Received 2 August 2005
Revised 27 January 2006

Adaptation in context-aware pervasive information systems: the SECAS project

Tarak Chaari and Frédérique Laforest
LIRIS-CNRSUMR, Toulouse, France, and
Augusto Celentano
Dipartimento di Informatica, Università Ca' Foscari Venezia, Mestre, Italy

Abstract

Purpose – The simple environment for context aware systems (SECAS) Project deals with the adaptation of applications to the context (user preferences and environment, terminal, etc.). The authors aim to develop a platform which makes the services, data and the user interface of applications adaptable to different context situations.

Design/methodology/approach – Previous research has concentrated on how to capture context data and how to carry it to the application. The present work focuses on the impact of context on the application core. A case study in the medical field is also analysed.

Findings – The paper illustrates a new definition of the context which separates the application data from the parameters of the context. This definition helps to establish a complete study on how to adapt applications on their three dimensions (services, content and presentation) to the context.

Originality/value – The paper presents the SECAS platform, one that ensures the deployment of adaptive context-aware applications.

Keywords Computer applications, Information systems

Paper type Research paper

1. Introduction

Nowadays, new challenges have appeared in information systems: users want to receive the needed information at any time and everywhere. These challenges have solicited developers to integrate mobile terminals in their applications creating a new research domain called pervasive (or ubiquitous) computing (Birnbaum, 1997). Such applications have to adapt to a set of parameters such as the type of the terminal, the connection state, the user environment, which can evolve during the application use. All these parameters characterize a contextual situation.

In different context situations, users may access different data and exploit different aspects of an application. For example, in one context a doctor accesses a health database from her office for screening patients for prevention cares, while in a different context, the same doctor accesses the same database at the patient home for post-treatment analysis. While data are the same, the way they are returned may vary according to the doctor's situation. Often, in different contexts, users access almost the same data and the same services but receive answers shaped differently, with different presentation and possibly different content details. For example, a doctor examines a patient record at the hospital using a desktop computer connected to the hospital database, or consults the same record stored on a PDA while visiting the patient at home, or receives an audio description of the patient record during a surgical operation.

Applications supporting such diversity, known as contextaware applications, have to perceive the situation of the user in his/her environment and consequently adapt their behavior [including services, data and user interface (UT)] to that situation, without explicit demand from the user.

International Journal of Pervasive Computing and Communications
Vol. 3 No. 4, 2007
pp. 400-425

1742-7371
DOI 10.1108/17427370710863130

The development of context-aware, adaptable applications require two goals to be assessed: design an architecture supporting context-awareness at run-time, and design the application itself in order to be context-aware. The SECAS project aims to achieve these goals by providing a generic platform to design and deploy context-aware applications. Many existing efforts provide interesting techniques to capture, interpret and model context information but there is no precise and complete solution on how to adapt the application to the captured context. Our work focuses on this point by providing the necessary tools to adapt existing applications to new contextual situations that were not taken in consideration at their design stage. Our goal is to ensure this type of adaptation using minimum engineering effort to not develop new versions of the application to accept new contexts.

The paper is organized as follows: after reviewing the state of the art in context-awareness in section 2, we present our definition of context in Section 3. A context-aware architecture based on Web services is proposed in section 4. Section 5 focuses on the context-aware adaptation strategy of services, data and UI of an application. In section 6, we present a case study in the medical field where we give concrete examples of our adaptation approach, and in section 7, we discuss technical details on the SECAS implementation. Section 8 draws the conclusions and the perspectives of our future work.

2. State of the art in context-awareness

The first attempt to use context to change the behavior of an application is the active badge system, developed at Olivetti Research Ltd in 1992 (Want *et al.*, 1992). Since then, this topic has received increasing attention and the field is today populated by several models and architectural proposals; standards are emerging for describing the context in a unified way, in order to guarantee device interoperability. Above all, the definition of context has been broadened from its first conception. Early works were focused on the relationships between the user location and the information processed (Bennett *et al.*, 1994; perkins and Johnson, 1996; Satyanarayanan, 2001); still now many context-dependent applications are locationaware applications: changes in user location, in a discrete or continuous way, make the application to propose different information and different services. Tourism has been one of the most explored domains, with guiding systems and user assistants able to perceive, through different types of sensors, the user movement.

A generalized notion of context, going beyond location (and time), has been proposed by Dey and Abowd (2000) as "any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object". Dey has described three main steps that an application has to do in order to be context-aware. First, it must capture the context as a set of low level data from different sensors (e.g. GPS coordinates, time, physical parameters of the environment, etc.) Second, an interpreter of the captured data must build high level contextual information, more meaningful to the application; for example, it can map GPS coordinates to a street address, values of captured noise and light parameter to environment classification (indoor, outdoor, street, office, etc.). Finally, it must carry the interpreted information to the application, which uses it together with other data to offer an adapted computation or service. The context toolkit (Dey *et al.*, 2001) is one of the first context-aware architectures considering these three main steps, as shown in Figure 1: sensors capture low-level context signals and present

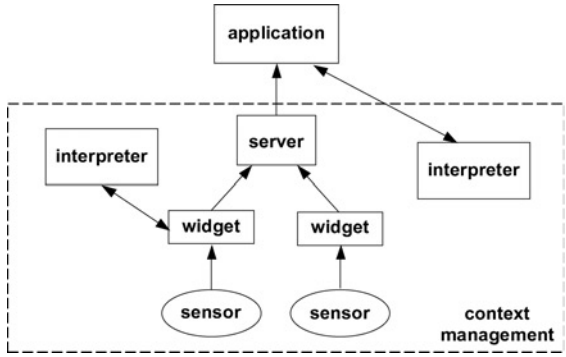

Figure 1.
The context toolkit architecture

them to the context widgets, which use the interpreter to carry high-level context data to the application through a context server.

Context data is also meaningful as a series; for example, user motion can be computed, hence foreseen, by sampling the location to build a history. Different users have different behaviors, and their history can exhibit recurrences that allow a system to anticipate the user expectation. To build a context history, which can involve different facets, a reliable representation of all its aspects must be devised.

There are three basic approaches to context modeling, at different levels of complexity. The simplest approach, used by the context toolkit, stores context as an unstructured set of attribute/value pairs, e.g. {Name = "context1", User = "*x*", Location = "*y*", Time = "*t*"}. A more complex approach, aiming at building a standard representation suitable for processing in a Web based environment, uses RDF. The most consistent example is an extension of the CC/PP profile of W3C (Indulska *et al.*, 2003) called comprehensive structured context profiles (CSCP), proposed by Held *et al.* (2002). The third approach goes a step further, modeling the context using ontologies (Chen *et al.*, 2004). The most consistent proposal is CoOL (Strang and Linhoff-popien, 2003) that presents a context parameter as a set of entities with properties that represent its characteristics. Table I presents a summary of these approaches with the advantages and the drawbacks of each one.

The use of ontologies to model context is the best choice if we want to guarantee a high degree of expres-siveness and semantic richness. Ontologies can also offer means to avoid conflicts that we can find when identi-fying context situations (Chen *et al.*, 2004). The major drawback of this choice is the complexity of implementation of ontolo-gies and the heaviness of reasoning on their facts and their entities. A pervasive environment needs high performance analysis and a short response time due to the limitations of the capabilities of devices that we use in such environments. Therefore, we will use an RDF based representation of context that ensures a sufficient degree of

Model characteristics	Expressiveness and semantic richness	Implementation ease	Conflict resistance
Attribute/value pairs	–	+	–
RDF based	+	+	–
Ontologies	+	–	+

Table I.
Existing context models

expressiveness and semantic richness. We will define priority rules to avoid context situations conflicts.

The adaptation of the application to the context can be driven by four approaches (Dockhor Costa *et al.*, 2005):

- Conceptual frameworks focus on the architectural aspect of context-aware systems and provide means to facilitate capturing, interpreting and carrying context data to the interested parties. The context toolkit (Dey *et al.*, 2001) and the cooltown (Kindberg and Barton, 2001) projects are examples of this approach.
- Service platforms aim at providing the pertinent services to the user depending on context. This includes dynamic service discovery, dynamic deployment of adaptive services addressing issues of scalability, security and privacy. M3 (Henricksen *et al.*, 2005) and platform for adaptive applications (Efstratiou *et al.*, 2002) are examples of contributions to this approach.
- Appliance environments try giving solutions to the heterogeneity problem by providing interoperability techniques and frameworks. Ektara (De Vaul and pentland, 2000) and universal information appliance (Eustice *et al.*, 1999) are projects which use this approach.
- Computing environments for pervasive applications focus on designing the physical and logical infrastructure to hold ubiquitous systems. The PIMA (Banavar and Bernstein, 2002) and Portolano (Esler *et al.*, 1999) projects are examples of this approach.

Table II presents a synthetic view of these approaches by comparing the most relevant issues.

Despite such a rich landscape in context related research, a complete, comprehensive model is still missing. The lack of a reference model is mainly apparent in the relationships between the application and the context. Defining how the application can adapt to the context is still a question that has no definite answer. We reformulate the question to put the user at the center of the adaptation process: what is the impact of context on the perceivable behavior of the application?

Satyanarayanan (2001) has shown the importance of adaptation in pervasive computing. He said that "adaptation is necessary when there is a significant mismatch between the supply and demand of a resource" which is typical in pervasive computing and context awareness domains. In fact, the heterogeneity of the user profiles and their environment makes the adaptation of the application a needed goal.

In the following sections, we present and discuss a complete adaptation strategy of applications to different contexts.

Issue	Conceptual frameworks	Service platforms	Appliance environm.	Computing environm.
Device heterogeneity			×	
Device mobility		×		
Context management	×	×		
Adaptation		×		×
RAD/deployment		×		×
User context	×			

Table II.
Approaches in context awareness

3. A new vision of the context

Researches in the context-awareness domain have not yet led to a generic and pragmatic definition of context. The definitions issued so far are very abstract or very specific to a particular domain, making the formalization of the context very difficult. The definition of Dey and Abowd (2000) is widely accepted as a "good" definition, but does not help in separating the contextual data from the application data. In our opinion, the core of the application should be designed in a contextindependent way, i.e. by abstracting from the different contexts in which it will be used; in such a way, a designer should be able to identify the data which are inherently associated to the application, and to distinguish them from the data which specify the context, which should be considered in a second step. It should also help in turning a legacy application into a context-aware one, leaving the legacy application unmodified.

To reach this goal, the boundary between application data and context data must be defined clearly; it may depend on the application domain, since some data that are at the application level in one domain can be seen as context in another domain. For example, GPS localization is part of application data in a traffic regulation system, but is part of context data in a telemedicine application. A definition of what is context data is therefore complex: it is not retrieved from permanent storage of the application, and is not provided by equipments or other input sources directly related to the application domain. Context data is a variable input of the application which is provided by means other than the user every time he/she uses the services of the application.

In general, we can define the context as the set of the external parameters that can influence the behavior of the application by defining new views on its data and its available services. These parameters may be dynamic and may change during the execution. Moreover, they must be transparent for the user, since they are not significant as application data.

An instance of these parameters characterizes a context situation which does not modify the application data but may lead to process them in a different way. For example, a contextual situation can be characterized by the following parameters {user = "doctor", terminal = "PDA", location = "patient X home"}. The application will be adapted by processing application data according to the doctor profile and role (e.g. in terms of access authorization), to the terminal capabilities (e.g. by presenting data in small chunks instead of in large tables), and to the specific location (e.g. accessing locally stored data instances instead of remote, centralized archives).

To formally define a contextual situation, we consider an n-dimension space, where each dimension represents a context facet. We define five basic facets, but more can be added in specific application domains: network profile, user description/preferences, terminal characteristics, location and environment. A change of a parameter's value on a dimension defines a new contextual situation. It is worth to note that the five axes are rarely used at the same time in an application, whose architecture delegates to different components or to different services the data processing functions, the authorization checks, the network related issues, the presentation, and so on. In Figure 2, we present two examples of contextual situations C_1 and C_2 in the 3D space {location, user, terminal}.

C_1 presents a situation where the user is a general practitioner situated in his office, using a standard PC, e.g. modifying the treatment of a patient. In context C_2, a nurse consults the same medical record at the patient home using a PDA, to apply the prescription of the practitioner. In both contexts users interact with the same service, "patient treatment". However, the application does not behave in the same way in the two situations, since differences along the context dimensions lead to changes in the

application: the user profile (practitioner and nurse) influences the access rights to data; the location (office and patient's home) modifies the list of the available services, since some medical devices are not present at the patient's home); the terminal dimension changes the way data is displayed and the interaction of the user with the application, e.g. decomposing the information returned to the user in small chunks for small screens, displaying all the data on a standard screen, performing text to speech synthesis on a mobile phone, and so on.

To concretely store and exchange context information with the application we use an XML representation based on CSCP (Held *et al.*, 2002). We use the richness and the generality of this model to define an XML element for each dimension (or facet) of the context. All these dimensions are attached to a general context session profile for every user of the SECAS applications. The attributes of each dimension are not limited to (parameter, value) pairs and can be defined by hierarchical structures. Figure 3 shows the general structure of our context model, while Figure 4 shows a concrete example of a context session profile.

4. The SECAS architecture

The global architecture of SECAS is illustrated in Figure 5. It is based on four subsystems: the application core, the adaptation layer, the context management system, and the client-side system. A set of components for each subsystem manages

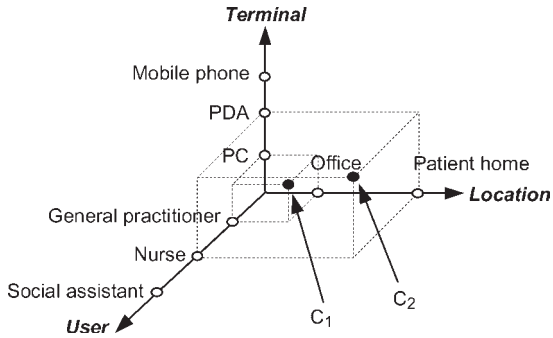

Figure 2.
Multi-dimension representation of context

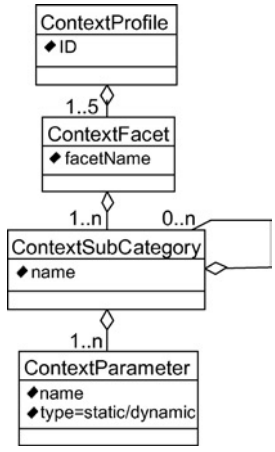

Figure 3.
The SECAS Context model

```
<?xml version="1.0" encoding="UTF-8"?>
  <contextProfile ID="U17T355008002783631">
    <contextFacet name="terminal">
      <contextParameter name="type" type="static">cldc</contextParameter>
      <contextParameter name="model" type="static">NOKIA6600</contextParameter>
      <contextParameter name="serialNumber" type="static">355008002783631</contextParameter>
      <contextSubCategory name="hardwarePlatform">
        <contextParameter name="totalMemory" type="static">2</contextParameter>
        <contextParameter name="availableMemory" type="dynamic">1.3</contextParameter>
        <contextSubCategory name="screenSize">
          <contextParameter name="width" type="static">208</contextParameter>
          <contextParameter name="height" type="static">320</contextParameter>
        </contextSubCategory>
      </contextSubCategory>
      <contextSubCategory name="softwarePlatform">
        <contextSubCategory name="acceptedDataTypes">
          <contextParameter name="Images" type="static">png, bmp</contextParameter>
          <contextParameter name="Text" type="static">plain, String, html</contextParameter>
          <contextParameter name="Nuemric" type="static">int, short, long</contextParameter>
          <contextParameter name="Videos" type="static">3gp</contextParameter>
          <contextParameter name="Applications" type="static">vnd.nokia.ringing-tone
          </contextParameter>
          <contextParameter name="Audio" type="static">wav, midi, mp3, 3gp</contextParameter>
          <contextParameter name="Other" type="static">date</contextParameter>
        </contextSubCategory>
        <contextSubCategory name="api">
          <contextParameter name="virtualMachine" type="static">Monty 1.0 VM</contextParamete
          <contextParameter name="userInterfaceVocabulary" type="static">
              http://liris-7024/midpUIVocabulary.xml</contextParameter>
          <contextParameter name="serviceInvocationDescriptor" type="static">
              http://liris-7024/midpServiceInvocation.xml</contextParameter>
        </contextSubCategory>
      </contextSubCategory>
    </contextFacet>
    <contextFacet name="network">
      <contextParameter name="connectionType" type="static">GPRS</contextParameter>
      <contextParameter name="bandWidth" type="static">33</contextParameter>
      <contextParameter name="delay" type="static">135</contextParameter>
      <contextParameter name="connectionState" type="dynamic">disconnected</contextParameter>
    </contextFacet>
    <contextFacet name="userProfile">
      <contextSubCategory name="basicInformation">
        <contextParameter name="userID" type="static">17</contextParameter>
        <contextParameter name="generalDomain" type="static">medical</contextParameter>
        <contextParameter name="specificDomain" type="static">nephrology</contextParameter>
        <contextParameter name="profession" type="static">specialist</contextParameter>
      </contextSubCategory>
      <contextSubCategory name="preferences">
        <contextParameter name="language" type="dynamic">FR</contextParameter>
        <contextSubCategory name="display">
          <contextParameter name="numericValues" type="static">curves</contextParameter>
          <contextParameter name="images" type="static">fullScreen</contextParameter>
        </contextSubCategory>
      </contextSubCategory>
    </contextFacet>
    <contextFacet name="location">
      <contextSubCategory name="pysical">
        <contextParameter name="country" type="dynamic">France</contextParameter>
        <contextParameter name="city" type="dynamic">Villeurbanne</contextParameter>
        <contextParameter name="zipcode" type="dynamic">69100</contextParameter>
        <contextParameter name="streetname" type="dynamic">Jean Capelle</contextParameter>
        <contextParameter name="streetNumber" type="dynamic">7</contextParameter>
        <contextParameter name="building" type="dynamic">501</contextParameter>
        <contextParameter name="room" type="dynamic">330</contextParameter>
      </contextSubCategory>
      <contextSubCategory name="logical">
        <contextParameter name="IPAdress" type="static">134.214.107.20</contextParameter>
      </contextSubCategory>
    </contextFacet>
  </contextProfile>
```

Figure 4.
An example of a context session profile

the operations needed for sensing and interpreting the context, and adapting consequently the application core and the UI.

Different technologies can be used to build applications, and no one appears to dominate the current scenarios. The adoption of Web services, however, is widespread

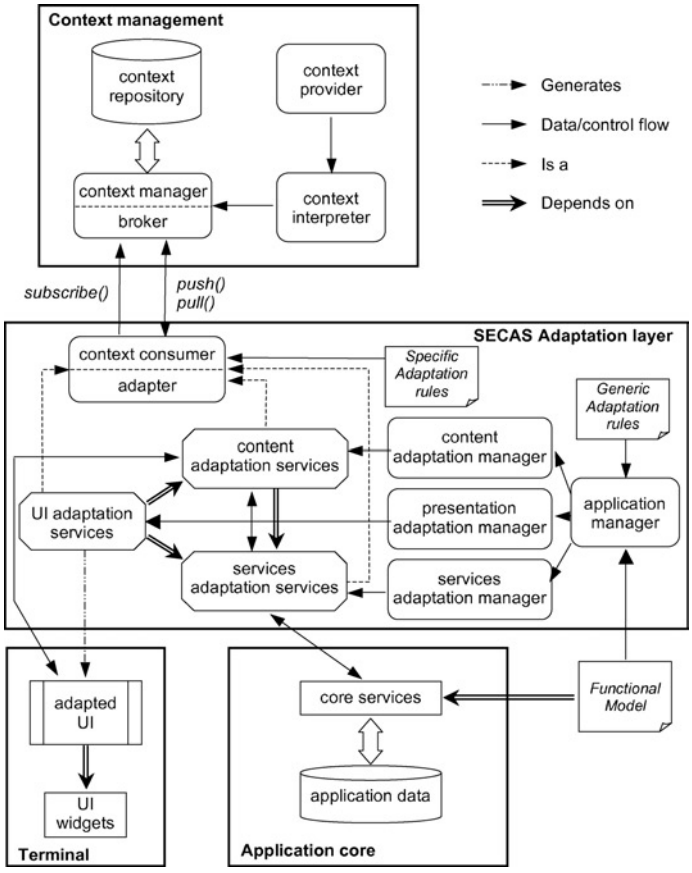

Figure 5.
The SECAS architecture

and is considered today a viable architecture for evolving applications, mainly due to its "loosely coupling" approach for the integration of application functions (Austin et al., 2002). We therefore adopt a Web services paradigm for discussing about context-aware application design.

4.1 Context management

Context management is highly dependent on the environment, i.e. on the physical context properties and on the sensors which capture and transmit raw data; a general, comprehensive model is therefore difficult to define. We encapsulate such dependences in a generic context provider, which is also responsible for managing the aspects of context related to the user, such as profile and history. Since the captured context may not be meaningful to the application in its raw format, a context interpreter module translates the low level context into a high level representation (e.g. it maps geographical coordinates to a street address).

Context parameters are represented in XML documents as described in section 3. A context parameter needed in an application can be useless in another one. Thus, this XML structure can be extended by additional subcategories and parameters according to the application domain. In particular, the facet environment can hold extra context

information that is not described into the other basic four facets. In general, the context parameters of this facet are at high level and can be interrelated. Many techniques can be used to structure this information (e.g. ontologies) and maintain a coherent set of context data to the application.

A part of the context is dynamic and volatile, and is consumed as it is acquired (e.g. location and time). Another part of the context does not change frequently, and its value may survive different execution sessions (e.g. user profile). A context repository holds the non volatile part of the context, while the volatile part is maintained in internal data structures.

In order to get the context, the application must subscribe to a context broker. In our case, due to the clear decoupling between the application core and the adaptation system, this function constitutes an interface between the context management system and the adaptation system. The context broker carries the pertinent data to each service in the application through the adaptation system. While subscribing, the service tells the broker which part of the context is relevant to it. The broker can therefore provide a specialized context view for each service.

Pull context consumers define logical rules while subscribing to the context broker: when the expression of the rule turns to "true", the context is pushed to the pull consumer. Push consumers retrieve the pertinent parameters of the context defined while subscribing. This view can dynamically evolve during execution, requiring some intelligence in the broker that is therefore tightly coupled with the context manager: it detects the changes of the context parameters and makes the necessary operations to refresh the context repository.

4.2 Adaptation of the application to the context

Context adaptation can be applied to the services of the application (services adaptation), to the exchanged data with the user (content adaptation) and to the visualization (UI adaptation). All these adaptations can be static or dynamic. Static adaptation is obtained by providing different pre-built versions of a resource for different context situations. Dynamic adaptation is done at runtime by filtering the service input and output according to the context. An application manager holds a session object for each client, containing the service references and their dependences (defined by a service Petri net that will be described in section 5.1). It is responsible for adapting the application by calling and properly linking the adaptation services. After the execution of the services adaptation process, the content adaptation module ensures the adaptation of the output data to the context situation. Finally, the UI adaptation module generates the suitable presentation to the user, based on the available interface widgets, depending on the context and on the adapted content.

5. Adaptation strategy to the context

5.1 Preparing the application for adaptation

The adaptation of the application must be established independently from the design and even from the implementation of the application. In fact, we want to add context-awareness after designing all the basic, non-adapted services that the application offers to the users.

We model a service with a function $R = f(X)$ getting input X and computing some output value R, where $X = (x_1, x_2, \ldots, x_m)$ and $R = (c_1, c_2, \ldots, c_n)$ are vectors of typed values. Each output component c_i of R (that in the following we shall denote $R[i]$) is a class that has a name and a type ($R[i]$.name and $R[i]$.type, respectively). To each

component c_i we associate a vector $r_i = (r_{i_a}, r_{i_b}, \ldots)$ where $r_{i_a}, r_{i_b}, \ldots$ are the possible values for the component c_i (i.e. instances of $R[i]$). This generic representation offers a standard exchange format between functions, which facilitates the composition and the adaptation of services. The software entities have functional dependences between them, since the input of a service may depend on the output of other services.

We use the notation $(f_1, f_2, \ldots, f_n) \rightarrow f$ to model the dependence "f depends on f_1, $f_2, \ldots, f_n$". This dependence can be illustrated by a Petri net representation where the places are the services and the transitions are the conditions that allow the application to invoke the next service. The main condition is the correct execution of the previous services. In the case depicted in Figure 6, the transition cannot be fired, therefore the service f cannot be offered to the user, before completing the services $f_1, f_2, \ldots, f_n$.

We define the functional model of the application by a non-autonomous Petri net (David and Alla 1992) that describes the dependences between the services offered to the user of the application. Formally, a functional model of an application is a tuple $M = (F, T)$ satisfying the following requirements:

(1) F is a finite set of service functions $\{f_1, F_2, \ldots, f_n\}$.

(2) T is a finite set of transitions $\{t_1, t_2, \ldots, t_m\}$.

(3) each t_i is a tuple (d, gc, A), where d is the maximum delay for passing the transition, gc is the general condition of the transition, and A is a finite set of associations $\{a_1, a_2, \ldots, a_l\}$ between the services. Each association a_i is a finite set of pairs (sourceParameter, destination- Parameter) linking the outputs of the source service to the inputs of the destination service; *sourceParameter* can be a complex expression involving some basic operators (like number arithmetic, string concatenation, boolean evaluation, etc.) on the elementary output parameters of the source service.

Each transition of the Petri net is fired when the execution of all its entry services is completed successfully and the logical conditions on the output parameters are satisfied (e.g. "userID is not null"). An extra general logical condition that involves external events can be defined for each transition. By default, when the exit services of the transition are not context-aware, the value of the general condition is the "true" constant.

The root place of the net is the initial service offered to the user (e.g. user authentication). Each transition is timecontrolled by a predefined expiration delay to avoid execution time deadlocks at the client side. The service can generate an error when it is not suitably invoked (e.g. due to a bad input vector). This type of error is controlled by logical expressions specifying conditions on the output of the services.

At any time in the execution, the user can go back to the previous place/service through an implicit transition which is fired when the user makes an explicit action on

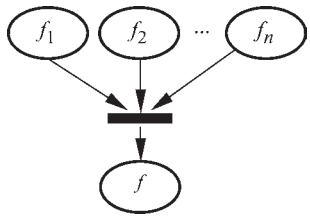

Figure 6.
A Petri net representing service dependences

the client application, such as a click on a back button, or when the maximum delay of the transition is expired.

The deployment of the functional model is ensured by an XML descriptor which is a tagged representation of the application functional model. We have developed an extension of the Petri net markup language (PNML)[6] to describe services and their dependences.

Figure 7 shows an example of a medical application, while Figure 8 presents a part of the corresponding XML description.

The Petri net representation constitutes a formal way of modeling service dependences and interoperability that guarantees their correct composition and adaptation. There are many other techniques for composing services in different contexts; generally, they focus on how to compose the same services in different application domains. Our focus, however, is not on the composition of application services, because they have already been designed to work together for the same application; rather, our main goal is to provide the most relevant and practical technique that facilitates the application adaptation to new context situations without redeveloping its services.

After the deployment of the descriptor, we apply our adaptation strategies to the services, the data and the presentation of the application. These strategies are detailed in the following of this section.

5.2 Services adaptation

The service adaptation consists in transforming the Petri net of the services dependences into another adapted nonautonomous Petri net where the transitions can be controlled by external events coming from context providers. To guarantee this adaptation we add an adaptation layer on top of the application core. The adapted services can have many versions or instantiations, according to different context situations. These versions can be added statically or dynamically to the basic existing services of the application.

The selection of a service among the available versions is made by a tier service that we call adapter (Figure 9).

In general, while the application services do not change, different adaptations may not be equivalent at the outer level, e.g. due to the different types of handled data in different contexts. The adapter can be written as ad(X, cad(c))/f_a, f_b, . . . , where ad is the adapter service that chooses among the f_i according to the current context situation, X is the application data initially provided for the non-adapted service f, and cad(c) is the necessary view of the context c for the adapter service ad to perform the adaptation. The adapter service ad knows the list f_a, f_b, . . . , of the available versions for a given service f.

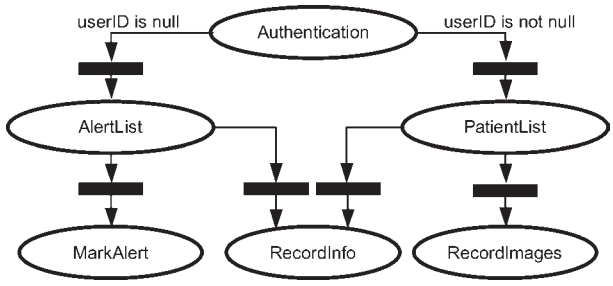

Figure 7.
A functional model of a medical application

```
<?xml version="1.0" encoding="iso-8859-1"?>
<application>
<!-- A place is described by its service id, url and methods
     (with parameters) -->
  <place id="Authentication">
    <url>http://secasserver.insa-lyon.fr/xmlrpc</url>
    <method id="sicom.identification">
      <description>checks login and password of a Sicom
                   user; returns userID and userName
      </description>
      <input name="sicom.login" type="secas:string"/>
      <input name="sicom.password" type="secas:string"/>
      <output name="sicom.userID" type="secas:int"/>
      <output name="sicom.userName" type="secas:string"/>
    </method>
  </place>
    <place id="PatientList">
    <url>http://secasserver.insa-lyon.fr/xmlrpc</url>
    <method id="sicom.listePatient">
      <description>returns the list of the dialyzed
          patients treated by the practionner identified
          by "sicom.userID"
      </description>
      <input name="sicom.userID" type="secas:int"/>
      <ouput name="sicom.recordID" type="secas:int"/>
      <ouput name="sicom.patientName" type="secas:string"/>
      <output name="sicom.patientBirthDate"
              type="secas:date"/>
      <output name="sicom.dialysisMode"
              type ="secas:string"/>
      <output name="sicom.dialysisType"
              type ="secas:string"/>
      <output name="sicom.recordCreationDate"
              type="secas:date"/>
    </method>
  </place>
  <!-- other places ... -->
  <!-- A transition specifies at least one association
       between a source service and a destination service.
       For each association we list source and destination
       parameters, and define a maximum delay in seconds for
       the execution of all the source services of all the
       associations of the transition. A transition can have
       a condition for every entry service. The condition is
       represented by a logical expression on the output
       parameters of the entry service. -->
  <transition delay="100">
    <generalCondition>
        <expression value=always true>
    </generalCondition>
    <association sourceService="Authentication"
                 destinationService="Patientlist">
      <sourceParameter methodID="sicom.identification"
                   inputExpression="sicom.userID"
                   condition="sicom.userID is not null"/>
      <destinationParameter methodID="sicom.listePatient"
                   parameterName="sicom.userID"/>
    </association>
  </transition>
  <!-- other transitions ...-->
</application>
```

Figure 8.
A fragment of the XML description of the medical application of Figure 7

The selection of the service instances is not the only task of the adapter. It intercepts services calls and applies adaptation operators on them. We distinguish two types of adaptation operators: the first one specifies the operations on the service output and the operations applied on the service instances; the second one collects all the

adaptation transformations on the functional model (i.e. the Petri net of service dependences). The first type of adaptation operators (that we call Services Adaption operators, SAoperators) is managed by the adapters, which hold a database of adaptation rules. The rules define the adaptation operations that the adapter has to compute on the service it adapts, and the context parameters related to it.

The second type of operators (that we call Functional-model Adaption operators, FAoperators) is guaranteed by the services adaptation manager (see Figure 5). It holds a database of adaptation rules, to perform the adaptation process on the functional model of the application. It also holds a database of generic rules that all the adapters of the applications have to compute to guarantee the services adaptation. The rules are organized by priority order (i.e. the first rule in the database will be applied before the next ones) to define an adaptation plan for the application.

Each rule is modeled by a pair (Expression, Adaptation operators). Expression is a logical expression defining a pertinent contextual situation for the adapter. In general, these logical expressions are formulated in terms of the context and the output of the considered service. Adaptation operators define the operations that must be computed to adapt the service to the considered contextual situation. These rules are executed until expression turns to false.

For example, the rule

$$(\neg contextProfile.terminal.acceptedDataTypes.acceptImages \wedge \exists f|(\exists_i|R[i].type = '' image'')) \rightarrow lockService(f)$$

defines a contextual situation in which the terminal does not support images but the service provides an image output. The action that must be executed in this condition is lockService(f), described in section 5.2.3, which informs the application manager that it must lock the service f that provides the image because the terminal cannot display it.

We use XML to model and store the rules. Figure 10 shows the XML representation of the rule shown above.

5.2.1 Operators on the service output.

- projectOutput: this operator is applied to the output vector R of the service and projects it on a subset of its components. For example, if $c_1, \ldots, c_n$ are the names of the components of vector R, and $r_1, \ldots, r_n$ are the component values, the projection of R on $(c_2, \ldots, c_n)$ is the vector $(r_2, \ldots, r_n)$. In this example, the elementary output information r_1 of the service is not returned to the user. It is denoted projectOutput(R, $c_2, \ldots, c_n$). It is useful when we need to hide some output values, or to prepare an entry data set for another service that uses only a part of the output of the previous service.

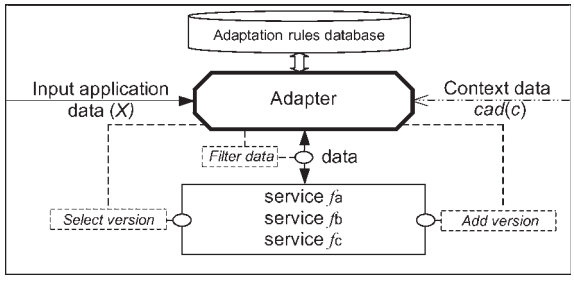

Figure 9.
Adapter of a business service f

```
<rule ID="3">
  <expressions>
    <expression operator="equals">
      <parameter type="context"
        name="context.terminal.softwarePlatform.acceptedData
Types.acceptImages"
          value="false"/>
    </expression>
    <expression operator="exist">
      <parameter type="serviceOutput" name="R[i].type"
                value="secas:image"/>
    </expression>
  </expressions>
  <actions>
    <action type="FAaction">
      <operator name="lockService">
        <parameter name="serviceID">serviceID</parameter>
      </operator>
    </action>
  </actions>
</rule>
```

Figure 10.
The SECAS XML representation of an adaptation rule

- selectOutput: this operator is applied to the output R of a service to remove some instances of the output data which are not relevant in the current context, identified through some intensional specification (e.g. like in a WHERE clause in SQL). For example, selectOutput(R, $c_3 >$ "17") means that we select all the instances where the value of the third component of the service output is >17.
- joinOutput: this operator joins the output of a service with extra components, selected from the output of the same or another service. The resulting output vector is extended to include the joined components, e.g. joinOutput (R, $c'_1, \ldots,$ c'_m) extends the output $R(R, r_1, \ldots, r_n)$ with the components $r'_1, \ldots, r'_m$. For example, a set of services could provide many different series of data which are normally used one at a time. Joining them into a unique vector allows the application to present them in a comparative style.
- addOutput: this operator adds new instances to the output of a service due to a change in the context situation. It is generally used to merge the results of several invocations of the same service with different parameters. For example, if we consider a service that returns the list of the restaurants by area and the user is at the border between two areas, the adaptation of this service to this situation is guaranteed by the union of the two results: *addOutput*(*selectOutput*(R, *area* = "*area1*"), *selectOutput* (R, *area* = "*area2*")).

5.2.2 Operators on the services instances.

- selectVersion: this operator is used to select a specific version of a service among the available versions, according to the context situation.
- addVersion: this operator is used to add a new version of a service to adapt its behavior to a new context situation.
- removeVersion: this operator is used to administrate the management of the associated versions to an adapted service.

5.2.3 Operators on the application's functional model.

- lockService: this operator locks a transition in the functional model of the application by setting the general condition of the transition to the "false"

constant. The services that depend on the locked transition become inaccessible. Using this operator, we obtain a sub-net of the original functional model.

- lockService: this operator unlocks a transition in the functional model of the application by setting the general condition of the transition to the constant "true". This operator can be called when the context changes to a new situation where a locked service should be offered again to the user.
- addService: this operator inserts a new service into the functional model as a leaf of the Petri net, so helping in creating a context-aware application incrementally. All the dependences between the newly added service and the existing services must be specified to guarantee the application consistency.
- insertService: this operator replaces a non-adapted service f_i of the functional model of the application by an adapted service f_j that uses f_i. This operator uses the original service and adapts its behavior to the context (e.g. it splits the output of a service in chunks to avoid memory overload in terminals of limited capabilities). The adapter associated to the initial service f_i must have the same dependences as the service f_i and must provide the same output structure R to the services that initially depend on f_i. This condition is necessary to maintain the application consistency.

To implement the service adaptation we use lightweight web services that exchange a generic data structure in an XML-RPCformat. We have made this technical choice after noticing a considerable slowness in exchanging SOAP messages especially at the client side. To deploy the adapters and link them to the context broker, we use the OSCAR OSGI container (Hall and cervantes, 2004).The main bundle in this container is the application manager which is an XML-RPC server. It provides a generic service to deploy the XML descriptor of an application (as in Figure 8), instantiates and deploys an adapter for each service of theapplication. Each adapter is a new XML-RPC web service in the OSGI container. The application manager provides another service to deploy the adaptation rules for each application in an XML format. The application manager links between the instantiated adapters and the context broker concerning the given rules. Then, it applies the adaptation operations to the functional model of the application (FAoperators). The services adaptation process is completed by applying the adaptation operations to the adapters (SAoperators). Figure 11 presents the general architecture of the services adaptation layer.

5.3 Content adaptation

Content adaptation follows the services adaptation, and consists in modifying some properties of the data delivered to the user. Such an adaptation has been studied at

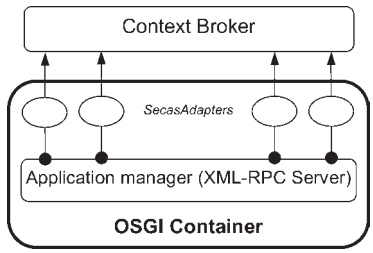

Figure 11.
General architecture of the services adaptation layer

depth; we recall here a non exhaustive list of transformations which can be well supported by web services of general applicability.

Format and type adaptation. This transformation consists in changing the data format to suite device capabilities and user profile, e.g. by reducing the size and the color depth of an image, or by synthesizing speech from a text for a sight impaired user.

Language translation. User preferences may express strict requirements about the language; therefore texts could requireto be translated into other languages. Translation services exist, that can produce good results for selected domains, and can be reused in many applications.

Data compression. Besides raw data compression (e.g. zip), used for reducing the size for permanent storage and transmission of generic data types, and multimedia compression (e.g. jpeg), which is generally processed on the user terminal, semantic compression can be used on textual data to compute a summary of a document.

Data decomposition and aggregation. Data decomposition consists in extracting some part of a medium or of a set of data. The user could ask for a selection or an aggregation of different media objects: e.g. a tourist could ask for a part of a map (decomposition) with a list of the restaurants nearby (an aggregation of two media objects). The decomposition and the aggregation transformations can also be temporal; e.g. the user could be interested only in a few short sequences from a large video file. The service can present them separately or together, after their aggregation. While the decomposition can be highly independent from the data semantics, aggregations like the one described in the first example are very dependent on the data meaning and on their relations, and could require proper cross-links to be defined at application level.

Figure 12 shows the general architecture of the content adaptation module. Content adaptation is implemented by modeling each transformation with a logical expression transformation(URI, SecasInType, SecasOutType) where URI is the general resource identifier pointing to the adaptation web service that computes the adaptation transformation. SecasInType and SecasOutType are customized content types defined by the metadata of the content (type, format, language, compression rate, etc.). These types are stored in the SECAS metadata directory. The content adaptation manager stores all the content adaptation web services in a transformations database (Figure 12). These transformations may be chained in a nested style, as defined in Berhe et al. (2005). This leads us to look for a list of transformations that compute the necessary adaptation operations to have a final content supported by the terminal. An adaptation engine uses the session accepted types database to perform this process. This database is extracted from the AcceptedDataTypes section in the context profile. For each elementary output of each service, the adaptation engine iterates the transformations

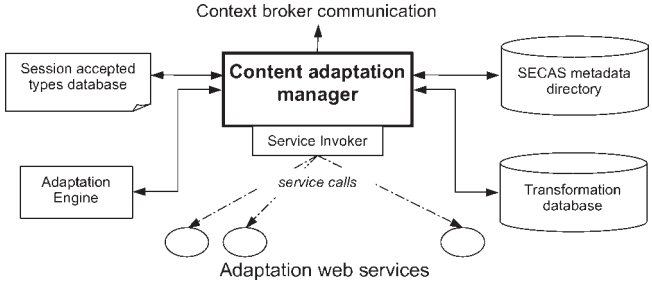

Figure 12.
General architecture of the content adaptation module

searching process until finding a type supported by the terminal. If this process fails, i.e. if no suitable transformation list allows us to compute an adapted output, the content is replacedby a substitution resource, if provided, or by an error message.

5.4 Presentation adaptation

To guarantee the adaptation of the presentation of a software entity, we automatically generate its user interface according to the characteristics of the terminal and to the input/output parameters of its services (Figure 13). For each service, we generate an input window and an output window for the interaction with the software entity. Each window is a collection of elementary user interface components.

The presentation adaptation process follows the identification of the necessary content adaptation transformations. Our user interface generation process is based on the types of the data delivered to the user. In a first step, SECAS maps between the delivered data types and the SECAS abstract user interface components (SecasTextComponent, SecasImageComponent, SecasTextBox, SecasCheckComponent, SecasNumeric-Component, SecasBooleanComponent, SecasVideoComponent, SecasAudioComponent, etc.). This defines a generic logical model of the window. This model is independent from the target device. It only depends on the data types returned to the user and on his/her preferences.

The second step consists in selecting the concrete visual components and their layout on the screen to provide the final physical model of the window. This selection is based on a specific user interface vocabulary which describes the available user interface components on the target device, their behavior (instantiation and methods) and their layout on the target screen. The vocabularies are referenced by the terminal profile stored in the "context repository" (API section in Figure 4).

The last step of the presentation adaptation consists in providing navigation facilities between the software entities. The navigation commands are defined by the transitions between them in the application's functional model. The interaction with application services is ensured by a service invoker.

To implement the presentation adaptation we have developed a generic XML user interface vocabulary that can describe any object oriented API. To describe a new software platform on a terminal, we start by describing the logical structure of the user interface entities (components and containers). After that, we describe their behavior (i.e. how to instantiate these entities and interact with them). Finally, we describe how we layout the existing components on the containers. These vocabularies can be given by the API developers or can be extracted automatically from the existing APIs using reverse engineering techniques. Many existing tools can help the automatic extraction of these vocabularies like javadoc and jad (java decompiler) for Java APIs.

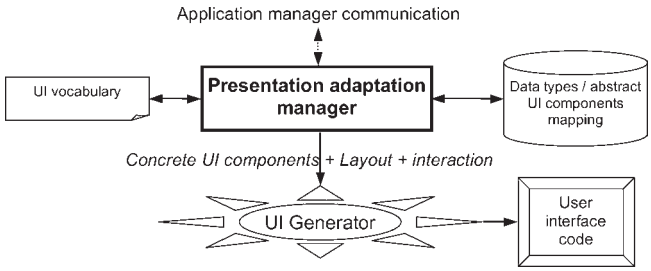

Figure 13.
SECAS presentation adaptation process

We have also designed and implemented a user interface generator that automatically provides adaptive multi-terminal UIs (Chaari and Laforest, 2005). This platform facilitates the interaction with web services by generating Java user interfaces for different kinds of terminals. The generated code is automatically adapted to the user preferences and the terminal used (see Figure 14 and Figure 15).

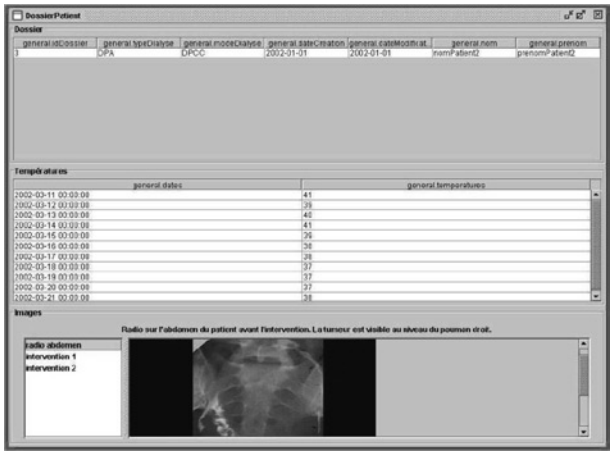

Figure 14.
A dialysis patient record on a standard PC

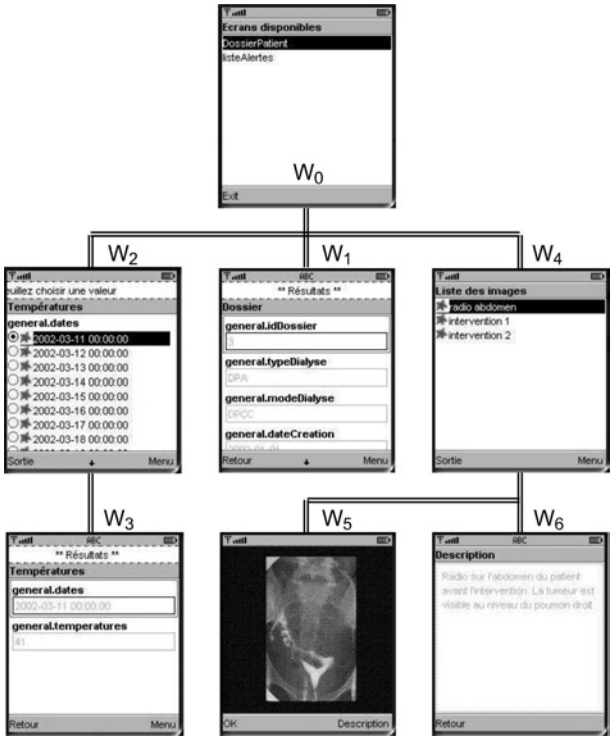

Figure 15.
The same dialysis patient record on a smartphone

There are other approaches to automatic generation of user interfaces that use XML-based languages like UIML (Ali and Pérez-Quiñones, 2004) to describe the target windows and screens. These languages are complete but very complex and heavy to use. In many cases, describing the user interface is harder then coding it. Chaari and Laforest, (2005) has a detailed related work section on these approaches.

6. A case study

In the scope of the SICOM project of the Rhône-Alpes region in France, we have analyzed the software needs of home healthcare of dialyzed patients. In this project, we have collaborated with the Edouard Herriot hospital, Baxter France Enterprise and the France Telecom R&D group. The software offers a set of services to the healthcare practitioners, such as a dialysis record including the prescribed treatment and the description of each dialyze action, a list of medical images for visualizing the catheter state when needed, a graphical representation of the evolution of some parameters (temperature, weight, filtration volume, etc). We have also developed services to social assistants, nurses, and other participants for accessing the prescribed treatments, managing the supplies, etc.

Figure 14 illustrates an application for consulting a dialysis patient record. It is composed of three software entities: the first one offers a service returning general information about the dialysis record of a patient and his/her prescribed treatment; the second one offers a service that returns a trace of the patient's temperature; the third entity displays a list of images from the same record. A menu bar ensures the navigation among the software entities.

Figure 16 presents the initial functional model of this medical application, as used by a practitioner on a standard PC. Figure 17 presents the adapted functional model of the same application when the context situation has changed on the terminal dimension: in this new context situation the practitioner uses a smartphone.

Figure 15 shows the adapted application as it appears to the user. The menu bar is replaced by a first window (W_0) displaying the list of the available windows. The first software entity is automatically adapted to the window (W_1) where data is displayed vertically, while data is displayed horizontally for the standard PC. In this case, the same service is used (i.e. there is no service adaptation, only presentation adaptation). The application manager decomposes the service displayTemperatures() of the second entity into two services by applying the adaptation rule 1 of Figure 18.

In this rule, the service firstValues() is defined as returning the projection of the result R (the outout vector of a service) on its first column. The second service, selectedInstance() returns the selection in R of the lines whose first component c1

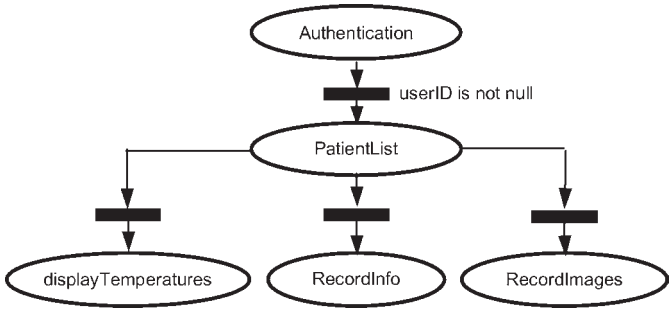

Figure 16.
The initial functional model of the medical application illustrated in Figure 14

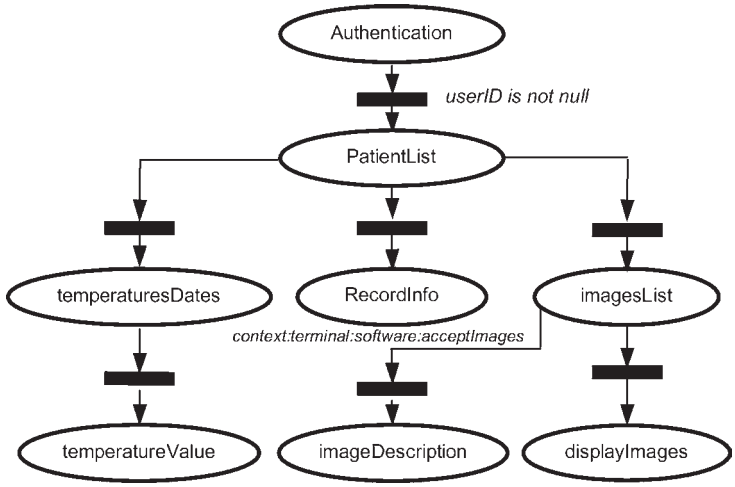

Figure 17.
The adapted functional model of the medical application where the terminal context has changed (PC → smartphone)

$$\{(context.terminal.hardwarePlatform.memory < 3Mb) \wedge (card(R) > 1) \rightarrow firstValues = projectoutput(R[1]) \wedge insertService(f, firstValues) \wedge selectedInstance = selectOutput(R[1] = selected(firstValues)) \wedge insertService(f, selectedInstance)\} \quad (1)$$

$$\{(context.terminal.softwarePlatform.acceptedDataTypes.acceptImages) \wedge (context.terminal.type = \text{“cldc”}) \wedge (card(R) > 1) \wedge (\exists i | R[i].type = \text{“image”}) \rightarrow (displayNoImage = projectOutput(R, \neg R[i])) \wedge (displayImage = projectOutput(R, R[i])) \wedge insertService(f, displayNoImage) \wedge insertService(f, displayImage)\} \quad (2)$$

$$\{(\neg context.terminal.acceptedDataTypes.acceptImages) \wedge (\exists i | R[i].type = \text{“image”}) \rightarrow lockService(f)\} \quad (3)$$

Figure 18.
Three adaptation rules

corresponds to the selection of the user among the results of the service firstValues(). The insertService operator replaces the call to the original service by calls to the adapted service. The adapted service itself calls the original service.

The third entity providing access to a list of images is also firstly adapted by applying rule 1. At a second stage, rule 2 of Figure 18 is applied. As a final result, we obtain three software entities in the new context situation: the first, firstValues(), resulting from applying rule 1, gives the list of the returned images by the initial non adapted service. The second, displayImage(), resulting from applying rule 2, presents the selected image. The third service, displayNoImage(), also resultig from applying rule 2, gives the textual description of the selected image. These three adapted services are inserted in the functional model, and the calls to the original service are redirected to the new services. Finally, the image is adapted to the size and to the color depth of the used terminal, as shown in window (W_5) (content adaptation).

If the terminal does not support images, the selection of an image directly leads the user to its textual description (window W_6) and the displayImage() service is locked by the generic rule 3 of Figure 18.

7. Implementing the SECAS architecture

To comprehensively recall our adaptation strategy, we describe in this section a generic scenario for adapting an existing application to new contexts using SECAS; the scenario is based on ten steps:

(1) A SECAS administrator uses the application manager to deploy the functional model descriptor of the considered application.

(2) SECAS prepares the adaptation layer by instantiating an adapter for each service of the considered application.

(3) SECAS asks the designer to provide the necessary adaptation rules for services adaptation.

(4) The application manager uses adaptation rules to link between the adapters and the context broker to make the context parameters accessible to them.

(5) The application manager computes the service adaptation operations according to the given adaptation rules and the default context profile.

(6) The application manager notifies the content adaptation manager to searching the necessary content transformations for each output of each adapter.

(7) The application manager notifies the presentation adaptation process to launch the generation of the input and output windows for each adapter.

(8) The generated user interface is sent to the user when connecting to the application.

(9) The identified content adaptation transformations are computed when the user interacts with the application new services, i.e. the adapters.

(10) Each time a push context parameter is modified (by the connection of a new user or a new device for example), the scenario is restarted from Step *e*.

Figure 19 presents an activity diagram of a generic scenario of using the SECAS platform. The administrator of the platform provides two simple configuration files only (the functional model and the adaptation rules of the application) and all the adaptation process is done automatically by SECAS.

To refine the adaptation process, the designer of the application can add an adaptation rule to the application manager or to a specific adapter rules database at any moment during the life cycle of the application. The designer has also to provide a priority order for every rule entered in the databases to avoid conflicts between them. Figure 20 shows the sequence diagram corresponding to Steps *a* to *e*.

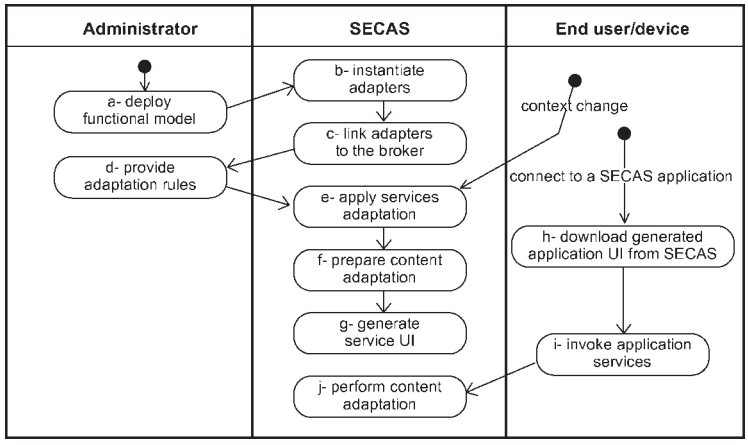

Figure 19.
A generic scenario presenting the different functionalities of the SECAS platform

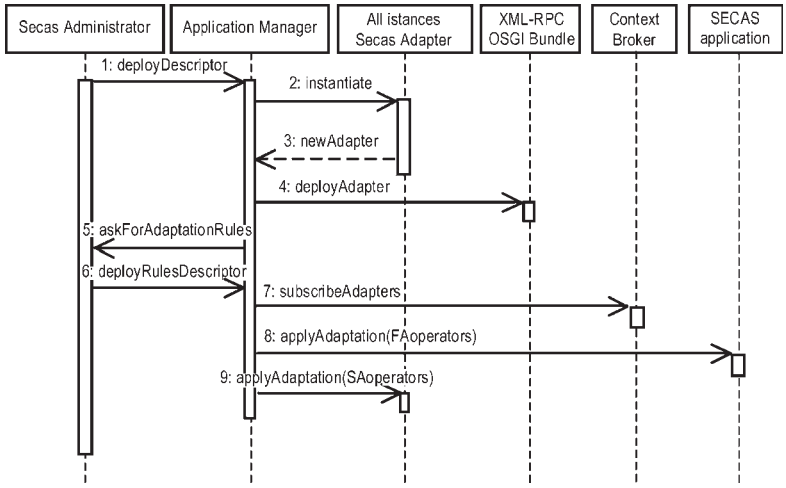

Figure 20.
Sequence diagram of the functional adaptation of an application

The content adaptation process follows the services adaptation. The content adaptation manager verifies that all the outputs of each service are supported by the terminal and fit user preferences. If not, the manager notifies the adaptation engine to look for the necessary content adaptation operations in the transformations database (see Figure 12). Then, it creates a content adaptation path for every non adapted output. The transformations list is executed when the user invokes the corresponding service. A caching repository stores the results of the adaptation for each path to minimize response delays for further use of the same content. Figure 21 presents the sequence diagram of the content adaptation process corresponding to step *f*.

Presentation adaptation (step *g*) follows the content adaptation. The SECAS presentation adaptation manager generates an input window and an output window for the interaction with each adapter. It starts by matching the final output types of the adapters and the SECAS abstract components. Then it identifies the concrete visual components and their layout using the user interface vocabulary corresponding to the default context profile (default terminal and user preferences). While identifying the necessary SECAS abstract components, the presentation adaptation manager adds the navigation tools to reach the other services of the application according to the services dependences defined in the adapted functional model of the application. Figure 22 presents, the sequence diagram corresponding to the presentation adaptation process.

8. Conclusion and future work

In this article, we have presented the SECAS platform that ensures the deployment of adaptive context-aware applications. We have proposed a new definition of the context which separates the application data from the parameters of the context. This definition helped us to establish a complete study on how to adapt applications on their three dimensions (services, content and presentation) to the context. We have analyzed a case study in the medical field, and have developed a static adapted application to validate our adaptation strategy.

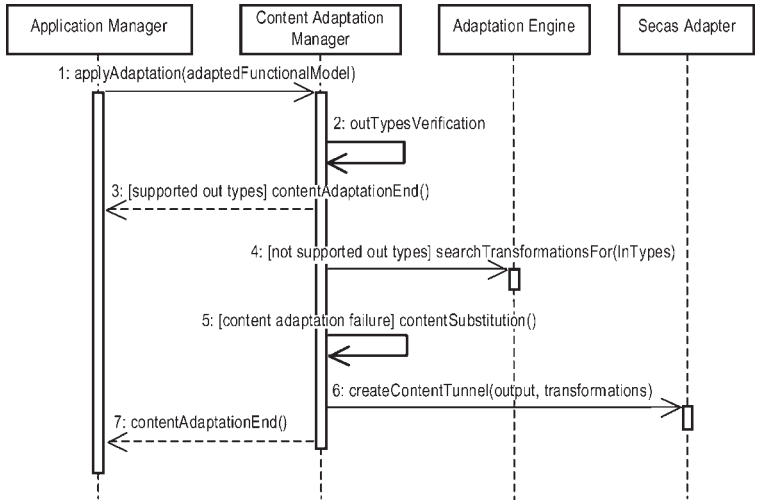

Figure 21.
Sequence diagram of the content adaptation process

A Petri net representation describes the services of the application and their dependences. This representation, coupled with Java XML-RPC lightweight web services, helped us to establish our adaptation strategy and to have shorter response times than other techniques that are based on SOAP messages exchange.

We have made a thorough technical design of the SECAS architecture. We have developed the presentation adaptation module and a simple prototype of the content adaptation module (Chaari and Laforest, 2005). We are now completing the development of the services adaptation module on the OSCAR OSGI container using XML-RPC web services because they offer an easy, flexible and lightweight means for deploying and integrating the adaptation to the context in many levels of the application. Compared to other approaches in context awareness (Table II), the SECAS platform supports device heterogeneity, context management, adaptation and rapid development of contextaware applications.

Finally, to minimize the development efforts to adapt legacy applications, we intend to automate the construction of their functional models. In fact, a high-level monitor (for high-level protocols) could identify the elementary services of the existing application by monitoring the exchange of messages between the server and the client.

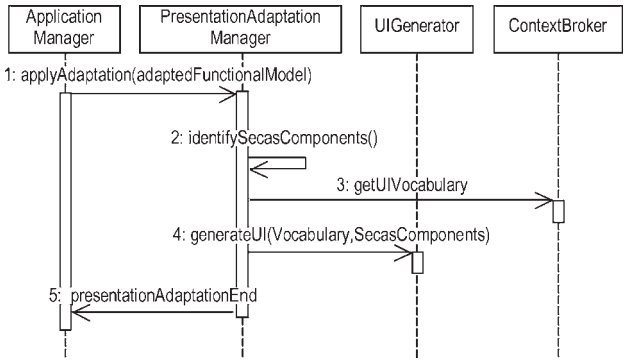

Figure 22.
Sequence diagram of the presentation adaptation process

The monitor could detect three types of services: an input service, if it detects a message from the client to the server not followed by a response; an exchange service, if it detects a message from the client to the server followed by a response from the server; a push service, if it detects a message from the server without an explicit request from the client.

References

Ali, M.F., Pérez-Quiñones, M.A. Shell, E. and Abrams, M. (2004), "Building multi-platform user interfaces with UIML", in Seffah, A. and Javahery, H. (Eds.), *Multiple User Interfaces: Cross-Platform Applications and Contextaware Interfaces*, John Wiley & Sons, New York, NY, pp. 95-116.

Austin, D., Barbir, A., Ferris, C. and Garg, S. (2002), "Web services architecture requirements", W3C working draft, 19 August 2002, available at: www.w3.org/TR/2002/WD-wsa-regs20020819

Banavar, G. and Bernstein, A. (2002), "Software infrastructure and design challenges for ubiquitous computing applications", *Communication of the ACM*, Vol. 45 No. 12, pp. 92-6.

Bennett, F., Richardson, T. and Harter, A. (1994), "Teleporting – making applications mobile", *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz*, CA, pp. 82-4.

Berhe, G., Brunie, L. and Pierson, J.M. (2005), "Distributed content adaptation for pervasive systems", *ITCC 2005, International Symposium on Information Technology: Coding and Computing, Las Vegas, Nevada, USA, 4-6 April*, pp. 234-241.

Billington, J. *et al.* (2003) "The petri net markup language: concepts, technology, and tools", in van der Aalst, W. and Best, E. (Eds), *24th International Conference on Application and Theory of Petri Nets*, LNCS 2679, Springer, pp. 483-505.

Birnbaum, J. (1997), "Pervasive information systems", *Communication of the ACM*, Vol. 40 No. 2.

Chaari, T. and Laforest, F. (2005), "SEFAGI: simple environment for adaptable graphical interfaces", *Proceedings of the 7th International Conference on Enterprise Information Systems, ICEIS, Miami, FL*, pp. 232-7.

Chen, H., Perich, F., Finin, T.W. and Joshi, A. (2004), "SOUPA: standard ontology for ubiquitous and pervasive applications", *MobiQuitous 2004, 1st Annual International Conference on Mobile and Ubiquitous Systems, Networking and Services*, Cambridge, MA, pp. 258-267.

David, R. and Alla, H. (1992), *Petri Nets and Grafcet: Tools for Modeling Discrete Event Systems*, Prentice-Hall, London.

DeVaul, R.W. and Pentland, A.S. (2000), *The Ektara Architecture: The Right Framework for Context-Aware Wearable and Ubiquitous Computing Applications*, MIT Technical Report.

Dey, A.K. and Abowd, G.D. (2000), "Towards a better understanding of context and context-awareness", *CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*, The Hague.

Dey, A.K., Salber, D. and Abowd, G.D. (2001), "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications", *Human-Computer Interaction Journal*, Vol. 16 No. 2-4, pp. 97-166.

Dockhorn Costa, P., Ferreira Pires, L. and van Sinderen, M. (2005), "Architectural patterns for context-aware services platforms", in Mostéfaoui, S.K. and Maamar, Z. (Eds), *Proceedings of the 2nd International Workshop on Ubiquitous Computing, IWUC 2005, Miami, FL*, pp. 3-18.

Efstratiou, C., Friday, A., Davies, N. and Cheverst, K. (2002), "A platform supporting coordinated adaptation in mobile systems", *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002)*, *Callicoon*, *NY*, pp. 128-37.

Esler, M., Hightower, J., Anderson, T. and Borriello, G. (1999), "Next century challenges: data-centric networking for invisible computing", *Proceedings of the 5th Annual International Conference on Mobile Computing Networking (Mobi- Com'99)*.

Eustice, K., Lehman, T.J., Morales, A., Munson, M.C., Edlund, S. and Guillen, M.A. (1999), "Universal information appliance", *IBM Systems Journal*, Vol. 38 No. 4, pp. 575-60.

Hall, R.S. and Cervantes, H. (2004), "An OSGi implementation and experience report", *Proceedings of the 2004 IEEE Consumer Communication and Networking Conference*, pp. 394-9.

Held, A., Buchholz, S. and Schill, A. (2002), "Modeling of context information for pervasive computing applications", *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI2002)*, Orlando, FL.

Henricksen, K., Indulska, J., McFadden, T. and Balasubramaniam, S. (2005), "Middleware for distributed context-aware systems", in Meersman, R. and Tari, Z. (Eds), *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, LNCS Series 3760, Springer, pp. 846-63.

Indulska, J., Robinson, R., Rakotonirainy, A. and Henricksen, K. (2003), "Experiences in using CC/PP in context-aware systems", *Proceedings of the 4th International Conference on Mobile Data Management*, *Melbourne*, *Australia*, pp. 247-61.

Kindberg, T. and Barton, J. (2001), "A web-based nomadic computing system", *Computer Networks*, Vol. 35 No. 4, pp. 443-56.

Perkins, C.E. and Johnson, D.B. (1996), "Mobility support in IPv6", in *Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking*, White Plains, NY, pp. 27-37.

Satyanarayanan, M. (2001), "Pervasive computing: vision and challenges", *IEEE Personal Communications*, Vol. 8 No. 4, pp. 10-17.

Schilit, B.N. and Theimer, M.M. (1994), "Disseminating active map information to mobile hosts", *IEEE Network*, Vol. 8 No. 5, pp. 22-32.

Strang, T. and Linnhoff-Popien, C. (2003), "Service interoperability on context level in ubiquitous computing environments", *International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet (SSGRR2003w)*, *L'Aquila*, *Italy.*

Want, R., Hopper, A., Falcao, V. and Gibbons, J. (1992), *The Active Badges Location System*, Technical Report, Olivetti Research Ltd.

About the authors

Tarak Chaari is a PhD student in the CNRS UMR 5205 LIRIS Laboratory (Lyon Research Center for Images and Intelligent Information Systems) in Lyon, France. His research interests focus on adaptation in the scope of pervasive computing and contextaware systems. He holds a master degree in information systems from INSA Lyon on automatic user interface generation. He is a Member of dynamic adaptation to runtime environments action and member of the research working group on impact of grid computing, peer-to-peer-computing and mobile computing on database systems. He teaches information systems modeling and development at INSA Lyon. Tarak chaari is the corresponding author and can be contacted at: chaari@gmail.com

Frédérique Laforest is associate professor in the CNRS UMR 5205 LIRIS Laboratory (Lyon Research Center for Images and Intelligent Information Systems) in Lyon, France. She teaches pervasive information systems and applications modeling at the INSA Lyon high school. Her research interests concern document-based user interfaces to databases, multi-terminal user interfaces generation, and adaptation of applications to the context.

Augusto Celentano is full professor of Information Technology at Ca' Foscari University in Venice, Italy. He received a Master Degree in Electronic Engineering from Politecnico di Milano. Augusto Celentano has been member of scientific committees in research and educational centers of Politecnico di Milano, and has been consultant and scientific coordinator of research projects of the European Union. His current research interests are in mobile, pervasive and context-aware information systems, in multimedia systems and in human-computer interaction.