End-User Development: New Challenges for Service Oriented Architectures

Christian Dörner
University of Siegen
Hölderlinstr. 3
57068 Siegen
+49 271 740 4070
christian.doerner@uni-siegen.de

Volkmar Pipek
University of Siegen
Hölderlinstr. 3
57068 Siegen
+49 271 740 4068
volkmar.pipek@uni-siegen.de

Moritz Weber
University of Siegen
Hölderlinstr. 3
57068 Siegen
+49 271 740 4070
moritz.weber@uni-siegen.de

Volker Wulf
University of Siegen
Hölderlinstr. 3
57068 Siegen
+49 271 740 4036
volker.wulf@uni-siegen.de

ABSTRACT
Service Oriented Architectures (SOA) evolved as an important architectural concept in software engineering. Although the services of an SOA are loosely coupled and reusable, there have been few considerations of SOA in terms of End User Development (EUD). In this paper we will analyze the potential of SOA for the development of adaptable systems and propose challenges, which have to be solved to reach this goal. Our analysis is based on empirical studies and on requirements for EUD systems, taken from earlier research. If SOAs are extended with structures for in-use-modifications (even beyond software technologies), it will be possible to design a new generation of user-adaptable systems.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures – *Service Oriented Architectures.* D.2.1 [**Software Engineering**]: Requirements/Specifications – *Tools. H.5.2* [**Information Systems**]: User Interfaces – *User-centered design. H.1.2* [**Information Systems**]: User/Machine Systems – *Human Factors.*

General Terms
Design, Human Factors.

1. INTRODUCTION
Service Oriented Architectures (SOA) promise the development of a new generation of adaptive and adaptable software applications. In comparison to other software architectures, SOA's key features are loosely coupled 'functions', which have a standardized and open interface on the basis of XML. Such service interfaces ease the creation of distributed systems, especially across borders of different software infrastructures (e.g. of different organisations). This practical advantage from an enterprise point of view will probably make SOA the architecture of choice in the near future.

WEUSE IV'08, May 12, 2008, Leipzig, Germany.

SOAs are often used as a basis for the implementation of processes, which are part of the applications running on it. This way of thinking about software and its construction puts (business) processes in the centre of the whole software development process. This paradigm shift allows new kinds of end user participation in the software development process, as many users from organisational contexts are now able to think about information systems in a more 'natural' way, as they are usually familiar with (business) processes. The direct implementation of processes on top of a SOA will partially eliminate the need of transformation form users' requirements to UML or other modelling languages. This moves the development tasks closer to the context of users, making SOA an interesting topic for end user development (EUD) research. Imagine for example a scenario, where a procurement manager could dynamically create special cases of the purchasing process in the ERP system by himself, based on the web service of the company and its contract partners, to buy goods from a new supplier.

In the past, many EUD researchers thought about ways to ease software development for end users, even if it is considered to be a 'wicked problem', having a high inherent complexity [3]. The research about tailorable systems was often based on component architectures, as those architectures were state of the art in software engineering during that time. Famous tailorable systems are BUTTONS [14], OVAL [15], FREEVOLVE [22] or AGENTSHEETS [18] and KIDSIM [21]. Besides those systems, researchers also created end user appropriate programming languages and alternative programming philosophies, like programming by demonstration/example or natural programming approaches (e.g. [5, 10, 16]). Other researchers focused on debugging and testing support for end users [4, 9] and the development of software development models [8].

Many of the systems presented 'holistic' approaches that required applications to be completely developed in the respective programming framework. In our understanding, flexibilization technologies like component-based systems or SOA, although mainly developed to improve software reuse, also offer the potential for in-use-development in existing software infrastructures. But what are the framing conditions and architectural requirements for end user developable/tailorable SOAs to support this idea? This paper delivers first answers by identifying new challenges, which EUD poses for SOA.

In the next section we present our research methodology. Afterwards we will present a short introduction of SOA before we

analyse, on an empirical and theoretical basis, which aspects are important for the development of EUD tools. In section six, we will use the results of the analysis to formulate challenges SOA has to deal with, if EUD should be made possible. Before we conclude the paper, we sketch some possible solutions for the previously identified challenges.

2. METHODOLOGY

To consider EUD and SOA at the same time, we decided to design a methodology similar to the Domain Analysis approach, which is a concept to *"[…] manage the identification, capture, and evolution of domain knowledge and make the information reusable when creating new systems in the same application domain."* [11] This is especially useful in our case, as challenges for EUD systems can be acquired best in the field, where end users do their daily work with the systems. Repenning used a similar approach to identify 13 design guidelines for End-User Development suggestions, by analyzing the AGENTSHEETS system [19]. He stated: *"These guidelines have emerged from observing people using the AgentSheets system."*

We base our research on two information sources: empirical studies and existing surveys. Empirical studies provide valuable information about users' needs and skills for developable systems (e.g. problem statements, adaptation tasks). By taking surveys into account, we had a solid basis of findings/suggestions of how to design EUD systems.

3. SOA IN BRIEF

Like component-based architectures, Service Oriented Architectures are designed for reuse and provide the possibility of changing parts of a system without (re-)implementation. Even if the basic ideas underlying SOAs are not new [12, 23], they become more and more important for the development of modern software systems. The main reasons are the platform-, vendor- and language-independence as well as the reuse of services. Other reasons are the improved ability to react on changing system requirements and the automation of (business) processes.

The most popular technology for the implementation of SOAs are web services. A web service represents a function that can be called remotely over a network, making them similar to remote procedure calls [12]. Web services were developed to support the interaction of machines over a network and are based on well-defined XML standards [20] for a universal service description and communication. Per definition they are self-contained, self-describing, modular software artifacts that can be combined with other services. Earlier concepts like DCOM (Distributed Component Object Model) or Java RMI (Remote Method Invocation) left designers struggling with platforms, languages and encoding schemes, resulting in limited interoperability [1].

Because web services can be combined with other web services to a process, an application based on such a composition is not programmed, but 'orchestrated'. The term orchestration describes generally a business process that involves the invocation of several services [17]. There are various possibilities for the creation of such orchestrations, from which the most famous one is BPEL (Business Process Engineering Language). As BPEL itself is 'only' a language, graphical modeling tools usually use BPMN (Business Process Modelling Notation) for the graphical representation of a (business) process. BPEL was designed with automated processes in mind, making it impossible to have user-interactions within a process. This is often insufficient, because user-interactions already played a major role in workflow management systems. Therefore, the BPEL extension BPEL4People was defined, which allows the integration of people as 'service processors' into a (business) process.

The web service concept defines at least two actors: the service provider and the service consumer. Furthermore it defines an optional service broker, which can be questioned by the service consumer to find services of the service provider(s). UDDI (Universal Description, Discovery and Integration) is the most common implementations of a service broker and can be compared to a phone book [12].

The users' familiarity with process thinking may also afford the management of the underlying software configuration by 'simply' modelling (business) processes. Amongst existing modelling tools we basically find two categories: professional tools and web-based 'Web 2.0' tools. IBM WEBSPHERE BUSINESS MODELLER, IDS SCHEER ARIS, ORACLE BPEL PROCESS MANAGER, SAP NETWEAVER COMPOSITION ENVIRONMENT and SERENA BUSINESS MASHUPS are examples for well-known professional tools, which have the capability of modeling complex (business) processes in an organizational environment. Using those tools demands a certain technical knowledge in process modeling. In the second category, we find tools/web pages like YAHOO! PIPES and MS POPFLY or the end user programming tool MARMITE [24]. Those tools allow end users the creation of mash-ups, which are basically a combination (organized as process) of existing (dynamic) web contents (e.g. data from web sites, services).

4. EMPIRICAL STUDIES

Considering that the tools described above already reflect quite a history of programming ergonomics, the question arises: What makes the tools so hard to use for end users? In our analysis, we make use of our own empirical work, done in an industrial setting as well as of a study from the implementation process of a SOA in the financial industry.

4.1 PROBLEMS OF END USERS

The results we present here are taken from one of our recent studies, where we identified problems and problem solving strategies – related to software – of employees of small and medium sized enterprises (SME) [7]. There are two problem categories, which are highly relevant in the context of this paper. The first one can be called 'functional problems', as it denotes mainly data exchange problems and missing functions of the software. The second category is closely related to the first one and covers collaborative aspects of problem solving, like delegation support issues.

Functional problems can be reduced to the (business) functionality of the software. Many users of the study work with the companies' SAP system and described problems with the analysis of data. For example one user struggled to collect the necessary data to do his credit limit checks: *"There are often problems, when I want to compare things. I sometimes have the problem that I have to access four or five thing in order, to get the things I need […] for example the annually credit limit check of our customers. Therefore I need master data and data from the SD, some data from financial accounting and I don't get them by pressing a button."* To overcome shortcomings like this one, users started to export the relevant data to Microsoft Excel and created their reports within this

tool. Unfortunately, it is not possible in any case to do an automated export from the SAP system to Excel. In this case users had to collect their data within the SAP system manually and copy and paste this data into their Excel sheet. One user described this issue that way: *"Statistics […] my colleague wanted to know, how much overrun each employee of his department had daily, during the last month […] now it would be wonderful, if we were able to transfer this into an Excel file; no Mrs. X has to type each number herself."* Another example for missing functionality was the problem of a user with a system, called *HWS*, which is an optimized standard software product for his particular industry. The program did not offer him the needed flexibility to do his calculations for a particular quote, because the program did not allow him to make "a verifiable determination of the needed masses". Therefore he used Excel to do this task. His colleague used the HWS system to manage and print quotes. The use of Excel for the calculation led to a new problem. It was impossible to exchange data between the HWS system and Excel. As a solution, the two colleagues used printouts to exchange data between the HWS system and Excel. The user described the whole process like this: *"When I get the prices back, I open my Excel template and start typing the whole text of the quote – if I have enough time. […] the problem is that this text is not automatically available in the HWS system. Therefore it has to be typed again completely. […] We have to type it at least twice. It would be nice, if we had some kind of program for the HWS system, which is directly connected with my Excel template."*

Aside from this tendency to design small solutions with familiar tools, the study revealed further detailed information about the 'escalation patterns' of users. It was interesting to see that many different persons and responsibilities/expertises were involved, some even outside the organization where the problem occurred. In most cases, the users asked one or more persons for help. This diversity in the structure of problem solving patterns is important for system design, as the involvement of different persons affords the delegation of tasks, which should be supported by the systems.

4.2 ORGANIZATIONAL PROBLEMS

The analysis of organizational problems of getting (business) processes implemented in an SOA is based on an empirical study that Brahe and Schmidt [2] conducted in the financial industry. They name several problems, from which we present the most relevant ones in the context of this paper. These problems supplement the end users' perspective, which we discussed in the last section.

Missing information was the first issue that was identified. The problem occurred during the implementation of the (business) processes. The developer did not have enough detailed information about the process. Therefore he had to talk to people who know the business requirements and could answer his questions. The next issue was to find appropriate services for the 'implementation' of the process activities. Services have varying locations, because they are provided by different systems. A central service library allowed searching for services in a common place, but the service documentation was rather poor. Consequently, the process developer had to contact every service developer to find out about the services' functionality. If an appropriate service was finally found, the granularity of the service was sometimes too general or too specific. For example, some services needed data for their invocation, which was not supplied by the process. As a consequence, the service had to be re-implemented to be useful in this context.

Besides these more general issues, the tools used were not appropriate for all tasks, even if they were powerful ones like the IBM WEBSPHERE BUSINESS MODELER. The most significant problem was the transformation of the process or solution model into real code. The authors name the missing extensibility of the tool as main issue in this case.

5. SURVEYS

The first survey we analyzed is one of our previous works about component-based tailorability [25]. Wulf et al. identified three requirements to make systems tailorable. The first requirement is to *provide an architecture for re-designing software*. The second requirement is to *provide end-user-oriented concepts and interfaces* and the third requirement is to *provide a strong congruency between architectural and interface concepts*. Those requirements ensure that users have at least some basic tailoring functions at hand.

The second survey by Repenning and Ioannidou extends those requirements by two other important aspects [19]. The first aspect is that end user development systems must *carefully balance the user's skill and the challenges posed by a development process. The second aspect* demands that such systems should *enable an end-user developer to gradually acquire necessary skills for tackling development challenges*. Taking those requirements into account lowers the entry-level for adaptations and eases the learning process of users to acquire design skills over the time.

If we think about the development of software, we also have to consider maintenance and reuse aspects, as software is usually used over a long period of time. The third of survey by Dittrich et al. [6] adds some important requirements, concerning such issues. They demand that EUD systems should have a good *balance between traditional software (re-) development and maintenance on the one hand and tailoring and use on the other*. If systems provide such a balance, it will be very hard to distinct between them, because *software maintenance and (re-) development will to a growing extent be mixed and interlaced with tailoring*. Putting end users in the role of developers will furthermore create the *need for giving end users better tools for testing of tailoring constructs as well as the necessary training to use them*. In section 4.1 we claimed that users have a demand of delegating tasks to others. This is especially interesting, as the development or adaptation of software usually happens collaboratively [13]. Dittrich et al. support this thesis by their requirement of creating *better tools for reuse/sharing of tailoring constructs*.

6. NEW CHALLENGES FOR SERVICE-ORIENTED ARCHITECTURES

By combining the problems we sketched in the empirical section and the requirements we described in the previous sections, we had a solid basis of requirements, which we used to evaluate Service Oriented Architectures. The result of the analysis is a number of challenges that SOAs have to meet for an optimal support for EUD applications and interfaces.

One of the most important aspects of a SOA is the possibility to exchange services and change service orchestrations during the runtime of the system. This enables in-use-redesign. End users are able to 'change' a SOA in their personal work context during the use-time of the system. This provides an infrastructure that allows

not only changes from a software engineering perspective, but also from users' work-domain perspective. The loose coupling of services leads to another important aspect of SOA. Its concepts are designed to support language-, platform-, and vendor-independent system-to-system communication. But, as EUD requirements were not considered during the design of those concepts, end users are not able to use the interfaces of SOAs 'directly' or to do some modifications of them. End users expect (domain specific) graphical user interfaces as representations for applications and consequently for services as well. Additionally, the service interface descriptions are 'only' of technical nature and must therefore be enhanced by understandable non-functional information and an end user oriented documentation.

SOA modeling tools are designed to provide 'programming in the large', which is similar to the idea of end user development, because it abstracts form technical details and translates the modeling process to the description of domain specific (business) processes. This advantage is unfortunately limited, because the modeling tools are designed for software architects, making them far too complex for end users. Process modeling requires using appropriate services for the processes' tasks. The search mechanisms for such services exclude end users, because service descriptions inside the repositories predominantly use a technical (e.g. location URL, service parameters) style. Some web sites, like *http://www.programmableweb.com/* demonstrate how suitable services categorizations for end users may look like. Closely connected to this issue is the question of service granularity. This is important, as end users have a different understanding of functions in comparison to programmers. A first step in this direction is the definition of the so-called business service of SAP. Business services represent business functions (e.g. get sales volume) instead of technical functions (e.g. get value from table), making it easier for end users to understand them.

End users are usually not familiar with technical aspects of SOAs from the beginning on. The requirement of a strong concurrency between a graphical user interface and the architecture concept or application logic behind it is therefore hard to achieve. Furthermore users should be able to learn more about architectural concepts of SOAs over the time, making it even harder to bring the divergent aspects together. If it comes to the modeling process of services in a SOA, we have to deal with the requirement of balancing the skills of end users and the modeling process. Furthermore end users should be enabled to gradually acquire necessary skills for dealing with development challenges. This requirement for a 'gentle-slope' of the development process leads to the need of an incremental development process. Such a development process eases not only the work for professional developers, but also for end users. Therefore end users must have mechanisms, which allow them to reuse self-made adaptations and adaptations of other users, leading to a balance of software maintenance and tailoring at the same time. This is already basically supported by the process-oriented design of SOA, as processes can be reused and stored in service repositories.

The last aspect that has to be considered, are testing and debugging mechanisms. Debugging of processes, running on a SOA, is currently problematic and time-consuming. Error messages are often very long and unspecific, as many of the used protocols are based on XML. Besides those technical errors, users also need also need tools to test their processes in a separated environment with real data, as business exceptions have to be handled as well.

7. POSSIBLE SOLUTIONS

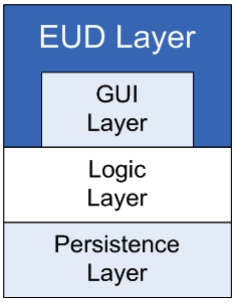
In this section we like to present first thoughts for the solution of the challenges we described. The current SOA-standards are comparatively young, which poses problems and chances at the same time. The chance is to enrich metadata and protocol standards with EUD concepts, to ease the development of adaptable/tailorable systems. The problems can be seen in the implementation of applications, because young standards change often and subsequent versions may be incompatible to previous versions. The following ideas could help to make the vision of End User Development in Service Oriented Architectures a bit more real.

An important precondition for successful end user developments is the provisioning of a suitable development model. The model should apply a visual model approach, as graphical models are usually easier to understand for users. The inherent process model of SOAs is a good basis for that. One of the biggest technical challenges is to keep the process model and the implementation of the process synchronized, even when end users return to the process level at use time. Since the development phases of process design, service orchestration and (to a certain extent) service implementation may all be involved in an in-use tailoring activity, the usual division into application logic and user interface level becomes problematic. Therefore it may be helpful to integrate interface descriptions into the metadata of a service.

If we look at the development process, we have to deal with the previously mentioned problem of service search. The first problem is the service interface, which contains 'only' technical information. That information should be extended with non-technical information that explains the service and its possible working contexts to end users. Since problems in the context of the orchestration or use of a service may relate to organizational aspects, it may be advisable to allow the information to have different scopes and visibility. It should be possible to annotate a service to ease the sharing of orchestrations and document issues that arose during use. A user should be able to 'trace' the service down in other orchestrations in order to estimate its use potential from other examples. In general, this calls for something like an 'end user wiki' for services. The second problem is the granularity of a service. The granularity should be aligned to business functions, rather than technical considerations. This would also foster users' understanding of services, as business terms are more familiar to them. This may be partially solved by using service wrapper concepts that specialize services, but in general it may be advisable to build hierarchical service structures, whose 'leaf services' have a very fine granularity, in order to provide more flexible service representations at the process level. The third problem sphere is finding an appropriate service search algorithm. It should not only take the additional non-technical information into account, but could also be intelligent in the sense of allowing service recommendation. Recommendation functions could take for example the user profile, the user's experience and the modeling context into account to advise for the use of a specific service.

Aside from providing orchestration interfaces appropriate for end users it is necessary to address the collaborative nature of EUD in SOA. Supporting the delegation of tasks to other users (or professional maintenance services) is necessary as well as providing testing tools for assuring quality. Testing should go beyond formal syntax checks and also include business semantics to reduce logical errors in the process flow. Testing tools could also provide an

exploration environment, to enable users to explore their solution in a simulated practical setting.



Besides the design time of the system, we also have to consider its use-time. One requirement demanded to provide graphical user interfaces, which have a strong congruency with the underlying architecture. The figure on the left depicts a possible solution to this requirement. A special EUD layer could extend a typical three-tier architecture of an application (the logic layer could be a SOA for example). This layer is implemented as a kind of wrapper around the GUI layer and has a connection to the logic layer. The implementation as a wrapper is an advantage, because the classical three-tier architecture concept is not affected by this kind of implementation.

8. CONCLUSIONS

Service Oriented Architectures have a huge potential for the implementation of tailorable systems, because in addition to flexibility and platform-independence, the formulation of software in terms of services and processes is closer to business domains. Using processes as a 'modelling language', leads to a unification of system design and organizational change. In this paper we tried to identify the most difficult passages on the way by describing the challenges of SOA, which have to be solved in the future to make them tailorable by end users.

Many of the ideas we described call for additional metadata of service descriptions. Especially for a potentially fast growing collection of use experiences with a service, the amount of data to be handled raises questions with regard to storage locations and synchronization issues of this information, and serious concerns about service performance if this additional contextual information defers the functional communication of a service. Therefore, the requirements EUD brings to SOA may require extending protocol and server structures of SOA standards.

9. ACKNOWLEDGMENTS

Research funded by the German Federal Ministry of Education and Research (BMBF) ['SE 2006'] and the German Research Society (DFG) [SFB/FK 615 'Media Upheavals'].

10. REFERENCES

[1] Adamopoulos, D.X., Enhancing Web Services in the Framework of Service-Oriented Architectures. in *Proc. PDCAT '06*, (2006), IEEE Press, 260 - 265.

[2] Brahe, S., BPM on top of SOA: Experiences from the Financial Industry. in *Proc. BPM'07*, 2007, LNCS, 96 - 111.

[3] Brooks, F.P.J. No silver bullet: essence and accidents of software engineering, IEEE Press, 1987, 10-19.

[4] Burnett, M., Cook, C. and Rothermel, G. End-user software engineering. *Communications of the ACM*, *47* (9). 53-58.

[5] Cypher, A. *Watch What I Do: Programming by Demonstration*. MIT Press, 1993.

[6] Dittrich, Y., Lindeberg, O. and Lundberg, L. End-User Development as Adaptive Maintenance. in *End-User Development*, Springer, 2005, 308 - 326.

[7] Dörner, C., Heß, J. and Pipek, V. Improving Information Systems by End User Development: A Case Study *Proc. ECIS2007*, University St. Gallen, 2007, 783-794.

[8] Fischer, G., McCall, R., Ostwald, J., Reeves, B. and Shipman, F., Seeding, evolutionary growth and reseeding: supporting the incremental development of design environments. in *Proc. CHI '94*, 1994, ACM, 292 - 298.

[9] Ko, A.J. and Myers, B.A., Designing the whyline: a debugging interface for asking questions about program behavior. in *Proc. CHI '04*, (2004), ACM Press, 151 - 158.

[10] Lieberman, H. *Your Wish Is My Command: Programming by Example*. Morgan Kaufmann, 2001.

[11] Lung, C.-H. and Urban, J.E. An approach to the classification of domain models in support of analogical reuse. *SIGSOFT Softw. Eng. Notes*, *20* (SI). 169 - 178.

[12] Ma, K.J. Web Services: What's Real and What's Not? *IT Professional*, *7* (2). 14 - 21.

[13] Mackay, W.E., Patterns of sharing customizable software. in *Proc. CSCW '90*, (1990), ACM Press, 209 - 221.

[14] MacLean, A., Carter, K., Lövstrand, L. and Moran, T., User-tailorable systems: pressing the issues with buttons. in *Proceedings of the CHI '90*, (1990), ACM Press, 175 - 182.

[15] Malone, T.W., Lai, K.-Y. and Fry, C. Experiments with Oval: a radically tailorable tool for cooperative work. *ACM Trans. Inf. Syst.*, *13* (2). 177 - 205.

[16] Myers, B.A., Pane, J.F. and Ko, A. Natural programming languages and environments. *Commun. ACM*, *47* (9). 47 - 52.

[17] Reichert, M., Stoll, D.: Komposition, Choreographie, Orchestrierung von Web Services in: *EMISA Forum* 2004, 21-32.

[18] Repenning, A. Agentsheets: A Tool for Building Domain-Oriented Dynamic, Visual Environments *Department of Computer Science*, University of Colorado, Boulder, 1993.

[19] Repenning, A. and Ioannidou, A. What makes End-User development Tick? 13 Design Guidelines. in *End User Development*, Springer, 2005.

[20] Shi, X. Sharing service semantics using SOAP-based and REST Web services. *IT Professional*, *8* (2). 18 - 24.

[21] Smith, D.C., Cypher, A. and Spohrer, J. KidSim: programming agents without a programming language *Commun. ACM 37* (7). 54-67.

[22] Stiemerling, O. Component-Based Tailorability *PhD Thesis*, Universität Bonn, Bonn, Germany, 2000.

[23] Vinoski, S. Old measures for new services. *IEEE Internet Computing*, *9* (6). 72 - 74.

[24] Wong, J. and Hong, J.I., Making mashups with marmite: towards end-user programming for the web. in *Proceedings of the CHI '07*, (2007), ACM Press, 1435 - 1444.

[25] Wulf, V., Pipek, V. and Won, M. Component-based tailorability: Enabling highly flexible software applications. *Journal of Human-Computer Studies*, *66* (1). 1 - 22.