# Towards a Context Binding Transparency[1]

Tom Broens, Dick Quartel, Marten van Sinderen

Center for Telematics and Information Technology, ASNA group, University of Twente,
P.O. Box 217, 7500 AE Enschede, the Netherlands,
{t.h.f.broens, d.a.c.quartel, m.j.vansinderen}@utwente.nl,
http://asna.ewi.utwente.nl

**Abstract.** Context-aware applications use context information, like location or identification of nearby objects of interest, to adapt their behavior to the current situation of the user. These applications acquire context information from distributed context sources, like GPS receivers and RFID beacons. Consequently, context-aware applications must be able to discover, select and bind to suitable context sources. Furthermore, due to the dynamic nature of context sources, their (lasting) availability is not guaranteed and the quality of their context information may vary. This makes maintaining a context binding – i.e., a binding between a context source and an application - complex. In this paper, we propose a context binding transparency that simplifies creating and maintaining a context binding. The proposed solution encompasses a language to specify context requirements and offerings, and interfaces to retrieve the requested context information. The responsibility for discovery, selection, binding and maintenance of required bindings is delegated to our underlying middleware, coined CACI. By providing this context binding transparency, we reduce the required development effort for creating context-aware applications.

## 1 Introduction

The user's environment is increasingly equipped with a multitude of devices, ranging from laptops and mobile phones to Internet connected refrigerators. In the vision of ubiquitous computing [1], these devices should cooperate to unobtrusively offer relevant services to users. 'Unobtrusiveness' is defined by the Merriam-Webster dictionary as not being obtrusive, meaning not being undesirably prominent[2]. In relation to ubiquitous computing this means that, amongst others, offered services should take the current situation of the user into account to tailor the behavior to that situation. For example, when a telephone call comes in and a user cannot be disturbed, his phone will not ring but vibrate.

A way to enable unobtrusive services is context-aware computing [2]. Context-aware applications take besides explicit user input also the situation of an entity (i.e.

[1] This work is part of the Freeband AWARENESS Project. Freeband is sponsored by the Dutch government under contract BSIK 03025. (http://awareness.freeband.nl)
[2] http://www.m-w.com/dictionary/obtrusive

person, place or object relevant to the functioning of the application) into account, to provide tailored functionality. Common examples of context information are user location and presence, temperature of a room and available communication bandwidth. Context-awareness is particularly interesting for mobile applications, because these type of applications function in constantly changing environments. For example, a mobile tourist guide application could benefit from context by offering personalized tourist information based on the current physical location of the user.

Generally, a context-aware system consists of software components that can have the following two roles: context producer and context consumer. Context producers acquire context information from the environment and make it available to context consumers. For example, a software component that can offer the location provided by GPS receivers or RFID beacons. We call these software components context sources. A context consumer retrieves context information from a context producer. Typically, a context-aware application fulfills a context consumer role. However, a context-aware application can also be a context producer when its application logic creates context information. For a context-aware application to use context information, it has to associate with a suitable context source. We call this association a context binding.

In first generation context-aware applications context bindings were hard coded inside the application logic [3]. This leads to fixed couplings between context-aware applications and specific context sources, resulting in inflexible applications that cannot cope with changing circumstances and future evolutions. Currently, there is a trend towards middleware infrastructures for context-aware applications [4]. These infrastructures offer solutions to recurring problems like context discovery, adaptation and security. However, some key challenges remain, driven by the inherent characteristics of context sources: (i) context information can be offered by a multitude of physically distributed context sources. Problems that arise are how-to find relevant context sources and how to easily bind to these remote context sources, (ii) (similar) context sources can be provided by different context providers using different data models for storing and accessing context information. Problems that arise are how-to create interoperability between context sources, (iii) context sources have fluctuating properties. First, they can appear and disappear at arbitrary moment. Secondly, their quality, which is called Quality of Context (QoC)[5], can vary among context sources and also among context samples provided by context sources.

Without supporting mechanisms, coping with these challenges is hard for application developers. Creating and maintaining context bindings requires substantial development effort. In this paper, we propose a context binding transparency that provides means to simplify the development of creating and maintaining context bindings. This transparency encompasses a language to specify context requirements and offerings (see [6]), and interfaces to retrieve the requested context information. The responsibility for discovery, selection, binding and maintenance of required context bindings is shifted to our underlying middleware context binding mechanisms, coined CACI. By providing this context binding transparency, we reduce the required development effort for creating context-aware applications.

In this paper, we focus on the high-level design aspects of our context binding transparency to position it in the overall development of context-aware applications. In comparison, in [6] and [7] we take a more bottom-up approach describing the de-

tailed design of our underlying context binding mechanisms (CACI), the context requirement language and proof-of-concept implementation.

The remainder of this paper is structured as follows: Section 2 discusses a basic model of context-aware applications, which forms the framework to position our proposed binding transparency. Furthermore, it is used to identify involved functions relevant for the design of our binding transparency. Section 3 continues by describing the high-level design of our binding transparency. Section 4 briefly discusses our proof-of-concept implementation and evaluation. In Section 5, we present some conclusions and directions for future work.

## 2 Model of context-aware applications

Since the 1980's, context-awareness has gained momentum in the ubiquitous computing, pervasive computing and ambient intelligence research communities. Context is defined as any information that characterizes the situation of an entity (i.e. user, place or computational object) relevant to the application behavior [2]. Examples of context are location, presence, temperature, number of emails, conversation partner etc. We define context-aware applications as applications that adapt their default behavior to the context at hand.

Context-aware applications are characterized by the use of context inputs additional to user input. Optionally, context-aware applications can produce context, which can be made available to the environment (for example to other context-aware applications). Therefore, context-aware applications can also act as context sources when they produce and output context. Figure 1 illustrates these characteristics.

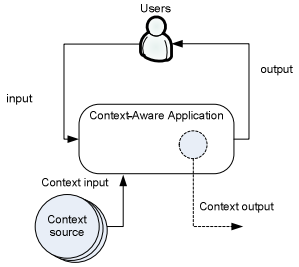


**Fig. 1.** Context-aware application

We consider context-aware applications as an extension of non-context-aware applications. Context-aware applications have a basic non-context-aware behavior, which is adapted when context is used. We assume that a context-aware application can function without context but can do its job better when considering context.

Let us consider the previous intuitive notion on context-aware application and discuss it from a modeling point of view. We start with a high-level black-box description of a context-aware application and its supporting context middleware (see Figure 2).

A context-aware application uses context to adapt its behavior. Furthermore, it can produce context. We consider a context middleware that facilitates these needs by

offering a context retrieval and context publishing service. The context retrieval service facilitates the context-aware application to retrieve context. The context publishing service facilitates the context-aware application to publish its context to the environment.

The context-aware application in itself can be further detailed into two main functional elements: application logic and context logic (see Figure 3). Application logic is the behavior of the application (that fulfils the users need), which is influenced by context information and possibly can produce context. Context logic is the behavior needed for the application logic to retrieve its required context information or publish its offered context information.

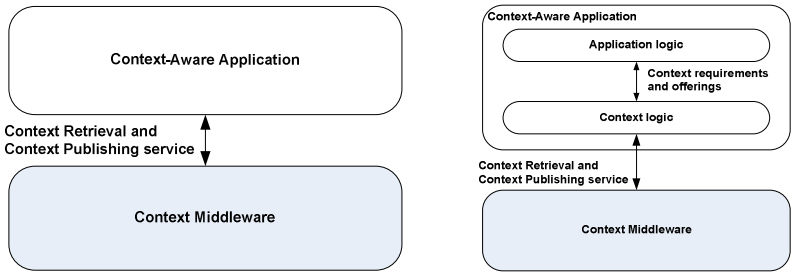


**Fig. 2. & Fig. 3.** Zooming into context-aware applications

When zooming into the context logic (see Figure 4), two functional elements can be distinguished. First, the context consumer element consists of behavior to retrieve context required by the application logic. For an application to be context-aware, it requires to have context consumer functionality. The context producer element is optional and consists of behavior to publish the offered context of the application logic. In this paper, we also use the term context consumer and producer role. With this we indicate that a context-aware application consist of a context consumer and respectively context producer element.

Both the context retrieval service and the context publishing service are provided by the context middleware. We denote the specific middleware functionality that provides these services as context management. Context middleware may also consist of other elements like communication and security mechanisms. These are out of the scope of the model presented in this paper.

We model context sources similarly to context-aware applications (see Figure 5). It consists of application logic responsible for sensing, acquiring and processing context into context offerings. The context logic has a mandatory context producer function that is responsible for publishing offered context produced by the application logic.

As you can see, a context-aware application A can appear as a context source for context-aware application B that is using the context of context-aware application A. However, there also exist non-context-aware applications that are context sources. They have as sole purpose producing context. For example, an application part that wraps a GPS to produce location information.

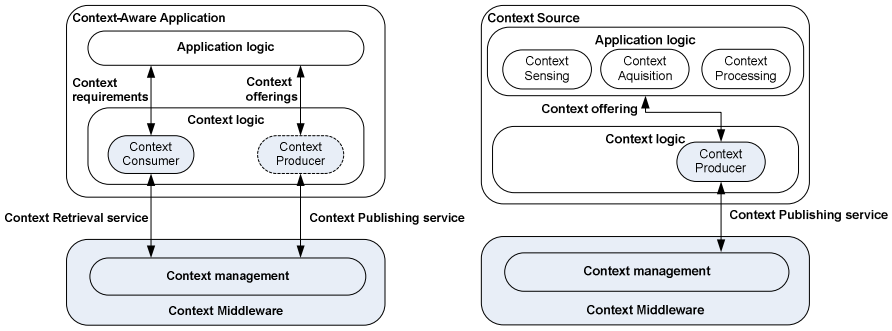


**Fig. 4. & Fig. 5.** Zooming into context-aware applications and context sources

Figure 6 shows the relationship between a context consumer (i.e. context-aware application) and a context producer (i.e. context source). For a context consumer to retrieve context it needs a binding with a context producer. Our binding transparency hides physical context producers and provides ways, via interactions with the context middleware for context consumers to create and maintain context bindings. The application developer of a context-aware application (context consumer) is unaware of the context producer with which a binding is created, how this binding is created and how this binding is maintained to overcome the dynamicity of context producers. The next section discusses this context binding transparency in more detail.

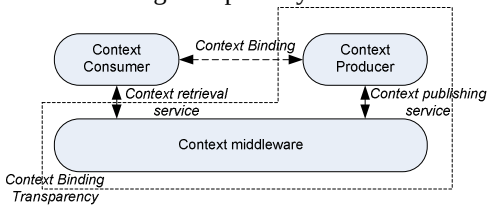


**Fig. 6.** Relation between context consumer and context producer

## 3 High-level design of a Context Binding Transparency

The concept of transparencies was introduced in the context of distributed system in the Open Distributed Processing (ODP) reference model [8]. Transparencies are mechanisms that hide certain complexities for the application developer to simplify the development of their application. For example, location transparency [8] masks out the problems of locating distributed objects by enabling them to be found using logical names rather than physical addresses.

We propose a binding transparency that hides the complexities of developing context-aware applications dealing with creating and maintaining context bindings. When re-considering figure 4, our binding transparency can be positioned between the context-aware application and the context middleware offering a specific instance of the context retrieval and publishing service. Due to space limitations, we focus in this

paper on the context retrieval service. Key features of our proposed transparency (i.e. for context retrieval) are:

- Application developers specify their context requirement in a context requirement language rather than programming code.
- Our binding mechanism is responsible for parsing and interpreting the context requirements to:
    - Initialize a context binding by discovering (using an underlying discovery mechanism), selecting and binding to suitable context sources.
    - Hide the fact that bound context sources may disappear by re-binding at run-time to other suitable context sources.
    - Hide the fact that the QoC of the context sources may fall below a specified level by re-binding at run-time to other suitable context sources.
    - Re-bind to context sources with a higher QoC when they become available.
- Supporting interoperability of different available underlying context discovery mechanisms.
- Offering a uniform interface to retrieve context information without being aware of the heterogeneity of physical context source.

Figure 7 presents the internal perspective of the context retrieval service showing functional elements and their interaction. A context-aware application with its specific application logic formulates some context requirements (i.e. specified by the application developer) (1). These are translated into in one or more binding creation requests by the context logic and transferred to the context middleware (2). These requests consist of two parts. The first part specifies the required context information. The second part specifies the acceptable QoC. The context middleware in terms of the context retriever stores the binding requests and tries to resolve the context requirements by invoking context discovery requests at on or more available context managers (3). A context manager is a repository of physical context sources (4) which offers a context discovery service. This service enables context-aware applications to find context sources that can offer specified context.

The context logic of the application can use the context retrieval service to retrieve context in a (i) request-response and (ii) subscribe-notify manner. The subscription mechanism is controlled by application specified subscription criteria (e.g. periodical updates, updates when the context information changes, updates based on criteria that are more complex).

When an established binding fails (e.g. application out of range of the sensors) or new context sources (i.e. other QoC) become available, the middleware will try to (re)establish a suitable binding. When this is not possible, the context-aware application is notified. When the application does not require any context anymore, for example because it quits, it can notify the middleware to destroy the established binding.

When a binding creation request is invoked by the context-aware application, the context retriever creates a context producer proxy (CP'') (5) which acts as the single point of access to context used by the application. This context retriever discovers and selects a suitable context producer (CP*, reference of a context source (CS)). The CP'' is bound to the selected CP* (6). From this moment the CP'' can deliver context to the context-aware application. When a binding destroy request is invoked by the

context-aware application the CP'' is unbound to the CP* and is cleaned-up from the context retriever administration.

Recapitalizing, from the perspective of the application, the physical context sources and their dynamicity are hidden. The application interacts with the context middleware to retrieve context rather than with individual context sources.

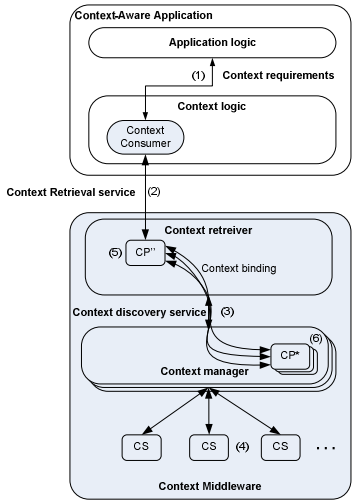


**Fig. 7.** Context retrieval binding transparency

## 4 Implementation and Evaluation

In [6] and [7], we discuss more details on our underlying context middleware (CACI) and our proof-of-concept implementation. The implementation is based on Java and the OSGi component framework. The language to express context requirements is based on XML. The prototype is tested on a regular PC using a standard VM and a mobile device running the IBM J9 virtual machine. For testing, we create a component generator that can generate and deploy a template component with a user-defined context requirements document. SimuContext [9], a framework to simulate context sources, is used as the underlying context discovery mechanism.

Preliminary evaluation of our binding transparency using the implemented proof-of-concept (see [7]) showed a decrease of programming effort, needed to create and maintain bindings, from approximately 1000 to 60 lines of code. The learning curve to use our transparency is limited due to the simple XML-based context requirement specification language and straightforward interfaces. Furthermore, without our transparency also the underlying discovery mechanisms have to be learned. Preliminary performance evaluations, performed by inserting probes in the proof-of-concept, showed neglectable (< 1ms) overhead for establishing a binding, for instance compared to remote invocation of a context discovery mechanism, introduced by the binding mechanisms.

## 5 Conclusion

This paper discusses a model of context-aware applications and focuses on the design aspects of our proposed binding transparency. We take the perspective of the application developer and aim, with this transparency, to hide the complexity of creating and maintaining context bindings, needed for context-aware applications to retrieve context information. We propose a context retrieval service that implements this transparency. This service has as goal to offer the 'best possible' context to the service user during the existence of a context binding. With the 'best possible' context, we mean: (i) guaranteed continuity of available context information when possible and (ii) deliver context information that has the highest possible quality of context. Initial evaluations, by applying the transparency in applications develop in the AWARENESS project have shown the feasibility of our approach. However, several aspects need to be tackled in our future work: (i) design of the context providing service, (ii) extending the binding transparency to (dynamically) interoperate with multiple discovery mechanisms and (iii) perform more extensive performance evaluations.

Other approaches that propose a similar kind of mechanism that uses a specification language to bind services are the service binder [10] and the extended service binder [11] proposed for the OSGi framework. However, they focus on generic OSGi services and therefore lack support for key aspect required for context exchange like quality of context and maintenance of the binding.

## References


[1] M. Weiser, "The Computer for the Twenty-First Century", Scientific American, vol. September, pp. 94-110, 1991.

[2] A. Dey, "Providing Architectural Support for Context-Aware applications", PhD thesis, Georgia Institute of Technology, 2000.

[3] M. Korkea-aho, "Context-Aware Applications Survey", 2000, http://users.tkk.fi/~mkorkeaa /doc/context-aware.html.

[4] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam, "Middleware for Distributed Context-Aware Systems", in DOA 2005. Agia Napa, Cyprus, 2005.

[5] T. Buchholz, A. Kupper, and M. Schiffers, "Quality of Context: What it is and why we need it", in HPOVUA 2003, Geneva, Switzerland, 2003.

[6] T. Broens, A. Halteren, and M. v. Sinderen, "Infrastructural Support for Dynamic Context Bindings", presented at 1st European Conference on Smart Sensing and Context (EuroSSc'06), Enschede, the Netherlands, 2006.

[7] T. Broens, M. v. Sinderen, A. Halteren, and D. Quartel, "Dynamic Context Bindings in Pervasive Middleware", in Middleware Support for Pervasive Computing Workshop (PerWare'07). White Plains, USA, 2007.

[8] G. Blair and J. Stefani, "Open Distributed Processing and Multimedia", Addison-Wesley, 1998.

[9] T. Broens and A. van Halteren, "SimuContext: simulating context sources for context-aware applications", presented at Intl. Conference on Networking and Services (ICNS06), Silicon Valley, USA, 2006.

[10] H. Cervantas and R. Hall, "Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model", in ICSE 2006. Edinburgh, Scotland, 2004.

[11] A. Bottaro and A. Gerodolle, "Extended Service Binder: Dynamic Service Availability Management in Ambient Intelligence", in FRCSS 2006. Vienna, Austria, 2006.