A Service-Oriented Approach to Implementing an Adaptive User Interface

E.K.Senga, A.P.Calitz and J.H.Greyling

Department of Computing Sciences, Nelson Mandela Metropolitan University
P.O. Box 77000, Port Elizabeth, 6031, South Africa
e-mail: andre.calitz@nmmu.ac.za

Abstract

Service-Oriented Architectures (SOA) are increasingly being adopted to integrate the disparate computational assets in organisations. A major hurdle in the integration process is the provision of user interfaces (UIs) for applications based on SOA. A popular approach to this problem is to generate the UI whenever a user seeks to interact with an application. End users of applications are, however, increasingly different in their needs, capabilities and traits. Adaptive user interfaces (AUI) have been proposed as a means to cater for such differences. This paper outlines research undertaken to develop an AUI prototype using a SOA. A hybrid approach was used to analyse and design the prototype. An evaluation was conducted firstly to determine whether the components of the prototype adhere to established SOA principles by analytically evaluating the prototype based on these principles. Secondly, to determine the effectiveness of the prototype by evaluating the prototype and finally, to evaluate the effect the generated UI has on the performance of end-users by using a usability study. Results of the evaluation indicate that the prototype components indeed adhere to SOA principles, the prototype is effectively implemented and the UIs do not negatively affect the performance of end-users.

Keywords

Service-oriented architectures, adaptive user interfaces, web services user interfaces.

1. Introduction

Service-Oriented Architectures (SOA) is a computing and design paradigm as well as an architectural style which focuses on the design of computing systems by way of services (Oasis, 2006; Papazoglou, 2006; Josuttis, 2007; Shen, 2007; Erl, 2008). A major hurdle in the integration process when using SOA is the provisioning of user interfaces (UIs) for SOA based applications (Tibco, 2006). A popular method of integrating the UI in SOA, as a result of this challenge, is to generate the UI whenever a user seeks to interact with an application (Kassoff, Kato and Mohsin, 2003; Ellinger, 2007; He and Yen, 2007; Song and Lee, 2007; Spillner, Braun and Schill, 2007; Nestler, 2008; Gonzalez-Rodriguez, Manrubia, Vidau, and Gonzalez-Gallego, 2009).

The end users using these applications have different expertise and competencies and they have different needs, capabilities and traits. Adaptive user interfaces (AUI) have been proposed as means to cater for such differences; thus allowing users with

different capabilities and needs to interact with applications. The provision of an UI, that could adapt, depending on the user's expertise and dynamic data obtained from user interactions, can improve productivity and task completion time. Jason (2008) indicated that novice users require a different UI from expert users, where expert users utilise short-cuts and control-keys more extensively.

This paper discusses key concepts such as SOA, Automated User Interfaces, Adaptive User Interfaces. This is followed by an overview of existing service-oriented (SO) analysis and design methods and the combination of two popular methods: SOMA by IBM and Service Oriented Analysis and Design Method (SOADM) by Erl (2008). The application of this method to an AUI is presented, and the outcome, a model for AUI services, is discussed. The implementation of a prototype as a proof of concept is presented and its evaluation to answer various research questions, as well as the results of the evaluation are provided. Finally, conclusions drawn from the research are presented.

2. Background

This section provides background discussions on SOA, automated user interfaces, adaptive user interfaces and service oriented analysis and design.

2.1. SOA

The most basic component of SOA is the service and the other components include the Service Provider, Service Consumer and the Registry. The Service Provider is the creator, owner or host of the service. Service providers register their services in registries which are repositories or databases with a list of services and their descriptions. The function of the registry is to keep a searchable list of services, in order to allow service consumers to find appropriate services for their needs and bind to them. Service Consumers are applications or other services looking to make use of the capabilities of a service. They are able to search the Registry for appropriate services and bind to them using a Universal Resource Locator (URL) provided in the Registry.

2.2. Automated User Interfaces

SOA, as an architectural paradigm, advocates encapsulating units of computation or capabilities and making them accessible via a defined interface. In practice, these capabilities may be written in a variety of programming languages, deployed on different platforms, and defined by an interface using WSDL. WSDL is an XML specification that defines the operation(s) of a web service; the input and output messages; the data types of input and output message; the messaging protocol (e.g. SOAP) and bindings of the web service. This level of detail provided in the WSDL provides limited information from which to generate a UI. Several authors have proposed methods for creating UIs for web services (He, Ling, Peng, Dong and Bastani, 2008; Spillner et al., 2007; Gonzalez-Rodriguez, Manrubia, Vidau, and Gonzalez-Gallego, 2009). Some approaches simply generate UI's from the WSDL of web services while others make use of additional documentation to supplement the generation of the UI (Kassoff et al., 2003). He and Yen (2007) propose an approach

that uses an Object Layout Hierarchy (OLH) to define the layout of UI elements as nested groups.

2.3. Adaptive User Interfaces

AUIs use different techniques to achieve an increase in the flow of information between computers and users, and adapting the UI is one such technique. AUIs adapt the UI to match the needs of diverse users. AUIs consist of three components which work together in order firstly to capture user-interaction information and store this information in a meaningful way that models the user. This is achieved by using models of the user (User Model), the task (Task Model) and various other models of the AUIs environment (Jason, 2008). Secondly, the components analyse the stored information to make inferences about the user. Finally, the components are able to adapt the AUI by changing aspects of the UI to match the characteristics of the current user and thus facilitate the human-computer interaction between the application and the user.

2.4. Service Oriented Analysis and Design

A number of service oriented design methods exist for the analysis, design and implementation of SOA-based applications (Arsanjani, 2004; Zimmermann, Krogdahl and Gee, 2004; Mittal, 2006; Arsanjani, Ghosh, Allam, Abdollah, Ganapathy and Holley, 2008). SOMA is a popular method developed by IBM which provides details of the analysis design and implementation of SOA applications and systems (Arsanjani, 2004; Arsanjani et al., 2008). Erl's (2005) Service-Oriented Analysis and Design Methodology (SOADM) provides details that are not available in SOMA. Implementing SOADM, the analysis and design phase of a model includes three steps. The first step in system oriented analysis and design is to identify relevant services. This is followed by deriving specifications of the derived services. Lastly the realisation of the services involves decisions on how to implement, deploy and maintain these services.

3. Implementation of the prototype

This section discusses the implementation of an AUI prototype using SOA, based on an AUI services model (Figure 1). Firstly, the implementation domain is described in order to provide a background of the domain in which the prototype is implemented.

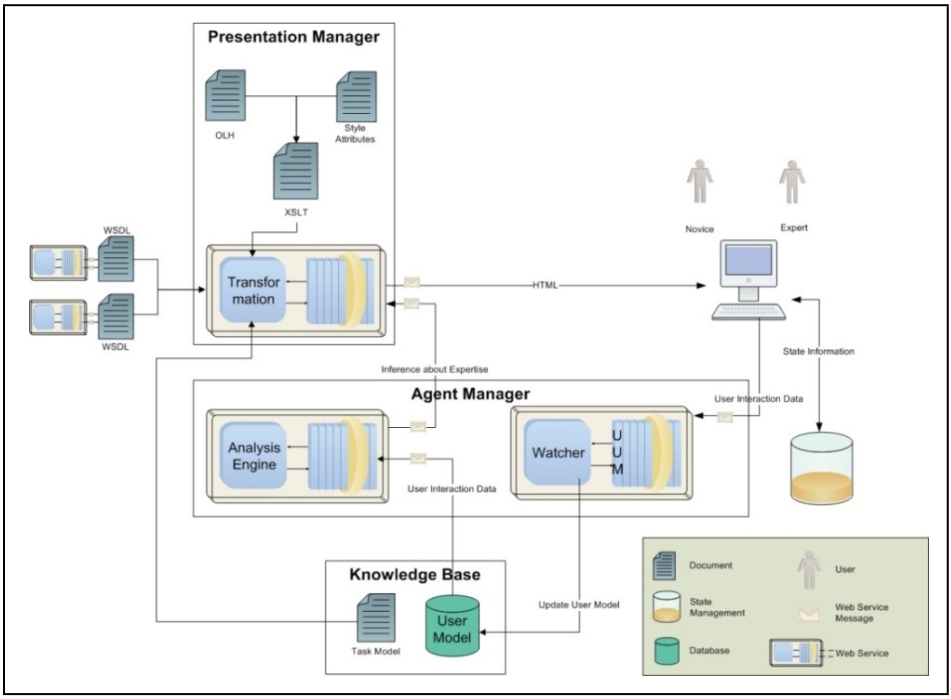


Figure 1: AUI Services Model

3.1. Implementation Domain

Contact Centres (CCs) are the main point of contact between companies and their customers. Contact Centre Agents (CCAs) are the personnel responsible for interacting with customers in a CC and they respond to customers' queries concerning company products or services. The query resolution process begins with logging the customer's query, followed by providing customer details, assigning the call and providing solution details. Jason (2008) implemented an AUI to improve the performance of novice CCAs by providing different UIs based on the user's inferred level of expertise. A prototype, consisting of two UIs was created. One UI caters for novice users and the other for expert users. For the current study, Jason's prototype was redeveloped within the SOA environment.

3.2. Prototype

An AUI services model (Figure 1) was defined after service identification and specification were conducted. The AUI services model using SOA, defines how the AUI services interact in order to provide adaptive functionality. The following sections discuss the implementation of the components of the AUI services model, as part of the service realisation phase.

3.2.1. Knowledge base

The knowledge base stores the user and task models. Both the user and task models are implemented using XML. These components are discussed further below.

3.2.1.1. User model

AUIs generally utilise a user model. The purpose of the user model is to store user-related information; which is used in the adaptation process. The user's unique characteristics are either stored or derived from the data stored within the user model. User performance data are stored as an XML document within a user's profile in a database. In addition, the user profile contains the user's log-in times, and the time taken to complete tasks. Every user begins as a novice regardless of expertise and experience. The Keystroke Level Model (KLM) is used to determine whether a user is an expert or a novice (Hurst, Hudson and Mankoff, 2007) and the low level behavioural data is obtained and analysed using Jason (2008) model. Once the performance matches the performance of predicted expert users, the prototype ceases to generate the novice UI and now generates the expert UI.

3.2.1.2. Task model

The task model maintains a model of a task or goal the user is trying to achieve. The model defines a task and its sub-tasks in a hierarchical structure. The sub-tasks are marked as being complete or incomplete as the user interacts with the UI and completes different sub-tasks. In this study, the task model is stored as an XML document. It is also exploited to provide additional capabilities, for example, element dependencies within the UI are addressed using the task model.

3.2.1.3. Agent Manager - Watcher Service

This research defines Informative Moments (IM) as UI elements with which users interact. For each IM with which users interact, various metrics, referred to as Predictive Features (PFs) are captured. A PF relates to a specific, measurable action (Hurst et al., 2007). It can be measured for any UI element such as a drop down list. The function of the Watcher component is to capture user-interaction information and store this information in the knowledge base. In order to achieve this as a service, the generated UI is created with JavaScript code to collect user-interaction information for each IM. An AUI object is created for each IM and updated if the user interacts with the IM.

The PFs that have time as a unit of measurement *(Dwell Time, Total Time and Selection Time*) are measured using the start-time and the end-time of the action. The *Dwell Time* measures the time (in seconds) during which a user was inactive for longer than 1 second while making a list selection. The *Total Time* measures the cumulative total time that a user has interacted with an IM. The *Selection Time* measures the time taken from when a user selects a list until an item is selected in the list. Jason (2008) identifies various other PFs such as *Mouse Velocity and Mouse Acceleration.* The KLM Predicted Time is a constant value (2.65 seconds) and is obtained from the design of a KLM (Jason 2008). The *KLM Difference* is obtained

by subtracting the current *Selection Time* from the *KLM Predicted Time*. Experts are therefore expected to have a smaller KLM than novice users (Jason, 2008). The user model is updated when all the values for the PFs have been obtained, storing a separate record of each task performed.

3.2.1.4. Analysis Engine – Analysis Engine Service

The role of the Analysis Engine Service is to make inferences about users from the information stored in the user model. When invoked, this service uses statistical inference techniques to determine whether the current CCA's performance is equal to or better than the performance of previously defined (not part of this study) expert users, determining whether a user is classified as an expert or not.

3.2.2. Presentation Manager – Transformation Service

The Analysis Engine Service is invoked by the Transformation Service to determine which type of UI to generate for the user. The presentation manager, which satisfies the efferential component of adaptivity, specifies how an AUI should adapt. In this study, adaptation is provided by generating a UI for a user, based on the user's inferred level of expertise. The functionality of the presentation manager is provided by the Transformation Service. The appropriate UI for a user's level of expertise is generated once the expertise of the user is determined by using the Analysis Engine Service. The Transformation service performs its function by using Extensible Stylesheet Language Transformation (XSLT) (W3C, 2009) rules to combine information from the Task Model, the WSDL for the services that support Call Logging steps and the Object Layout Hierarchy to create the UI. The following sections discuss the elements used by the Transformation Service.

3.2.3. Object Layout hierarchy

The Object Layout Hierarchy (OLH) is an XML document which uses nested XML elements to define groups of UI elements in a UI and the layout that these groups have (He et al., 2008). This document is accessed with the Task Model and the XSLT documents to determine the UI layout during the generation process. Various horizontal and vertical groups (either sorted or not) can be defined. Figure 2 illustrates how the layout definitions are applied to the UI in order to achieve the layout of elements. Additional styling of the UI is required, however.

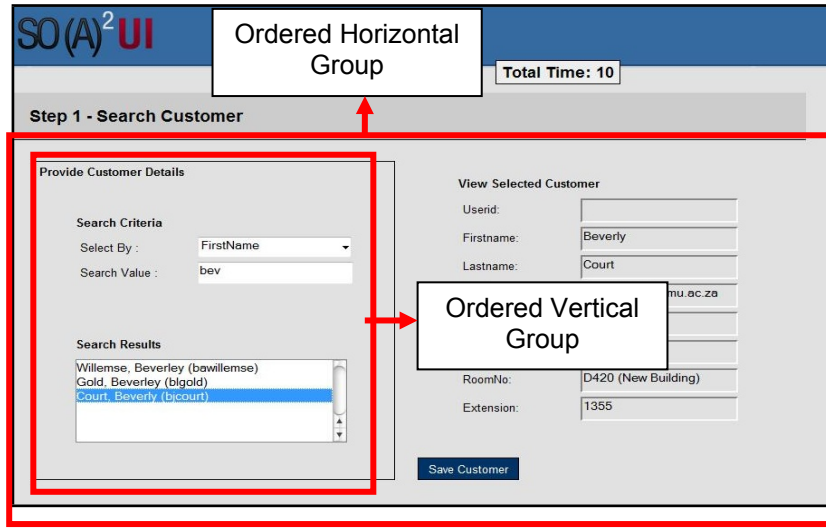


Figure 2: Example of the application of the layout groups to the Novice Step 1

4. Evaluation

The evaluation consisted of a three-component evaluation of the prototype, namely, an analytical evaluation, evaluation by software engineering metrics and finally, a usability evaluation of the generated UI.

4.1. Analytical Evaluation

Erl (2008) proposed a set of design principles to which application services for SOA applications should conform. An analytical evaluation was conducted to evaluate the prototype based on these principles:

A. **Service composability:** "*Services are effective composition participants, regardless of the size and complexity of the composition";*
B. **Service coupling:** "*Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment*";
C. **Service abstraction:** "*Service contracts only contain essential information and information about services is limited to what is published in service contracts*";
D. **Service statelessness**: "*Services minimize resource consumption by deferring the management of state information when necessary;*
E. **Service re-usability**: "*Services contain and express agnostic logic and can be positioned as reusable enterprise resources*";
F. **Service autonomy**: "*Services exercise a high level of control over their underlying runtime execution environment*";
G. **Service discoverability:** "*Services are supplemented with communicative metadata by which they can be effectively discovered and interpreted*".

Table 1 provides a summary of the service evaluation using SOA design principles. The table shows the rating assigned to the AUI services based on the characteristics portrayed by the services. Green cells in the table represent full adherence to the design principle by the service. Yellow cells represent partial adherence. The information in Table 1 is evidence that the AUI services conform to SOA principles.

		TRANSFORMATION	WATCHER	ANALYSIS
A. SERVICE COMPOSABILITY		*COMPOSABLE*	*COMPOSABLE*	*COMPOSABLE*
B. SERVICE COUPLING		CENTRALISED	CENTRALISED	CENTRALISED
C. SERVICE ABSTRACTION		CONCISE	CONCISE	CONCISE
D. SERVICE STATELESSNESS		PARTIAL ARCHITECTURAL	PARTIAL ARCHITECTURAL	PARTIAL ARCHITECTURAL
E. SERVICE REUSABILITY	TACTICAL	HIGH	LOW	LOW
	ACTUAL	HIGH	HIGH	HIGH
F. SERVICE AUTONOMY		PURE (FULL)	PURE (FULL)	PURE (FULL)
G. SERVICE DISCOVERABILITY		*SUFFICIENTLY DESCRIBED*	*SUFFICIENTLY DESCRIBED*	*SUFFICIENTLY DESCRIBED*

Table 1: Summary of analytical evaluation

4.2. Software Engineering Metrics

A number of software engineering metrics was measured in order to evaluate how effectively the prototype was achieved. Decoupling metrics measured the level of dependency between the AUI services, while architectural metrics measured how effectively the prototype was architected.

4.2.1. Decoupling metrics

A single decoupling metric, the degree of coupling within a given set of services (DCSS) was measured. The formula for determining this metric is shown in Figure 3. This metric was first proposed by Quynh and Thang (2009) who state that a lower value DCSS value equates to a lower degree of coupling between services. Any result between 0 and 1 means that coupling for that set of services is low. The measured DCSS for the prototype in this study was found to be 0.33, thus showing that the coupling between services was indeed low. This means that services were developed with minimal dependencies in accordance with the loosely coupled principle of SO design.

$$DCSS = \frac{Max - \sum_{u \in V} \sum_{u \in V} d(u,v)}{Max - Min}$$

WHERE

1. U AND V ARE TWO SERVICES IN THE SET OF SERVICES
2. D (U,V) IS THE DISTANCE BETWEEN SERVICES U AND V
3. MAX = K*V*(V-1)
4. MIN = V*(V-1)

   WHERE,

   a. K = MAXIMUM VALUE BETWEEN ANY TWO SERVICES IN THE GRAPH
   b. V = NUMBER OF SERVICES (NODES IN THE GRAPH)

Figure 3: Formula for DCSS

4.2.2. Architectural design metrics

Characteristics of an application such as structural, data and system complexity can be measured by using software engineering metric models. High complexity values mean that complex code had to be written in order for the modules to function in an SOA, while low values mean that the complexity is low.

$$S = \frac{\sum f^2(i)}{n}$$

Figure 4: Structural complexity formula

Where, $f^2(i)$ *is the fan-out of each class or module being evaluated and* n *is the number f modules in a system.*

$$D(i) = \frac{v(i)}{f(i)+1} \quad [A] \qquad D = \frac{\sum D(i)}{n} \quad [B]$$

Figure 5: Data complexity formulae

Where:

$v(i)$ is the number of input and output parameters passed to and from the module.

$F(i)$ is the fan-out of each class or module being evaluated

n is the number of modules in a system.

4.2.3. Structural complexity

Structural complexity measures the complexity of a module using the Fan-out approach (Card and Glass, 1990; Pressman, 2004). For the purposes of this evaluation, fan-out refers to procedural calls to dependent classes and web services, i.e. calls to other web services as well as subordinate classes (e.g. data access class

for a service). Fan-out is calculated for each procedure in a module, and the sum of all procedures' fan-out values is the fan-out of the module. Figure 4 shows the formula used to calculate the structural complexity of a system S by adding up the fan-out for each module i (Kan, 2002).

Data complexity

Data complexity is a measure of the complexity in the internal interface for a given module (Card and Glass 1990; Pressman 2004). Figure 5 (A) shows how the data complexity is measured for each module of a system, while Figure 5 (B) shows how the data complexity of each module in a system is added up to get the average data complexity of a system.

System complexity

System complexity is a measure of the overall system complexity. Overall system complexity is affected when the structural and data complexity of components within a system change (Kan, 2002; Pressman, 2004). System complexity is measured by adding the structural and data complexity of a system.

4.3. Results from applying metrics

Table 2 provides a summary of the metrics when applied to the AUI services of the prototype. Increased structural complexity increases the problem and perceived complexity of a system (Bundschuh and Dekkers, 2008). A complex system requires more effort to implement. Low complexity values, therefore, indicate less effort in implementing a module. The values in Table 2 show the structural and data complexity for the AUI services. The transformation and expertise services have extremely low structural complexity values. This was done intentionally to decouple the services from their external environment. The watcher class, however, has higher level coupling and dependency as indicated by the complexity values. This indicates that complex code had to be written to perform the required functionality.

Service	Architectural Design Metrics		
	Structural Complexity	Data Complexity	System Complexity
Transformation	0	2	2
Watcher	9	1.43	10.43
Expertise	1	1.5	2.5
Overall	10	4.93	14.93

Table 2: Summary of Architectural Design Metrics for AUI services.

4.4. Usability Evaluation

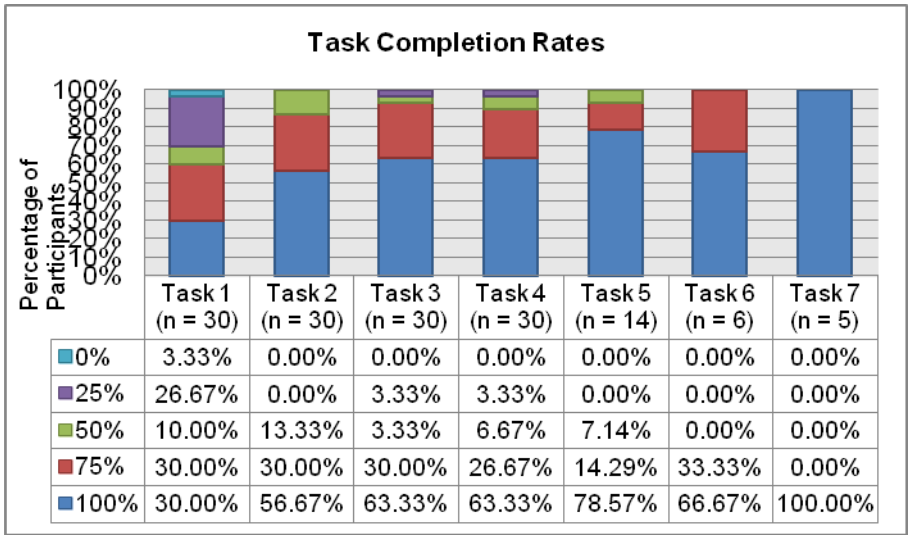
The purpose of this section is to report on the evaluation of the generated UI by testing its usability using usability evaluation guidelines (Nielsen, 1993; Scholtz, 2000). The majority of the 30 participants were recruited from the NMMU Department of Computing Sciences Department. Thirty participants were recruited for the evaluation. Participants were required to have a high level of computer experience, a sound knowledge of IT and no application or domain experience. A

role-playing scenario was used for the evaluation (Pretorius, 2005) and a simulated CC environment was created whereby the participant played the part of a CCA, while the instructor took the role of a customer calling into a CC with a query.

A test plan which provided instructions on how to complete the tasks and information on the tasks to be performed was created for the evaluation using the same set of heuristics as Jason (2008). The final task plan consisted of 7 tasks in total. The effectiveness and efficiency of the prototype was evaluated, which included eye-tracking evaluations (not included in this paper).

Effectiveness

Task success (or completion rate) can be used to measure how effectively a user is able to complete a given set of tasks on a UI (Tullis and Albert, 2008). Each task consisted of 4 steps, with each step contributing 25% to the task completion. Figure 6 shows the task success and failure rates for tasks 1 to task 7. Task one shows poor performance, with only 30% of the participants (n=9) completing the task with a 100% success rate and only 60% (n=18) completing more than 75% of the task successfully. Over the course of the evaluation, however, the task completion rate is observed to increase. Task 4 has 90% (n=27) of the participants completing more than 75% of the tasks and task 7 has 100% of the participants completing 100% of the task. Only five participants attempted task 7, however, since the UI was adapted for all the other participants before they attempted this task.



	Task 1 (n = 30)	Task 2 (n = 30)	Task 3 (n = 30)	Task 4 (n = 30)	Task 5 (n = 14)	Task 6 (n = 6)	Task 7 (n = 5)
0%	3.33%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
25%	26.67%	0.00%	3.33%	3.33%	0.00%	0.00%	0.00%
50%	10.00%	13.33%	3.33%	6.67%	7.14%	0.00%	0.00%
75%	30.00%	30.00%	30.00%	26.67%	14.29%	33.33%	0.00%
100%	30.00%	56.67%	63.33%	63.33%	78.57%	66.67%	100.00%

Figure 6: Task Completion rates for Tasks 1-7.

Efficiency

Time-on-task is a metric that measures the length of time a participant takes to complete a task and thus user efficiency. Combining this metric with task success shows the participant's task efficiency as the completion rate per unit of time. Figure 7 shows the mean time-on-task achieved by all participants (n=30) for task 1 to task

7. The average time to complete tasks reduced drastically after the first task showing that users required only one task to familiarise themselves with the UI.

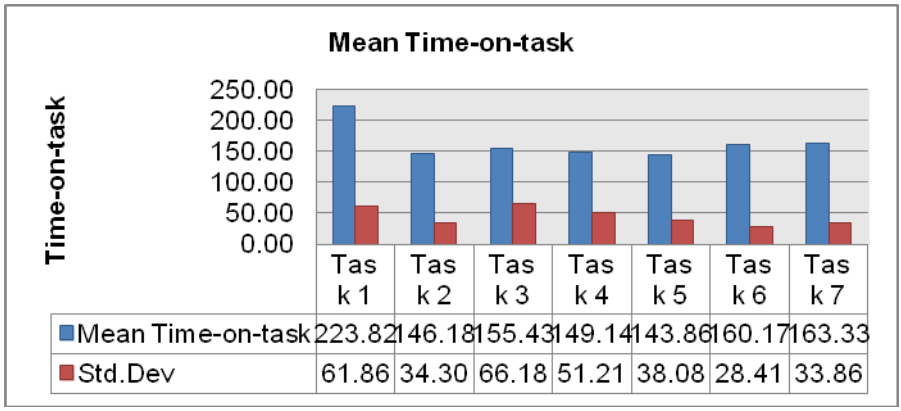


Figure 7: Mean Time-on-Task.

The success rate for each task was combined with the time-on-task to give a value for efficiency. CCAs from the NMMU ICT helpdesk are given approximately two minutes to resolve a query, after which the call must be assigned to a technician who can resolve the query (Vermaak, 2008). Efficiency was therefore measured as the task completion rate per two minutes, that is, how many calls an agent resolves every two minutes. Tasks 1-7 were typical tasks CCA's perform working at the NMMU helpdesk.

Figure 8 shows the efficiency rates for all the tasks completed by participants. This was done by measuring the efficiency as the completion rate per unit of time (two minutes in this case). Task 1 had the lowest efficiency rate of 37% which means that users were only capable of completing 37% of tasks every two minutes. This can be explained by the learning required to complete tasks. Task 5 has the highest efficiency rate of 77%.

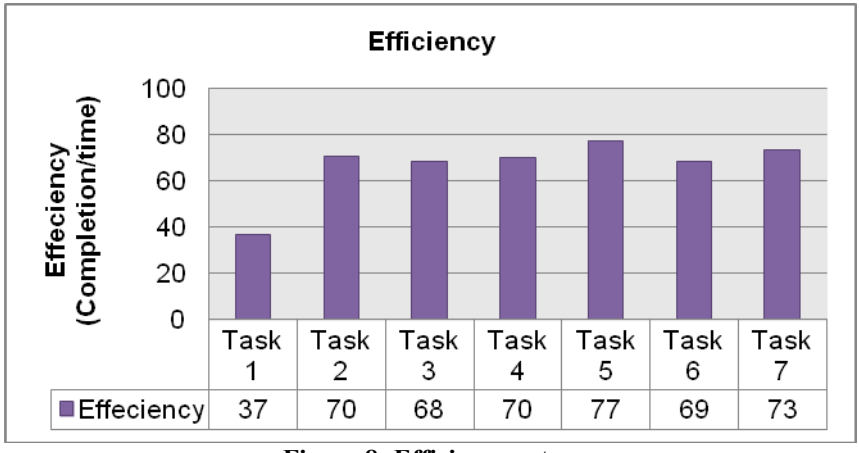


Figure 8: Efficiency rates.

These results indicate that the users could, effectively and efficiently, complete the tasks outlined in the task plan. By completing the tasks in good time, it can be inferred that the generated UIs did allow users to complete the tasks.

5. Conclusions and Recommendations

The aim of this research was to determine if an AUI could be implemented by using a SOA. In order to meet this objective, an AUI services model was designed and a proof-of-concept prototype was implemented and evaluated using an analytical evaluation and a usability evaluation. The evaluation of the prototype showed that users' productivity was not negatively affected by using a SOA. It can therefore be concluded that an AUI can be implemented effectively by using an SOA.

The scope of this research was limited to the use of AUI services in a controlled environment. Future work could involve a research study building on the work of Gonzalez-Rodrigues, et al. (2009) by implementing an adaptive user interface management system and comparing the results to the findings in this study. The envisaged benefits of this research would be increased by the availability of AUI services in organisations for improved UI usage by agents and for training purposes.

6. References

Arsanjani, A. (2004), Service-Oriented Modeling and Architecture (SOMA) (Online). Available at: https://www.ibm.com/developerworks/webservices/library/ ws-soa-design1/, Date Accessed: 20 May 2009.

Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S. and Holley, H. (2008), SOMA: A method for developing service-oriented solutions. *IBM Systems Journal,* 47, pp 377-396.

Bundschuh, M. and Dekkers, C. (2008), *The IT Measurement Compendium: Estimating and Benchmarking Success with Functional Size Measurement,* Berlin / Heidelberg, Springer.

Card, D. N. and Glass, R. L. (1990), *Measuring Software Design Quality,* University of Michigan, Detroit, USA, Prentice Hall.

Ellinger, R. S. (2007), Service Oriented Architecture and the User Interface Services: The Challenge of Building User Interface Services. *Technology Review Journal,* 15, pp 43-61.

Erl, T. (2005), *Service-Oriented Architecture: Concepts, Technology, and Design,* Upper Saddle River, NJ Prentice Hall PTR.

Erl, T. (2008), *SOA Principles of Service Design,* Upper Saddle River, NJ, Prentice Hall.

Gonzalez-Rodriguez, M., Manrubia, J., Vidau, A. and Gonzalez-Gallego, M. (2009), Improving accessibility with user-tailored interfaces. *Applied Intelligence, 30,* pp 65-71.

He, J. and Yen, I.-L. (2007), Adaptive User Interface Generation for Web Services. *In Proceedings of e-Business Engineering, 2007. ICEBE 2007,* pp 536-539.

He, J., Yen, I. L., Tu, P., Jing, D. and Bastani, F. (2008), An Adaptive User Interface Generation Framework for Web Services. *In Proceedings of the Congress on Services Part II, 2008. (SERVICES-2. IEEE)*, pp 175-182.

Hurst, A., Hudson, S.E. and Mankoff, J. (2007), Dynamic detection of novice vs. skilled use without a task model. *Proceedings of the SIGCHI conference on Human factors in computing systems*. San Jose, California, USA. ACM. pp 271-280.

Jason, B. A. (2008), An Adaptive User Interface Model for Contact Centres. *Department of Computer Science and Information Systems.* Port Elizabeth. South Africa., Nelson Mandela Metropolitan University.

Josuttis, N. M. (2007), *SOA in Practice: The Art of Distributed System Design,* Sebastopol, CA, USA, O'Reilly Media, Inc.

Kan, S. H. (2002), *Metrics and Models in Software Quality Engineering,* Reading, Mass, Addison-Wesley Professional.

Kassoff, M., Kato, D. and Mohsin, W. (2003), Creating GUIs for Web Services. *IEEE Internet Computing,* 7, pp 66-73.

Mittal, K. (2006), Build your SOA, Part 3: The Service-Oriented Unified Process (Online). Available at: http://www.ibm.com/developerworks/webservices/library/ws-soa-method3 /index.html, Date Accessed: 21 October 2008.

Nestler, T. (2008), Towards a Mashup-driven End-User Programming of SOA-based Applications. *In Proceedings of the 10th International Conference on Information Integration and Web-based Applications and Services*, pp 551-554.

Nielsen, J. (1993), *What is usability?,* San Francisco, Morgan Kaufmann.

Oasis (2006), Reference Model for Software Oriented Architectures (Online). Available at: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm Date Accessed: 15 March 2008.

Papazoglou, M. P. (2006), Web Services Technologies and Standards. ACM Computing Surveys.

Pressman, R. (2004), *Software Engineering: A Practitioner's Approach,* New York, NY, USA, McGraw-Hill Science/Engineering/Math.

Pretorius, M. (2005), The Added Value of Eye Tracking in the Usability Evaluation of a Network Management Tool. *Department of Computer Science and Information Systems.* Port Elizabeth. South Africa, Nelson Mandela Metropolitan University.

Quynh, P. T. and Thang, H. Q. (2009), Dynamic Coupling Metrics for Service--Oriented Software. *International Journal of Computer Science and Engineering,* 3, pp 46-46.

Scholtz, J. (2000), Common industry format for usability test reports. *In Proceedings of the Conference on Human Factors in Computing Systems*, pp 301-301.

Shen, H. T. (2007), Service-Oriented Architecture *Future of IT: SERVICE-ORIENTED ARCHITECTURE.* University of Queensland, Australia.

Song, K. and Lee, K.-H. (2007), An Automated Generation of XForms Interfaces for Web Service. *In Proceedings of the IEEE International Conference on Web Services 2007*, pp 856-863.

Spillner, J., Braun, I. and Schill, A. (2007), Flexible human service interfaces. *In Proceedings of the 9th International Conference on Enterprise Information Systems*, pp 79-85.

Tibco (2006), Rich Portals: The Ideal User Interface for SOA. Palo Alto, CA, USA.

Tullis, T. and Albert, W. (2008) , *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics,* Burlington, MA, USA, Morgan Kaufmann.

Vermaak, R. (2008), ICT Helpdesk Manager at the Nelson Mandela Metropolitan University. Port Elizabeth. South Africa.

W3C (2009) XSLT (Online). Available at: http://www.w3.org/TR/xslt. Date Accessed: 15 August 2009.

Zimmermann, O., Krogdahl, P. and Gee, C. (2004), Elements of Service-Oriented Analysis and Design. IBM developerWorks.