: D.2.2 Design Tools and Techniques

1. Introduction

Engineering widely distributed systems has long been a challenge for the software engineering community. In the last few years a trend has emerged towards Service-Oriented Architectures (SOA) that aims at simplifying this problem. SOAs are an attempt to modularize systems in such a way that they are composed of independent software components that offer services to one another through well-defined interfaces. The service approach is ideally suited to more loosely coupled systems, where individual parts may be developed by different people or organizations. Wilson et al. describe the three main advantages of such a system as Modularity (dynamic coupling), Interoperability (standard interfaces), and Extensibility (encapsulation) [15].

Service-orientation is a philosophical approach to creating distributed systems, but there are a number of standards and approaches to providing them at an implementation level (including Web Services based on SOAP, GRID Services based on OGSI, and REST services based on HTTP and XML). Because of the difference in these approaches, and due to a lack of common notation and engineering experience, developing a service-oriented system can be difficult. Decisions must be made about how to divide a problem into logical services, how those logical services should be interfaced to maximize reuse, how they should be gathered together to create composite services, and what service-oriented implementation is best suited to each service, or to the design as a whole.

A particular danger is in *over-engineering* the web service design, resulting in a system which is highly granular (introducing performance overheads) but which doesn't benefit from that

granularity in terms of reuse and extensibility (due to tight coupling of data models or high dependency on a particular order of interaction).
In this paper we give an overview of the Service Responsibility and Interaction Design Method (SRI-DM), an agile approach for the modeling of services at an abstract level that is independent of implementation. SRI-DM:

- Defines a scenario with a use case diagram.
- Factors a set of services based on individual use cases.
- Represents these services at a high level using Service Responsibility and Collaboration cards (SRCs).
- Refactors these SRCs as necessary.
- Defines how services could interact to fulfill the scenario using a sequence diagram.

Through two case studies (PeerPigeon and ASDEL) we show how using SRI-DM can identify the problem of over-engineering, and help to create simpler more pragmatic web service designs.

2. Background

Service orientation is an approach to creating stand alone components such that their potential for reuse is maximized. A number of standards, infrastructures and protocols have emerged which provide for this at an implementation level.
Web services have received a great deal of recent attention, and are defined around a set of standards (such as SOAP, WSDL, UDDI) developed by the W3C to make functionality available over the Web as simply as data [4, 7]. A new generation of Web Service standards (such as WS_Security) is now being introduced to add a standard layer of authentication and security to Web Services. This will make Web Services attractive for systems builders as it will become possible to build virtual organisations using relatively lightweight middleware.
Another approach to service provision is represented by Representational State Transfer (REST) [6], where HTTP and XML are used to send and retrieve data to a remote script or application residing on a Web server. REST services are popular, but are not secure enough to build virtual organizations and therefore will not be able to support the growing number of sophisticated service-based systems.
We believe that each approach is appropriate in different situations, and that an agile methodology for service design should be agnostic about the service technology itself.

2.1. Establishing SOA

The take-up of Web services within enterprises may be problematic. Weatherley suggests that in the educational domain there are a number of barriers that prevent the widespread use of Web services for delivering Web-based educational materials [14]. These barriers relate to the need for understanding Web service protocols and the dynamic nature of the communication with Web browsers. Mukhi et al. believe that an increase in the adoption of SOA requires improvement to some of the non-functional features such as security, transactionality and reliability [10]. They have developed a framework that supports and uses transactional and reliable services, achieved by using a policy model based on WS_Policy.
SOA specifications are progressing toward standardization in a variety of ways, including small groups of vendors and chartered technical committees. The e-Framework is an initiative by the UK's Joint Information Systems Committee (JISC) and Australia's Department of Education,

Science and Training (DEST) to systematize a SOA for Education and Research [11,17]. We believe that substantive barriers to the establishment of SOAs include little shared understanding about how services should be developed, what granularity is appropriate for different problems, and no common notation to enable developers to share designs.

2.2. Modeling Services

Dijkman and Dumas explain the need for particular Service Oriented Design strategies [5], based on a number of characteristics that differentiate Service from component-based design: High Autonomy (of designers and developers), Coarse Granularity (of service interfaces), and Process Awareness (close relationship with business processes). Enterprise level service development is most affected by the latter two characteristics. Quartel et al describe the use of design milestones to help develop Web services from business practices [12], and Benatallah and Dumas have created environments to ease the creation of composite services [3]. Martin et al. suggest that the best way to implement Web Services in an enterprise is to start with a component-based architecture that exposes business process level services as Web services [9]. Wada et al have taken a model driven approach to this problem, building a model of the domain and then using this to derive an object design [13]; this kind of modeling has also been used with SOAs to validate a design [1].

Wilson et al. present Reference Models as a potential solution [15]. Broadly speaking a Reference Model can be thought of as a description of how a set of services within a Framework collaborate to provide the necessary functionality for a particular domain. Reference models are a way to help architects and software vendors make consistent logical divisions in their architectures and products. However, they require a method for describing services and their interactions at an abstract, logical level.

While model-driven approaches give you the benefits of automatic model transformations where there is a consistent/constrained understanding of the processes. We believe that this model-driven approach to service-design may be too high an overhead in more uncertain environments. In these situations an agile approach seems more appropriate.

2.3. Agile Methods

Agile methods are a number of software development methods that were proposed in the mid 1990s as a reaction to inflexibility of traditional approaches. An agile method could be defined as an adaptive process run by talented and creative people and controlled with iterative and incremental development [18]. Although agile methods were initially described as development methodologies, the term *agile* represents an attitude, a philosophy, and a way of thinking that was presented through the principles and practices in the agile manifesto [19]. This way of thinking can be applied to many other aspects of software creation including design and modeling. Agile techniques share common principles [16] such as:

- delivering working software frequently within a short timescale
- close communication
- simplicity
- programming over documenting
- customer involvement
- encouraging rapid and flexible response to change.

SRI-DM is agile as it enables a team of developers to define a scenario quickly and generate a number of services that will fulfill it. It is lightweight in that the documentation is limited to what is needed, and serves to drive the development forward as well as record it for others.

3. SRI-DM

In this section we present an overview of the Service Responsibility and Interaction Design Method (SRI-DM). It uses a collection of logical descriptions (Service Profiles) to describe how a number of services, regardless of implementation, could be combined to solve a particular problem defined as a use case scenario.
The method produces a design that has the following parts:

- **A Scenario**: presented as a Use Case Diagram and narrative that describes a problem for which a set of services can provide a solution.
- **Service Profiles**: a set of profiles that describe a number of services at an abstract logical level. These suggest granularity, and describe the individual capabilities of each service. They promote reuse and understanding of the design, while retaining flexibility in the implementation.
- **A Sequence Diagram**: showing one example of how the services can interoperate to fulfil the scenario.

SRI-DM separates abstract representations of services from their implementation; however as the design process is agile, it is iterative (as we will see in the Service Profile section) in order to cope with requirements change
Service Profiles are not concrete interfaces and so cannot be described using interface definition languages (such as WSDL). Instead they set the granularity of the model, and describe in a semi-formal way the role of each service and the potential ways in which they might rely on one another.
In the rest of this section we will look at each part of the SRI-DM — Scenario, Service Profiles, and Sequences — and describe their formal notation.

3.1. Scenarios

Our method takes as its starting point a scenario that describes a problem that is to be solved using a set of interacting services. We have chosen use case diagrams as our method of modeling because they are high level and implementation independent. From an agile point of view they are also useful in that they are relatively informal, simple, and help to define and structure a problem space without too much detail about the activities within that space. A brief narrative description is held alongside the diagram as a whole, as well as for each individual use case. These descriptions help disambiguate the use cases, explain the roles of the different actors associated with the use cases, and focus at a high level on what each use case involves [21].
Scenarios are developed in a community or user focused manner in line with agile principles to ensure that they are relevant. These use case diagrams capture the practice of an existing user community.

3.2. Service Profiles

Service Profiles are abstract descriptions of services that may be fulfilled by several different Service Implementations which may each expose different concrete interfaces. Service Profiles are thus modeled in an abstract way that does not prescribe a data model or dictate explicit methods. To do this we created Service Responsibility and Collaboration cards (SRCs) based on Class Responsibilities/ Collaborations, a modeling technique first described by Beck and Cunningham for eXtreme Programming [2].

Our SRCs model the capability of a service to realize a specific use case (a single bubble from a use case diagram). An SRC card is a small card (we use A5 address cards in our sessions). The name of the service appears at the top of the card. Down the left hand side of the card, we list the responsibilities of the service. On the right hand side we list and group other services which collaborate to fulfill the responsibilities listed on the left hand side. The responsibilities of a service describe at a high level: what it is for, what it does, and what it can provide.

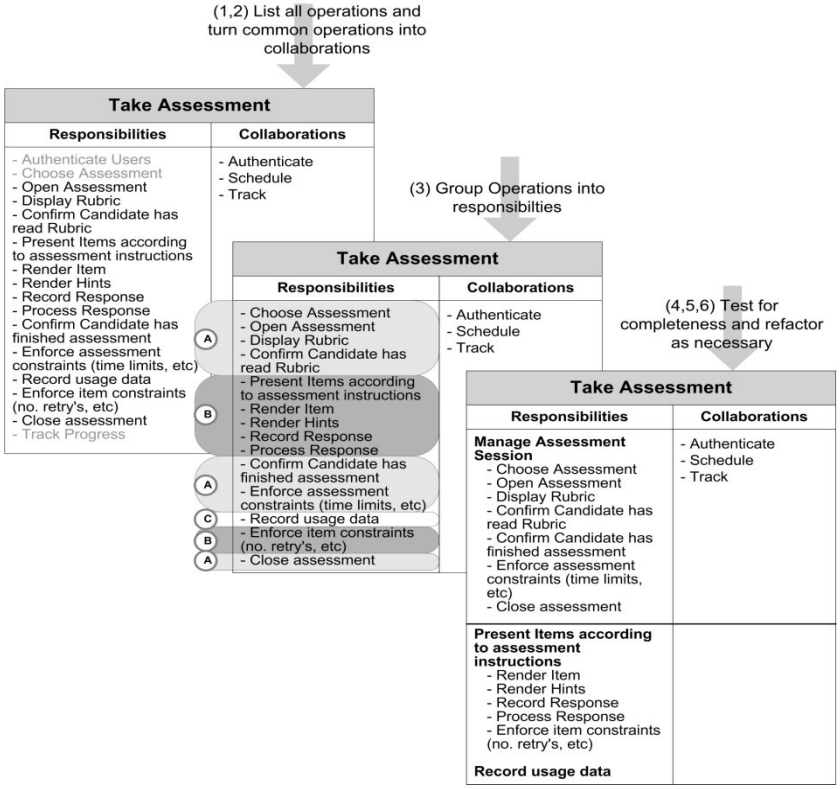

Figure 1: The factoring of the "Take Assessment" Use Case into a SRC

Deriving SRCs from the Use Cases is a six-step process:

1. Work through each use case. A traditional noun and verb analysis is a useful technique; verbs can indicate the responsibilities of the services that fulfill the use case, and nouns imply a data model and inform the narrative. From the verb analysis write down all of the operations needed for a use case.
2. Group the operations into responsibilities and collaborations.
3. Consider which operations might be common with other SRCs and move them from the responsibilities to the collaborations column.

4. Identify which responsibilities would benefit from which collaboration.
5. Test the completeness/accuracy of the design by working various scenarios.
6. Re-visit the SRC and re-factor as necessary as other SRCs are developed, and as common collaborations become apparent.

Figure 1 shows this process applied to a "Take Assessment" Use Case (the numbers above each card refer to the stages described above). The use case description is used to derive the initial list of operations, which are consequently factored into a set of responsibilities and collaborations. This example is taken from the work of the FREMA project [20] which built a number of reference web service models for the domain of assessment.

3.3 Service Sequence Diagrams

At the scenario level, services represented by SRCs must interact with each other to fulfil a wider purpose. We use Sequence Diagrams to represent the interactions, showing which services should communicate and in which order, and containing enough description to show how the individual services are responsible for moving and processing data, without having to specify the detail of the data model or the decision making logic. Figure 2 shows a sequence diagram from the FREMA "Take Assessment" example, and in particular the interactions around the *candidate* actor. Collaborations are modeled, although in this diagram they are grouped together into one column to aid clarity.

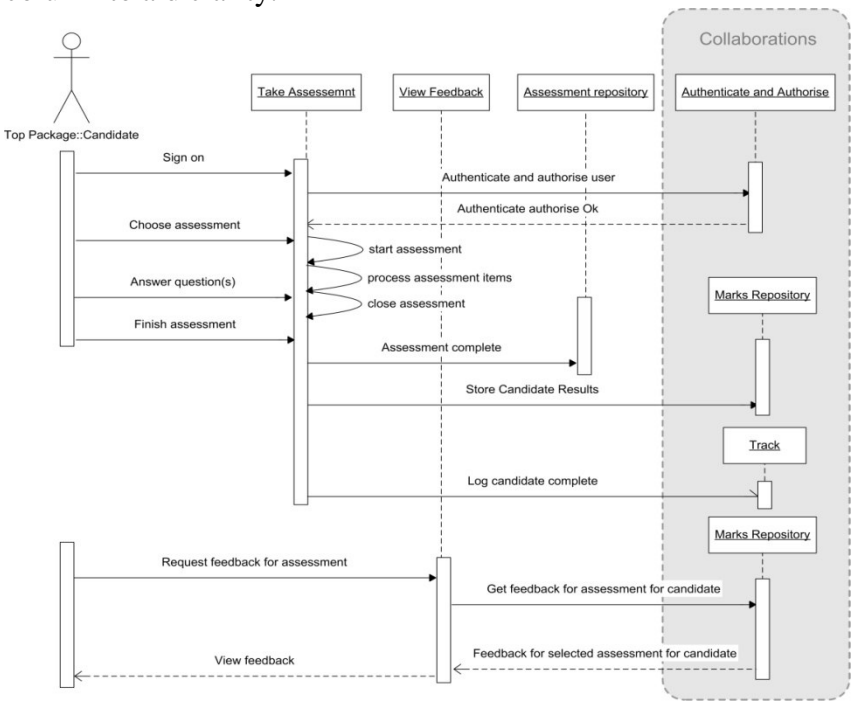

Figure 2: Sequence Diagram from the Take Assessment Use Case

The SRCs and sequence diagrams are not intended to provide a complete description of interacting services; they provide an overview model, and not an interface description or detailed process model. Developers can use the SRCs to decide what responsibilities their services will take, and the sequence diagrams to see the consequences for interfaces to other services.

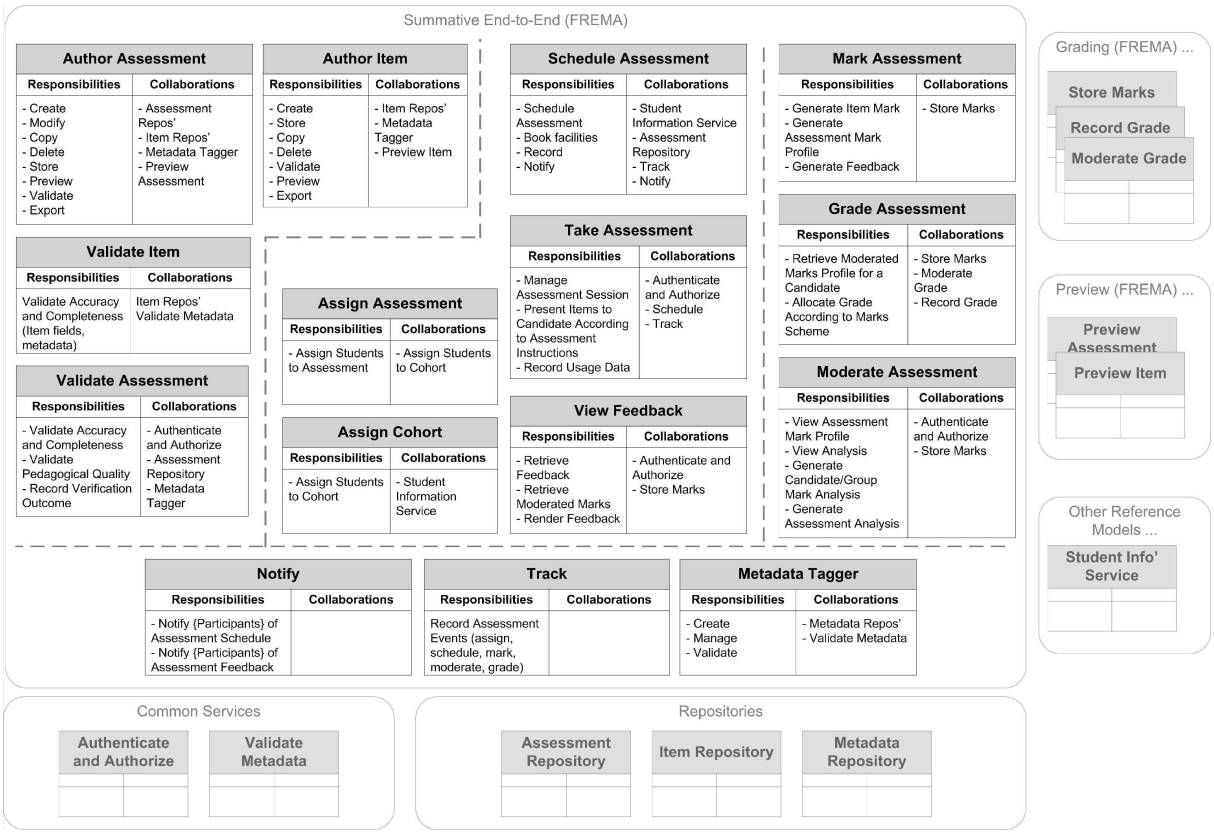

Figure 3: SRCs for Summative End-to-End CAA

3.4. Refactoring

The end of the SRI-DM process provides a set of SRC cards and semi-formal descriptions of the conversations that occur between them. This is sufficient understanding to begin to revise the Services, splitting and conjoining as appropriate. Figure 3 shows the entire set of SRC cards for FREMA. They have been arranged into three broad categories (left to right: authoring, running and marking) with a number of smaller supporting services below (notify, track and tagging). There is no single solution for a good design, and these services could be refactored in a number of different ways. SRI-DM gives the designer a good understanding of the complexity of Service interactions, and we would suggest that revising the design to minimize data and interaction complexity leads to more realistic service designs.

4. Case Studies

In this section we look at two case studies that show how SRI-DM has exposed unnecessary complexity, and resulted in a simpler and more pragmatic service design.

4.1. Case Study 1: PeerPigeon

PeerPigeon was a six-month JISC (UK) funded project to produce a set of services to support the peer review process in higher education institutions. It was an interesting application of SRI-DM

as it demonstrates how the method can identify data and conversation complexity between services, ultimately resulting in simpler design.

Peer Review, sometimes called Peer Assessment or Peer Evaluation, is an important technique for educators where students produce feedback (or grades) for each others' work. Peer Review activities can be formative or summative, and vary greatly in their complexity. To create PeerPigeon we needed to create a canonical model of Peer Review, which we did by examining a number of existing Peer Review systems and then generalizing to a common set of building blocks. The PeerPigeon Peer Review Pattern is based on *Peer Review Cycles* (the visible stages of peer review) and *Peer Review Transforms* (the invisible rules that dictate how documents move between peers within each stage). For example, in the case study, a course is run as a academic conference, the students who take the course are both authors and reviewers of the conference papers and also form the committee. The students are assessed through these peer assessment activities. Figure 4 shows how the peer review stages for a typical conference paper can be expressed as six different cycles with a single transform in each.

Cycle	**Transform**		
	Input	**Action / Participant**	**Output**
1. Author	-	Authors each write	a Paper
2. Review	Each Paper	is transformed by a Reviewer	into a Review
3. Decision	Each Set of Reviews	is transformed by the Committee	into one Decision
4. Decision Feedback	Each Decision	is given to the *appropriate* Author	-
5. Review Feedback	Each Set of Reviews	is transformed by *appropriate* Author	into a Revised Paper
6. Final Paper	Each Revised Paper	is given to the Committee	-

Figure 4: Cycles and Transforms for a Typical Academic Conference Paper

We assumed that a general Peer Review system would need to take this Pattern and instantiate it into a Plan, a set of concrete transforms involving real participants with an appropriate schedule. This context enabled us to create a generalized use case for peer review as shown in Figure 5.

Using SRI-DM we began the process of converting these use cases into initial Service Descriptions. Figure 6 shows the results of refactoring with SRI-DM, the method made it clear where the design was overly complex and allowed us to simplify by consolidating tightly coupled services. The first refactoring seems very plausible, with the core PeerPigeon System represented by four services (Author, Populate, Validate and Run Assessment Plan). However it quickly became clear that these services have to be tightly coupled in terms of data, in that they need to agree on the format of the Peer Review Plan. The second refactoring shows a much more pragmatic view of the system, with a user interface accessing the core PeerPigeon engine that has been exposed as a service, which itself uses separate logging, notification and packaging services. .

Using SRI-DM exposed the granularity and tight data coupling in the first factoring, and together with its explicit requirement to refactor for simplicity, resulted in a successful, efficient design.

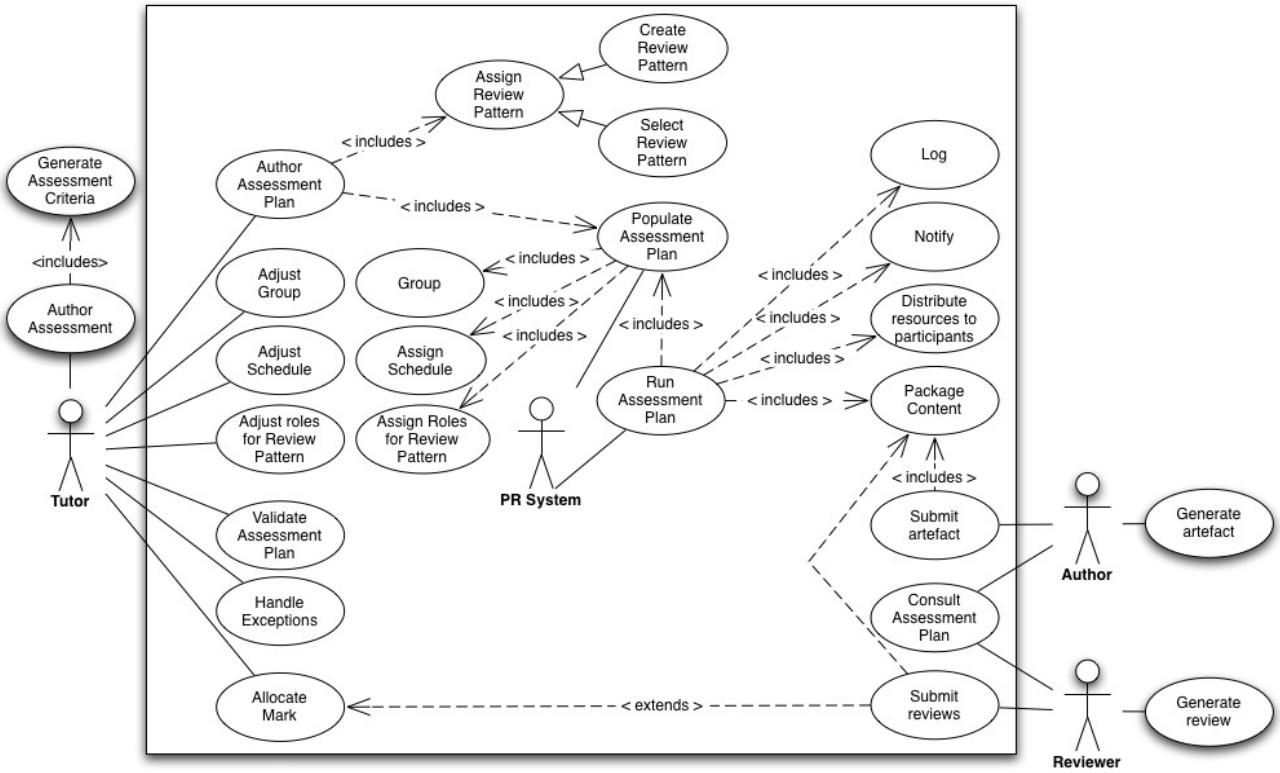

Figure 5: Generalized Use Case for Peer Review based on the PeerPigeon Patterns

Fine Grained Service	**First Refactoring**	**Second Refactoring**
Author Assessment Plan	Author Assessment Plan	PeerPigeon Service
Assign Review Pattern		
Create Review Pattern		
Select Review Pattern		
Populate Assessment Plan	Populate Assessment Plan	
Group		
Assign Schedule		
Assign Roles		
Adjust Group		
Adjust Schedule		
Adjust Roles		
Validate Assessment Plan	Validate Assessment Plan	
Run Assessment Plan	Run Assessment Plan	
Distribute Resources		
Log	Log	Log
Notify	Notify	Notify
Package Contents	Package Contents	Package Contents
Handle Exceptions	Tutor Interface	User Interface
Allocate Mark		
Submit artifact	Student Interface	
Submit reviews		

Figure 6: Refactoring process in PeerPigeon

4.2. Case Study 2: ASDEL

ASDEL was an eighteen-month JISC (UK) funded project to construct a set of services to run on-line assessments specified in the Question and Test Interoperability (QTI) standard developed by the IMS Consortium. QTI is a leading assessment standard. The specification describes a data model for representing questions and tests and the reporting of results, thereby allowing the exchange of data (item, test, and results) between tools (such as authoring tools, item banks, test constructional tools, learning environments, and assessment delivery systems). The ASDEL tool, which delivers a QTI test, is called *Playr*. Figure 7 presents the conceptual design diagram for the *Playr*. The external R2Q2 service allows a student to view a question, answer a question, and view the feedback. The R2Q2 engine is itself a loosely coupled architecture comprising of three interoperable services.

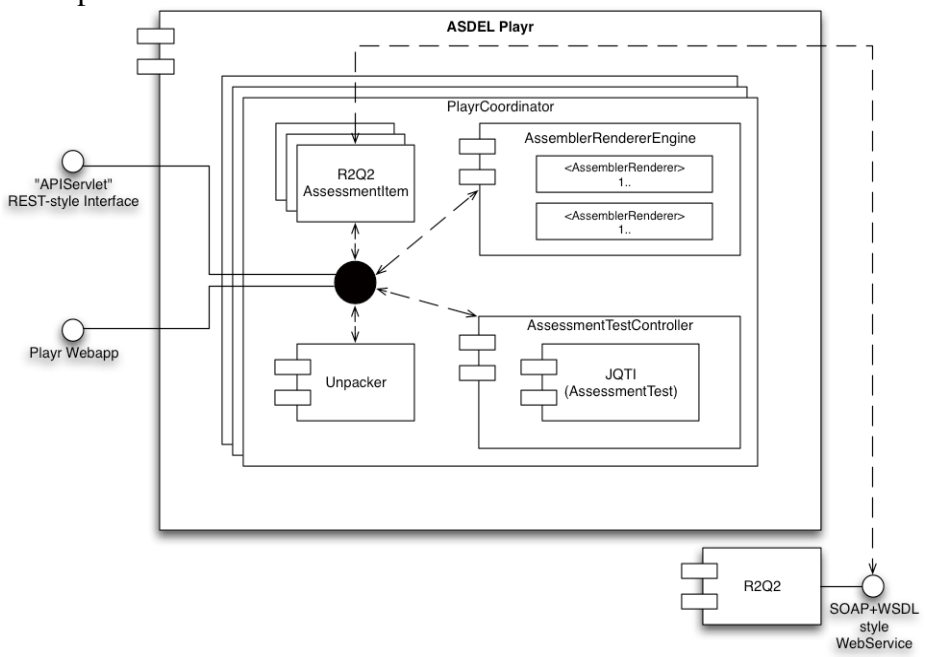

Figure 7: Architecture for the Assessment Delivery System

The original design for the ASDEL *playr* tool called for a number of small loosely coupled internal services communicating using SOAP. However, running load tests with tens of simultaneous users showed problems with quality of service. Using SRI-DM, the design was refactored into a simpler, more practical set of services. By redeveloping the services as internal components we removed the performance problems, and the system worked well in simulations with hundreds of simultaneous users. This supports the idea that small internal services can suffer serious performance issues due to the overhead of the service interface.

Figure 8 illustrates the performance differences between the original web service-based design and the componentized design (where the web services had been resolved into one service). The graph shows two sets of curves. *Throughput* is the number of requests the software is dealing with per second, and initially increases as a function of the number of users. It eventually peaks and then decreases as the server resources become exhausted (i.e. server runs out of available processing power, memory, file handles, etc). The *error rate* is the number of times the software fails to produce the expected outcome (for example, fails to load a page due to resource limits).

The curves on the graph clearly show that the componentized version of the *playr* (where the small services had been consolidated into one large service) performs much better than the fine grained web service version. The reasons for this somewhat dramatic improvement are numerous, but are mostly related to the reduction in memory and CPU resource usage from not having to continuously encode and decode SOAP XML messages.

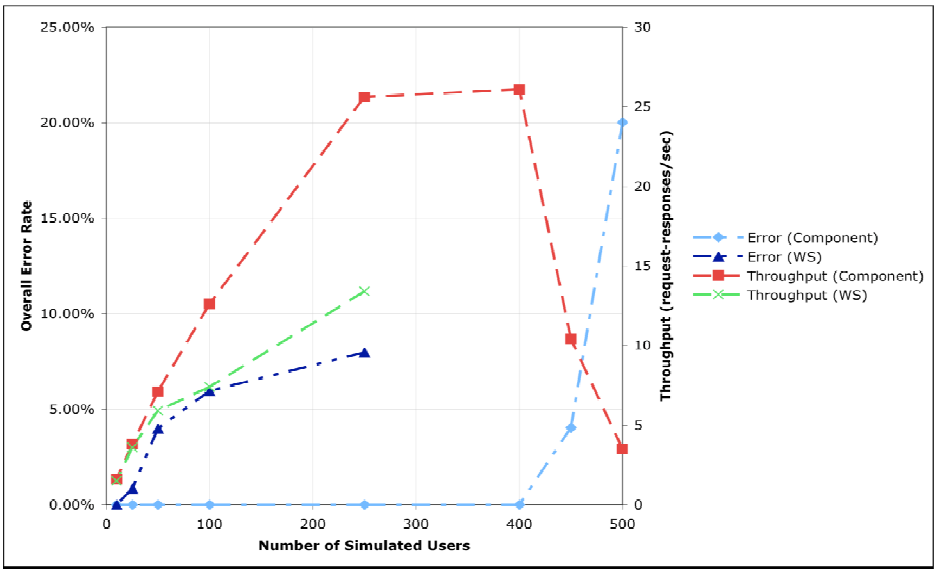

Figure 8: Performance of the original *playr* versus the improved design.

5. Conclusions

In this paper we have argued that there is a danger of over-engineering web service architectures, by creating designs with the wrong level of granularity. Highly granular service designs may seem to offer more benefits in terms of reusability, but in fact may be too tightly coupled in terms of their data models and the complexity of the service interactions. In these cases it is not worth paying the performance price of having many small services, and it makes more sense to refactor them into larger more effective services that hide the data and interaction complexity internally. We propose that using the agile Service Responsibility and Interaction Design Method (SRI-DM) can help to prevent the problem of over-engineering and result in more pragmatic web service designs, by helping to indentify and express complexity. In the method the scenarios are modeled as use-case diagrams, and the profiles as Service Responsibility and Collaboration cards (SRCs). SRCs capture the granularity of a service by defining its responsibilities, and the collaborations that it uses to fulfill those responsibilities. The SRI-DM includes a process of factoring abstract service profiles from formal domain scenarios. SRI-DM uses sequence diagram to show how the SRCs interact to fulfill the original scenario. This sequence diagram an example of one interaction that demonstrates the validity of the design.

We presented two case studies in which problems of tight coupling were addressed through the use of SRI-DM: PeerPigeon, where the data and state model were too complex to be effectively shared across a loosely coupled interface; and ASDEL, where the performance overhead of many small services was prohibitive and required a refactoring into more effective larger services.

The developers undertaking this case study are mixture of experienced and people new to agile techniques. We intend to carry out further study comparing the behaviors of differently composed teams. As SOAs become more reliable, and the standards underlying them more stable, it seems inevitable that they will form the basis of many distributed systems. If these systems are to be created as quickly and as flexibly as current software deployments then we must use design methodologies that are agile enough to cope with rapid turnaround, and that help us to create pragmatic solutions that take advantage of the benefits of SOA, but without sacrificing the effectiveness of the overall system.

7. References

1. Baresi, L., Heckel, R., Thöne, S., and Varró, D. (2003). Modeling and validation of service-oriented architectures: application vs. style. In Proc. of the 9th Euro SE Conf (Helsinki, Finland).

2.Beck, K. and Cunningham, W. (1989). A laboratory for teaching object oriented thinking. ACM SIGPLAN, Notices, 24(10):1-6, October 1989.

3.Benatallah B., Sheng Q., and Dumas M. (2003). The Self-Serv environment for Web services composition. IEEE Internet Computing, 7(1):40-48, Jan/Feb. 2003.

4.Curbera, F.; Duftler, M.; Khalaf, R.; Nagy, W.; Mukhi, N.; Weerawarana, S. (2002). "Unraveling the Web services Web: an introduction to SOAP, WSDL, and UDDI," Internet Computing, IEEE , vol.6, no.2, pp.86-93.

5.Dijkman, R. and Dumas, M. (2004). Service-oriented Design: A Multi-viewpoint Approach. Inter. Journal of Cooperative Information Systems 13(4), December 2004.

6.Fielding, R. T. and Taylor, R. N. 2002. Principled design of the modern Web architecture. ACM Trans. Inter. Tech. 2, 2 (May. 2002), 115-150.

7.Foster, I., Kesselman, C., and Tuecke, S. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. J. High Perform. Comput. Appl. 15, 3 (Aug. 2001), 200-222.

8.Highsmith, J. and Cockburn, A. (2001). “Agile software development: the business of innovation”. Computer, Sep 2001, Volume: 34, Issue: 9, pg 120-127, ISSN: 0018-9162.

9.Martin J., Arsanjani A., Tarr P., and Hailpern B. (2003). "Web Services: Promises and Compromises," Queue vol. 1, pp. 48-58, 2003.

10.Mukhi N. K. and Plebani P. (2004). "Supporting policy-driven behaviors in Web services: experiences and issues" in proc. of the 2nd Inter. Conf. on Service Oriented Computing ICSOC '04.

11.Olivier B., Roberts T., and Blinco K., (2005). "The e-Framework for Education and Research: An Overview". DEST (Australia). Downloaded 10 March 2007 from http://www.e-framework.org/Portals/9/Resources/eframeworkrV1.pdf

12.Quartel D.A.C., Dijkman R.M., and van Sinderen M.J. (2004). Methodological Support for Service-oriented Design with ISDL. In: Proc. of the 2nd ACM Inter. Conf. on Service Oriented Computing (ICSOC), pp. 1-10.

13.Wada, H., Suzuki, J., and Oba, K. (2005). Modeling turnpike: a model-driven framework for domain-specific software development. In Companion to the 20th Annual ACM SIGPLAN OOPSLA '05., New York, NY, 128-129.

14.Weatherley J. (2005). "A Web service framework for embedding discovery services in distributed library interfaces," in proc. of the 5th ACM/IEEE-CS Joint Conf. on Digital Libraries JCDL '05, Denver, CO, USA

15.Wilson, S., Blinco, K. and Rehak, D. (2004). Service-Oriented Frameworks: Modeling the infrastructure for the next generation of e-Learning Systems. A Paper prepared on behalf of DEST (Australia), JISC-CETIS (UK), and Industry Canada. Downloaded 10 March 2007 from http://www.jisc.ac.uk/uploaded_documents/AltilabServiceOrientedFrameworks.pdf

16.Larman, C. (2004). Agile and Iterative Development: A manager’s guide. Pearson Education.

17.JISC (2007). http://www.e-framework.org/.

18. Abbas, N., Gravell, A. and Wills, G. (2008) Historical Roots of Agile Methods: Where did “Agile Thinking” Come from? In: Proc. Of XP2008 Conf. Limerick, Ireland.

19. Highsmith J., Beck k., Cockburn A. and Jeffries R. (2001). "Agile Manifesto." from www.agilemanifesto.org.

20. Wills, G., Bailey , C., Davis , H., Gilbert , L., Howard , Y., Jeyes, S., Millard , D., Price, J., Sclater, N., Sherratt R., Tulloch , I. and Young , R. (2008) An E-Learning Framework for Assessment (FREMA). Assessment & Evaluation in Higher Education, 33 (4)

21. Ambler, S (2002). Agile Modeling, Wiley Publishing

Dr. David Millard received a BSc in Computer Science from the University of Southampton in 1997, and a PhD in Contextual Hypermedia Systems from the University of Southampton in 2001. He is now a Senior Lecturer of Computer Science at the University, is a member of the ACM Special Interest Group on the Web, and has published over 100 papers on hypertext, web and e-learning.

Dr. Yvonne Howard received a BSc in Computer Science from the University of Southampton in 1997, and a PhD in Evolutionary Software Process Modelling from the University of Southampton in 2004. She is a Senior Research Fellow in Computer Science at the University. She is the Project Manager for the Faroes project which has produced the Language Box. Her research interests include in the use of Agile Software Engineering techniques, particularly in placing users at the heart of the design process and using rapid feedback to build the capacity for innovation in the user community. Yvonne has published over thirty papers. She is a member of a UK JISC Special Expert Group that aims to provide a repository of contextualized models of the Higher Education domain to support innovation in the sector.

Noura Abbas received a BSc in Software Engineering from Al-Baath University in 2003 and an MSc in Software Engineering from the University of Southampton in 2006. She is now a final year PhD Student at the University of Southampton working on the impact of Agile methods on Software quality and customer satisfaction.

Dr. Hugh Davis gained a BSc in Ship Science from the University of Southampton (1981), before completing an MSc at City University (1987) and PhD at Southampton (1995) in Computer Science. He has worked as a social worker and teacher before starting an academic career at Southampton in 1987. He has a long history of research in Hypertext Systems and in Technology Enhanced Learning, with over 200 published papers, 3 best paper awards, and 30 grants. He is currently the University Director of Education responsible for e-learning at Southampton, and leads the Learning Societies Lab. His current interests focus on the use of technology as an agent for educational change. He is a member of the BCS and a professional member of the ACM.

Lester Gilbert is a Lecturer in Information Technology at the University of Southampton. He has published a textbook, Principles of e-Learning Systems Engineering, integrating his business-oriented practical experience of systems development with multimedia and Computer Aided Instruction to form the basis of his focus on e-learning and the use of technology in learning and teaching. Lester is the Principal Investigator on the JISC-funded REAQ and EFSCE projects, and a co-Investigator on a number of other projects including FREMA, EASiHE, mPLAT, MathsAssess, and R2Q2.

Dr Gary Wills is a Senior Lecturer in Computer Science at the University of Southampton. He graduated from the University of Southampton with an Honours degree in electromechanical engineering, and then a PhD in Industrial hypermedia systems. He is a Chartered Engineer and a member of the Institute of Engineering Technology and a Fellow of the Higher Educational Academy. He is also an visiting professor at the Cape Peninsular University of Technology, SA. Gary's main research interests are in Personal Information Environments (PIEs) and their application to industry, medicine and education. PIE systems are underpinned by Service Oriented Architectures, adaptive systems and advanced knowledge technologies. http://www.ecs.soton.ac.uk/people/gbw

Bob Walters worked for almost fifteen years working in commercial banking, before leaving to study Mathematics with Computer Science at University of Southampton. After completing his degree he worked for several years as a software developer before returning to Southampton as a research fellow in 1996. Since then he has completed his PhD in 2003 and is currently employed as a lecturer in the School of Electronics and Computer Science of University of Southampton.