Service-Oriented Context-Aware Application Design

Tarak Chaari¹, Frédérique Laforest¹ and Augusto Celentano²

¹ LIRIS, INSA Lyon, 20, Avenue Albert Einstein, 69621 Villeurbanne Cedex, France {tarak.chaari,frederique.laforest}@insa-lyon.fr ² Dipartimento di Informatica, Università Ca' Foscari di Venezia Via Torino 155, 30172 Mestre (VE), Italia auce@dsi.unive.it

Abstract. Context-aware systems are applications that adapt to several situations involving user, network, data, hardware and the application itself. Researchers in context-awareness have concentrated on how to capture context data and to carry it to the application. In this paper, we study the impact of context on the core of the application, give a new context definition useful for application design, and propose a context-aware architecture providing a functional adaptation to the context.

1 Introduction

Data centered applications exchange information with users at different levels of detail, content and presentation according to several parameters that depend on the user and his/her environment. They can also provide, within the same application, different services to different users, or to the same user in different situations. Hence, the notion of context has been developed, as a means to adapt the behavior and the interface of an application to the user situation and equipment, encompassing a large range of adaptation parameters.

In different contexts, users may access different data and exploit different aspects of an application. For example, in one context a doctor accesses a health database for screening patients for prevention cares, while in a different context the same doctor accesses the same database for post-treatment analysis of cases. While data are the same, the way they are returned may vary according to the doctor's goal. 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 detail. 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.

A context-aware application must manage the context as one of its inputs, processing any user request according to the different context instances. However, it is simplistic to consider the context as a part of the application input data, due to the diffi-

^{*} Work partly done while Augusto Celentano was on sabbatical leave at INSA Lyon, France.

culty of classifying in advance all the combinations of user situation, equipment and other context-dependent parameters. Context, therefore, must be managed separately and its influence on the application behavior must be described orthogonally with respect to the application data.

The paper is organized as follows: after reviewing the state of art in context-awareness in Section 2, we present our new definition of context in section 3. A context-aware architecture based on Web services is proposed in Section 4. Section 5 discusses the structure and the composition of adaptation services, with a focus on adaptation of the application's behavior. Section 6 draws the conclusions and the future work.

2 State of Art

Research on context-awareness has started with addressing the problem of mobility by hiding it to the user. The first real context-aware computing effort was initiated by researches at Olivetti Research Ltd and Xerox PARC Laboratory [1]. Since then, many other researches have studied this topic and contributed to this domain. Early works approached the problem by studying location-awareness (like Active Map [2] and Teleporting [3]), and still now many context-aware applications are limited in the scope of context management, using small pieces of contextual information and presenting ad hoc solutions for very specific needs.

In most contributions in this area, we distinguish three main steps that an application has to do in order to be context-aware. First, we have to capture low level contextual information from different sensors (for example, GPS coordinates). Second, we have to make some interpretation on what we capture to build high level contextual information which is more relevant to the application. For example, we can transform GPS coordinates to a complete address and compute physical, temporal and semantic relationships from the initial low level context values. Finally, we have to carry this interpreted information to the application. The Context Toolkit [4] is one of the first context-aware architectures considering these three main steps.

In the context-awareness domain, Dockhorn Costa [5] distinguishes four research approaches:

- 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 and the Cooltown [6] 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 [7] and Platform for Adaptive Applications [8] are examples of contributions to this approach.
- Appliance environments try giving solutions to the heterogeneity problem by providing interoperability techniques and frameworks. Ektara [9] and Universal Information Appliance [10] 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 [11] and Portolano [12] projects are examples of this approach.

Table 1. Approaches in context-awareness

Issue	Conceptual Frameworks	Service Platforms	Appliance Environments	Computing Environments
Device heterogeneity			X	
Device mobility			X	
Context management	X	X		
Adaptation		X		X
RAD/deployment		X		X
User context	X			

Table 1 presents a synthetic view of these approaches by comparing the most relevant issues.

In conclusion, we can say that in the existing context-aware applications there is a great interest to how to gather the context and how to carry it to the system, but there is no consistent answer to the question: How can the application adapt to the context? To be more precise, we reformulate this question: What is the impact of context on the perceivable behavior of the application?

3 A New Vision of Context

Context-aware applications usually mix context management code within the application code. The application code becomes more complex and more difficult to read and maintain. Decoupling context-independent activities of the application from contextual concerns would locally reduce the code complexity. In this article, we present solutions to make this effective. The first step, discussed in this section, concerns the separation of contextual data from application data, while the second step concerns the application architecture, and will be discussed in the next section.

Practical definitions of context—and more precisely of context data—have not yet drawn to a consensus. Definitions in the literature are often domain-oriented and thus too limited. Dey defines the context 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" [13]. It is a complete and general definition, but it does not help in separating the contextual data from the application data. We believe that the core of the application should be designed by making abstraction of the different contexts in which it will be used, which should be considered in a second step. Such a way of working should allow a designer to identify the data which are inherently associated to the application, and to distinguish them from the data which specify the context. It should also help in turning a legacy application into a context-aware one, leaving the legacy application unmodified.

To do so, the boundary between contextual data and application data has to be clearly defined; it depends 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 context data in a telemedicine application, but it is part of application data in a traffic regulation system. Context data 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. In general, context data is a variable input of the application which is provided by means other than the user him/herself every time he uses the services of the application.

The context describes the situation of the user in terms of his/her location, time, environment, used devices, profile, etc. In general, we can define the context as the set of the external parameters that can influence the behaviour of the application by defining new views on its data and its available services. 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 driver wants to take a break for lunch. He is looking for the list of restaurants nearby. Knowing that this person has some cardiac problems, the application has to filter the results by giving only the restaurants that propose the adapted food. The list of the restaurants, stored in the application's database, is a part of application data. However, the information "the person has cardiac problems" is a context parameter interpreted from the user profile.

In the following sections of this article, we use this definition and the separation between context data and application data to study the impact of context on the application core.

4 A Context-Aware Architecture Based on Web Services

The development of context-aware, adaptable applications requires two goals to be assessed: (1) design an architecture supporting context-awareness and adaptation at run-time, and (2) design the application itself in order to be context-aware. Different technologies can be used to build distributed applications (e.g., Web services, CORBA, RMI), and no one seems to dominate the current scenarios. The adoption of Web services, however, is widespread and is considered today a viable architecture for evolving applications, mainly due to its "loosely coupling" approach for the integration of application functions [14]. The low performance noted in early experiences has been overcome by improvements in more recent SOAP implementations. The independence from the language and from the platform improves interoperability across organizations, and the support of a more standard middleware makes Web services easier to use for composition and integration with respect to other approaches. A thorough comparison of the software architecture is, however, out of the scope and size of this paper, and a growing literature on such issues exists (see for example [15, 16]). We therefore ground our discussion about context-aware application design on a Web services paradigm.

Fig. 1 illustrates an adaptable service oriented architecture, in which the components devoted to context management and adaptation are separated from the application core. Each component corresponds to a phase of context management, as discussed in the following.

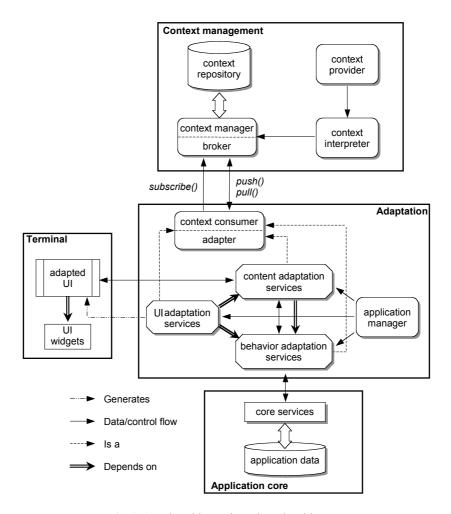


Fig. 1. An adaptable service oriented architecture

Context capturing and interpreting. Context capturing concerns mostly the sensors that acquire physical context properties and the raw data they generate, but also the acquisition of other contextual information, such as user preferences and history. This part is highly environment-dependent, and a generic model is difficult to build. In our architecture a *context provider* represents the context capturing system. Since the context representations that we initially capture may not be meaningful to the application, a *context interpreter* module translates the low level context into a high level representation, easier to use (e.g., an address instead of GPS coordinates).

Context storage. To model context parameters, we use XML documents (CC/PP extensions [17]) to store and exchange context values, maintained and processed by a *context manager*. 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 is less ephemeral, i.e.,

it does not change frequently and its value survives different execution sessions. A *context repository* holds the non volatile part of the context, while the volatile part is maintained in internal data structures.

Context dissemination. A context aware application has to consume a part of the context. In a service oriented architecture it must subscribe to the *context broker* that carries the pertinent data to each service in the application. While subscribing, the service tells the broker which part of the context is relevant to it. Then, the broker can provide a context view for each service. Pull context consumers define logic 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 must be tightly coupled with the context manager: it detects the changes of the context parameters and makes the necessary operations to refresh the context repository.

Adaptation to the context. Context adaptation can concern three levels: data flow (content adaptation), visualization (user interface adaptation), and application's behavior (service adaptation). All these adaptations can be static (i.e., a pre-built version of a resource is provided for each different context situation) or dynamic (adaptation is done at runtime, according to the current context). We must use both to ensure the best adaptation to the context. An *application manager* holds a *session object* for each client, containing the service references and their dependences (defined by a service graph, 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 adaptation by the behavior adaptation services, the content adaptation services ensure the adaptation of the output data to the context situation. Finally, the UI adaptation services generate the suitable presentation to the user, based on the available UI widgets, depending on the context and the content adaptation process.

Content and user interface adaptations are well studied in the literature, even if there are still aspects that need to be explored more deeply. In the next section we shall discuss mainly the application's behavior adaptation. The application behavior is defined by the set of the functions of the services it offers. These functions have to adapt to the context situation by processing information depending on the context parameters that define the actual context situation.

5 Adaptation of the Application to the Context

To ensure the adaptation of the application, we have defined the notion of a *software entity*, which represents the user's view of the application at a certain moment during its runtime. It is composed by the service offered to the user, by the user interface that guarantees his/her interaction with the service, and by the data processed and displayed to him/her. Therefore, the application can be seen as a set of software entities. The adaptation must be applied to the three components of each entity (service, data and user interface). Web services can be used to ensure the adaptation of these three components. Since we assume that the business core of the application is imple-

mented by Web services (or encapsulated by a Web service layer), such a way of working guarantees a variable degree of granularity at one hand, and a flexible and reusable application assembly and deployment at another hand. It also helps carrying out the context adaptation at many levels of the business core services.

5.1 Application's behavior adaptation

From the perspective of the functional architecture, a Web service based application is composed of a number of business services that perform the desired functionalities of the application. Business services do not distinguish between application data and context data. According to our view of context, discussed in Section 3, they are not context-aware, and their behavior remains the same in different context situations, unless context data are in some way used to control their execution, or conveyed to them together with application data.

From a general viewpoint, services interpret user input and access the data storage to produce changes in the application state. Each service is described by a function R = f(X) getting input X and computing some output values R, where X and R are vectors of typed values.

The functional model of the application describes its offered services to the user and the dependences between them. A service f_2 may depend on some outputs of another service f_1 . We use the notation $f_1 \rightarrow f_2$ to express such dependence. The functional model of the application can be represented by an oriented graph where the nodes are the services and the arcs are the dependences between them. In the example of Fig. 2, services f_2 and f_3 cannot be offered to the user before the invocation of service f_1 , and service f_4 cannot be accessed before the invocation of f_2 and f_3 .

To adapt the functional core to the context, we perform a transformation on this graph by replacing each node of the graph by an *adaptation entity*, as shown in Fig. 3. In order to add context awareness to an application, the designer must provide different versions of each service, tailored to different context situations. The versions are different instances of a same "virtual" service, each instance being specialized on specific context values. Some services may not exist in all the versions needed to cover all the contexts.

The selection of a service among the available versions, the handling of the proper context parameters for the service and the filtering of the outputted data to the user are made by a tier service that we call *adapter*. For example, using the strategy design

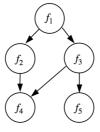


Fig. 2. The oriented graph representing the functional model of an application

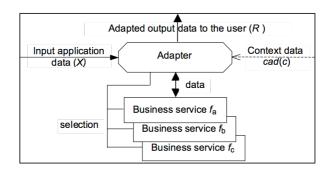


Fig. 3. Adaptation entity of a business service f

pattern [18], a strategist adapter chooses, among the available versions of the service, the correct version that guarantees the adaptation to the user device, and conveys the data to be processed to the selected version.

In general, different adaptations are not completely equivalent at the outer level, e.g., due to the different types of handled data. 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. ad knows the list $f_a, f_b, ...$, of the available versions for a given service f. cad(c) is the necessary view of the context c for the adapter service ad to perform the adaptation.

We denote $F = \{f_1, f_2...f_n\}$ the set of services offered as they appear to the user at a certain runtime moment. These services are functionally independent, and they only depend on the parent services in the graph of the functional model. F evolves when the user interacts with the services. In the example of Fig. 2, at a first step $F = \{f_1\}$. After the invocation of f_1 , F becomes the set $\{f_2, f_3\}$ and so on. The set of apparent services F may change in a specific context situation. For example, we can remove the access to a service in a context situation. This is one of the possible adaptation actions to context presented in the next section.

5.2 Behavioral adaptation actions

The situation is really more general, since services can use other services, in a nested style. Therefore, the execution of a service may be the result of a composition of services in the style f(g[h(..., ch(c)), cg(c)], cf(c)). Each level of composition requires its context values, using its context selection from the broker.

Composition is not the only adaptation action on services. In the remainder of this section we present a non exhaustive list of adaptation actions. We distinguish two types of adaptation actions. The first one includes the actions which are applied on the set of the application services (projection, composition and extension). The second one includes actions which are applied on the set of the versions of a service (selection and union). The first adaptation actions are executed by an *application manager* that manages the invocation of the services and their presentation to the user. The second set of adaptation actions is executed by the *adapters* (Fig 3).

Projection (II). This action can be applied to the list of the business services of the application. It consists in limiting the access to a subset of the services, e.g., for security reasons or access rights constraints. For example, $F' = \Pi(\{f_2, f_4\}, F)$ modifies the set of services F by manifesting to the user only the services f_2 and f_4 .

Composition (o). The composition can be applied to two services f_1 and f_2 and can be serial (o_s) , if the services are applied sequentially, or parallel (o_p) , if the services are applied independently and the results are joined. Note that in general the serial composition is not commutative: f_1 o_s f_2 is different from f_2 o_s f_1 . For example, a service f_1 can compute a text translation and in parallel, another service f_2 makes an image compression for the connected terminal. The composition f_1 o_p f_2 guarantees a complete adaptation of the output to the user.

Extension (extend). This action can be used to augment the set of the application services F by an additional service h which extends the application behavior. $F' = \text{extend}(F, h) = \{f_1, f_2, ..., f_n, h\}$. For example, we can add a new service that filters the output of a service to adapt it to a new device type or to a new user profile.

Selection or restriction (σ). This operator can be used to remove some instances $\{f_{ia}, f_{ib}, f_{ib}\}$ of a service f_i . For example, $\sigma(\{f_{ia}, f_{ib}\}, f_i) = \{f_{ia}, f_{ib}\}$. In this case, f_{ia} and f_{ib} are the only possible instances of f_i . For example, a service gives some information on the patient record. A first version outputs this information by displaying an image, a second gives a textual representation of the image and a third synthesizes a voice speech of this text. The selection action is computed to propose to the user the services that output the format supported by his/her device.

Union (U). This operator can be used to add another version of a service: $\bigcup (f_i, f_{i\alpha}) = \{f_{i\alpha}, f_{ia}, f_{ib}, \ldots\}$. $\{f_{ia}, f_{ib}, \ldots\}$ are the initial possible instances or versions of f_i . The added version guarantees the adaptation to a new context situation that wasn't supported in the initial set of application services.

6 Conclusion

In this paper, we have proposed a new definition of the context that separates application data from context parameters. This new definition helped us making a more precise study on how to guarantee context-awareness in the application core. We have given the guidelines for adapting the behavior of the application to context, and have proposed a solution based on Web services to guarantee these adaptations. A set of adaptation actions on the application's services ensures context-awareness as a layer onto the application core.

The proposal is complete from the conceptual and methodological points of view, but needs to be evaluated in a set of experiments. Therefore, we shall define some concrete rules to map the context situations to an adaptation route using a combination of our adaptation actions. Then, we shall develop an integrated prototype following the schema of Figure 1 and including the behavioral adaptation detailed in Section 5. Among the domains where such adaptations can be successfully tested, we plan to approach the home care domain, where the variance in offer of services and the needs of its different users make adaptation a valuable goal.

References

- 1. R. Want, A. Hopper, V. Falcão, and J. Gibbons. The Active Badge location system. *ACM Transactions on Information Systems*, 10(1), pp. 91-102, 1992.
- 2. B. N. Schilit and M. M. Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network* 8(5), pp. 22-32, 1994.
- 3. F. Bennett, T. Richardson and A. Harter. Teleporting Making Applications Mobile. *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, pp. 82-84, Santa Cruz, California, 1994.
- 4. A. K. Dey, D. Salber and G. D. Abowd. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction Journal* 16(2-4), pp. 97-166, 2001.
- P. Dockhorn Costa. Towards a Services Platform for Context-Aware Applications. Master Thesis. University of Twente, The Netherlands, 2003.
- 6. T. Kindberg and J. Barton. A Web-Based Nomadic Computing System. *Computer Networks*, Elsevier, 35(4), pp. 443-456, 2001.
- J. Indulska, S. W. Loke, A. Rakotonirainy, V. Witana, A. Zaslavski. An Open Architecture for Pervasive Systems. *Proc. 3rd Intl. Work. Conf. on Distributed Applications and Inter*operable Systems (DAIS 2001), Kraków, Poland, pp. 175-188, 2001.
- 8. C. Efstratiou, K. Cheverst, N. Davies and A. Friday. An Architecture for the Effective Support of Adaptive Context-Aware Applications. *Proc. 2nd Int. Conf. in Mobile Data Management (MDM'01)*, Hong Kong, pp. 15-26, 2001.
- 9. R. W. DeVaul, A. S. Pentland. *The Ektara Architecture: The Right Framework for Context-Aware Wearable and Ubiquitous Computing Applications*. MIT Technical Report, 2000.
- K. Eustice, T. J. Lehman. A. Morales. M. C. Munson, S. Edlund and M. Guillen. A Universal Information Appliance. *IBM Systems Journal*, 38(4), pp. 575-601, 1999.
- G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman and D. Zukowski. Challenges: An Application Model for Pervasive Computing. *Proc. 6th Annual Intl. Conf. on Mobile Computing and Networking (MobiCom 2000)*, pp. 266-274, Massachusetts, USA, 2000.
- M. Esler, J. Hightower, T. Anderson, and G. Borriello. Next Century Challenges: Data-Centric Networking for Invisible Computing. Proc. 5th Annual Intl. Conference on Mobile Computing Networking (MobiCom'99), 1999.
- 13. A. K. Dey, G. D. Abowd. 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, The Netherlands, 2000.
- D. Austin, A. Barbir, C. Ferris, S. Garg. Web Services Architecture Requirements, W3C Working Draft, 19 August 2002, http://www.w3.org/TR/2002/WD-wsa-regs20020819.
- D. Karastoyanova and A. Buchmann, Components, Middleware and Web Services, Proc. IADIS International Conference WWW/Internet 2003 (ICW12003), Algarve, Portugal, 2003.
- N. A.B. Gray, Comparison of Web Services, Java-RMI, and CORBA service implementations, Fifth Australasian Workshop on Software and System Architectures. In conjunction with ASWEC 2004, Melbourne, Australia, 2004.
- 17. J. Indulska, R. Robinson, A. Rakotonirainy, and K. Henricksen. Experiences in Using CC/PP in Context-Aware Systems. *Proc. 4th Intl. Conf. on Mobile Data Management*, Melbourne, Australia, pp. 247-261, 2003.
- 18. E. Gamma, R. Helm, R. Johnson and J. Vlissied. Design Patterns, Addison Wesley, 1995.