Usability Evaluation for Enterprise SOA APIs

Jack Beaton, Brad A. Myers, Jeffrey Stylos, Sae Young (Sophie) Jeong, Yingyu (Clare) Xie

Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
412-268-8277

jackbeaton@cmu.edu

ABSTRACT

SAP recently began offering access to web services through its Enterprise Service-Oriented Architecture (E-SOA) platform. It is in the best interest of SAP that its E-SOA service operations are easier for developers to use and understand, which will contribute to higher E-SOA adoption, and a more effective means of innovation on the part of business customers. To facilitate such a change, Carnegie Mellon University's Human-Computer Interaction Institute is working with SAP's E-SOA and Business Process Renovation Teams to analyze the E-SOA interfaces using HCI techniques and determine means by which developers assigned to create SOA APIs in general, and Enterprise SOA APIs in particular, can design superior interfaces. The identification of usable design patterns, and methodologies to determine these patterns, can streamline SOA projects for API developers and programmers who use SOA APIs.

Categories and Subject Descriptors

C2.4. [Distributed Systems]: Client/server; H5.2. [Information Interfaces and Presentation]: User Interfaces; D.2.6 [Programming Environments]: Integrated environments;

General Terms

Documentation, Design, Human Factors, Languages.

Keywords

API Design, Service-Oriented Architecture, Web Services, Enterprise SOA, Usability, SAP

1. INTRODUCTION

The interface between the developer and the library being utilized is known as the Application Programming Interface (API). Usability issues arising from tradeoffs in API design have severe implications for the productivity of developers. By studying API users, Human-Computer Interaction (HCI)

techniques may be applied to evaluate and improve APIs. In our previous studies, we have successfully accomplished this, while also designing effective and valid methodologies for measuring the effects of API design choices [5, 14, 15].

By analyzing the E-SOA web services APIs, SAP hopes to increase API usability, and therefore the speed, efficiency, and viability of projects which create SOA composite applications. To this end, we are adapting Human-Computer Interaction methodologies and techniques to inform and support the design of SOA APIs.

Although many resources focus on the design of web services, few tested design guidelines exist to enable speedy, intuitive, and effective discovery and consumption of web services by programmers [2, 6, 13]. It is in the best interest of SAP and other web services API designers to increase API usability, so as to reduce the costs and increase the effectiveness of all SOA projects using these APIs.

SAP's E-SOA usability effort faces interesting and novel challenges that may yield general solutions to other SOA API design efforts. First, SAP possesses a massive backend used to create composite applications, which require substantial complexity. There are over 1700 web services available, and these services are grouped in at least two hierarchies: one by licensed process component (Enterprise Resource Planning, Customer Relationship Management, and more) and the other by likely scenario of use. Relationships between service operations complicate their use, and are not always clear. Little is known about how to organize large API frameworks to maximize usability. We are studying SAP's Enterprise SOA service hierarchies to better understand this challenge.

Second, SAP's users may experience conceptual challenges that are not normally anticipated or understood. Part of SAP's business model relies on users licensing software that is installed on servers owned by the customer, resulting in the interesting situation that the service provider and consumer are technically the same, although the users did not create and are not necessarily familiar with the services. The Enterprise SOA user experience is one of both the provision, and the consumption, of hundreds of available services. We plan to explore the usability implications of SOA's asymmetry between creation and consumption.

Third, current stub generators result in API designs that mimic the creation of XML documents, a form of programming that is inconsistent with what is commonly used in the Java and .NET environments. It may be necessary to adhere to document-style Web Service Description Language (WSDL) conventions because the Remote Procedural Call (RPC) format, designed to encourage consistency with non-distributed method calls, is criticized as too tightly-coupled to be interoperable enough to support SOA [1, 9]. However, the design flexibility of stub generators raises the possibility that XML document-style API design does not necessarily need to be imposed by document-style WSDL conventions. Usability research into the generation of API designs from WSDL may result in improvements that could speed the adoption of both E-SOA and SOA as a whole.

Fourth, use of APIs is significantly affected by the IDE used, as well as the documentation provided. After the needed service is located in the documentation, stub generation tools integrated into the IDE are required to produce the client-side code needed to call it. Support for both discovery of services and their technical configuration with the user's IDE is necessary to ensure intuitive consumption.

By examining web service provisioning and consumption in an Enterprise SOA environment, we hope to identify pain points, barriers, and means of improvement that could result in generalized solutions, and support API designers across other circumstances.

This paper begins by describing the usability challenges that have been identified so far in our ongoing research, and next describes HCI techniques that are being altered specifically to support API designers in judging design tradeoffs and making informed and justified decisions. We conclude with topics for future research.

2. USABILITY CHALLENGES
Multiple challenges face consumers of E-SOA services, including: the difficulty of web service configuration, fragmentation of the discovery user experience across multiple interfaces, confusing hierarchies of service navigation, business modeling support, hidden relationships between services, and inconsistent API design.

2.1 Difficulty of Web Service Configuration
There are multiple means by which web services may be provided and consumed, each with different implications for the usability of the resulting API design. These can be described in four dimensions: the format of the WSDL, the stub generation method, the stub provision method, and the API design of the resulting library.

- **Document versus RPC** – Although SAP supports both of these WSDL formats, the document method allows use across all languages, while the Remote Procedural Call (RPC) style creates a more consistent API design at the expense of interoperability [1, 9]. However, the use of

document-style does not itself impose a lack of consistency – that depends on the generated API design of the stub.

- **Generation method** – For a programmer to consume web services, code generation engines are necessary to interpret WSDL files. However, these engines may not be compatible with the development environment or the web services that a developer is using, and difficulties with installation are common. Even after installation, configuration is a complex task that restricts the use of web services to people with specialized knowledge, adding substantial time to the accomplishment of otherwise straightforward development tasks. It is possible to assign this task to the web service provider, or the web service consumer (see below).

- **Downloaded or Homemade** – Web services may be accessed via a WSDL file that is then used to generate a service, or a generated stub library may be downloaded in the language of the user's choice. This may make the service more accessible to users without the expertise or time needed to configure stub generators, but the target language and version must be identified in advance.

- **Consistency** – Stub generators included in the Axis SOAP engine and in Visual Studio .NET convert document-style WSDL files into an API design pattern that more closely resembles writing XML than the common API patterns used by C, Java, and C# libraries. This design pattern may not be necessary, and our usability testing may identify issues that could be globally impeding SOA adoption industry-wide.

In the case of our E-SOA project, developers are assumed to be reasonably proficient with web services, and so the bulk of our web services configuration research will focus on methods to improve the usability of generated APIs, and the implications upon SOA usability and adoption.

2.2 Multiple Interfaces for Discovery
Although UDDI (Universal Description, Discovery and Integration language) is valuable as a standard, the practical needs of the service provider and the service consumer differ, and this is not always reflected in the interfaces that rely on UDDI. Providers are mainly concerned with service governance; but consumers are interested in service discovery. It is not necessary that the interfaces for governance and discovery be at all unified or even similar in appearance, so long as the underlying registry remains centralized. To a service consumer, any means of accessing either a WSDL file or generated stub is a form of web service discovery.

Complicating the situation, SAP E-SOA customers are technically involved in governing and providing the same web services that they discover and consume, and so SAP must support interfaces for a user base that shifts perspective between consumer and provider.

SAP currently provides three different E-SOA interfaces for discovery:

- **Enterprise Services Registry** – This is an interface directly into the registry itself. While useful for certain functions, it is difficult to use for effective discovery and consumption because it does not include documentation describing the service, its intended use, its parameters, or any other information besides the WSDL file itself.

- **Enterprise Services Wiki** – In order to harness the power of online communities, documentation in the form of a wiki is available, providing more informal descriptions, including a bundling pattern of services meant to be informed by the community (see below).

- **Enterprise Services Workplace** – This valuable resource comprises the main documentation of the API. In addition to detailed explanations of E-SOA services, links to download WSDL files are available. However, the ES Workplace is currently only loosely integrated into hosted registries, introducing complexity.

Linking these interfaces together into a unified consumption experience for developers is a significant challenge that will require interesting innovations.

2.3 Hierarchies of Service Navigation

While browsing the 1700+ service operations available from SAP in the Enterprise Services Workplace, service consumers are presented with multiple forms of hierarchy. Service operations are classified in the Enterprise Service Index by original SAP process component (ERP, CRM, SRM, SCM, and other add-on components such as HCM) and also a new categorization method called the Enterprise Service Bundles, designed to reflect needs of service consumers presented with similar scenarios of use. In addition to these two main indexes, services that are available under industry-specific licenses are presented in the form of solution maps. It is unclear what effect these multiple hierarchies and presentation methods have on service discovery and consumption.

The complete Enterprise Service Index is a classification by process component, which is a hierarchy familiar to developers with an existing background in SAP. However, it is a goal of SAP to use Enterprise SOA to expand its user base to include developers unfamiliar with SAP products. Attempts to make the hierarchy more presentable to novice SAP developers may alienate expert developers by removing their familiar reference points.

On the other hand, the Enterprise Service Bundles are designed to group service operations into different scenarios of use. To encourage community definition of the Bundles, the lists of service operations that comprise them have been placed within the Enterprise Service Wiki. This presents the unusual activity of taking a detour through a wiki on the way through the documentation.

Finally, the solution maps represent a totally different browsing strategy, and exist separately from the Enterprise Service Index service operations. It is unclear how this may affect the browsing patterns of unfamiliar users.

2.4 Business Modeling for Enterprise SOA

Prior to the implementation of Enterprise SOA solutions, business process experts and architects must cooperate to model the intended automation, first by identifying business needs, and then by discovering and specifying the service operations that will be appropriate for the task. Business modeling must therefore proceed in parallel with discovery of enterprise web services. SAP's service documentation is designed to encourage an understanding of enterprise services for both business process experts and programmers. Programmers also may approach the documentation from backgrounds in different programming languages. The ESOA API documentation must therefore be understandable for programmers of multiple different languages, as well as for business process experts. The wide variety of audiences places conflicting demands on the design of the documentation, as each audience brings its own language and interpretation.

We intend to identify guidelines to improve the discovery and consumption of these services through the various SOA interfaces within this unique collaborative environment.

2.5 Backend Service Relationships

In addition to the complications above for each individual service, operations in multiple services may be linked together by invisible relationships in the hidden server backend, with detrimental results to developer productivity. For example, when synchronous write and read services depend on a hidden asynchronous relationship, changes to values in business objects will seem to appear after an inexplicable delay that can cause race conditions in client code written by developers unaware of the asynchrony.

As a result, there are multiple conditions under which a developer may become frustrated for lack of knowledge of the SAP backend architecture. Given the complexity of the SAP process components, it is unreasonable to expect that the backend relationships can be made clear for all developers. Instead, new design strategies are needed to simplify and abstract architectural complexity and enable developers to access the maximum amount of functionality with minimal information. The impact of these guidelines on the specifications used by the API designers must be provided in a manner that is comprehensible and useful to them. In the distributed SOA environment, these research challenges are highly relevant and essential to usability and adoption.

2.6 API Usability Evaluation

Just as in non-SOA programming, the consequences of API design can be far-reaching, yet little usability research has been dedicated to this subject. Recent research for non-SOA APIs has identified the effects of certain design patterns, such as the use of factory constructors having a negative effect on the productivity of developers [5, 14]. The SOA context presents new and interesting API design challenges to resolve.

The capability of accessing functionality across languages can expose forms of logic common in the provider's language, but nonexistent in the language of the developer. Developers may need to adapt to the mental models of a new and unfamiliar language, while still using their own. The best practices of service design under these conditions must be identified to ensure the usability of SOA APIs.

An example of this in the E-SOA API is the proliferation of optional fields, which are a possibility supported by ABAP, the SAP backend development language. Filling out various combinations of fields in a service operation will result in different results – but it is difficult to communicate which combinations accomplish what result. In this case, is it more effective to hide this form of functionality, or to determine a means by which field dependencies can be effectively documented and communicated? If the latter, what visualization would be most sensible? We are planning studies of this and other similar issues, to help inform SOA API design in ways that are generalizable and useful to software designers providing services across languages.

The products of such research will allow developers to quickly recognize common API design issues both by sight in an API and when observing API users, and also to quickly consider the tradeoffs of one or more possible solutions so a confident recommendation for improvement can be made.

3. HCI USABILITY TECHNIQUES
In order to identify usability issues and choose appropriate guidelines and recommendations, a variety of HCI techniques are currently being vetted for use. These techniques are tried and true methods of exploring interface design issues, determining their severity, and identifying solutions. Although they are predominantly applied to Graphic User Interfaces, they have all been altered to reflect the more in-depth environment experienced by developers, and address the unique challenges of evaluating API designs

3.1 Think Aloud Protocol for API Testing
Think Aloud Usability Evaluation is the gold standard of usability testing, but can be difficult to apply when in an unconstrained environment, such as during programming. If subjects can potentially choose any of a wide array of optional solutions, none of them obviously wrong, testing can be very time-consuming without identifying definite usability problems.

At Carnegie Mellon, methodologies for effectively performing Think Aloud studies on developers are currently being refined with a significant degree of success [5, 14].

Innovations include using a simple and highly-bounded programming task, and first having participants write pseudocode that the user expects to find in the API, and then having the user actually code the task using the real API using its documentation and IDE. This effectively reverses the standard methodology of first observing real use of the tool and then trying to guess what the user must have been thinking

after the fact, by identifying the user's original mental model, and then gauging how flexibly that model fits the circumstances.

During the study, the moderator must pay closer attention to the user than is often necessary during the testing of graphical interfaces, in order to deduce the mental model of the developer. It is not enough to simply record all errors that appear on the screen. Adept programmers may explore, so making an error is not necessarily representative of a task failure, but may be an intentional learning step.

Finally, the size of the programming design space calls for a bias towards a wider variety of tasks, to ensure that the conclusions are widely applicable. Time to task completion is an important indicator in the development setting, and also a useful quantitative metric [5].

3.2 Expert Evaluations
Pioneered by Jakob Nielsen, industry-standard *Heuristic Evaluation* depends primarily upon a list of ten heuristics, each reflecting an archetypical problem that can be identified by its symptoms in such a straightforward manner that the solution also becomes clear [11].

However, the training associated with applying this list is almost exclusively dedicated to evaluating GUI-type interfaces, making the technique difficult to use for API design. Heuristics particularly relevant to APIs include Consistency and Standards, Error Prevention, and Help and Documentation, with other heuristics being less applicable, or may require new interpretations that are not yet substantiated in research.

We are studying ways to develop and reinterpret the Nielsen heuristics in such a way that programmers could be trained to use them to improve styles of programming during development, when such changes will be cheapest and easiest to make, while also providing multiple real-life examples of the consequences of forgoing the recommended improvements.

Evaluation by *Cognitive Dimensions* [7] is a similar technique that has previously been used to identify tradeoffs in the design of programming languages. These measurements, including Consistency of patterns, Diffuseness of jargon, and Viscosity of program structure, provide a language that can describe solutions to conceptual barriers in API design, while pointing to side effects of those design decisions. [3, 4, 8, 10]

3.3 Cognitive Walkthrough for API Testing
The cognitive walkthrough technique is performed by obtaining assumptions about an ideal type user, possibly assembled into personas from gathered user data. Knowledge of jargon that the user may or may not be familiar with is especially valid. From this information, a series of four questions can be answered with each step of a task specified within a certain interface, referencing the user's goals, perceived options, chosen action, and conclusions [12]. Although admittedly subjective to an extent, the technique is

useful in circumstances where a "real" user cannot be found, either due to inaccessibility or simply because the product is so new that few users exist.

However, these questions become less applicable for programming tasks, because a development environment rarely has a single set process to solve any problem. Furthermore, there is no way to prove that any given group of coding actions should proceed in a certain order. Therefore we rephrased the questions to eliminate the black-and-white labeling of "good" and "bad" actions, and instead are used to predict, in this order:

- The Goal of the user when presented with this scenario;

- What option(s) are Perceived by the user;

- Which perceived option(s) seem most Action-able;

- What Result the user will conclude has occurred.

This alteration allows the designer more flexibility in judging how users will be predicted to behave, and discusses the prediction in terms of what mental model the designer wants to encourage the user to adopt, instead of what the user must do.

In addition, by eliminating the need for steps, it is possible to assert simple scenarios using an "encounter-based" method, in which a simple scenario describes a possible set of circumstances the user may encounter, disregarding set linear procedures, and then user reactions are predicted.

4. APPLICATION OF HCI TECHNIQUES

By definition, any barrier that prevents the user from using a product, tool, or service is a form of usability breakdown. While breakdowns may frequently result from technical issues, psychological and cultural barriers are also sometimes a contributing cause for a failure in the marketplace. Therefore, the use of the above techniques is not limited to only technological barriers to understanding, but also includes issues such as that of collaboration between programmers and business experts that are often not interpreted as within the mandate of developers.

Initial results from the use of Think Aloud studies point to severe conceptual barriers. Some appear to be triggered by a common stub generation method that exposes the structure of the SOAP XML messages in the code. Others may be caused by subtle inconsistencies between web service testing utilities and the web service APIs. These usability problems cannot be easily categorized into the issues discussed above, and are topics for future work.

5. RESEARCH CHALLENGES

Our research challenges for future work can be summarized into five areas: the design of the API itself, the usability of client-side tools, the effectiveness of the documentation, the meta-level questions of how best to alter HCI techniques to answer API design questions, and finally how to present HCI-derived recommendations to API designers.

For the design of the API itself, the first question is what challenges do SOA APIs face that non-SOA APIs do not? How can hundreds of existing systems be combined and managed in a SOA environment? To what degree must backend complexity be exposed through web services, and what effect does this have on service consumers? In what way can SOA and pre-SOA API design methodologies inform each other, and where are they different?

Current IDEs are not designed specifically for SOA, and stub generation is a common barrier to usability. What problems occur repeatedly, and can they be avoided? How can we gauge the relative effectiveness of stub generators, and how should they be redesigned if necessary? How can users be provided with the best tools for consumption of stubs generated for SOA, and is this worth the investment on the part of API designers?

SOA documentation faces the interesting challenge of having to bridge every programming language that is intended to access the web services. In the case of enterprise applications, business process experts are added to the audience of developers. How can documentation be designed to be equally useful to every reader? Where documentation is interactive, such as through the use of service test interfaces, what design best supports the work of developers?

API usability studies, relative to usability studies of graphic user interfaces and physical controls, are little explored in HCI. How can HCI usability techniques add value in the world of API design and development? If existing techniques are inappropriate, what alterations should be made?

Finally, it is important to present HCI research and design recommendations in a way that is useful and valuable to developers. How can usability-driven design recommendations be sensibly weighed against the tradeoffs, such as cost to fix, and possible alienation of the existing user base? How can usability data be presented in a way that is most usable to developers?

All the above questions are avenues of possible research, and we propose to address some of them in future studies.

6. CONCLUSIONS

By adapting Human-Computer Interaction techniques to the unique demands of software development tasks, valuable insights can be gained to help improve E-SOA web service API design, increasing the rate of SOA adoption, lowering the costs of business process redesign, and speeding business innovation. We hope that these insights will generalize to help others create more usable SOA APIs.

ACKNOWLEDGMENTS

REFERENCES

[1] Akram, A., and Meredith, D., "Approaches and Best Practices in Web Service Style, Data Binding, and Validation," 2006. http://epubs.cclrc.ac.uk/bitstream/1542/AsifAkramDave MeredithChapter.pdf

[2] Artus, D. J. N., "SOA Realization: Service Design Principles," 2006. http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design/

[3] Clarke, S., "API Usability and the Cognitive Dimensions Framework," 2003. http://blogs.msdn.com/stevencl/archive/2003/10/08/5704 0.aspx

[4] Clarke, S., "Describing and Measuring API Usability with the Cognitive Dimensions," in *Cognitive Dimensions of Notations 10th Anniversary Workshop*. 2005. www.cl.cam.ac.uk/~afb21/CognitiveDimensions/worksh op2005/Clarke_position_paper.pdf

[5] Ellis, B., Stylos, J., and Myers, B. "The Factory Pattern in API Design: A Usability Evaluation," in *International Conference on Software Engineering (ICSE'2007)*. May 20-26, 2007. Minneapolis, MN: pp. 302-312.

[6] Goetz, J. "Methodology for Service Identification and Service Design," 2007. Diploma Thesis, Berufsakademie Mannheim Staatliche Studienakademie University of Cooperative Education, Mannheim, Germany.

[7] Green, T.R.G., "Cognitive Dimensions of Notations," in *Proceedings of the Fifth Conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and computers V*, 1990. Univ. of Nottingham: pp 443-460.

[8] Green, T.R.G. and Petre, M., "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework." *Journal of Visual Languages and Computing*, 1996. **7**(2): pp. 131-174.

[9] Loughran, S. and Smith, E. "Rethinking the Java SOAP Stack," 2005. http://www.hpl.hp.co.uk/techreports/2005/HPL-2005-83.pdf

[10] Modugno, F., Green, T.R.G., and Myers, B.A. "Visual Programming in a Visual Domain: A Case Study of Cognitive Dimension," in *Proceedings of Human-Computer Interaction '94, People and Computers IX*. 1994. Glasgow, Scotlan: pp. 91-108.

[11] Nielsen, J. and Molich, R. "Heuristic evaluation of user interfaces," in *Proc. ACM CHI'90 Conf.* 1-5 April, 1990. Seattle, WA: pp. 249-256.

[12] Rieman, J., Franzke, M., Redmiles, D. "Usability Evaluation with the Cognitive Walkthrough," Conference Companion on Human Factors in Computing Systems, May 07-11, 1995, Denver, CO: pp. 387-388.

[13] SAP AG. "mySAP Business Suite: Service Provisioning," 2006. https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/li brary/uuid/6d19c8ee-0c01-0010-619d-92af980436d7

[14] Stylos, J. and Clarke, S. "Usability Implications of Requiring Parameters in Objects' Constructors," in *International Conference on Software Engineering (ICSE'2007)*. May 20-26, 2007. Minneapolis, MN: pp. 529-539.

[15] Stylos, J. and Myers, B. "Mapping the Space of API Design Decisions," in *2007 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC'07*. Sept 23-27, 2007. Coeur d'Alene, Idaho: pp. 50-57.