Context-Sensitive User Interfaces for Semantic Services

WANITA SHERCHAN, Monash University
SURYA NEPAL, CSIRO ICT Centre
ATHMAN BOUGUETTAYA, RMIT University
SHIPING CHEN, CSIRO ICT Centre

Service-centric solutions usually require rich context to fully deliver and better reflect on the underlying applications. We present a novel use of context in the form of *customized user interface services* with the concept of *User Interface as a Service* (UIaaS). UIaaS takes user profiles as input to generate context-aware interface services. Such interface services can be used as context to augment semantic services with contextual information leading to UIaaS as a Context (UIaaSaaC). The added serendipitous benefit of the proposed concept is that the composition of a customized user interface with the requested service is performed by the service composition engine, as is the case with any other services. We use a special-purpose language (called User Interface Description Language (UIDL)) to model and realize user interfaces as services. We use a real-life e-government application, human services delivery for the citizens, as a proof-of-concept. We also present a comprehensive evaluation of the proposed approach using a functional evaluation and a nonfunctional evaluation consisting of an end user usability test and expert usability reviews.

Categories and Subject Descriptors: H.5.2 [**User Interfaces**]: User-Centered Design

General Terms: Design, Human Factors, Management

Additional Key Words and Phrases: Semantic service, user interface, context-aware

ACM Reference Format:
Sherchan, W., Nepal, S., Bouguettaya, A., and Chen, S. 2012. Context-sensitive user interfaces for semantic services. ACM Trans. Internet Technol. 11, 3, Article 14 (January 2012), 27 pages.
DOI = 10.1145/2078316.2078322 http://doi.acm.org/10.1145/2078316.2078322

1. INTRODUCTION

Context is typically defined as a user's surrounding characterized by attributes such as location, environment, identity, date and time, season and temperature [Schilit et al. 1994; Brown et al. 1997; Ryan et al. 1997]. Other works have included the cognitive aspects of the user to define context in terms of the user's physical, emotional, social and informational state [Dey et al. 1998]. The definition of context relevant to this article is more in line with Abowd et al. [1999], in which context is considered to be any information that can be used to characterize the situation of an entity relevant to the interaction between a user and an application. If we apply this definition to services computing, context can be interpreted as the information that characterizes the operation of a service, specifically, information relevant to the interaction between a

Authors' addresses: W. Sherchan, Faculty of Information Technology, Monash University, Australia; email: Wanita.Sherchan@monash.edu; S. Nepal and S. Chen, Information Engineering Lab, CSIRO ICT Centre, Australia; emails: {Surya.Nepal,Shiping.Chen}@csiro.au; A. Bouguettaya, School of Computer Science and Information Technology, RMIT University, Australia; email: Athman.Bouguettaya@rmit.edu.au.

DOI 10.1145/2078316.2078322 http://doi.acm.org/10.1145/2078316.2078322

user and the service. Services typically represent complex real-life entities. Therefore, contextual information is paramount in interpreting and representing service output to users. Web services are the technology of choice for realizing service computing [Curbera et al. 2002]. Services are typically delivered through the Web; therefore, interfaces are a significant part of the interaction between the user and the Web service. As a result, information pertaining to user interfaces, such as user profiles, is a source of context in addition to the typical user location, identity, time, etc.

A Web service is considered to be context-aware if it has the ability to detect and respond to changes in the environment [Maamar et al. 2006]. For example, a flight booking service is context-aware if it can adapt the currency of the ticket price to the location of the client. Context-aware Web services should be easily accessible and adaptable so that they have potential to enrich user experiences and make daily life more productive, convenient, and enjoyable. Such a context-aware perspective also brings new challenges in the architecture, design, and implementation of existing Web services' infrastructures and applications. Some questions that need to be addressed are how to deal with large amounts of services based on a variety of users' personalized needs; how to handle context-aware service composition in a dynamic environment; and how to customize business processes according to users' preferences.

Issues. In many applications, such as e-government services, user interfaces are considered to be among the most important contextual information for the provisioning of services. In that respect, the use of appropriate user interfaces is an important part of making government services more easily and effectively accessible to a wider range of constituencies, including those with little computing skills, physical and mental impairments, and young people who are tuned into the latest in social network technologies. Therefore, user interfaces are key to providing rich context to many e-government services. Research in accessibility and usability has traditionally focused on typical users and domain specific applications. It is assumed that users' abilities do not change over time, and a system is designed for a specific application. The accessibility research has mostly focused on webpage content and how it can be made more accessible using features like text-to-speech, page magnification, enlarging menus and scroll bars, colour and font size changes, and keyboard-based navigation. The techniques for adaptive user interface usually depend on a user selecting the most appropriate options for layout. The challenge is how to provide an adaptive user interface in a way that is transparent to the user. Designing a good interface in general is challenging. A new approach is needed that will provide mechanisms to adapt to both short-term and long-term changes in user's abilities, as well as the requirements of different services accessed by the user.

Our Approach. To address the aforementioned issues, we propose an approach to design *User Interface as a Service* (UIaaS) to create interfaces that automatically adapt to the conditions of the users as well as the applications. The UIaaS leverages users' context information to allow for easy access to services while adapting the presentation medium (i.e., user interface (UI)) to suit their context. The UIaaS takes two types of input—the user's context from the user profile, and the service context from the service being provisioned—to generate the UI, which is itself a service. This considers user interface as a service with context as input, enabling the composition of UI services, assignment of quality attributes, and querying them based on the contextual information. This is a novel view of context which leverages and extends our previous work on aspect-oriented approach, weaving context and services [Li et al. 2009]. In our current proposal, we consider the User Interface as a Service, which is part of the service ecosystem. Since the UIaaS already has contextual information embedded in it, it can also serve as a source of context for service composition leading to UIaaS as a Context

(UIaaSaaC). The user interface service is considered as contextual information that adapts to the needs of the service input/output, as well as to the user's expectations and limitations. The UIaaS is key contextual information for generating the best UI elements for presentation of service input/output. It is important to note that the purpose of UIaaS is to generate UI elements in a standard and uniform format based on the available context information by using already existing interface services. The presentation of such UI elements to the end users is outside the scope of this article. It has been well-studied in the field of Human Computer Interaction (HCI).

In our approach, UIaaS is dynamically composed with other services. This is a form of an augmentation/weaving of their contextual presentation. The serendipitous effect of this approach is that no extra composition infrastructure is needed beyond what is required for normal service composition. The composition engine considers the UI as any other service that is to be composed. The set of UIaaS is managed through an ontology that maps all types of UIs to their respective service requirements as well as user expectations and their physical and mental challenges. We provide a description language that maps a service instance of a UI concept to a uniform representational context for a service. This language is called *User Interface Description Language* (UIDL). Each retrieved service is mapped to the best UI service instance and then composed with it. We use a real e-government application to model the provisioning of social services to a large and varied set of populations that include the unemployed, senior citizens, students, single mothers, expecting mothers, and those stricken by transient circumstances. This work is part of a larger research endeavour that aims to provide a seamless framework for efficiently managing the whole lifecycle of services [Yu et al. 2008].

Contributions. This article makes the following major contributions: (i) Use of context in Web services interfaces: we use user profiles as a source of context for customizing user interfaces. (ii) User interface as a service using contextual information: we use contextual information from user profiles as input and generate user interface services (UIaaS) as output. (iii) User interface as a service as a context for applications: the user interface services (UIaaS) can themselves be used as context for service composition, service query, etc. by other applications. (iv) No need for new infrastructure to support the proposed UIaaS concept: UIaaS can be realized using existing infrastructure, for example, the WSMS infrastructure [Yu et al. 2008]. UIaaS can also utilize existing functionalities such as search, compose, query, etc., and therefore, does not require addition of new functionalities to operate correctly.

The article is organized as follows. In Section 2, we give an overview of the Web Service Management System (WSMS) architecture that provides a framework for managing user interfaces as services. In Section 3, we introduce the concept of User Interface as a Service (UIaaS) with context as input and UIaaS itself as a context. In Section 4, we describe our proof-of-concept implementation using a real-life e-government application, human services delivery for the citizens, and present two evaluations of our approach—functional and nonfunctional. The nonfunctional evaluation consists of an end user usability test and expert usability reviews. In Section 5, we review the current literature in context-aware service delivery with a focus on context-aware interfaces. We conclude and point to future directions in Section 6.

2. MANAGING THE SERVICE LIFECYCLE

The Web Service Management System (WSMS) is an enabling technology for managing context-aware Web services. It provides the platform to test, deploy, and manage context-aware Web services. We use WSMS as the platform on which to demonstrate the concept of UIaaS. This section provides a brief description of WSMS and its key

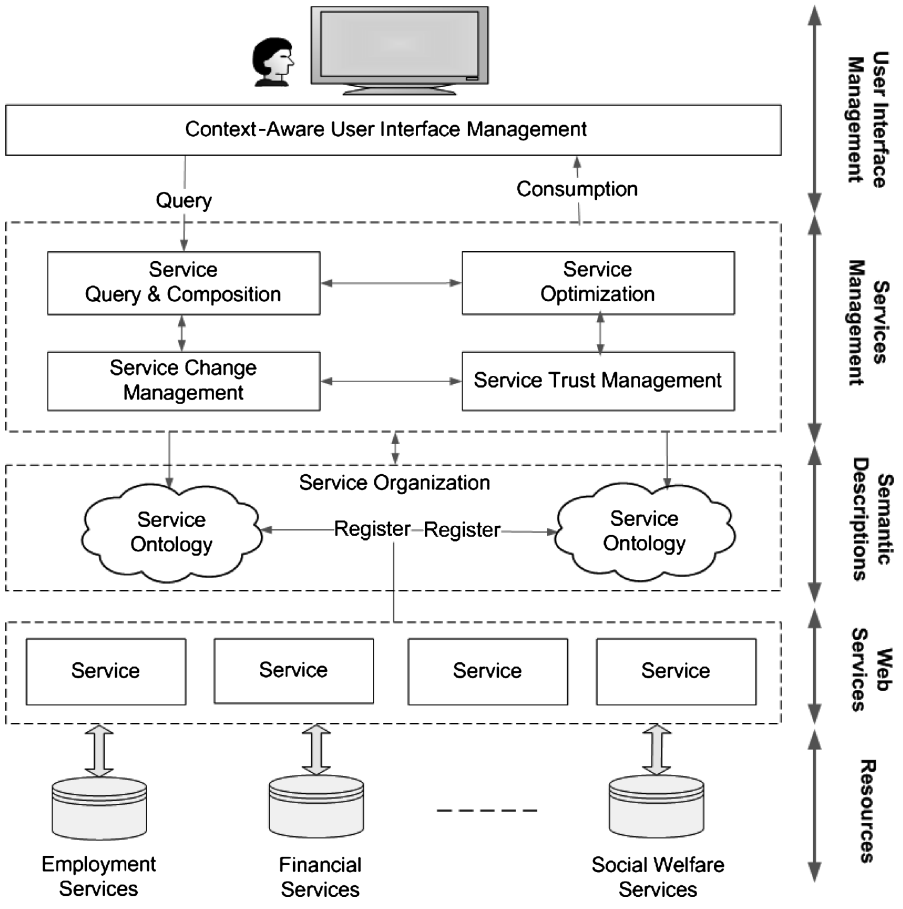


Fig. 1. Web Service Management System (WSMS) architecture.

components, more specifically, their roles in the proposed context-sensitive user interfaces for semantic services.

Figure 1 shows the WSMS framework architecture. The lowest layer in the architecture contains application-specific services. The second layer is the service organization that uses ontology to describe the underlying application-specific services and contextual information about these services. The third layer contains the core components of WSMS for managing service life-cycle, which include service trust management, service optimization, service change management, and service query and composition. The highest layer is the user interface. The discussion in this section will be limited to the second and third layer components and their roles in supporting context-aware user interface management at the highest layer. We describe how UIaaS enables context-aware user interfaces for semantic services in Section 3.2.

Service Organization. The service organization catalogues services and relevant context information so that they can be uniquely identified and managed within their life-cycle. We use an ontology-based approach to organize services. An ontology provides a common framework to represent service metadata, as well as domain and context information. The ontology also provides context information to compose context-aware composite services. Two types of context information are relevant to user interfaces.

The first consists of inputs that a user interface service takes, such as user profile, device characteristics, geo-spatial and temporal information. Such context information helps WSMS to create context-aware user interface as a service. The second type is user interface services as context, that the service composition takes as input to generate a composite service that includes user interface. The first type provides context information to the user interface service from users' perspective, whereas the second provides context from the system's perspective.

Service Trust Management. Service organization may contain several services, including user interfaces, that provide similar functionality. Service composition prefers that all the services selected during composition are the best services for the given context, compared to other similar services. Therefore, trust acts as a contextual information for service selection. However, selection of the best services becomes difficult because services may commit to provide a certain level of Quality of Service (QoS) but may fail to deliver. Thus, a major challenge in WSMS includes providing a trust framework for enabling the selection of services based on trust information. We have developed a number of reputation collection, assessment, and dissemination models for WSMS, presented in [Malik and Bouguettaya 2009]. Our trust models are based on the concept of community, where the reputation of a service is the collective perception of the service consumers within the community.

Service Optimization. WSMS may find a number of ways to create context-aware services that satisfy all of a user's functional and quality requirements. Service optimization aims to select the services that a user would prefer the most, based on the available context information. The proposed techniques for service optimization are based on QoS parameters. These techniques aim to identify a service with the best quality, based on context information. To conduct effective service optimization, WSMS adopts schemes aiming to handle both quantitative and qualitative properties of services. We propose some such techniques in Ouzzani and Bouguettaya [2004] and Wang et al. [2010].

Service Change Management. Service change management is an important aspect of context-aware user interfaces, as the services in a composition may need to be changed based on the contextual information. A user interface service may go through frequent changes, and the triggers for these changes could come from different sources. Changes in the services include changes in the functionality they provide, the way they work, the component services they are composed of, and the quality of service they offer. The service change management component is designed to manage such changes in the user interface services, as well as other services. WSMS uses a number of approaches to deal with the change in user and service context (refer to Ryu et al. [2008] for details).

Service Query and Composition. This component takes user interface as a context and generates a composite service. The component provides support for the underlying query model and composition algorithms. The WSMS query model consists of a query interface, a query language, and a query engine. The interface allows users to specify what they want (goals) at a semantic level and leave composite service generation to WSMS. The query interface is handled by the user interface component, which will be discussed in the next section. The underlying query language is service query language, the XML service query language we developed as an interface to WSMS query engine. Unlike keyword-based information searching, the service query language is able to specify nonfunctional requirements using standard SQL-like prediction clauses. Based on how services are composed to create composite services, WSMS allows three types of composition: horizontal, vertical, and hybrid [Bouguettaya et al. 2010].

Services in WSMS are organized using a semantic model in service organization. Context-aware user interfaces are generated by the user interface component and provided as user interface context to the service composition. Hence, context-aware composite services can be automatically generated by WSMS. Upon receiving a user's query, WSMS will first identify relevant services, including user interfaces, and perform logical reasoning to generate a composite service at the semantic level. WSMS uses a matchmaking algorithm proposed by Medjahed et al. [2003] and context-aware service-weaving algorithm proposed by Li et al. [2009] to generate a composite service with appropriate user interfaces. This is feasible since user interfaces are generated as services and provided as context to service composition.

3. CONTEXT-SENSITIVE USER INTERFACES

Service-oriented applications are by nature platform- and application-independent and designed to be used by a variety of users under different circumstances. Therefore, the application interfaces should be simple and easy to use. In addition, the query interface should be rich and powerful enough to be able to express various concepts and their relationships. Furthermore, the result of a query often is not a single service but a composition of atomic services from different applications. Considering these requirements, the query interface for WSMS is designed to have form-like features where users can declaratively specify their query. Different Graphical User Interfaces (GUIs) are needed for end users to generate composite services interactively in different application domains. The GUI presents the abstractions of underlying domain knowledge with the help of the domain ontology.

In order to address this issue, we introduce a novel approach to service composition by creating user interface as a service, and providing it as a context to service composition. This enables the service composer to create composite services composed of Web services and user interface services. The functionalities of the user interface management component in terms of context-awareness can be described as follows: it creates a context-aware user interface for users and provides User Interface as a Service (UIaaS) as a context for composition for a specific application. For example, a user interface for a color-blind person is generated as a service based on their user profile. The generated user interface is then used as a context for some human services application for composing other services, so that the outputs from the other services can be displayed to the user using the interface service generated for a color-blind person. Next, we describe two main contributions of this article—user interface as a service, and interface service as a source of context for applications.

3.1. User Interface as a Service (UIaaS)

User interfaces take different forms that include textual, audiovisual, and graphical interfaces. Their main purpose is to present results of queries, making them understandable to different categories of users. Some interfaces are general, such as text-based interfaces, while others are more specific, such as Braille-based interfaces. Historically, every type of user interface is designed for different types of environments (e.g., large/small screen) and types of users (e.g., visually impaired). There are commonalities and differences among the different designs. The core idea behind context-sensitive user interfaces is to consider user interfaces as services that take, as input, several types of context information, such as user profile, device characteristics, geo-spatial and temporal information, and generate the interface to suit the characteristics of the user, device, and geo-spatial location. The focus of our work is on the functional aspects of user interface rather than nonfunctional. Therefore, the UIaaS aims to provide functionalities for generation of artifacts necessary for user interface, but not necessarily how these artifacts are rendered and displayed to the user.

The UIaaS treats each interface service as a first-class object that can be manipulated and combined. This determines how the generated information is presented to the user. Instead of defining a user interface in terms of overall Webpage layout, the user interface is defined as a composition of user-specific Web service components. Each Web service provides a user interface component for input and output messages related to its functionality. The component's design takes into account users' characteristics and disabilities, available hardware, and service provided to create a composite user interface.

The core underlying principles of UIaaS are as follows.

—A pool of services in WSMS are responsible for generating UI-related artifacts. These services are advertised like other Web services, and take contextual information about the user, the application, and its environment as input to generate a set of UI artifacts as output. For example, a service takes device type as input and generates a widget to display textual information for that device as output. This is an example of a fine-grained UI service.
—UI services are organized by the service organizer like other services, using three layers of ontology. Therefore, UI services are also treated as first-class service artifacts that can be queried, searched, composed, changed, and trusted within WSMS.
—UI services can be composed with other application and domain-specific services using the same composition engine. This means realizing the UIaaS concept does not require a separate composition engine for UI services.
—Keeping with the service-oriented design principles, user interface services are also platform- and application-independent. UI services generate artifacts according to the given contextual information. However, the rendering and presentation of these artifacts are outside the scope of this article.

The UIaaS concept is not restricted to a specific design and implementation. In this article, we propose a design and implementation to realize the UIaaS concept based on the previous guidelines. It is important to note that our proposed design and implementation is one way of realizing the UIaaS concept. There could be other alternative ways of realizing UIaaS. We next describe our proposed approach.

The proposed design consists of two main elements.

(1) A User Interface Description Language (UIDL) for describing artifacts/elements of user interface for service-oriented applications. All UI elements can be described in UIDL. UIDL can be integrated with a standard workflow language such as XPDL to represent a workflow composed from Web services.
(2) A User Interface Execution Engine (UIEE) for (a) rendering contents as per the UIDL returned by the underlying services, and (b) invoking appropriate underlying services in response to user inputs.

The UIDL is only one way of realizing the proposed solution. Although UIDL is a powerful language to demonstrate the UIaaS concept, the contributions of the proposed solution is beyond the construct and expressive power of UIDL, as just explained. This proposed design enables user interfaces to be described, published, searched, queried, composed, and invoked as services.

As shown in Figure 2, three services publish their Web service interface (WSDL) along with their UI into a semantic service repository. An application can then lookup/query the UIs that meet its requirements. The application can directly request the UIDL from the corresponding UI service (or optionally via the service repository). Once an appropriate UIDL is retrieved, the UIEE can render the UI according to the specification in the UIDL. Similar to traditional Web services, the application can compose multiple UIs from multiple UI services to form their own UI as shown in Figure 2.

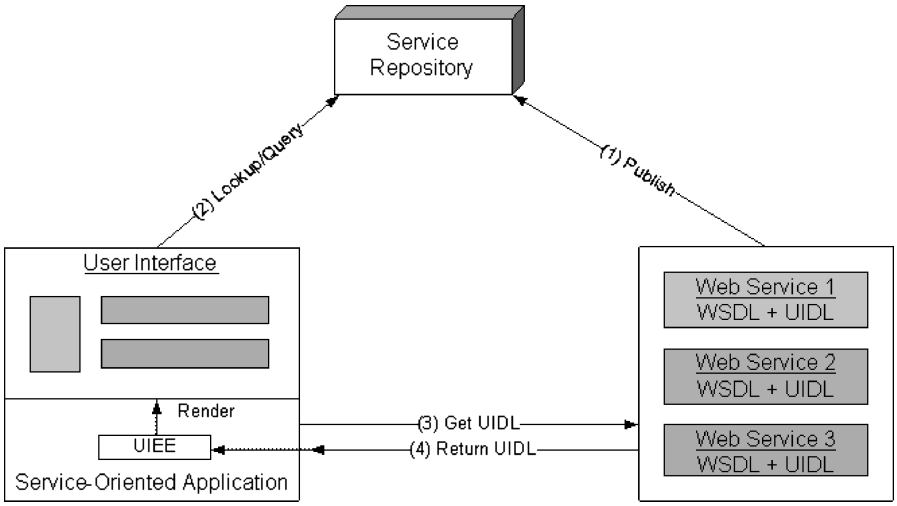


Fig. 2. Overview of UI as a Service (UIaaS).

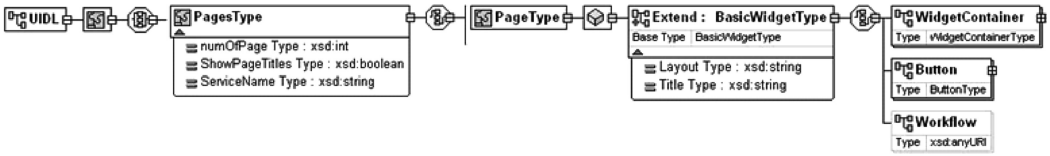


Fig. 3. High-level structure of UIDL schema.

The proposed design works in a way similar to Web Service User Interface (WSUI) proposed in Kassoff et al. [2003]. However, these existing solutions are not based on the concept of UIaaS, which inhibits the composition of user interfaces. Furthermore, they do not have independent representation of UI artifacts as in UIDL. We make further comparison with these works in a later section.

User Interface Description Language (UIDL). UIDL is an application-independent, platform-independent, and programming language-independent user interface description language that is able to express complex Rich User Interfaces (RUI). UIDL is designed as an XML schema which includes a variety of atomic UI artifacts/widgets including label, button, radio button, check list, text box, etc. It also has complex UI widgets that are composed from these atomic UI widgets to form a higher level of UI, such as widget container, page, pages, workflow, etc.

While each of these widgets has its own specific properties to serve particular interactions between user and system, they all are derived from a basic widget. The basic widget contains the common properties for most of the UI widgets, such as position, colour, font, and size. Figure 3 shows the high-level structure of UIDL schema.

Figure 4 shows the schema representation for `BasicWidgetType`, the basic widget from which all basic UI components such as label and button are derived. Figure 5 shows the XML representation of `ListItem`, a basic UI component. Figure 6 shows the schema representation for `WidgetContainerType`, a complex UI component consisting of basic UI components such as label, radio box, text box, and button.

User Interface Execution Engine (UIEE). The User Interface Execution Engine (UIEE) interprets the UIDL and renders a UI according to the description in the UIDL.

```
<xsd:complexType name="BasicWidgetType">
  <xsd:sequence>
    <xsd:element name="Position" minOccurs="0">
      <xsd:complexType>
        <xsd:attribute name="Xcoor" type="xsd:int"/>
        <xsd:attribute name="Ycoor" type="xsd:int"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="Colour" minOccurs="0">
      <xsd:complexType>
        <xsd:attribute name="Background" type="xsd:string"/>
        <xsd:attribute name="Front" type="xsd:string"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="Font" minOccurs="0">
      <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="size" type="xsd:int"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="Size" minOccurs="0">
      <xsd:complexType>
        <xsd:attribute name="height" type="xsd:int"/>
        <xsd:attribute name="width" type="xsd:int"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="WidgetName" type="xsd:string"/>
  <xsd:attribute name="WidgetType" type="xsd:string" use="required"/>
</xsd:complexType>
```

Fig. 4. Schema component representation for `BasicWidgetType`.

```
<...
 WidgetName="xsd:string [0..1]"
 WidgetType="xsd:string [1]"
 DefauItem="xsd:string [0..1]"
 Label="xsd:string [0..1]">
  <Position
   Xcoor="xsd:int [0..1]"
   Ycoor="xsd:int [0..1]"/> [0..1]
  <Colour
   Background="xsd:string [0..1]"
   Front="xsd:string [0..1]"/> [0..1]
  <Font
   name="xsd:string [0..1]"
   size="xsd:int [0..1]"/> [0..1]
  <Size
   height="xsd:int [0..1]"
   width="xsd:int [0..1]"/> [0..1]
  <ListItem> ListItemType </ListItem> [1..*]
</...>
```

Fig. 5. XML instance representation for `ListItem`.

UIEE is a programming language-independent and platform-independent UIDL interpretation and rendering engine. The basic UIEE process of rendering a UIDL is shown in Figure 7.

3.2. User Interface as a Service (UIaaS) as a Context (UIaaSaaC)

The proposed concept of UIaaS is mainly targeted for service-oriented applications. Such applications usually deliver a specific functionality (service) by composing other Web services. Interface services generated using UIaaaS have contextual information embedded in them. Therefore, user interfaces generated using UIaaS can themselves be used as contextual information for service composition. The concept of UIaaSaaC takes UIaaS as a context for a particular application domain, and composes and generates new services.

```
<xsd:complexType name="WidgetContainerType">
  <xsd:complexContent>
    <xsd:extension base="BasicWidgetType">
      <xsd:sequence maxOccurs="unbounded">
        <xsd:choice maxOccurs="unbounded" minOccurs="1">
          <xsd:element name="Lable" type="LabelType" minOccurs="0"
          maxOccurs="1"/>
          <xsd:element name="TextBox" type="TextBoxType" minOccurs="0"
          maxOccurs="1"/>
          <xsd:element name="RadioBox" type="RadioBoxType" minOccurs="0"/>
          <xsd:element name="CheckBox" type="CheckBoxType" minOccurs="0"/>
          <xsd:element name="ListBox" type="ListBoxType" minOccurs="0"
          maxOccurs="unbounded"/>
          <xsd:element name="Button" type="ButtonType" maxOccurs="unbounded"
          minOccurs="0"/>
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="Layout" type="xsd:string"/>
      <xsd:attribute name="Title" type="xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Fig. 6. Schema component representation for `WidgetContainerType`.

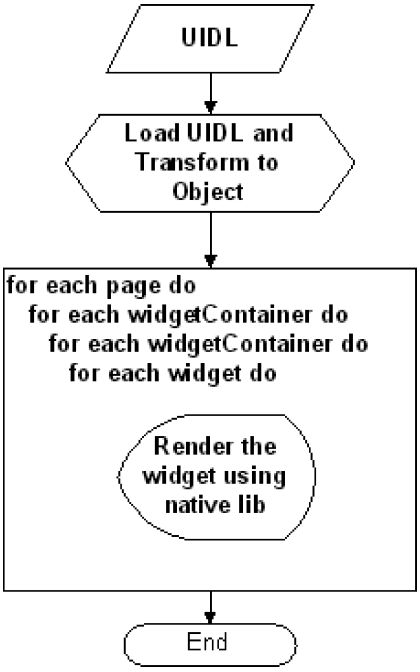


Fig. 7. UIEE flowchart.

Figure 8 shows a simple UI service realized using UIDL and UIEE, based on the concept of UIaaSaaC. The figure also shows the protocol for a typical interaction between the UI service and the back-end application. The interactions, managed by the UIEE, occur as follows.

—First, users can select an item from the application menu bar. The system creates a query based on the user selection and executes it on the service ontology. The execution of the query may result a return of a workflow or an execution of a service depending of the type of query. An example of execution of a service in our human

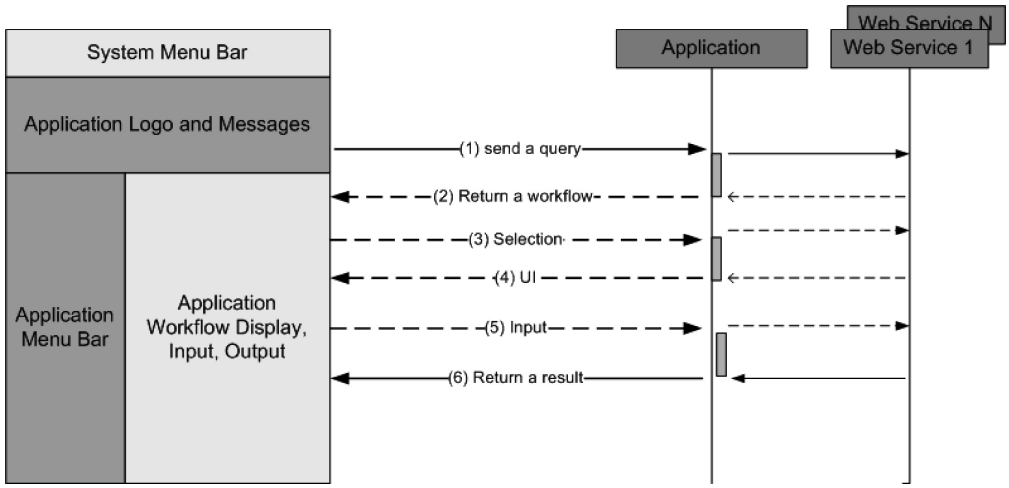


Fig. 8. A simple protocol for interaction between UI service and the back-end applications.

services application is shown in Figure 13 when a user selects a menu item "My Life Track".

—If the execution of an initial query is a (composite) service that requires user's selection or input, the application will return a workflow (composite service), which may reuse an existing composite service or generate one on demand, by composing existing Web services. The workflow should use a standard representation, such as XPDL and/or BPEL, which can be shown in the workflow display area. An example of the execution of a composite service for human services is shown in Figure 12.

—Optionally, each individual activity (can be a Web service) can have its specific user interface for data input.

—If the query cannot provide enough data to execute the activity, a new interface component will automatically pop-up to collect further input data. Such interfaces are generated from Web services using UIaaS concept.

—The collected input data may be used as an input to the Web service or may result in selection of new services.

—Once the execution is completed, the application will return the execution results to the UI service. The execution results can be formatted server-side, either in a standard presentation language such as HTML or in a format understandable by the UI data projector, that is, UIDL.

With a generic user interface, standard protocol and workflow/data representation, the strong dependency between applications and the user interface can be decoupled, thus achieving the purported design goal: User Interface as a Service (UIaaS) to serve multiple Web service-based applications. UIaaS treats each interface service as a first-class object that can be manipulated and combined. Thus generated user interfaces can be provided to service composition as context (UIaaSaaC). Additionally, user's contextual information can be used to generate personalized interfaces using UIDL. We next describe how we have implemented these concepts in WSMS for a human services application.

4. IMPLEMENTATION AND EVALUATION

In this section, we describe an implementation of UIDL and UIEE to realize the concept of UIaaS proposed in this article. The context of our implementation is e-government

services. We first describe the application domain and the need for UIaaS in this domain.

4.1. E-Government Services

E-government services refer to services offered by a government to its citizens, such as social welfare services, financial services, employment services, disability services, and services for the aged population. Our implementation was designed based on e-government services provided by Centrelink (www.centrelink.gov.au), an Australian Government Statutory Agency providing a range of Commonwealth services to the Australian community on behalf of the Australian government. Centrelink works in partnership with client agencies, a variety of government departments, agencies, and community organizations. Centrelink's objective is to design and deliver products and services efficiently and effectively to satisfy their stakeholders, including individual citizens, residents, and other agencies. Some of the products and services provided by Centrelink are listed here.

—Support for employment: Centrelink helps people through transitional periods in their working life by connecting people to the services they need, from vocational training to work experience, through Job Services Australia (JSA). The focus of the employment services is to transition people from welfare to sustainable employment.
—Support for families and guardians: Centrelink provides a range of services for families with children to help them meet costs and cope with the challenges associated with raising children. The financial support includes benefits such as Child Care Benefit, Family Tax Benefit, and Parenting Payment.
—Support for carers: Centrelink provides financial assistance to carers through a range of products, such as Carer Payment and Carer Allowance.
—Support for students: Centrelink provides a range of financial assistance services to students, such as Youth Allowance, Austudy, ABSTUDY, Fares Allowance, Pensioner Education Supplement, and Assistance for Isolated Children.
—Responding to emergencies in communities: Centrelink helps people during transition periods in their lives in case of emergencies such as bushfires, floods, and drought.

In many applications such as e-government services, user interfaces are considered to be among the most important contextual information for the provisioning of services. In that respect, the use of appropriate user interfaces is an important part of making government services more easily and effectively accessible to a wider range of constituencies, including those with little computing skills, physical and mental impairments, and young people who are tuned into the latest in social network technologies. Therefore, user interfaces are key to providing the rich context to many e-government services. Since e-government services are delivered to a wide variety of people with various characteristics and various needs, an important part of service delivery is the customization of services to the service consumer. This includes user interface/interaction customization provided by UIaaS. For example, if a customer is an elderly person, they would need larger font sizes in the user interfaces. Similarly, if a person is color blind, a user interface that uses only black and white is more suitable. Therefore, UIaaS provides an ideal way for providing the rich context to e-government services.

Centrelink uses a service-delivery model based on the notion of life events to deliver a variety of products and services to people on behalf of client departments. In the life events approach, the customers come to Centrelink at points of transition or crisis in their lives, such as leaving school, becoming unemployed, retiring from the workforce, separating from a partner, and natural disasters. Centrelink then tailors a range of government services to meet the customer's needs. For the purpose of our

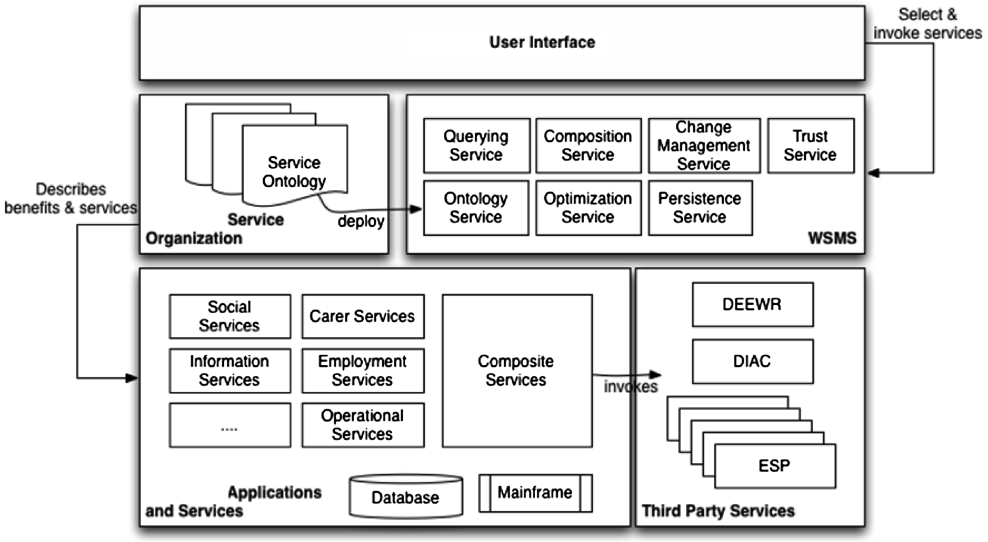


Fig. 9. UIaaS implementation architecture.

implementation, we consider the case of a person with the life event of "carer for sick family member".

4.2. UIaaS Implementation Architecture

Figure 9 depicts an implementation architecture of UIaaS in WSMS on a coarse-grained level. It is implemented using open source technologies and is based on the Java 6 platform. Each WSMS component is implemented as a dedicated Web service using the JAX-WS technology. WSMS uses Apache CXF as a Web service stack that implements the latest version of JAX-WS (v2.1). Apache Tomcat is used as a Java Servlet container for hosting the Web services.

The implementation is split into four core parts.

(1) User Interface Manager. This user interface (UI) component constitutes the UIDL-based user interfaces that allow customers to register for services, claim services, view/update their profiles, and view their current LifeTrack. The LifeTrack is a graphical view of all events reported by a customer, as well as all received benefits and services. The UI is implemented using the Google Web Toolkit (GWT), which contains a set of efficient tools for developing Web applications.
(2) Service Organization. This component declaratively describes all benefits, services, and products and their relationship in the human services domain, by using ontologies. In particular, the WSMS system uses WSMO (Web Service Modeling Ontology) and WSML (Web Service Modeling Language) to model and describe the different artifacts. An open source software WSMO Studio was used to create and edit the ontologies. The implementation also used corresponding tools and APIs (Application Programming Interfaces) to parse the ontology at runtime.
(3) Web Service Management System (WSMS). WSMS implements the core infrastructure that exposes all its functionality as services. These services are essentially middleware services, especially tailored for building service-oriented systems. The WSMS components are generic and not related to any particular application domain. The *Querying Service* acts as the central entry point and coordinator for WSMS. The user interface manager interacts with WSMS to find the corresponding atomic/composite services that need to be invoked. The user interface manager

invokes the services (either atomic or composite) identified by the querying service to accomplish user interaction.

(4) Applications and Services. This part constitutes the domain-specific part of the overall system. It comprises all existing applications, Web services, databases, etc. in the human services domain. A key requirement for using WSMS is that all relevant application services are designed in a service-oriented way, that is, by exposing them as Web services. In the current prototype, we simulate a small fraction of the e-government human services within our own environment.

(5) Third Party Services. External services constitute a core part of government business. These include services provided by external parties, such as government departments and agencies, and various private and non-profit Employment Service Providers (ESPs). Services provided by these organizations are typically invoked within the human services business processes. In the current prototype, we simulate these services.

All these different system components interact with and depend on each other. For example, a prerequisite for using WSMS is the creation of a service organization ontology. This requires a considerable amount of time to identify the core business services and describe them in the ontology. These descriptions are then deployed to the WSMS and managed by the *Ontology Service*. Current WSMS implementation does not enforce any particular ontology design, as long as the ontologies are modeled cleanly in WSMO and encoded in WSML. Additionally, the concrete Web services from the human services environment are annotated with the ontological concepts from the service organization ontology, which enable semantic interpretation of the services. The data of the concrete services (e.g., WSDL description) are then maintained by the *Persistence Service* in WSMS.

We next describe the three major components of this implementation relevant to the UIaaS proposed in this article.

Querying Service. The querying service retrieves a Web service query for a desired service. A query in WSMS is based on the functionality described in the Service Organization Ontology. It can also contain a number of nonfunctional aspects such as trustworthiness. The query language is an SQL-like language that enables users and/or developers to query required services at a semantic level, like querying databases, so that they do not have to deal with the complexity of interfacing to individual Web services. The query language is defined as an ANTLR grammar [Parr 2007]. This allows automatic generation of the parser of our query language using existing tools. Upon receiving a query, the querying service parses the query and interacts with the ontology service to check whether the desired services exist (i.e., are described as part of the service organization). If the service exists, the query returns the service to the requester so that it can be invoked. If no service can be found, the querying service will interact with the service composer to compose a service based on the existing services in the service organization.

Ontology Service. The ontology service encapsulates the access to various ontologies used for service management. It provides a simple and technology-independent access to various metadata encoded in ontologies. The current implementation uses WSMO/WSML as concrete ontology model and language. The ontology service manages various ontologies that are required for implementing WSMS, including service ontology, user interface ontology, trust ontology, and change ontology. This service simplifies the task of service querying by providing a unique API to all ontologies. We developed a user interface ontology (UI Ontology) to describe the UIaaS design proposed in this article. Figure 10 shows the visualization of UI ontology. The UI ontology describes the

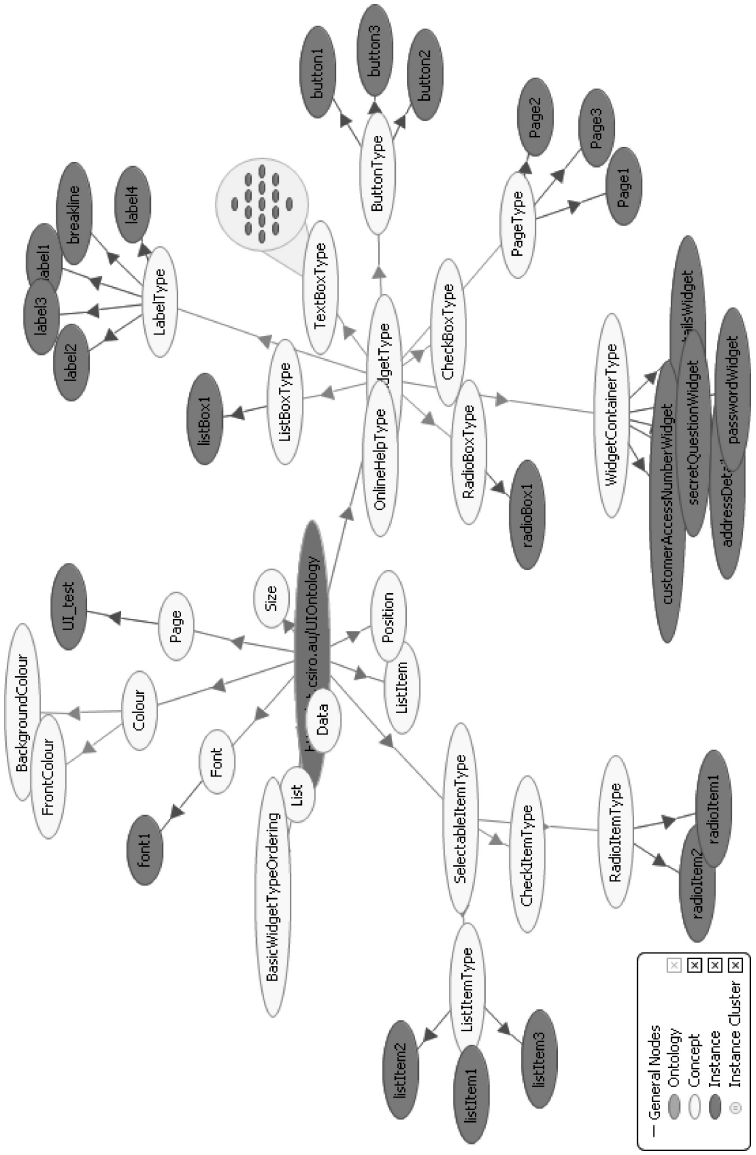


Fig. 10. UI ontology visualization.

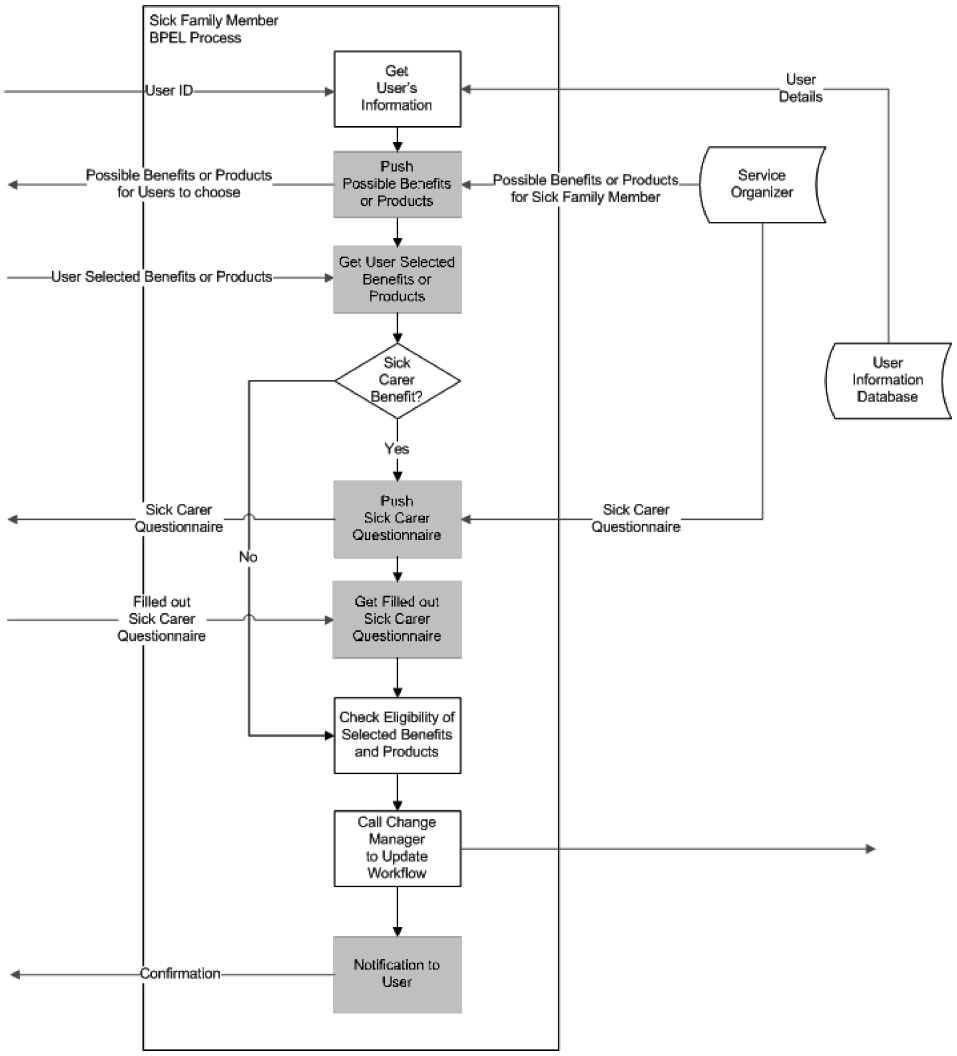


Fig. 11. SickFamilyMember service process interaction design.

various UI components and their relationships as described in Section 3.2. Complex UI concepts can be derived from the basic UI concepts in the UI ontology. The UI ontology can be extended to add more complex UI concepts.

Composition Service. The composition service creates service composition as a service by matching pre- and post-conditions of the services and generating the composite service flow, based on the matches [Medjahed et al. 2003]. The input of the composition service is a query that is received by the querying service. To successfully compose a set of services, these services need to be annotated adequately (using the ontology), especially with the pre- and post-conditions specified. Composed services may consist of a composition of Web services as well as interface services.

4.3. An Example Case Study

In the following, we describe an example application service that implements UI as a service. Figure 11 shows an example interaction design for an e-government service

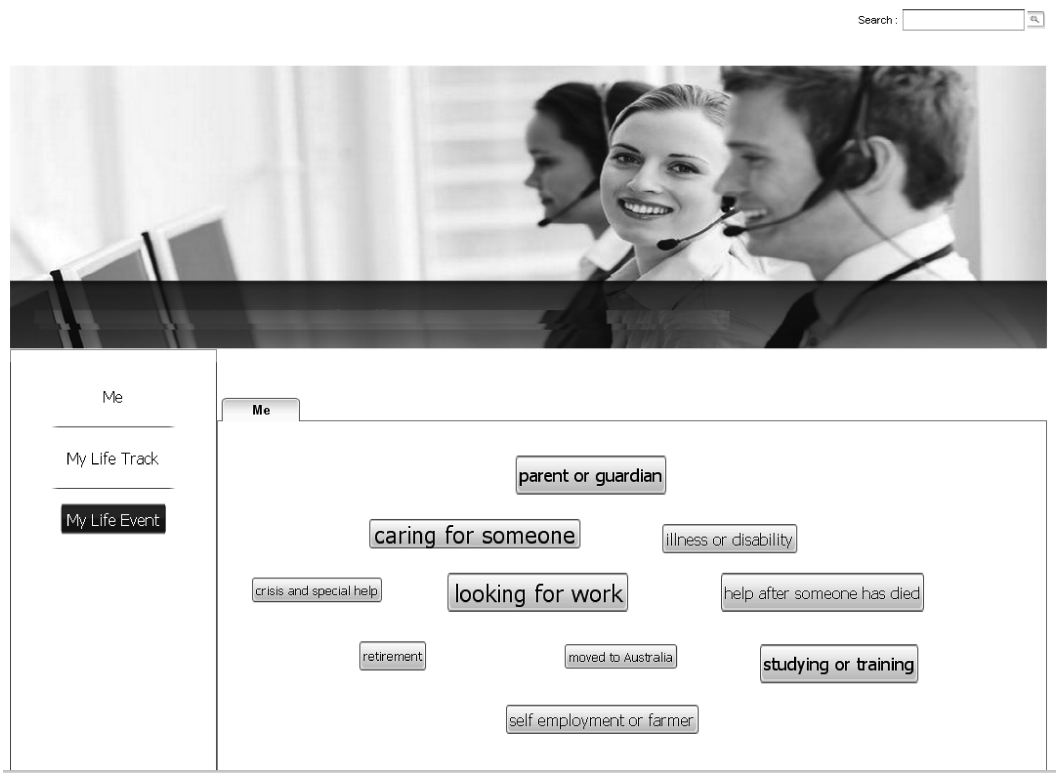


Fig. 12. Life events tag-cloud generated using the concept of UIaaS.

for a customer seeking services as a "carer for sick family member". The shaded boxes represent the user interfaces pushed to the user. The service shown requires multiple interactions with the user (service consumer). In addition, the questions to be asked to the user need to be customized to the user's circumstances. Two users requesting the same services may have different circumstances; hence, the questions to ask should be different. Additionally, some of the information may have already been provided by the user during registration or while accessing other services. Therefore, the user interface needs to be aware of the context of the service/application used and provide suitable interfaces accordingly. Our proposed UIaaS resolves this issue by considering UI as composed of atomic parts as described in the UIDL in Section 3.2. Based on the requirement at a particular instance, the questions in the UI can be composed of the atomic parts as required.

In addition to customization of the UI components as needed, our proposed UI service design also enables customization of usability/readability of the UI. Standard customer profiles can be saved for different demographics, such as aged and colour-blind. Before rendering the UI, the User Interface Execution Engine (UIEE) checks the profile of the current customer, and if a matching profile is found—say aged customer—then the UI is customized with larger font sizes automatically. Similarly, if the customer is using a different device for accessing the service, our interface service accepts the device profile and automatically adjusts the display and contents of the application interface to the device.

Figure 12 shows an instance of context-awareness in the user interface. The screen shows the available life events for the customer's selection, reported as a tag-cloud.

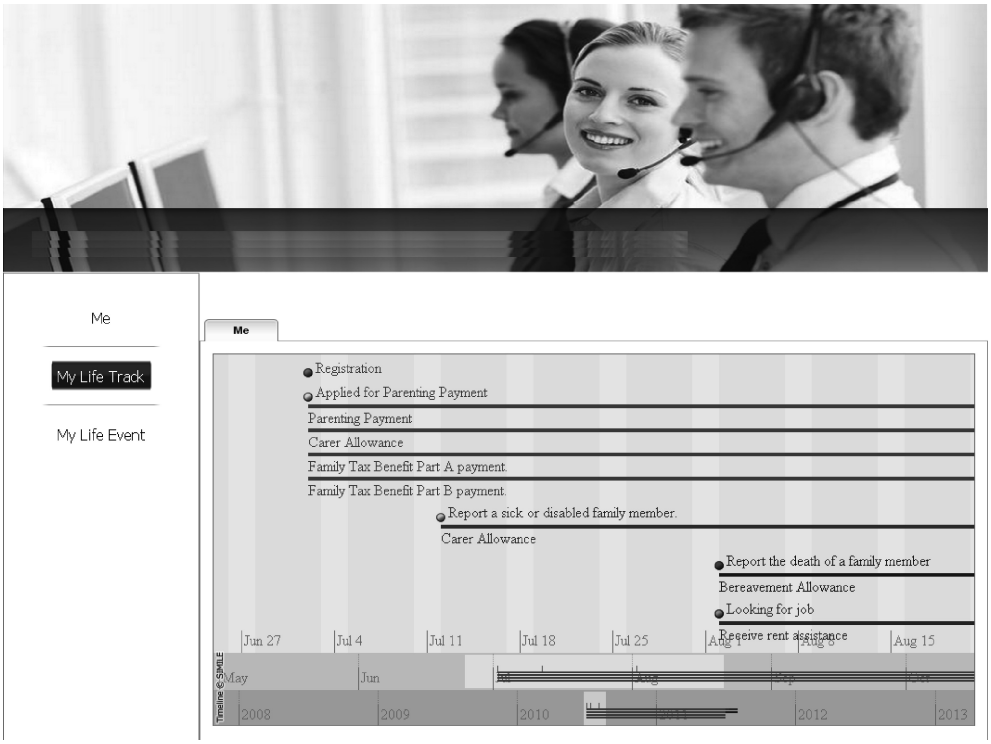


Fig. 13. LifeTrack generated using the concept of UIaaS.

Based on the customer's profile, certain life events are highlighted. As seen in this instance of a 30-plus married woman, the UI highlights "looking for work" and "caring for someone" since most people having similar profiles are likely to be experiencing these two life events. Such user profiling is performed using domain knowledge. In addition, if further information is available in the profile, such as the user is color-blind, elderly, or with low vision, the UI changes accordingly with black and white fonts and larger fonts, respectively.

Figure 13 shows an instance of LifeTrack—a mechanism for the user (human services customer) to track their life events and the benefits they are receiving/have received. This UI is also generated using the concept of UIaaS.

To summarize, our context-aware user interface design provides the following benefits to application developers.

—Application developers can register and publish generic user/workflow interfaces in the service repository.
—Application developers can search and query the user/workflow interfaces published in the registry and use them in new applications.
—Application developers can compose the UI for a new application by using already existing user interface services.
—Application developers can publish the composed UI services for future use, so that other application developers developing similar applications can use the published interface as a service using the principles of service-oriented systems.

Table I. Functional Comparison of UIaaS and Other Approaches

Approaches/Properties →	C	AI	CA	GI	SO
XUL [Butter et al. 2007]	×	✓	✓	×	×
User-Centric Approach [Mowafi and Zhang 2007]	×	✓	✓	×	×
Hydrogen Approach [Hofer et al. 2003]	×	×	✓	×	×
Adaptive Approaches [Liu et al. 2003; Kumar et al. 2006]	×	✓	✓	✓	×
Generic Approach [Weis et al. 2006]	×	✓	✓	×	✓
CAPNET [Repo 2004]	×	✓	✓	×	×
Connected User Interface [Kruger et al. 2004]	×	×	✓	×	×
GUI for Web Services [Kassoff et al. 2003]	×	✓	✓	✓	×
UIaaS	✓	✓	✓	✓	✓

Note: C: Composition, AI: Application Independent, CA: Context-awareness, GI: GUI Independent, SO: Service Orientation.

4.4. Evaluation

This section presents the evaluation of the proposed UIaaS concept, as well as the implemented user interface for a human services application. We classify our evaluation into two broad categories: functional and nonfunctional. We next describe the results of our evaluation in these two broad categories.

Functional Evaluation. Functional evaluation refers to the technology/concept evaluation and deals with the interactions between user interfaces and underlying application systems. This evaluation helps to differentiate the proposed concept from existing approaches proposed in the literature [Butter et al. 2007; Mowafi and Zhang 2007; Liu et al. 2003; Kumar et al. 2006; Weis et al. 2006; Kassoff et al. 2003]. We have investigated a number of characteristics related to the development of user interfaces, such as ease of generating and combining user interface artifacts within a service-oriented system; reusing already existing user interface artifacts; etc., and analyzed how the proposed technologies achieve them. We found that there is no standard set of metrics for conducting functional evaluation of the user interface from the point-of-view of technologies. Therefore, we first define a set of metrics for our evaluation purpose as follows.

—Composition (C): A user interface is said to be composable if it can be combined with other services (including other interface services).
—Application-independent (AI): A user interface can be used for different applications without significant modifications.
—Context-awareness (CA): A user interface takes into account contextual information while generating relevant artifacts.
—GUI-independent (GI): A user interface functionality is independent of the graphical user interface presented to the user.
—Service orientation (SO): A user interface is developed using "pure" service-oriented approach, meaning a user interface can be described and advertised like other web services.

A number of technologies or concepts have been proposed and used in developing user interfaces for web applications [Butter et al. 2007; Mowafi and Zhang 2007; Liu et al. 2003; Kumar et al. 2006; Weis et al. 2006; Kassoff et al. 2003; Repo 2004; Kruger et al. 2004]. We compare them with our proposed UIaaS approach using the just-defined metrics. Table I shows a high-level comparison result. A brief description of each of these other approaches is presented as related work in the next section.

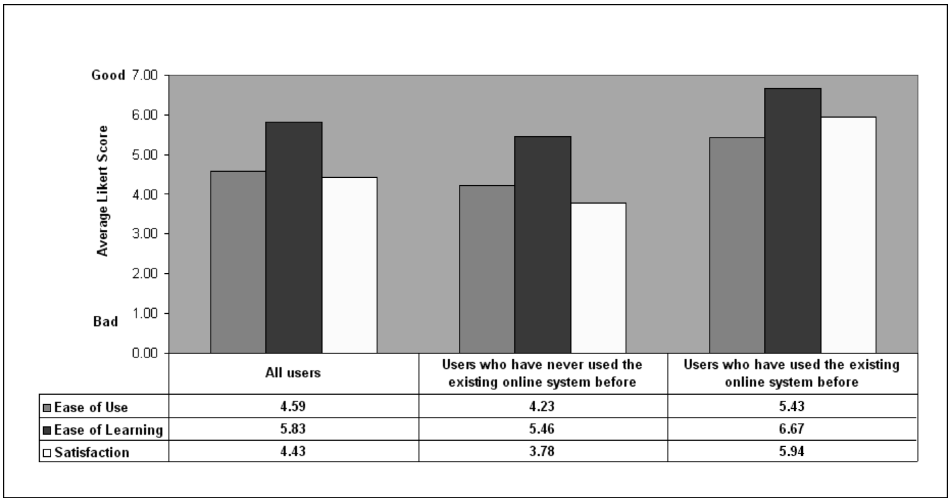
Nonfunctional Evaluation. The most popular method used for evaluating user interfaces based on nonfunctional properties is the *usability test* [Jeffries et al. 1991]. This method is very popular among researchers working in the field of HCI. The usability

test includes structural layout of the user interface artifacts, as well as their ease of use. The metrics used in this method include ease of use, ease of learning, satisfaction, usefulness, comfort, convenience, shape, color, brightness, etc. [Han et al. 2000]. This method of evaluation requires involvement of end users or experts. In summary, the nonfunctional properties evaluate the interactions between the users and user interfaces. There are two types of usability test: *expert review* and *end user evaluation*. We first conducted an end user evaluation. In order to better understand the results of the end user evaluation, we followed up with an expert review. We next describe our evaluation process and results of both types of usability test.

End User Usability Test. We first designed a set of questions. There were three types of questions: factual, opinion, and attitude. The first type capture generic information about the users such as age, sex, use of computer, use of already existing online system, etc. The later two types were used to evaluate users' experience with the user interface. There are a number of wellknown methods to conduct end user usability tests such as QUIS [Chin et al. 1988], PUEU [Davis 1989], NAU [Nielsen 1993], CSUQ [Lewis 1995], ASQ [Lewis 1995], PUTQ [Lin and Salvendy 1997], and USE [Lund 2001]. We studied these different methods and evaluated their appropriateness for our system. We have decided to use Lund's USE questionnaires to conduct the end user usability test. The decision was made based on our studies of the appropriateness of the method for our system and the advice from the usability experts. However, we have used only three criteria: *ease of use*, *ease of learning*, and *satisfaction*. We left out the *usefulness* criteria as it is not an appropriate criteria for our system. Ease of use has 11 questions as defined in Lund's USE questionnaires (e.g., it is simple to use, it is user friendly, etc.). The ease of learning criteria has four questions (e.g., I learned to use it quickly, I easily remember how to use it; it is easy to learn to use it, etc.). The satisfaction criteria has seven questions (e.g., I am satisfied with it, I feel I need to have it, etc.).

Users were asked to imagine themselves as a parent or guardian of two children. One of their children goes to primary school and the other is disabled and needs special care. They would like to get assistance from the government to help raise their kids. They were made aware that they could claim eligible benefits online and were given a link to our system. They were asked to perform four tasks: register on the online system, claim the benefits as a parent or guardian of two children, observe the successful transactions and states, and observe and update their profile. Users were asked to play with the system as much as they liked after performing the given tasks. Users were then asked to answer 22 questions from Lund's three categories on a *Likert* scale [Likert 1932] from 1 (strongly disagree) to 7 (strongly agree). Users were also given an option of choosing "not applicable".

We conducted the usability test with a variety of users, representative of citizens. There were ten total users. 70% of them had never used an existing online system for claiming benefits from the government before and, 30% of the users had used the existing system for claiming one or more types of benefits. Figure 14 shows the results of our evaluation. The average score for all users was roughly 5 out of 7. The score went up to above 6 when we consider only those users who had used the existing online system. This provides an indirect comparison with the existing system. Another observation is that users seem to find our interface easy to learn, independent of their experience with the existing system. However, the satisfaction level is very low among users who had not had any experience with the existing system. There is a significant difference at the level of satisfaction between the users who had not had any experience and those who had some kind of experience with the existing system. In order to better understand this gap, we conducted expert usability reviews. We next present the process and results on the expert usability reviews of our interface.



	All users	Users who have never used the existing online system before	Users who have used the existing online system before
Ease of Use	4.59	4.23	5.43
Ease of Learning	5.83	5.46	6.67
Satisfaction	4.43	3.78	5.94

Fig. 14. Results for end user usability test.

Expert Usability Reviews. The expert usability review is also known as expert review, or usability audit, or heuristic evaluation. This is an evaluation of a user interface versus common usability best practices and heuristics by a trained usability professional. We invited five usability experts from the HCI and user experience research group within CSIRO ICT Centre. They were asked to perform some critical tasks in the system. The tasks were the same as those of the end user usability test: registration, claiming a benefit, observing the status of the claim, and updating the profile. The same set of tasks were chosen with an aim of correlating the results between the two usability tests. In addition to performing the tasks, experts were allowed to play with the system as much as they liked. We then asked experts to provide scores to Neilson's ten usability heuristics [Nielsen 1993]. The scoring scale is set to a Likert scale from 1 (bad) to 7 (good).

Figure 15 shows the results of the expert usability test. It shows that experts have given a high score (>5) for the first four criteria. This result is consistent with the end user usability test, as these criteria have direct influence on the ease of learning. The expert reviews identify two main areas of concerns: lacking or minimal error handling and clearly marked exits. We observed this as the main reason for low satisfaction scores among inexperienced users in our end user usability test. We plan to use these results as a guide for improving our user interface in the future.

5. RELATED WORK

Context has been the center of sustained research in several areas, including databases [Kashyap and Sheth 1996], distributed computing [Yau et al. 2002], semantics [Noy and Musen 2001], and recently, service computing [Baldauf et al. 2007]. Context has been used to model a range of situations including geo-spatial and temporal information [Moldovan et al. 2005], mobile computing [Riva and Toivonen 2007], and computing environments [Chen et al. 2003]. Context is broadly defined as the environment characterizing the situation in which the entity of interest is in [Abowd et al. 1999]. If this definition is applied to Web services, context is interpreted as the information that characterizes the situation in which a Web service operates. Example contexts include location and time.

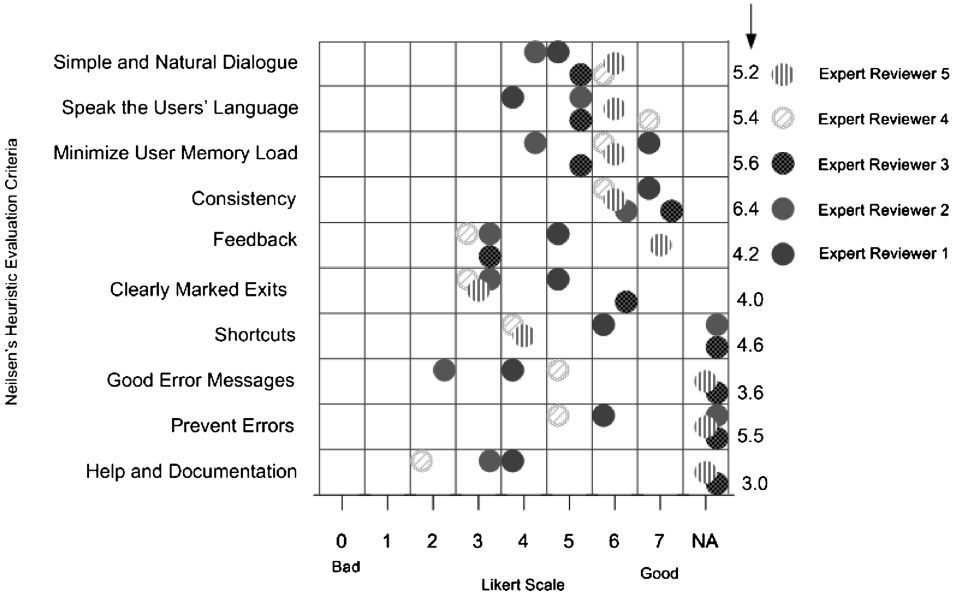


Fig. 15. Results for expert usability test.

Context-awareness is particularly important for Web service composition since context-aware Web services can adapt to diverse situations caused by the diversity of users. Current composition methods can hardly cope with context-aware service composition. Usually, whenever a new context is available, most current composition methods have to reconsider the whole composition. Our own Aspect-Oriented Programming (AOP)-based approach overcomes this problem by leveraging the key features of AOP of dividing a service into two parts [Li et al. 2009]. The first part is the main service-implementing business logic and the second is the context aspects concerning context services. A context aspect describes the context service which should be executed to adapt the main service to a particular context. The main part implements the application functions, while the aspects part adds some auxiliary features to the main functions. These two parts are separated clearly in syntax. When the auxiliary features are not needed or changed, the aspects part can be independently removed or updated. For example, logging the execution results of some Java methods for debugging can be implemented in an aspect; after debugging is finished, the logging aspect will be removed without affecting the debugged Java methods. An important technique underlying AOP is the *weaving* mechanism. Weaving, in this instance, refers to the ability of the context services, to be meshed (composed) into the main service, such that the context services may be invoked when executing the main service. The weaving approach is semantic-based: a context service is inserted into a main service according to its semantics.

This section reviews the literature related to context-aware user interfaces from the point of view of Web services and service-oriented architectures. We classify the relevant literature into three broad categories as follow.

5.1. Context-Aware Web Services

The centre of service computing has been in automatic service composition [Berardi et al. 2005; Narayanan and McIlraith 2002; Kona et al. 2008]. However, the focus in the past has been on how to use functional and nonfunctional attributes of the services

to achieve automatic composition. Recently, there has been increasing interest in the content-aware Web service and service composition. A body of work has been done in this area in recent times [Mostéfaoui and Hirsbrunner 2003; Zhou et al. 2007; Maamar et al. 2007; Medjahed and Atif 2007; Maamar et al. 2005; Maamar et al. 2006; Sheng and Benatallah 2005].

Medjahed and Atif [2007] introduced a generic definition of Web service context through an ontology-based categorization of contextual information. The rule-based service matchmaking was proposed to consider the relevant context. Similarly, Li et al. [2009] construct an ontology (context ontology) to facilitate the detection of join-point semantically. However, they are more concerned about the support of changing context as they treat the context as an aspect in Aspect-Oriented Programming (AOP).

Gu et al. [2005] proposed a service-oriented context-aware middleware (SOCAM) architecture for the building and rapid prototyping of context-aware services, but with focus on infrastructure support to context-aware systems. A number of context-aware architectural approaches has been proposed in literature [Gu et al. 2004; Riva and Toivonen 2007; Pawlak et al. 2001]. However, it is difficult to develop complex context-aware Web services. This issue has been addressed to some extent by Sheng and Benatallah [2005] by proposing a modeling language for model-driven development of context-aware Web services based on the Unified Modeling Language (UML), called ContextUML. Sheng et al. [2009] have provided a development platform, ContextServ, for rapid development of context-aware Web services using ContextUML.

Mrissa et al. [2007] proposed a context-based approach for semantic Web services composition. The approach enables developers to annotate WSDL descriptions to describe contextual details; to deploy a context-based mediation architecture to allow explicit assumptions on data flow; and to automatically generate and invoke Web service mediator to handle data heterogeneities during Web service composition. The context ontology was defined to make context explicit for each concept of a domain ontology.

Maamar et al. [2005] proposed an approach for context-oriented Web service composition by using agents. The context model presented [Maamar et al. 2007; Maamar et al. 2005] comprises four types of context: W-context deals with Web services' definitions and capabilities; C-context addresses how Web services are discovered and combined; S-context handles the semantic heterogeneity that arises between Web services; and R-context focuses on the performance of Web services. After identifying these different types of context in Web service composition, the authors presented a policy-based approach for developing a context-oriented Web service. Three types of policies were introduced to support transitions between the four context levels: engagement, mediation, and deployment. While the proposed context model is advantageous in terms of context categorization, it is less generic than the one defined by Medjahed et al. [2003].

5.2. Context-Aware User Interfaces

A user interface is context-aware if it is cognizant of its situation and is able to modify its behaviour according to the changing situation. Context-awareness is particularly important for user interfaces since context-aware user interfaces can adapt to diverse situations caused by the diversity of the users. Achieving context-aware user interface in service computing is a challenging task. When developing a user interface for a particular application, it is often difficult to predict all possible contexts in which the interface is used. Moreover, contexts for user interfaces are dynamic in nature and keep evolving with the change in the situation of use.

Context-aware user interfaces have been well studied in the context of mobile devices [Hofer et al. 2003; Butter et al. 2007]. Mobile devices available today are heterogeneous with regard to their display and input capabilities and used software platform

configurations. In addition to adapting the interface of an application to the device characteristics, the interface needs to be adapted to the current context of the user. This adds extra burden to developers. Butter et al. [2007] have overcome this limitation by proposing an XUL-based User Interface Framework. This framework separates the UI adoption from the application logic and offers portability to different Java ME platform configurations. Using this framework, the User Interface (UI) adapts itself automatically on context changes and changes to different screen resolutions or orientations without increasing code complexity for the developers. The context here is defined around both device characteristics and user behaviour. Mowafi and Zhang [2007] have defined the context based on user characteristics. They proposed a user-centric approach for obtaining and parsing context, driven by users' context of interest, that offers more interactive and construed context to users.

There is a body of work in adaptive user interfaces [Liu et al. 2003; Kumar et al. 2006]. For example, Liu et al. [2003] have proposed a technique of adapting user interfaces for applications using user interface events and frequencies. By computing episode frequencies and implication relations, they can automatically derive application-specific episode associations, and therefore, enable an application interface to adaptively provide just-in-time assistance to a user. Kumar et al. [2006] have developed an adaptive user interface in media applications. Not all user interface adaptation is beneficial in context-aware systems. Some adaptations introduce poor usability. Paymans et al. [2004] have presented a study on usability trade-offs. Similarly, Lavie et al. [2010] examine the positive and the possibly adverse effects of adaptive user interfaces in the context of an in-vehicle telematic system.

5.3. User Interfaces for Web Services

It is obvious from these discussions that today's context-aware services are isolated systems designed for special scenarios. For example, a context-aware airline service knows when your plane leaves and whether it is on time or not. Similarly, context-aware taxi service knows how long it takes to the airport from your home. There is a need to federate context-data retrieved from different services. Weis et al. [2006] identified several challenges to address this need. One of them is defining a general-purpose user-interface for such applications that allows users to deal with context-data and interact with context-aware services. They have developed a general-purpose user interface which is a collage of instant messenger, roadmap, and Web browser. However, the concept is still limited to a specific travel application. It is not suitable for capturing context information about other application domains, such as human services. Furthermore, user interface itself is not context-aware and is fixed for five views: people, places, maps, objects, and activities.

Repo [2004] addressed this problem, to some extent, in the pervasive-networking environment. There is a need for providing seamless access to surrounding services using a personal mobile device in such an environment. Repo proposed a reference model architecture to facilitate the development of adaptable user interfaces in a context-aware pervasive-networking environment. Though the paper was able to accommodate the idea of context-awareness in the user interface, the proposed solution was still for a domain-specific application.

Kruger et al. [2004] uses a different approach. Rather than designing one unified user interfaces, they focus on invoking specialized user interfaces for different situations. However, the proposed solution is still targeted to specific application. This means the user interface is contextualized for limited contexts. We have seen similar limitations with many other approaches proposed in the literature. For example, Vermeulen et al. [2008] have proposed to augment service descriptions with high-level user interface models to support automatic user-interface adaptation in a mobile

computing environment. Their method was built by defining ontology for a hierarchical task structure and selected presentation information. This allows end-users to interact with services on a variety of platforms.

It is clear from these discussions that context-aware Web services and context-aware interfaces have been studied from two different perspectives. The novelty of the approach presented in this article is to bring these two perspective together to provide context-aware user interface as a Web Service.

6. CONCLUSION

We propose a new concept of User Interface as a Service (UIaaS) to model context-aware services. We describe a specialized XML-based language for modeling UIaaS as a context. Contextual information related to user interface types is implicitly mashed-up with the requested services when the results are displayed. Context consisting of UIaaS is organized in an ontology, which describes the different classes of user interfaces. We address the interface complexity issue from two aspects. First, we design a UI service with uniform GUI and standard communication protocol to serve different applications. Second, we leverage our own Web Service Management System (WSMS) prototype to manage UIaaS. They represent a special type of service, thus enabling UIaaS to be managed as any other service. The serendipitous effect is that context services consisting of user interfaces may now be organized, composed, queried, changed, and managed as any other services. We implemented the designed UI service and demonstrated its usefulness leveraging our own WSMS prototype. We evaluated the proposed interface design in two ways: (i) based on functional properties, and (ii) based on nonfunctional properties. We conducted two types of nonfunctional evaluation: (i) end user usability test, and (ii) expert reviews. Based on these evaluations, we intend to refine the user interface in the future.

ACKNOWLEDGMENTS

We wish to thank the WSMS project members for their contribution in the design and implementation of the WSMS prototype including the UIaaS manager. In particular, we would like to acknowledge the contribution of Florian Rosenberg, Xuan Zhou, Jemma Wu, Payam Aghaei Pour, Armin Haller and Hongbing Wang.

REFERENCES

ABOWD, G. D., DEY, A. K., BROWN, P. J., DAVIES, N., SMITH, M., AND STEGGLES, P. 1999. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*. Springer-Verlag, 304–307.

BALDAUF, M., DUSTDAR, S., AND ROSENBERG, F. 2007. A survey on context-aware systems. *Int. J. Ad Hoc Ubiq. Comput. 2,* 4, 263–277.

BERARDI, D., CALVANESE, D., GIACOMO, G. D., HULL, R., AND MECELLA, M. 2005. Automatic composition of web services in Colombo. In *Proceedings of the 13th Italian Symposium on Advanced Database Systems*. A. Calì, D. Calvanese, E. Franconi, M. Lenzerini, and L. Tanca, Eds., 8–15.

BOUGUETTAYA, A., NEPAL, S., SHERCHAN, W., ZHOU, X., WU, J., CHEN, S., LIU, D., LI, L., WANG, H., AND LIU, X. 2010. End-to-end service support for mashups. *IEEE Trans. Serv. Comput. 3,* 3, 250–263.

BROWN, P., BOVEY, J., AND CHEN, X. 1997. Context-aware applications: From the laboratory to the marketplace. *IEEE Pers. Comm. 4,* 5, 58–64.

BUTTER, T., ALEKSY, M., BOSTAN, P., AND SCHADER, M. 2007. Context-aware user interface framework for mobile applications. In *Proceedings of the 27th International Conference on Distributed Computing Systems Workshops*. 39–40.

CHEN, H., FININ, T., AND JOSHI, A. 2003. An ontology for context-aware pervasive computing environments. *Knowl. Engin. Rev. 18,* 3, 197–207.

CHIN, J. P., DIEHL, V. A., AND NORMAN, K. L. 1988. Development of an instrument measuring user satisfaction of the human-computer interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, New York, NY, 213–218.

CURBERA, F., DUFTLER, M., KHALAF, R., NAGY, W., MUKHI, N., AND WEERAWARANA, S. 2002. Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Comput. 6,* 2, 86–93.

DAVIS, F. 1989. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quart. 13,* 3, 319–339.

DEY, A. K., ABOWD, G., PINKERTON, M., AND WOOD, A. 1998. Cyberdesk: A framework for providing self-integrating context-aware services. *Knowledge-Based Syst*. 47–54.

GU, T., PUNG, H. K., AND ZHANG, D. 2005. A service-oriented middleware for building context-aware services. *J. Netw. Comput. Appl. 28,* 1, 1–18.

GU, T., PUNG, H. K., AND ZHANG, D. Q. 2004. Toward an OSGi-based infrastructure for context-aware applications. *IEEE Perv. Comput. 3,* 4, 66–74.

HAN, S. H., YUN, M. H., KIM, K.-J., AND KWAHK, J. 2000. Evaluation of product usability: development and validation of usability dimensions and design elements based on empirical models. *Int. J. Indust. Erg. 26,* 4, 477–488.

HOFER, T., SCHWINGER, W., PICHLER, M., LEONHARTSBERGER, G., ALTMANN, J., AND RETSCHITZEGGER, W. 2003. Context-awareness on mobile devices—the hydrogen approach. In *Proceedings of the Hawaii International Conference on System Sciences*. 292–2101.

JEFFRIES, R., MILLER, J. R., WHARTON, C., AND UYEDA, K. 1991. User interface evaluation in the real world: a comparison of four techniques. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, New York, NY, 119–124.

KASHYAP, V. AND SHETH, A. P. 1996. Semantic and schematic similarities between database objects: A context-based approach. *Int. J. VLDB 5,* 4, 276–304.

KASSOFF, M., KATO, D., AND MOHSIN, W. 2003. Creating guis for web services. *IEEE Int. Comp. 7,* 5, 66–73.

KONA, S., BANSAL, A., BLAKE, M. B., AND GUPTA, G. 2008. Generalized semantics-based service composition. In *Proceedings of the IEEE International Conference on Web Services*. 219–227.

KRUGER, A., BUTZ, A., MÜLLER, C., STAHL, C., WASINGER, R., STEINBERG, K.-E., AND DIRSCHL, A. 2004. The connected user interface: Realizing a personal situated navigation service. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*. ACM Press, New York, NY, 161–168.

KUMAR, M., GUPTA, A., AND SAHA, S. 2006. An approach to adaptive user interfaces using interactive media systems. In *Proceedings of the 11th International Conference on Intelligent User Interfaces*. ACM Press, New York, NY, 312–314.

LAVIE, T. AND MEYER, J. 2010. Benefits and costs of adaptive user interfaces. *Int. J. Hum. Comput. Stud. 68,* 8, 508–524.

LEWIS, J. R. 1995. IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use. *Int. J. Hum. Comput. Interact. 7,* 1, 57–78.

LI, L., LIU, D., AND BOUGUETTAYA, A. 2009. Semantic weaving for context-aware web service composition. In *Proceedings of the International Conference on Web Information System Engineering*. 101–114.

LIKERT, R. 1932. A technique for the measurement of attitudes. *Arch. Psych. 140*, 1–55.

LIN, H. X., C. Y.-Y., AND SALVENDY, G. 1997. A proposed index of usability: A method for comparing the relative usability of different software systems. *Behav. Inf. Technol. 16,* 5, 267–277.

LIU, J., WONG, C. K., AND HUI, K. K. 2003. An adaptive user interface based on personalized learning. *IEEE Intel. Sys. 18,* 2, 52–57.

LUND, A. 2001. Measuring usability with the USE questionnaire. *Usability User Exp. Newsl. STC Usabil. 8,* 2.

MAAMAR, Z., BENSLIMANE, D., AND NARENDRA, N. C. 2006. What can context do for web services? *Comm. ACM 49,* 12, 98–103.

MAAMAR, Z., BENSLIMANE, D., THIRAN, P., GHEDIRA, C., DUSTDAR, S., AND SATTANATHAN, S. 2007. Towards a context-based multi-type policy approach for web services composition. *Data Know. Eng. 62,* 2, 327–351.

MAAMAR, Z., MOSTEFAOUI, S. K., AND YAHYAOUI, H. 2005. Toward an agent-based and context-oriented approach for web services composition. *IEEE Trans. Knowl. Data Engin. 17,* 5, 686–697.

MALIK, Z. AND BOUGUETTAYA, A. 2009. Rateweb: Reputation assessment for trust establishment among web services. *Int. J. VLDB 18,* 4, 885–911.

MEDJAHED, B. AND ATIF, Y. 2007. Context-based matching for web service composition. *Dist. Parall. Datab. 21,* 1, 5–37.

MEDJAHED, B., BOUGUETTAYA, A., AND ELMAGARMID, A. K. 2003. Composing web services on the semantic web. *Int. J. VLDB 12,* 4, 333–351.

MOLDOVAN, D., CLARK, C., AND HARABAGIU, S. 2005. Temporal context representation and reasoning. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 1099–1104.

MOSTÉFAOUI, S. K. AND HIRSBRUNNER, B. June 2003. Towards a context-based service composition framework. In *Proceedings of the International Conference on Web Services*. L.-J. Zhang, Ed. CSREA, 42–45.

MOWAFI, Y. AND ZHANG, D. 2007. A user-centered approach to context-awareness in mobile computing. In *Proceedings of the 4th Annual International Conference on Mobile and Ubiquitous Systems, Networking & Services*. 1–3.

MRISSA, M., GHEDIRA, C., BENSLIMANE, D., MAAMAR, Z., ROSENBERG, F., AND DUSTDAR, S. 2007. A context-based mediation approach to compose semantic web services. *ACM Trans. Internet Technol. 8,* 1, 4.

NARAYANAN, S. AND MCILRAITH, S. A. 2002. Simulation, verification and automated composition of web services. In *Proceedings of the 11th International World Wide Web Conference*. ACM Press, New York, NY, 77–88.

NIELSEN, J. 1993. *Usability Engineering*. Morgan Kaufmann Publishers, Inc., San Francisco, CA.

NOY, N. F. AND MUSEN, M. A. 2001. Anchor-PROMPT: Using non-local context for semantic matching. In *Proceedings of the Workshop on Ontologies and Information Sharing at the 17th International Joint Conference on Artificial Intelligence*.

OUZZANI, M. AND BOUGUETTAYA, A. 2004. Efficient access to web services. *IEEE Int. Comput. 8,* 2, 34–44.

PARR, T. 2007. *The Definitive Antlr Reference: Building Domain-Specific Languages (1st Ed.)*. Pragmatic Bookshelf.

PAWLAK, R., SEINTURIER, L., DUCHIEN, L., AND FLORIN, G. September 2001. JAC: A flexible solution for aspect-oriented programming in java. In *Proceedings of the 3rd International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*. A. Yonezawa and S. Matsuoka, Eds., Lecture Notes in Computer Science Series, vol. 2192. Springer, 1–24.

PAYMANS, T. F., LINDENBERG, J., AND NEERINCX, M. 2004. Usability trade-offs for adaptive user interfaces: ease of use and learnability. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*. ACM Press, New York, NY, 301–303.

REPO, P. 2004. Facilitating user interface adaptation to mobile devices. In *Proceedings of the 3rd Nordic Conference on Human-Computer Interaction*. ACM Press, New York, NY, 433–436.

RIVA, O. AND TOIVONEN, S. 2007. The dynamos approach to support context-aware service provisioning in mobile environments. *J. Syst. Softw. 80,* 12, 1956–1972.

RYAN, N., PASCOE, J., AND MORSE, D. 1997. Enhanced reality fieldwork: The context-aware archaeological assistant. *Comput. Appl. Archaeology*.

RYU, S. H., CASATI, F., SKOGSRUD, H., BENATALLAH, B., AND SAINT-PAUL, R. 2008. Supporting the dynamic evolution of web service protocols in service-oriented architectures. *ACM Trans. Web 2,* 2, 13:1–13:46.

SCHILIT, B., ADAMS, N., AND WANT, R. 1994. Context-aware computing applications. In *Proceedings of the 1st Workshop on Mobile Computing Systems and Applications*. Computer Society, 85–90.

SHENG, Q. Z. AND BENATALLAH, B. 2005. ContextUML: A UML-based modeling language for model-driven development of context-aware web services. In *Proceedings of the International Conference on Mobile Business*. 206–212.

SHENG, Q. Z., POHLENZ, S., YU, J., WONG, H. S., NGU, A. H. H., AND MAAMAR, Z. 2009. ContextServ: A platform for rapid and flexible development of context-aware Web services. In *Proceedings of the 31st International Conference on Software Engineering*. 619–622.

VERMEULEN, J., VANDRIESSCHE, Y., CLERCKX, T., LUYTEN, K., AND CONINX, K. 2008. Engineering interactive systems. In *Augmenting Services with User Interface Models,* Springer-Verlag, Berlin, 447–464.

WANG, H., ZHOU, X., ZHOU, X., LIU, W., LI, W., AND BOUGUETTAYA, A. 2010. Adaptive service composition based on reinforcement learning. In *Proceedings of the International Conference on Service Oriented Computing*. 92–107.

WEIS, T., SATERNUS, M., KNOLL, M., BRÄNDLE, A., AND COMBETTO, M. 2006. Towards a general purpose user interface for service-oriented context-aware applications. In *Proceedings of the International Workshop on Context in Advanced Interfaces, in conjunction with the International Working Conference on Advanced Visual Interfaces*. ACM Press, New York, NY, 53–55.

YAU, S. S., KARIM, F., WANG, Y., WANG, B., AND GUPTA, S. K. S. 2002. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervas. Comput. 1,* 3, 33–40.

YU, Q., LIU, X., BOUGUETTAYA, A., AND MEDJAHED, B. 2008. Deploying and managing web services: Issues, solutions, and directions. *Int. J. VLDB 17,* 3, 537–572.

ZHOU, J., NIEMELA, E., PERALA, J., AND PAKKALA, D. 2007. Web service in context and dependency-aware service composition. In *Proceedings of the 2nd IEEE Asia-Pacific Service Computing Conference*. 349–355.

Received January 2011; revised May 2011, August 2011, October 2011; accepted October 2011