DHQ: Digital Humanities Quarterly
2009
Volume 3 Number 4

Service-Oriented Software in the Humanities: A Software Engineering Perspective

Nicolas Gold <nicolas_dot_gold_at_kcl_dot_ac_dot_uk>, King's College London, Department of Computer Science

Abstract

Software Engineering, as a sub-discipline of the broader field of computer science, is concerned with the production, use, and maintenance of large, complex software systems. On first inspection, the set of managerial and technical activities involved in software engineering appears to be somewhat orthogonal to core research activity in the humanities, being concerned more with the production of research-enabling software systems than the research itself. However, as the scale of software used in digital humanities has increased, it is becoming clear that there are ways in which software engineering can inform, inspire, and aid in the management of the larger-scale software systems now being constructed in these disciplines. In particular, the development of service technology to aid in the production of flexible software systems for business now offers opportunities, not only for collaborative data sharing, but also the modelling, capture, provenancing, and replay of the research (and possibly creative) process itself.

This paper examines, from the perspective of a software engineer relatively new to the digital humanities, how the recent developments in service-oriented architectures could be used to enable new approaches to digital enquiry in the arts and humanities. The first part of the paper presents a brief history of software engineering, with particular reference to the aspects that have led to service-oriented architectures. In the second part, the paper offers some thoughts on how certain aspects of service-oriented architectures could be used to enable new kinds of computer-based research and practice in the arts and humanities. It also introduces important national initiatives in this area, such as the JISC e-Framework programme for Higher Education.

The Road to Services: An SE Perspective

Recent developments in the context of arts and humanities e-science have highlighted the possibilities of service-oriented approaches in arts and humanities research. These centre around new ways of describing and documenting research workflows, and in connecting academic users with the kinds of tools and data resources described elsewhere in this volume. Some of these methods and related projects were discussed at a series of international seminars, *Service Oriented Computing and the Humanities*, organized jointly by the AHRC ICT Methods Network http://www.methodsnetwork.ac.uk and the EPSRC Service Oriented Software Research Network http://sosornet.dcs.kcl.ac.uk/, the most recent of which was held in conjunction with the Digital Humanities 2008 conference in Oulu, Finland http://www.ekl.oulu.fi/dh2008/. 1

Looking back, one can see that as software systems began to increase in size and complexity, organisations increasingly came to depend on them and it became clear that the production of software was a discipline requiring far more than simply programming. In 1968, the term “software engineering” was first used to describe a particular branch of the nascent computing field concerned with building software systems on time and on budget [Naur 1969]. Software engineering (SE) is now defined as 2

> The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software... [IEEE 1990]

As such, SE encompasses both technical and managerial (perhaps also organisational and inter-organisational) processes and techniques.

Although much emphasis is placed on the delivery of new systems, the maintenance of existing software consumes at least 50% of the lifetime cost of a software system [Lientz 1980]. Constant and rapid organisational change leads to new requirements for functionality and changes to accommodate new operating environments, and the implementation of these changes creates bugs that must be fixed (perfective, adaptive, and corrective maintenance, respectively [Swanson 1976]). The increasing complexity of software has led to the development of abstraction mechanisms to allow the well-defined "packaging" of functionalities in such a way that larger-scale software can be built from these discrete elements. This allows the software engineer to maintain his or her understanding through manipulating the higher-level abstractions and ignoring the detail. 3

In all cases, these abstractions rely on the principle of "information hiding" first introduced by Parnas [Parnas 1972]. This principle establishes the idea that information about the way in which a section of a program (or a data structure) is implemented should not be made available to other parts of that program. The advantage of such an approach, beyond the fact that details can be abstracted away, is that other programmers cannot rely on the method of implementation to achieve a particular result, thus that method can be changed with no adverse effect elsewhere. From a software maintenance perspective, the "ripple" effect of a change is then minimised. 4

Initially, such abstractions took the form of programming language features like SECTIONS in COBOL (for more information on COBOL see the latest version of the international standard [ISO/IEC 2002]) where all variables are visible to the whole program (*global scope*) but functional code can be separated into blocks linked by PERFORM and GOTO statements. At this stage, the abstraction operated only on the sequence of code statements (so statements relating to a particular operation could be gathered together separately). As the field developed, other languages (e.g. C [Kernighan 1988]) were created to allow *local scope*, variables that exist only in the context of a particular block of code. Local scoping allowed a better segmentation of the program into functional units since data relevant to a particular computation (e.g. a temporary result) could be stored in a variable that existed only for the duration of the computation. Transfer of values from one part of a program to another takes place through global variables (those visible everywhere) or through a mechanism of procedure calls. The advantage of these "block-structured" approaches is that reusing a procedure elsewhere becomes nothing more than supplying data that conforms to the prescribed sizes, types, and number of data items required when calling it. In addition, if a change is to be made to the body of the procedure (e.g. perhaps one sorting algorithm is replaced with another) then no other part of the program is affected. The functionality is thus separated by an interface from the rest of the program. 5

Although functionality and data could now be separated, reused, and maintained more easily, data itself was still somewhat a second-class citizen since it was stored separately in databases and files. The development of *abstract data types* (ADTs) began to raise data to a first-class entity in programming, by going beyond simple procedures for operating on data (such as calculating with integers or splicing strings), to allow the definition of entirely new data types and the operations on them. Again, a well-defined interface was used as the means of communicating the necessary incoming and outgoing parameters. From here it was a small step to object-orientation and the modern programming languages used today. 6

It is now the norm for global variable usage to be minimised, for data to be managed inside an object through an interface that defines the operations and types of that data, and for data to persist (i.e. be stored beyond the timespan of a single program execution) behind that interface also. The encapsulation of data and function in this way has progressively allowed the construction of larger and more complex software as the amount of functionality captured in a single "chunk" of functionality has increased. 7

From an organisational perspective, the development of interface-oriented programming allowed companies to sell independent components of software for others to buy. This Component-Based Software Engineering (CBSE) approach (e.g. see [Szyperski 2002]) allowed organisations to buy-in pre-tested components of software with well-defined interfaces that could be integrated into systems under construction. 8

Despite all these advances in the construction of software systems and similar advances in the management of software projects, the problem of maintaining old legacy systems has become increasingly difficult. Making a change to a software system involves impact analysis (assessing the extent of a change and its impact on the rest of the system), design, implementation, regression testing (to ensure nothing that was working has broken), and upgrade management. These problems can be more complex for organisations relying on externally-sourced components since a change or update in the component (outside the customer's control) could impact internal systems. Equally, failure of the component supplier could lead to support for a potentially critical piece of software vanishing without warning. Many of the lessons of component-based software engineering are relevant to the service-oriented approaches currently in vogue. 9

Against this backdrop, in 1995 BT http://www.bt.com formed a Distributed Centre of Excellence (DiCE) in Software Engineering to study the future of software. This group identified a service-oriented approach as a way of increasing software flexibility through the apparently simple means of changing the emphasis of organisational IT from ownership to use [Brereton 1999]. The group envisaged a situation where software would be created by composing (through well-defined interfaces) a set of remotely-offered services, procured at the time of need to meet current requirements (see [Bennett 2001a], [Bennett 2001b]). When system execution was complete, the services would be de-coupled until the next execution. At that point, if the requirements had changed, a slightly different set of services would be procured, composed and executed. In this way, there would never be a single legacy system to be maintained and thus greater flexibility and rapid change would be available. This approach requires considerably more inter-organisational co-operation and exacerbates the problems first found in component-based software engineering when component (service) interfaces changed or their suppliers failed. 10

Web service technologies that support this, and similar approaches, have been developed since about 2000 and are now widely used (e.g. Web Service Description Language (WSDL) for endpoint description, Simple Object Access Protocol (SOAP) for messaging, and Web Service Choreography Description Language (WS-CDL) for choreography description). 11

In a sense, services represent the ultimate extension of the information-hiding principle. Procedures allowed the separation of related parts of code, scope rules allowed the separation of local and global data, ADTs and objects allowed related data and functionality to be associated and have persistence, and services now allow the hiding of the execution and location of the code and data behind a well-defined interface. 12

Key Principles of a Service-Oriented Architecture

Krafzig et al. define a service-oriented architecture thus: 13

> A Service-Oriented Architecture is a software architecture that is based on the key concepts of an application frontend, service, service repository, and service bus. A service consists of a contract, one or more interfaces, and an implementation. [Krafzig 2004]

Although this definition reflects its authors' terminology, it nonetheless sets out clearly the key elements required to adopt a service-oriented approach to software engineering. There must be a user interface (frontend), a populated collection of services (the repository), and a mechanism for connecting them (the bus). In much the same way as users/customers of commercial services need a frontend, repository and bus when using a Service-Orented Architecture, so researchers employing the academic research cycle have well-established needs for published outputs, formal research methods, and raw data. For each service, the contract describes informally what it does, the interfaces define this technically, and the implementation is self-explanatory [Krafzig 2004].

Whilst this approach is sufficient within a single organisation that has control over all aspects of the architecture and can work at any functional granularity in combining services, cross-organisational systems (such as those likely to be used in the digital humanities) also require standardisation in terms of the description of execution ordering (choreography and orchestration). Languages such as Web Service Business Process Execution Language (WS-BPEL) (OASIS 2007) and WS-CDL (defined above) (W3C 2005) provide the framework within which such knowledge can be captured, 14 represented, and used. Choreography allows each party to describe their role in an interaction whereas orchestration defines an executable process specifying how services work with each other from the perspective of a single party [Peltz 2003]. The two are complementary allowing both single-point and global views of the control and data flow through a process.

Services in the Humanities

15 The opportunities afforded by services have been recognised in many fields including the humanities since the promise of such enabling technology is very great. The JISC e-framework presents the utopian vision of a fully-connected, totally-interoperable environment in which data sources can simply be connected on demand in Higher Education [JISC 2007]. It aims to facilitate technical interoperability within the education and research communities through the use of, among other things, a service-oriented approach to system and process integration [JISC 2007]. However, it should noted for balance that many fundamental technical problems in services-technology have yet to be fully resolved, e.g.,

- reliable and predictable composition (in other words, a means by which services can be put together to achieve a known goal, every time, and with predictable characteristics)
- version control (the means by which revisions to services for bug fixes or new functionality can be managed and publicised to users)
- data description
- function description (the means by which the operation of a service can be described for both humans and machines)
- problem description (the means by which the operation or problem solved by workflows and choreographies can be described)
- performance
- and reliability

There has been a strong emphasis on encoding (rather than content) that has led to the development of many description and representation languages for web services. Less understanding has been developed of how best to use these. The problem of describing functionality has long been recognised and became particularly pertinent when component-based software engineering became more widespread. Describing the function of a service can be difficult because the way in which that description should be expressed often reflects the domain of application rather than the anticipated usage of the service-creator. Some progress has been made using ontologies but it is not clear that these will resolve all the outstanding issues.

16 Beyond the technical, major organisational and infrastructure issues have yet to be resolved. Although service-orientation is beginning to become more widespread in commercial IT, cross-organisational services are still relatively rare, particularly on a large scale, and most service-oriented implementations are intra-organisational. This is unsurprising since IT is business-critical to many organisations and the necessary trust and payment mechanisms have not yet matured sufficiently in the technical realm to be relied upon for ad-hoc commercial collaboration. When collaborating off-line, many hours are devoted to the construction of complex contractual agreements between organisations to ensure that obligations are clearly stated, can be monitored, and penalties applied in the event of non-compliance. These exist within the legal framework of the jurisdiction in which the contract is made. The international and multi-jurisdictional nature of the internet makes it very difficult to make such contractual arrangements, especially on an ad-hoc basis and monitoring is similarly difficult. On-line payment mechanisms for flexible and re-configurable tasks do not really exist yet. In addition, legal restrictions on the transmission and use of personal data limit the ability of organisations to collaborate in this way.

17 Organisations can, nonetheless, derive significant benefit from *internal* service-oriented systems. The nature of the architecture means that, for medium to large organisations, their software maintenance burden can be decreased through the incremental adoption and use of service-orientation (see [Krafzig 2004]).

18 By espousing an inter-organisational data- and process-sharing vision for the humanities, the field is placing itself at the forefront of research in service-oriented software engineering. It is likely that many of the challenges faced by

commercial implementations of services will also be faced by those adopting this technology in the digital humanities. However, this less commercial nature of services use may allow the necessary time and space to experiment and drive forward the field as a whole.

One further significant issue is that of long-term maintenance of the services-infrastructure. Commercial organisations have long recognised the risk of supplier failure (e.g. in component-based software engineering and now in inter-organisational services) and this has to some extent restricted the adoption and use of these technologies. If a component supplier fails, the customer organisation is usually insulated for a short period from the effect of this by virtue of owning the executable code and thus being able to continue operating their system even if they are unable to change it as rapidly as desired. In a service-based software system, the effect of supplier failure is immediate since workflows using the services offered will be unable to continue. 19

What is being proposed for the digital humanities (by visions such as the e-framework [JISC 2007]) is effectively a very large, multi-organisational, distributed and somewhat uncoordinated IT infrastructure, where many service providers are autonomous and with differing “business” goals. There has been some work aimed at data integration in such situations, for example, the IBHIS project (see [Kotsiopoulous 2003]). This project developed a broker for integrating healthcare data sources on-the-fly using a meta-ontology and registry [Kotsiopoulous 2003]. Although there have been proposals for adapting and enhancing such approaches for applications in the humanities, the field is by no means mature in this respect [Dunn 2007]. 20

Long-term support for the archives and services created as this technology is adopted is vital to carry out the software maintenance that will, as long experience in the software evolution field has shown, be necessary to sustain the infrastructure. In many respects, this is more critical in an academic field than commercially. An organisation finding that it no longer has a current need for a particular piece of IT can retire it without significant loss, thus freeing resources for new developments to support current business needs. It would be considerably more difficult to plan obsolescence in an academic field where services encapsulating data may have lain dormant for some years before being found to be critical to some enquiry in the future. Organisations providing service-oriented access to data, archives, or functionality will therefore need to ensure that a reliable delivery platform exists in the long-term. Moreover, they will also need to make provision for adaptive maintenance to keep pace with changing interconnection languages, and also to meet perfective maintenance requests for exposing that data or functionality in new ways. Without such planning and long-term support, there is a risk that considerable investment will be made in service-oriented “island” solutions, precisely the type of solutions that service-oriented architectures are designed to avoid. 21

Beyond Building the Systems

Having discussed how software engineering experience has and continues to provide a perspective on IT in the humanities, the next sections set out some ideas for the adaptation of services-technology to become part of the research and creative processes themselves. 22

Contribution to the research process

SOAs offer the opportunity to go beyond just providing the infrastructure for sharing resources, allowing the steps of a research process to be made explicit and reused in the form of a workflow or choreography. For example, by assembling appropriate functional and data services using a workflow expressed in WS-CDL, the research method itself is documented in use. Service-based systems for undertaking research thus become self-documenting and repeatable. In addition to providing the capability for transferring traditional research methods to the digital realm, this offers the opportunity to develop new research methods that can only be used with online resources. These can then be studied, criticised and improved because their representation is explicit. Similar approaches have been used to great effect in scientific discovery where the process of reaching a result can be as important as the result itself (e.g. [Syed 2006] and [DeRoure 2008]). Using an explicit representation also allows for versioning and provenancing to allow a historical record of the development of such techniques. Many of these ideas are embodied in the myExperiment platform (see [DeRoure 2008]). The application of similar principles has also been discussed in the context of archaeology [Dunn 23

2007].

Current technology, whilst sufficient to describe the necessary control and data flow conditions to connect services, may need to be extended to allow the expression and representation of domain-specific constraints. For example, a text-processing service pipeline designed to extract metadata under a given methodology may need to have constraints applied to describe the kinds of text sources or services to which it may be applied. This kind of constraint goes beyond simply describing the data type and format and relates more to the semantics of the domain of application. 24

Contribution to Creative Practice

Moving even further from simple infrastructure provision, an additional step beyond representing research methods as service-workflows is to involve services and workflow definitions in creative practice, thus creating a self-documenting record of the creative process. The naturally distributed paradigm may also allow for new forms of collaboration. 25

Whilst not applicable to all artistic forms, one can imagine digital visual art created from the successive application of services (perhaps written by one artist, perhaps by many), the sequence, repetition, and conditions of service application being defined using workflow languages by the artist “composing” the work. Since services do not need to exist locally, complexity in image rendering, for example, could take place on-the-fly at a remote, high-power computing facility with the results delivered to the artist. A forerunner of this kind of approach can be found in Whitbread’s “In the Womb of the Rose” project where artists collaborate by placing images using placeholder slots to form an artwork as a whole [Whitbread 2007]. A service-oriented approach would allow alternative creative flexibility since collaboration could take place in the form of a re-arrangeable production process (i.e. the software embodying methods for producing art has been created as services), not just the results. Indeed, it is possible to envisage distributed, interactive installations based on services technology where the execution of the service functionality may not be fully automated (the information hidden behind the interface being the presence and interaction of a human artist, rather than software). Although approaches to distribution have been previously used (e.g. distributed performances and installations using Open Sound Control [CNMAT 2007]), the use of standardised service languages would allow the easier study of the artistic process as well as the resulting artwork. 26

Essence and Accidents

This paper has envisioned a world where fine-grained services can be composed to allow new possibilities for enquiry in the digital humanities. It is an attractive vision with the potential to transform the nature of research in this field. However, consideration should be given to the possible effect (or lack of effect) of such a change. 27

In 1987, Brooks analysed the field of software engineering and technological development therein [Brooks 1987]. He characterised software engineering in terms of its essence (aspects intrinsic to software itself), and its accidents (aspects related to the production of software but not inherent in it). Brooks’ position was that, no matter what technological means may be brought to bear on the production of software, it would remain an essentially conceptual and difficult activity. Progress in software engineering has largely borne out this position. 28

It is possible, therefore, that despite the apparent potential of services, they are only the next step in a line of “accidental” developments in the software engineering field. If so, then they will only achieve a small improvement in system flexibility. 29

However, system flexibility is only one advantage posited by this article. A case has also been made for the opportunities to transform and record the research process in digital humanities using service-orientation. Whether services will have this envisaged impact on the digital humanities will depend to some extent on what might be seen as the “essence” and “accidents” of enquiry in the digital humanities. There is no question that digitisation offers many advantages, but the granularisation inherent in service-orientation may actually restrict rather than enhance research practice. A comparison might be drawn between modelling with clay or plastic construction bricks. Both achieve similar ends but have different characteristics, advantages and disadvantages. Modelling with clay offers limitless possible shapes for the end result but it takes a comparatively long time to reach simple shapes. Modelling with bricks is quick, 30 especially for simple shapes but ultimately there is less flexibility. It is clear that, whatever advantages may be conferred by technological developments such as services (such as quick access and integration of data), there will always be a role for the individual researcher to deal with the “essential” enquiry. The insight, intuition, knowledge, and expertise that the researcher brings to bear on a research question (the “essence”), combined with the ability to digitally encapsulate and combine data and methods (the “accidents”) offers great potential for the future of digital humanities.

Summary

This paper has attempted to draw together key concepts from software engineering, chart their development, and discuss their application to the digital humanities. In particular, the principle of information hiding and its embodiment in service-oriented architectures is discussed and related to possible applications in both research and creative practice. The paper has also discussed the organisational and inter-organisational implications of widespread adoption of service-technology in the digital humanities. 31

Acknowledgements

As a relative newcomer to the digital humanities, I am very grateful to the many people with whom I have discussed problems, solutions and ideas over the past year. In particular, I would like to thank Lorna Hughes, Stuart Dunn, Neta Spiro, John Rink, Nicholas Cook, Daniel Leech-Wilkinson and Craig Sapp for the many useful meetings we have held. 32

Works Cited

Bennett 2001a Bennett, Keith, Malcolm Munro, Nicolas Gold, Paul Layzell, David Budgen and Pearl Brereton. “An Architectural Model for Service-Based Software with Ultra Rapid Evoluation”. Presented at *ICSM 2001*. *17th IEEE International Conference on Software Maintenance* (2001).

Bennett 2001b Bennett, Keith, David Budgen and Pearl Brereton. “An Architectural Model for Service-Based Flexible Software”. Presented at *IEEE Computer Society*. *Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development* (2001).

Brereton 1999 Brereton, Pearl, and David Budgen. “The Future of Software”. *Communications of ACM* 42: 12 (1999), pp. 78-84.

Brooks 1987 Brooks Jr., Frederick P. “No Silver Bullet: Essence and Accidents of Software Engineering”. *IEEE Computer* 20 (1987), pp. 10-19.

CNMAT 2007 The Center For New Music and Audio Technology (CNMAT), U. B. (2007). “Open Sound Control.” Retrieved 21st September 2007, from http://opensoundcontrol.org.

DeRoure 2008 De Roure, David, Carole Goble and Robert Stevens. “The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows”. *Future Generation Computing Systems* (2008).

Dunn 2007 Dunn, Stuart, Nicolas Gold and Lorna Hughes. “CHIMERA: A Service Oriented Computing Approach for Archaeological Research”. Presented at *CAA 2007*. *Proceedings of the Computer Applications and Quantitative Methods in Archaeology Conference* (2007).

IEEE 1990 Warning: Biblio formatting not applied. IEEE. *IEEE Standard Glossary of Software Engineering Terminology, IEEE Standard 610.12-1990*. 1990.

ISO/IEC 2002 Warning: Biblio formatting not applied. ISO/IEC. *ISO/IEC 1989:2002 Information technology - Programming languages - COBOL International Organization for Standardization/International Electrotechnical Commission*. 2002.

JISC 2007 Joint Information Systems Committee. *e-Framework for education and research*. http://www.e-framework.org/.

Kavantzas et al 2005 Kavantzas, Nickolas, David Burdett, Gregory Ritzinger, Tony Fletcher and Yves Lafon, eds. *Web Services Choreography Description Language Version 1.0*. W3C, 2005. http://www.w3.org/TR/ws-cdl-10/.

Kernighan 1988 Kernighan, Brian W., and Dennis M. Ritchie. *The C Programming Language*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

Kotsiopoulous 2003 Kotsiopoulos, Ioannis, John Keane, Mark Turner, Paul Layzell and Fujun Zhu. “IBHIS: Integration Broker for Heterogeneous Information Sources”. Presented at *COMPSAC 2003*. *Proceedings of the 27th Annual*

International Computer Software and Applications Conference (2003).

Krafzig 2004 Krafzig, D., K. Banke, et al. (2004). *Enterprise SOA: Service Oriented-Architecture Best Practices*, Prentice Hall PTR.

Lesk 2004 Lesk, Michael. *Understanding Digital Libraries*. Morgan Kaufman, 2004.

Lientz 1980 Lientz, B.P., and E.B. Swanson, eds. *Software Maintenance Management*. Addison-Wesley Publishing, 1980.

Naur 1969 Naur, P. and B. Randell, Eds. (1969). *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968*, Brussels: Scientific Affairs Division, NATO.

OASIS 2007 OASIS. (2007). “Web Services Business Process Execution Language Version 2.0.” Retrieved 21st September 2007, from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

Parnas 1972 Parnas, D.L. “On The Criteria to be Used in Decomposing Systems into Modules”. *Communications of the ACM 5* 12: 5 (1972).

Peltz 2003 Peltz, C. (2003). Web Services Orchestration and Choreography. 36: 46-52.

Swanson 1976 Swanson, E. B. (1976). *The Dimensions of Maintenance*. Second International Conference on Software Engineering, San Francisco.

Syed 2006 Syed, J., M. Ghanem, et al. (2006). “Supporting Scientific Discovery Processes on Discovery Net.” *Concurrency Computat.: Pract. Exper.* 19 (2).

Szyperski 2002 Szyperski, C, and D. Gruntz. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley/ACM Press. 2002.

Whitbread 2007 Whitbread, D. *In the Womb of the Rose*. 2007. http://www.wombrose.co.uk.