# Capturing Context Requirements*

Tom Broens, Dick Quartel, and Marten van Sinderen

Centre 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 require context information to adapt their behaviour to the current situation. When developing context-aware applications, application developers need to transform specific application context requirements into application logic to discover, select and bind to suitable sources of context information. To facilitate the development of context-aware applications, we propose a Context Binding Transparency that simplifies the process of retrieving context information. A major element of this transparency is the declarative approach to capturing context requirements. This enables application developers to specify their context requirements at a high level of abstraction rather than in programming code, and thus to separate the transformation of context requirements into context binding logic from the development of the actual application logic. In this way, we try to decrease the development effort and facilitate maintenance and evolution of context-aware applications. This paper discusses the design of this binding transparency; especially focusing on the language we developed to capture context requirements.

**Keywords:** Context-Aware applications, Context Requirements, Context Binding Transparency, Context Binding Description Language (CBDL).

## 1 Introduction

Ubiquitous computing envisions a situation in which users are surrounded by computing devices that offer unobtrusive services. Unobtrusiveness is defined by Merriam-Webster's dictionary as not being undesirably prominent. 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 service behaviour to that situation. For example, when a user receives a telephone call but his situation is such that disturbance by audible signals would be inappropriate, his phone should vibrate rather than ring.

A way to enable unobtrusive services is context-aware computing. Context-aware applications use, besides explicit user inputs, context information to adapt the

* This work is part of the Freeband AWARENESS Project. Freeband is sponsored by the Dutch government under contract BSIK 03025. (http://awareness.freeband.nl).

G. Kortuem et al. (Eds.): EuroSSC 2007, LNCS 4793, pp. 223–238, 2007.

application behaviour to the situation at hand. Context is defined as any information that characterizes the situation of an entity [1] (e.g. user location, availability, weather conditions).

Context information is provided by so-called context sources. These context sources are software entities distributed in the user's environment that acquire context information and make it available to context-aware applications. For a context-aware application to use context information, it has to associate with a suitable context source that can provide the required context information. The association between a context-aware application and a context source that can provide the required context information is called a context binding.

Context sources exhibit certain characteristics that make developing context bindings complex: (i) context information can be offered by a multitude of physically distributed context sources. Problems that arise are how to discover relevant context sources and how to retrieve context information from 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 interoperate between context sources and their discovery mechanisms and (iii) context sources are dynamic. Firstly, they can appear and disappear at arbitrary moments (i.e. dynamic availability). Secondly, their quality, which is called Quality of Context (QoC) [2], can vary in time or it can be different from other context sources.

To facilitate the development of context-aware applications, we propose the *Context Binding Transparency* that facilitates the process of developing context-aware applications by simplifying the creation and maintenance of context bindings. Our Context Binding Transparency includes the following three main elements:

1. A *context binding description language* that enables developers to specify their context requirement at an abstract level rather then directly programming them.
2. A *context binding mechanism* that, based on a context requirement specification, creates and maintains context bindings, thereby hiding the distribution, heterogeneity and especially the dynamicity of context producers for the application developer.
3. A *context discovery interoperability mechanism* which hides the heterogeneity and dynamic availability of context discovery mechanisms.

In this paper, we focus on the first element of our proposed transparency: the *Context Binding Description Language* (CBDL). This language enables application developers to specify their context requirements at a high level of abstraction rather than in programming code, and thus to separate the transformation of context requirements into context binding logic from the development of the actual application logic. In this way, we try to decrease the development effort and facilitate maintenance and evolution of context-aware applications. Furthermore, we briefly discuss the second element (i.e. context binding mechanism), however, for details the reader is referred to [3, 4]. For more information on the third element of the proposed transparency (i.e. context discovery interoperability) see [5]. For more information on the overall AWARENESS project see [6].

The remainder of this paper is structured as follows: Section 2 gives a high-level overview of our proposed Context Binding Transparency. Section 3 identifies the requirements of the context binding description language (CBDL). Section 4 presents the design of CBDL. Section 5 discusses the usage of CBDL in the development process of context-aware applications, and it discusses the integration of CBDL with our context binding mechanism. Section 6 gives an example how to apply CBDL in developing a context-aware application and presents a generic reflection on the usability of CBDL. Section 7 discusses related work. Finally, in Section 8, we present a summary and future work.

## 2 Overview of the Context Binding Transparency

The transparency concept was introduced in the context of distributed system in the Open Distributed Processing (ODP) reference model [7]. Transparencies are offered by mechanisms that hide certain complexities for the application developer to simplify the development of the application at hand. For example, location transparency [7] hides the problems of locating distributed objects by enabling them to be found using logical names rather than physical addresses.

Our Context Binding Transparency hides certain complexities of developing a context binding. A context binding exists between a context consumer and a context producer (see Figure 1). A context consumer is typically a context-aware application, which consumes context information to be able to adapt its behaviour. A context producer is typically a context source, which acquires (produces) context information and makes it available to its environment. We propose to shift the recurring problem of creating and maintaining a context binding from the application to (context) middleware that offers a Context Binding Transparency. This transparency offers a context retrieval and publishing service used for easy exchange of context information. By using these services, 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.

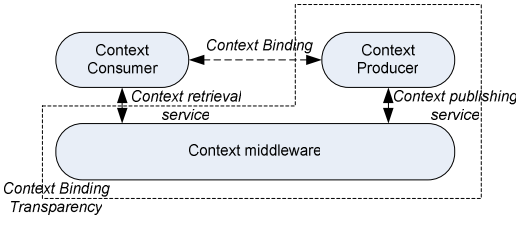


**Fig. 1.** Context Binding Transparency

Key features of our binding mechanisms offering the proposed Context Binding Transparency are:

- *Initialization:* based on the context requirement specification (expressed in CBDL) the context binding mechanism tries to resolve a context binding by discovering (using available underlying discovery mechanisms), selecting and binding to one ore more suitable context sources.
- *Maintenance:* based on specified criteria (e.g. QoC) the binding mechanism maintains the binding by:
    - Re-binding at run-time to other suitable context sources when already bound context sources disappear.
    - Re-bind at run-time to other suitable context sources when the QoC that is provided by the already bound context source may fall below a specified level.
    - Re-bind to context sources with a higher QoC when they become available.
- *Releasing:* when the application no longer needs context information, the established bindings are released.

For a more elaborate discussion on the Context Binding Transparency, see [8].

## 3 Context Requirement Analysis

In this section, we discuss the requirements for our context binding description language (CBDL). The context requirement specifications, expressed in CBDL, are used by the binding mechanism to create and maintain context bindings. Thereby, the context binding mechanism has to bridge the gap between the requirements specified by the developers of context-aware applications and the (heterogeneous) context delivery capabilities of underlying context discovery mechanisms capable of discovering available context producers (see Figure 2).

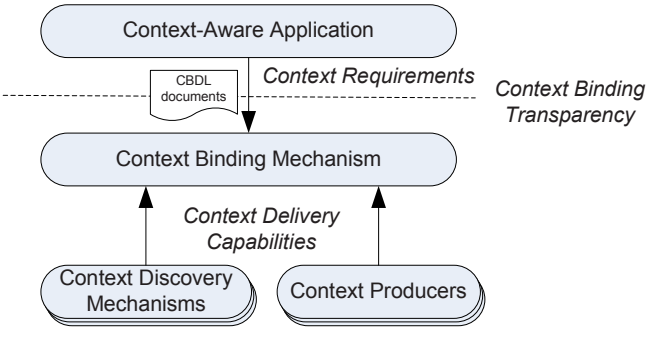


**Fig. 2.** Bridging the gap between context requirements and context delivery capabilities

We consider the following generic non-functional requirement in the design of CBDL:

- *Generality:* specification of context requirements in CBDL should not be restricted to specific application domains. CBDL should apply to a broad range of context-aware applications.
- *Usability:* specification of context requirements in CBDL should be easy and should not require a steep learning curve.

To capture the functional requirements of CBDL, we take a two-step approach. First, we analyse the capabilities of current context (discovery) middleware mechanisms to identify common capabilities currently offered (Section 3.1). Secondly, we analyse use-cases (from which we present two) to complement our requirements (Section 3.2). Together, these lead to requirements on what should be possible to express in CBDL to be able to capture context requirements used for creation and maintenance of context bindings by our underlying context binding mechanism (Section 3.3).

### 3.1 Analysis of Context Discovery Middleware

Currently several context middleware mechanisms are developed to facilitate the development of context-aware applications [9]. These mechanisms solve recurring development problems, such as dealing with privacy issues when exchanging context information, creating new context information by reasoning on existing context information, and discovery of distributed context sources. In this section, we analyse current context discovery mechanisms. First, because they implement solutions that fulfil context requirements application developers have and secondly, because our proposed Context Binding Transparency builds on top of these solutions.

We analyse nine different context discovery mechanisms. The first four originate from the Freeband AWARENESS project (CMF, CCS, CDF and Jexci) [10]. These mechanisms are developed for different domains (e.g. telecommunication operator domain, different administrative domains, ad-hoc situations) [10]. Secondly, we review the context discovery mechanism originating from the IST Amigo project (CMS) [11]. Thirdly, we complete our analysis with four external context middleware mechanisms (Context Toolkit [12], PACE [9], Solar [13], and JCAF[14]).

The analysis consisted of reviewing the following aspects of the different discovery mechanisms:

- *Interaction mechanism:* What interaction mechanism do the analyzed discovery mechanisms support?
- *Interaction data:* what type of information is expressed in the context discovery request and response?

The result of our analysis is presented in table 1. From the analysis, we distinguish the following common aspects provided by current context discovery mechanisms:

- All mechanisms support the common request-response and subscribe-notify interaction mechanism to retrieve context information.
- All mechanisms require information on the type of context and the entity to which the context relates, to discovery context sources.
- The majority of the mechanisms introduce the notion of quality of context in the request for context information.

- Some mechanisms require a form of security token (i.e. identity information on the entity that is requesting context) to be able to discover context sources.

**Table 1.** Comparing context discovery mechanisms

| | Interaction mechansism | | Interaction data | | | | |
|---|---|---|---|---|---|---|---|
| Frameworks | Req-Resp | Sub-Not | Entity | Type | QoC | Sec. info | Format |
| CMF | v | v | v | v | v | v | RDF |
| CCS | v | v | v | v | v | v | SQL/PIDF |
| CDF | v | v | v | v | v | - | RDF/PIDF |
| Jexci | v | v | v | v | v | v | *Negotiable (PIDF/java objects)* |
| CMS | v | v | v | v | v | - | RDF |
| Context Toolkit | v | v | v | v | - | v | XML |
| Pace | v | v | v | v | v | - | Context Modelling Language |
| Solar | v | v | v | v | - | - | N/A |
| JCAF | v | v | v | v | - | - | Java objects |

### 3.2 Analysis of Use-Cases

We complement the previous results by analysing use cases. Here we present two uses cases, which we consider representative for a broad range of context-aware application.

**Healthcare Use-Case: Epilepsy Safety System (ESS)**
The ESS monitors vital signs of epilepsy patients and determines upcoming epileptic seizures. When a likely seizure is detected, the system notifies nearby and available caregivers with instructions on the location (e.g. in lat/long context format) of the patient and route information to the patient. The application uses context information on the location of the patient and the caregiver and context information on availability of the caregivers to provide this functionality. The quality of the location data of the patient should have a minimal precision of 5m (i.e. the specified location of the patient may differ 5m from the actual location) to be able to dispatch caregivers to the right location. The location data of caregivers only has to be minimally 100m precise to be able to determine which one is nearby.

Additionally, the vital signs of the patient are transferred to the healthcare centre where care professionals monitor the patient's state and stays in contact with the dispatched caregiver. Context information on the available bandwidth (e.g. in kb/s) of the patient's device is used to tailor the granularity of transferred vital signs (e.g. increase or decrease sample frequency) and the amount of vital signs (e.g. decrease the number of send channels) to ensure transfer of vital signs to the healthcare centre.

**Office Use-Case: My Idea Recorder (MIR)**
During meetings, users can use their camera phones to take high-resolution pictures of whiteboard sketches to capture their ideas for future use. The MIR system distributes copies of these pictures to meeting participants. The phone automatically determines the persons that are currently in the meeting based on meeting information (e.g. in Boolean context format) from user's calendars and nearby Bluetooth devices. When the meeting information is not at least 75% correct (i.e. probability of 75% that the participant is actually in/out a meeting), the application asks the participant if he is in

the meeting. The system delays the data transfer until an adequate network becomes available (i.e. GPRS, UMTS, WLAN or Bluetooth) taking into account the cost and bandwidth characteristics of each network type and the battery status of her phone.

**Discussion**
We analysed multiple use-cases, from which we consider the previously discussed two, representative for a broad range of context-aware applications. From these use-cases, we derive the following characteristics of context and context-aware applications:

- Context is defined by its c*ontext type* (e.g. location, availability, bandwidth, meeting status).
- Context is always related to a *context entity* (e.g. patient, doctor, voluntary care giver, meeting participant).
- Context information can be offered in different *context formats* (e.g. lat/long, xyz, nmea, Boolean).
- Relevancy of context information for applications can depend on different QoC criteria (e.g. precision, probability of correctness). See also [2, 15].
- Context transfer might occur during the whole life-span of the application or during a limited period (e.g. during a seizure).
- Delivery costs resulting from using context (e.g. use of a certain communication mechanism, commercial value of context) might pose criteria for the suitability of context bindings.

### 3.3 Overall Conclusions and Identification of CBDL Requirements

Based on the analysis of current discovery mechanisms and use cases, we identify the following requirements for CBDL:

- *Basic context elements:* Context type, entity and format are basic elements needed in CBDL to describe context requirements.
- *QoC criteria:* Application have QoC requirements and may react differently when these QoC are not met. Therefore, CBDL should enable application developers to specify quality levels on the required context information.
- *Costs:* Additionally to QoC, context delivery costs pose additional criteria on the suitability of a context binding. Application developers should be able to specify in CBDL cost criteria related to QoC criteria.
- *Binding characteristics:* Transfer of context information can be continuous during the life span of the application or can be limited to a certain period in the life span of the application. Context bindings are therefore not always required. An application developer should be able to specify in CBDL the characteristics of the required binding. This includes re-binding strategy (in case of losing a bound context source) and scope of the discovery. Furthermore, they should be able to specify if re-binding is necessary in case a QoC level cannot be maintained or better quality context sources may appear.

- *Notification:* Although our transparency strives for continuous availability of high quality context information, this might not always be possible. Application developers have to be able to specify in CBDL a notification strategy in case a lost binding cannot be recovered or QoC level cannot maintained, such that the context-aware application can adapt its behaviour to these situations.

## 4 Design of the Context Binding Description Language

We distinguish three types of information in a CBDL document:

- *Context specification:* basic information on what context information the context-aware application requires.
- *Quality criteria:* information on the quality levels which are acceptable for the context-aware application to function.
- *Binding options:* configuration information required to control the discovery, selection, binding, and maintenance process of a context binding.

These categories are represented in the UML meta-model of the CBDL language as depicted in Figure 3.

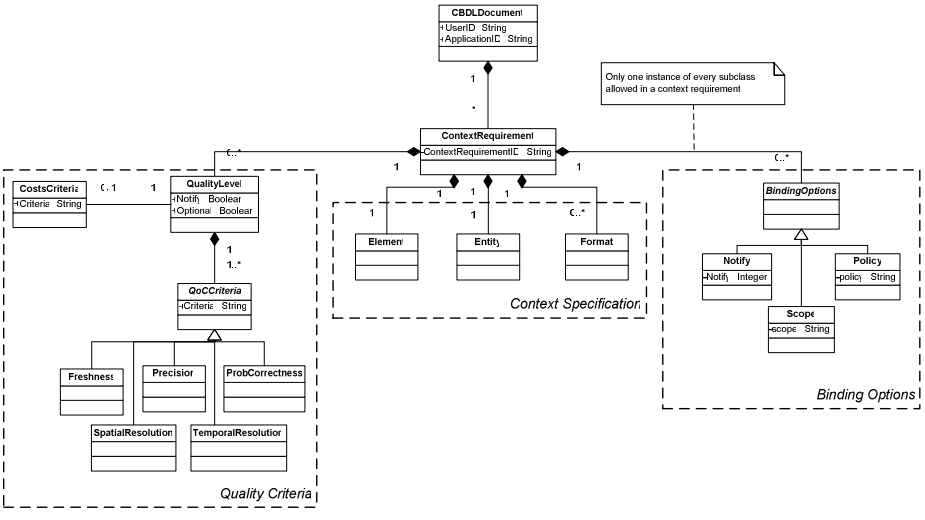


**Fig. 3.** CBDL language meta-model

The root of the CBDL language is the *CBDLDocument* element, which specifies which user is requesting a context binding (*UserID*) and to which application this binding belongs (*ApplicationID*). This information can be used as security information (e.g. identity to retrieve a security token) to be able to invoke underlying context discovery mechanisms. Furthermore, a CBDL document (*CBDLDocument*) enables application developers to specify multiple context requirements (*ContextRequirement*).

These requirements have to be uniquely identified by an ID (*ContextRequirementID*). This ID can be used to retrieve a handle on the established binding, to enable the context-aware application to retrieve context information.

Every context requirement (*ContextRequirement*) consists of mandatory context specification information. This information specifies: (i) a single type of context information that the application requires (*Element*), (ii) one entity to which the required context is related (*Entity*) and (iii) zero or more data formats the required context may have (*Format).*

Optionally, an application developer can specify multiple quality levels (*QualityLevel*). These quality levels consist of one or more quality criteria coupled with an optional cost criterion. We distinguish five possible types of QoC criteria based on [2, 15]. These are: (i) *Precision*: "granularity with which context information describes a real world situation", (ii) *Freshness:* "the time that elapses between the determination of context information and its delivery to a requester", (iii) *Temporal Resolution*: "the period of time to which a single instance of context information is applicable", (iv) *Spatial Resolution*: "the precision with which the physical area, to which an instance of context information is applicable, is expressed" and (v) *Probability of Correctness*: "the probability that an instance of context accurately represents the corresponding real world situation, as assessed by the context source, at the time it was determined" [15].

Additionally, the application developer may specify if the application needs to be notified when the QoC/Costs of the delivered context information comes into the range of the specified level or falls out of the range (*Notify,* default= true). Furthermore, the application developer specifies if the re-binding mechanisms needs to be triggered when the QoC of the delivered context information falls below the specified QoC level (*Optional,* default=false).

Furthermore, an application developer can optionally specify binding options (*BindingOptions*) to control the binding process of the context binding mechanisms. The following options can be specified:

- *Notify*: the application developer can specify the level of notification he wants to receive on the binding process. The following cumulative levels are identified:
  - *0*: no notifications.
  - *1*: notification when a binding is established.
  - *2*: notification when a binding is established and broken.
  - *3*: notification when a binding is (re-)establishing and broken (default).
- *Policy:* the application developer can specify what binding policy should be taken:
  - *Static:* when a binding is broken, no re-binding is necessary.
  - *Dynamic:* when a binding is broken re-binding is necessary (default).
- *Scope:* the application developer can specify if context sources should be searched only inside the scope of the local infrastructure (i.e. producers deployed inside the local application container) or also in external context discovery mechanisms (default = global).

# 5 Using CBDL and the Context Binding Mechanism

First, we present a general discussion on how to use CBDL in the development of context-aware applications (Section 5.1). Secondly, we describe how CBDL is integrated with our context binding mechanism (Section 5.2).

## 5.1 Using CBDL for the Development of Context-Aware Applications

Figure 4 presents the development trajectory of a CBDL based context-aware application using our underlying context binding mechanism, called Context-Aware Component Infrastructure (CACI). CACI is the implementation of the context binding mechanism exposing the proposed Context Binding Transparency.

On design-time, the application developer creates the application logic of the context-aware application. Furthermore, he specifies the context requirements relevant for its application in a CBDL document. During the design of the application logic, the application developer has to take the following aspects in mind:

- Create application logic that is able to retrieve context using the interfaces offered by the context binding mechanisms and the context requirement identifiers (*ContextRequirementID*) specified in the CBDL document.
- Create application logic that can receive notification by the underlying binding mechanisms of changes in QoC and binding status based on the notify flags (*notify*) specified in the CBDL document.
- Create application logic that can adapt to unavailability of context or availability of context with too low quality.

Both the application logic and the CBDL document are bundled into a context-aware application component, which can be deployed in the Context-Aware Component Infrastructure (CACI).

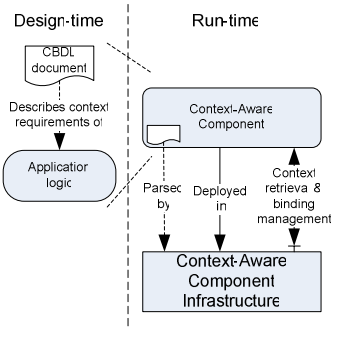


**Fig. 4.** Development trajectory of CBDL-based context-aware applications

## 5.2 Integration of CBDL and CACI

Figure 5 presents a functional decomposition of the binding mechanism deployed in the CACI infrastructure. After deployment of the context-aware application

component, binding requests are extracted from the CBDL document by the parser. These request are transformed in a discovery request forwarded to available context discovery mechanisms (see [5]). The discovery results are analyzed and a context producer that can fulfil the context requirement (i.e. binding request) is selected. The selected context producer is bound to an internally created context producer proxy (see [3, 8]) from which the context-aware application component can retrieve context information. This proxy is monitored for disappearing of its bound physical context producer. In case of a lost binding to a context source, this triggers a re-binding processes starting from discovery of suitable context producers for a new binding. Furthermore, some context discovery mechanisms offer active discovery, which enable the binding mechanisms to subscribe to discovery changes (e.g. new producers become available), this triggers a new selection process to determine if the new producer is more suitable for the context-aware application component. Status information on the binding can be notified to the application component based on the flags in the CBDL document.

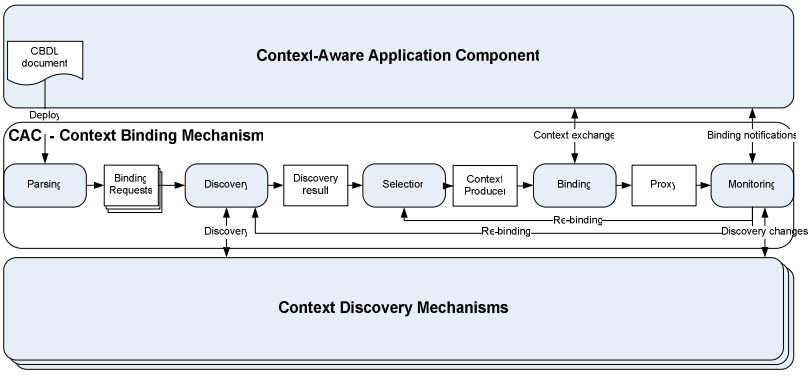


**Fig. 5.** Functional decomposition of the CACI Context Binding Mechanisms

We chose to represent the CBDL language using XML, as it is currently the de-facto standard for structured data. Consequently, tool support for creating CBDL document is widely available. Furthermore, usage of XML enables easy parsing and validation of the correctness of CBDL documents using for example XML Schema. Therefore, we derived a XML Schema of the CBDL language meta-model. The proof-of-concept of the CACI infrastructure is implemented using java and the OSGi component framework (see [4]). Context-aware components are OSGi components, which have in their component descriptor a pointer to the XML-based CBDL document.

## 6 Case Example and Reflection

Here we present an example on how to use CBDL for describing context requirements for the Epilepsy Safety System (Section 6.1). Furthermore, we present a general reflection on the usability of CBDL (Section 6.2).

### 6.1 Case Example: ESS

Let's reconsider the Epilepsy Safety System discussed in section 3.2. The ESS deploys a sensor system on the patient's body (called a Body Area Network (BAN)) which collects and transfers vital signs when a seizure is detected. This data is stored and analyzed in healthcare centres for diagnosis, first aid and treatment. In case a seizure is detected, caregivers dispatched by the healthcare centres may offer help to the patient in this life-threatening situation.

Amongst others, possible beneficial context types in the ESS are: patient and caregiver location, caregiver availability and patient BAN bandwidth usage. Location information helps to decrease travelling time to the patient in case of emergencies. First, because the precise location of the patient (destination) is known and second because a nearby caregiver can be dispatched to the patient. Availability information of caregivers helps to decrease false dispatches of unavailable caregivers. Bandwidth usage information assists to tailor the transferred vital sign data to decrease costs in case a of non-emergency situation, while this information also assists to prevent congestion and failing transfer of vital sign data in case of emergency situations.

Developers create bindings by adding CBDL specifications to their application components. In these descriptions, they describe the context requirements of the application. Figure 6 presents a simplified CBDL description, which is added to the ESS component at the health-care centre to create the binding to location and availability producers. For the other components at the patient and caregiver side, the descriptions are similar.

```
<?xml version="1.0" encoding="UTF-8"?>
<CBDLDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="CBDL-schema.xsd" UserID="Healthcarecentre"
ApplicationID="ESS_Healthcarecentre">
   <ContextRequirement BindingID="patient_location">
       <Element>Location</Element>
       <Entity>Patient.Tim</Entity>
       <Format>lat/long</Format>
       <QualityLevel>
           <QoCCriteria>
               <Precision>5m</Precision>
           </QoCCriteria>
       </QualityLevel>
   </ContextRequirement>
   <ContextRequirement BindingID="patient_bandwidth">
       <Element>Bandwidth</Element>
       <Entity>Patient.Tim</Entity>
       <Format>kb/s</Format>
   </ContextRequirement>
   <ContextRequirement BindingID="caregiver_location">
       <Element>Location</Element>
       <Entity>Caregiver.John</Entity>
       <Format>lat/long</Format>
       <QualityLevel>
           <QoCCriteria>
               <Precision>100m</Precision>
           </QoCCriteria>
       </QualityLevel>
   </ContextRequirement>
   <ContextRequirement BindingID="caregiver_availability">
       <Element>Availability</Element>
       <Entity>Caregiver.John</Entity>
       <Format>boolean</Format>
   </ContextRequirement>
</CBDLDocument>
```

**Fig. 6.** Example of CBDL document specifying context requirement of part of the ESS

The CBDL documents are handled by the CACI binding mechanism. The application developer only needs to retrieve the bound producer (see figure 7) by subscribing a call-back to the IContextProducerManager service (i.e. the local services mechanism is provided by OSGi) offered by CACI. This notification strategy is applied to cope with timing differences between the application and the binding performed by CACI. CACI notifies the component when a producer is bound.

```
// standard OSGi code to retreive the CACI service
ServiceReference ref = bc_.getServiceReference(IContextProducerManager.class.getName());
IContextProducerManager manager = (IContextProducerManager)bc_.getService(ref);
// Retreival of the bound context producers (id's correspond with the CBDL document from
fig6)
IContextProducerCallback cb = new Callback(this);
try{
   manager.subscribe("patient_location", cb);
   manager.subscribe("patient_bandwidth", cb);
   manager.subscribe("caregiver_location ", cb);
   manager.subscribe("caregiver_availability", cb);
} catch(ConsumerSubscribeException e){
   System.out.println("Wrong binding ID.");
}
```

**Fig. 7.** Retrieval of context bindings

## 6.2 Reflection

Usability of CBDL depends on three major factors (see Figure 8). The first is expressiveness; are context-aware application developer capable of expressing context requirements suitable for their applications. Secondly, learning curve; how difficult is it for the context-aware application developer to learn the CBDL language. Finally, performance; how does the introduced layer of indirection (i.e. transformation of CBDL specification to context bindings) perform.

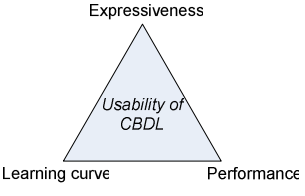


**Fig. 8.** CBDL usability triangle

- *Expressiveness:* By performing an extensive requirement analysis thereby reviewing current context management mechanisms and analyzing use cases, we created a language capable of specifying a broad range of context-aware applications. Furthermore, we added support for QoC criteria levels and binding process control.
- *Learning curve:* The application developers are required to learn how-to specify context requirements using CBDL and how-to use the CACI infrastructure. We do not think this presents a serious drawback, for two reasons. First, CACI provides simple interfaces to use the established bindings Furthermore, CBDL uses XML to express context requirements. XML

schemas are provided to ease this process. Future extensions could include a GUI that can enable developers to graphically generate CBDL descriptions and CACI integration code. Second, CA application development without CBDL and CACI also requires similar learning efforts to cope with the underlying discovery mechanism.

- *Performance:* Another possible drawback of adding a layer of indirection by CACI is its performance penalty. We performed some initial measurements on the time spent deploying a component, parsing the CBDL, discovery of context producers (making sure a discovery match can be made), selection of a context producer and returning the selected producer to the deployed component. This resulted both on the PC and the windows mobile device in insignificant overhead (less than 1ms time spent). Although more performance measurements are needed, our preliminary conclusion is that the delay introduced by CACI is considerably less or can be neglected compared to the delay for the (remote) discovery of context producers (i.e. communication and processing delay).

## 7 Related Work

In this paper, we propose a language to specify context requirements, which can be interpreted by our CACI infrastructure [3-5], to create and maintain context bindings in dynamic environments. The importance of coping with the dynamicity of (context) bindings in the infrastructure has also been recognized by others, who proposed several mechanisms for this purpose, such as context-sensitive bindings [16], service-oriented network sockets [17] and OSGi (Extended) Service Binder [18, 19]. Compared to CACI, these mechanisms have a similar goal but are not tailored to more advanced context-aware applications. Context producers and consumers have distinct characteristics that have to be incorporated in the binding mechanism to be able to fully support the application developer. For example, context binding mechanisms should be based on an extensible context model and the notion of quality of context (QoC) should be incorporated in the mechanisms.

To the best of our knowledge, no other initiatives exist to develop a language, which enables application developers to specify requirements for bindings with context producers at a high level of abstraction. Although Hong [20] recognizes the need for such a language, coined the context specification language (CSL), this language has not been detailed.

On the other hand, several types of languages have been proposed for other purposes, facilitating the development of context-aware applications in different ways. For example, Chan et al. [21] define a mathematical rule-based context request language. This language, implemented in XML, enables developers to specify context reasoning rules, using predicate calculus, interpreted by an infrastructure inference engine to retrieve required context information. Yau et al. [22] define a Situation-Aware object interface definition language (SA-IDL), which can be used to generate base classes for a situation-aware object. Etter et al. [23] describe a rule-based approach to specify context-aware behaviour in the ECA-DL language and to delegate the execution of this behaviour to the infrastructure using Event-Condition-Action rules. Robinson et al. [24] describe the Context Modelling Language (CML) which can be used to capture context information requirements to be used in the design of

context-aware applications. Chen [25] discusses a context ontology (SOUPA) that can be used to exchange context among entities in a uniform manner.

## 8 Summary and Future Work

In this paper, we discuss the Context Binding Description Language (CBDL). This language enables application developers of context-aware applications to specify their context requirements at a high level of abstraction rather than at the programming code level. CBDL thus enables a separation between the development of the application logic and the development of context bindings. The responsibility for creating and maintaining context bindings is shifted to our Context-Aware Component Infrastructure (CACI), which can interpret context requirements and use these to drive its discovery and binding mechanisms. In this way, we try to decrease the development effort and facilitate maintenance and evolution of context-aware applications.

The requirements for CBDL are derived from an analysis of current context management systems and future use scenarios. Elements incorporated in the CBDL language support (i) specification of context, (ii) specification of quality criteria, and (iii) specification of binding control information. We implemented the language using XML and integrated it with our CACI infrastructure. We believe that the CACI infrastructure offers a useful new transparency, which we call the Context Binding Transparency.

We plan the following research activities to further improve the CBDL concept and CACI prototype:

- Extending the CBDL language to support the development of applications with context producer capabilities or both context consumer and producer capabilities.
- Further evaluation of usability of the Context Binding Transparency featuring the CBDL language for development of context-aware applications.

## References

1. Dey, A.: Providing Architectural Support for Context-Aware applications, Georgia Institute of Technology (2000)
2. Buchholz, T., Kupper, A., Schiffers, M.: Quality of Context: What it is and why we need it. In: 10th Workshop of the HP OpenView University Association (HPOVUA 2003), Geneva, Switzerland (2003)
3. Broens, T., Halteren, A., Sinderen, M.v.: Infrastructural Support for Dynamic Context Bindings. In: Havinga, P., Lijding, M., Meratnia, N., Wegdam, M. (eds.) EuroSSC 2006. LNCS, vol. 4272, Springer, Heidelberg (2006)
4. Broens, T., et al.: Dynamic Context Bindings in Pervasive Middleware. In: Middleware Support for Pervasive Computing Workshop (PerWare 2007) White Plains, USA (2007)
5. Broens, T., Poortinga, R., Aarts, J.: Interoperating Context Discovery Mechanisms. In: 1st Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC 2007), Barcelona, Spain (2007)
6. Sinderen, M.v., et al.: Supporting Context-aware Mobile Applications: an Infrastructure Approach. IEEE Communications Magazine 44(9), 96–104 (2006)

7. Blair, G., Stefani, J.: Open Distributed Processing and Multimedia. Addison-Wesley, Reading (1998)
8. Broens, T., Quartel, D., Sinderen., M.v.: Towards a Context Binding Transparency. In: Broens, T. (ed.) 13th EUNICE Open European Summer School, Enschede, the Netherlands. LNCS, vol. 4606, Springer, Heidelberg (2007)
9. Henricksen, K., et al.: Middleware for Distributed Context-Aware Systems. In: DOA 2005, Agia Napa, Cyprus, Springer, Heidelberg (2005)
10. Benz, H., et al.: Context Discovery and Exchange. In: Pawar, P., Brok, J. (eds.) Freeband AWARENESS Dn2.1, Freeband AWARENESS Dn2.1 (2006)
11. Ramparany, F., et al.: An Open Context Management Information Management Infrastructure. In: Intelligent Environments (IE 2007) Ulm, Germany (2007)
12. Dey, A.: The Context Toolkit: Aiding the Development of Context-Aware Applications. In: Workshop on Software Engineering for Wearable and Pervasive Computing, Limerick, Ireland (2000)
13. Chen, G., Kotz, D.: Solar: An open platform for context-aware mobile applications. In: International Conference on Pervasive Computing, Zurich, Zwitserland (2002)
14. Bardram, J.: The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. In: Pervasive Computing, Munchen, Germany (2005)
15. Sheikh, K., Wegdam, M., Sinderen, M.v.: Middleware Support for Quality of Context in Pervasive Context-Aware Systems. In: PerWare 2007. IEEE International Workshop on Middleware Support for Pervasive Computing, New York, USA (2007)
16. Sen, R., Roman, G.: Context-Sensitive Binding, Flexible Programming Using Transparant Context Maintenance, in Technical Report WUCSE-2003-72. Technical Report WUCSE-2003-72, Washington University (2003)
17. Saif, U., Palusak, M.: Service-oriented Network Sockets. In: MobiSys 2003. International conference on mobile systems, applications and services, San Francisco, USA (2003)
18. Cervantas, H., Hall, R.: Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model. In: 26st International Conference on Software Engineering, Edinburgh, Scotland (2004)
19. Bottaro, A., Gerodolle, A.: Extended Service Binder: Dynamic Service Availability Management in Ambient Intelligence. In: FRCSS 2006 International Workshop on Future Research Challenges for Software and Services, Vienna, Austria (2006)
20. Hong, J.: The Context Fabric: An Infrastructure for Context-Aware Computing. In: CHI 2002. Doctoral Workshop, Human Factors in Computing Systems Minneapolis, USA (2002)
21. Chan, A., Wong, P., Chuang, S.N.: CRL: A Context-Aware Request Language for Mobile Computing. In: Cao, J., Yang, L.T., Guo, M., Lau, F. (eds.) ISPA 2004. LNCS, vol. 3358, Springer, Heidelberg (2004)
22. Yua, S., Wang, Y., Karim, F.: Development of Situation-Aware Application Software for Ubiquitous Computing Environments. In: COMPSAC 2002. International Software and Applications Conference, Oxford, England (2002)
23. Etter, R., Dockhorn Costa, P., Broens, T.: A Rule-Based Approach Towards Context-Aware User Notification Services. In: ICPS 2006. International Conference on Pervasive Services, Lyon, France (2006)
24. Robinson, R., Henricksen, K.: XCML: A runtime representation for the Context Modelling Language In: PerCom 2007. Pervasive Computing White Plains, USA (2007)
25. Chen, H., Finin, T., Joshi, A.: The SOUPA Ontology for Pervasive Computing. Ontologies for Agents: Theory and Experiences (2005)