Design, Implementation, and Evaluation of the Java Context Awareness Framework (JCAF)

Jakob E. Bardram
Centre for Pervasive Healthcare
Department of Computer Science, University of Aarhus
Aabogade 34, 8200 Århus N, Denmark
bardram@daimi.au.dk

DRAFT – April 2005
Date: 2005/06/20 21:07:14 | RCSfile: jcaf.v15.tex,v | Revision: 1.2

Contents

List of Figures

Abstract

Context-awareness is a key concept in ubiquitous computing. This technical report described the *Java Context-Awareness Framework* – JCAF, which is a Java-based context-awareness infrastructure and programming API for creating context-aware applications. The report describes:

- The background for JCAF which is research into a context-awareness infrastructure in hospitals.
- A system overview, including the key concepts in JCAF.
- The JCAF Architecture.
- The JCAF Programming Model.

Finally, the report provides some examples.

Chapter 1

Introduction

This technical report describes version 1.5 of the *Java Context-Awareness Framework* – **JCAF**. JCAF is designed to support *Context-Aware Computing* and has come out of our work with the design of context-aware applications in a hospital environment. This background if not discussed in this report – interested readers are refered to the publication on JCAF [4, 2, 3]. The purpose of this document is to describe the technical design and implementation of the JCAF Framework.

This document describes the overall goal of the ABC Framework (section 1.1); describes the high level architecture of the ABC software system (chapter **??**), including the key concepts; describes the programming model (chapter 4); and gives some examples of how to use JCAF (chapter **??**).

1.1 Goals of the JCAF Platform

A common goal for programming frameworks for context-aware computing is to enable programmers to easily develop and deploy context-aware applications. Programmers can focus on modeling and using context information and functionality specific for their application while relying on a basic infrastructure to handle the actual management and distribution of this information. Requirements for context-awareness systems and/or frameworks have been widely discussed and described [7, 11, 9, 8, 10, 1, 6, 2].

JCAF incorporates many of the lessons from these previous contributions. But JCAF is also distinctive in at least three ways: (i) JCAFs service-oriented infrastructure is a distributed system based on the idea of dividing context acquisition, management, and distribution in a *network of cooperating context services*; (ii) JCAF embodies a general-purpose, robust, modifiable, event-based, secure *architecture*; and (iii) JCAF has a generic, extensible, and expressive *Java programming model* for the deployment and development of context-aware applications and context models.

The three distinctive features have emerged out of our analysis of the existing proposed context-awareness frameworks as well as from our empirical work within healthcare [2, 5]. The PERVASIVE 2005 paper [4] provides more details on how JCAF differs from the other related middleware support for context-aware applications.

1.2 Environmental Assumptions

JCAF relies on the existence of a network of reasonable speed connecting computers running and using context services. We assume the latency of the network is reasonable.

The JCAF system is Java technology-centered. The JCAF architecture gains much of its simplicity from having the Java programming language as the implementation language. Java serialization, Java RMI, and the ability to dynamically download and run code is central to a number of the features of the JCAF architecture.

Chapter 2

Key Concepts

2.1 Context Services

The most important concept within the JCAF architecture is that of a *context service*. A service receive, manage, store, and distribute context information for entities.

Several context services can cooperate in a peer-to-peer fashion. In this way dedicated context services can be designed and deployed. For example a context service may be reponsible for maintaining entities and their context in an operating room or a context service may be designed and deployed to support a specific application.

2.2 Entities and Context

An *entity* models something that you want to manage context information for. Examples of entities are a person, a patient, a place, a TV, and a PC. Each entity has a *context* which is made up of a set of *context items* each having a specific *relationship* to this entity. Hence we can model tuples like '*Peter uses his PC*' with the tuple:

```
(peter:Entity , uses:Relationship , peter's_pc:ContextItem)
```

2.3 Context Clients

Context clients can submit context information and can listen to changes in context information for entities. Context clients that specialize in sensing, resolving, and submitting context information is often called *context monitors*. Context clients which are specialized in using context information is often called *context actuators*. Monitors and actuators can register at one or more context services and is notified when they need to provide or use up-to-date context information.

2.4 Context Events

The JCAF architecture supports distributed events. A context service allow special context client (*entity listeners*) to register interest in events in specific entities and receive a notification of the occurrence of such an event. The JCAF programming model also supports *type-based subscription*, i.e. enabling entity listeners to subscribe to types of entities, like all patients, persons, or places. This event mechanism enables distributed event-based programs to be written with a variety of reliability and scalability guarantees.

Chapter 3

The JCAF Runtime Infrastructure

The JCAF Runtime Infrastructure is illustrated in figure 3.1. Figure 3.1a illustrates a deployment situation with a range of *Context Services* which are connected in a peer-to-peer setup, each responsible for handling context in a specific environment like the operating room. A network of services can cooperate by querying each other for context information. All connections in figure 3.1 are remote and hence all components in JCAF can be distributed in a network.

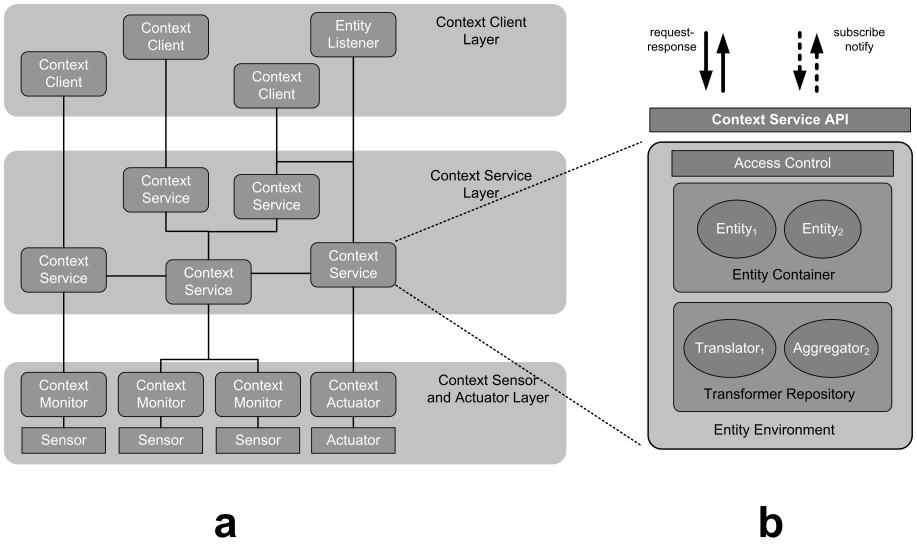

Figure 3.1: The Runtime Architecture of the JCAF Framework. a – An example of a deployment situation of a set of context monitors, context actuators, and a set of cooperating context services. b – Details of a context service.

3.1 Context Client Layer

Context Clients are the context-aware applications using the JCAF infrastructure by accessing one or more context services. Clients can access entities and their context; they can add or remove context information (and hence work as a context monitor, see section 3.3); they can add, query for, and use context transformers; and they can adjust the topology of the context service network. Clients can access entities and their context information in two ways. Either following a request-response schema, requesting entities and their context data, or by subscribing as an *Entity Listener*, listening for changes to specific entities. JCAF also supports *type-based* subscriptions of entity listeners, allowing a client to subscribe to changes to all entities of a specific type, e.g. patients. Context clients and entity listeners can access and subscribe to several context services.

3.2 Context Service Layer

Figure 3.1b illustrates the details of a *Context Service*, which is a long-lived service process analog to a Web Service, for example. An *Entity* with its *Context* information is managed by the service's *Entity Container*. An entity is a small Java program that runs within the Context Service and responds to changes in its context. The life cycle of an entity is controlled by the container in which the entity has been added. The entity container handles subscribers to context events and notifies relevant clients on changes to entities. An entity, its context and its life cycle are further discussed in section 4.2.

The Entity components in a Context Service work together and with other components to accomplish their tasks. Hence they must have ways to access each other, and to access shared resources. This is the accomplished through the *Entity Environment*, which all Entities has a handle to when executing [1]. Besides access to general resources like initialization parameters and logging facilities, the Entity Environment provides methods for accessing:

- *Context Transformers*, which are small application-specific Java programs that a developer can write and add to the *Transformer Repository*. Currently the framework contains two type of transformers, namely *Translators*, which can translate between one kind of context information to another, and *Aggregators*, which can aggregate two types of context information into one. These can be put together in a chain of transformers to obtain the desired transformation. The Transformer Repository can be queried for appropiate transformers on runtime.

- *Key-Value Attributes*, which are application-specific Java objects accessible across entities via a key. This can be used to share resources such as special purpose objects existing only in the environment – e.g. a simple counter tracking the number of requests from various computers. It can also be used to access handles (RMI stubs or database connections) to remote resources – e.g. a handle to an Electronic Patient Records and its database.

[1] The Entity Environment is analog to the Web Context in a J2EE Application Server, where handles to databases, shared objects, and other resources are maintained across servlets.

Acces to a Context Service is controlled by a *Secure Context Service*, which ensures correct authentication of client requests. This component consists basically of two parts, namely an access control list, specifying what the requesting clients can access, and mechanisms for authenticating the client.

3.3 Context Monitor and Actuator Layer

There are two special kinds of context clients: the *Context Monitor* and the *Context Actuator*. A monitor is a client specially designed for acquiring context information in the environment by cooperating with some kind of sensor equipment, and associate it properly with an entity. A context actuator is a client designed to work together with one or more actuators to affect or 'change' the context.

The JCAF framework can handle the acquisition and transformation of context information in two modes. In the *asynchronous* mode monitors constantly deliver context information to one or more context services, which then can notify listeners or be queried. In the *synchronous mode*, the monitor is asked to sense context information. This is done when the context information for an entity is requested by a client. In this case, monitors associated with this context information are asked to refresh their context information. A user's current activity according to his calendar is an example where the activity monitor asks the calendar about the activity at the time of calling.

The interaction diagram in figure 3.2 illustrates the dynamics of this synchronous mode. First, a monitor registers itself at a context service by indicating what type of context information it can provide. When a client, who is an Entity Listener, is requesting context information by using the `getContext()` method, then relevant registered context monitors are called to acquire context information by calling their `getContextItem()` method. To avoid deadlocks (e.g. if the calendar system does not answer), the `getContext()` method starts a separate thread to handle monitors and returns immediately with whatever context information is available currently. When the Context Monitors starts reporting back (which might take some time), then clients are notified using the `contextChanged()` method in the `EntityListener` interface.

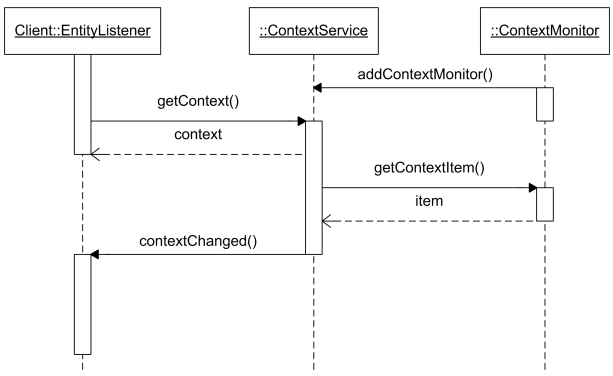

Figure 3.2: Interaction Diagram for asynchronous context acqucition using Context Monitors registered at the Context Service.

Similarly, Context actuators can register at a context service by specifying what type of context items it is an actuator for. When a context item is changed in the context service (i.e. the `contextChanged()` method is triggered), then all context actuators registered as interested in this type of context item are notified with information about this new context information. This can, for example, be used to keep context information synchronized across a distributed network of JCAF components and applications.

Chapter 4

The JCAF Programming Model

The JCAF programming model enables the programmer to create context-aware applications that are deployable in the JCAF infrastructure. The infrastructure both enables the programming model and makes use of it. The most important parts of the programming model is how to use the API of the context services, how to model context information for entities, and how to make use of the event-based infrastructure of JCAF.

4.1 The Context Service API

Figure 4.1 shows the UML diagram for the runtime infrastructure for context services.

The `ContextService` interface has methods for adding, removing, getting and setting entities. The `getEntity()` method returns the service's copy of an entity object, whereas the `lookupEntity()` method contacts other known services trying to locate the entity object. The `lookupEntity()` method takes as arguments the id of the entity to look for, the maximum number of allowed hops between services in the search, and an `DiscoveryListener` which is called when the entity is discovered. The method is non-blocking and relies on notifying the discovery listener if a matching entity is found.

Embedded in the context service's API are the APIs for the `TransformerRepository`, containing methods for adding and getting transformers, and the `ContextClientHandler` interface, containing methods for adding and authenticating a clients, including context monitors and actuators. The `EntityListenerHandler` interface contains methods for adding, removing, and accessing entity listeners (see section 4.3). The `EntityEnvironment` is shared by all entities in a service and has methods for setting and getting attributes, accessing information about the local context service, and accessing the transformer repository.

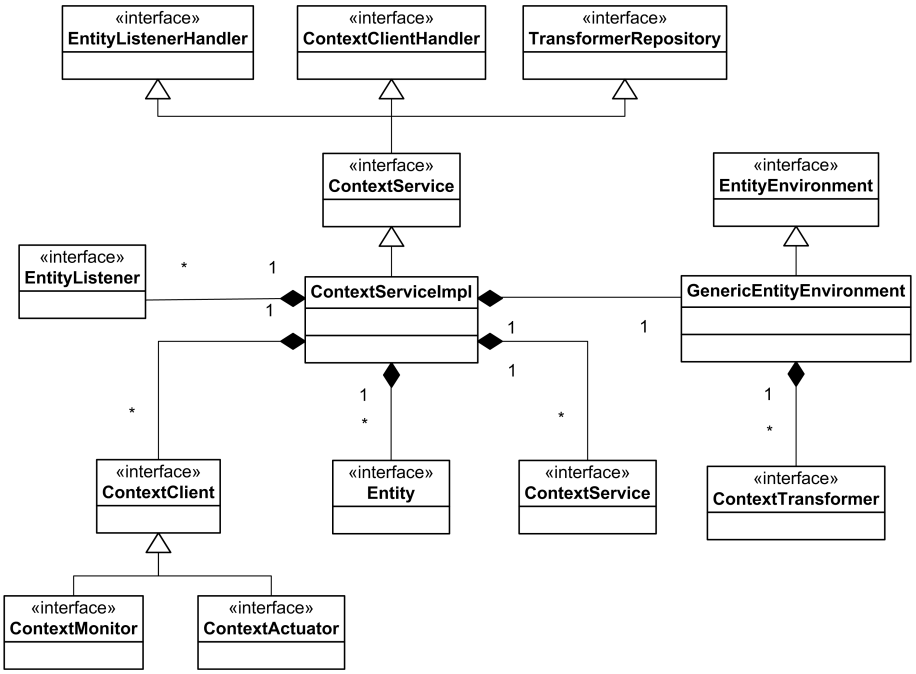

Figure 4.1: The UML diagagram for the Context Service runtime architecture.

4.2 Modelling Entity and Context

Context modeling in JCAF is done by making object-oriented models in Java. The core modeling interfaces provided by JCAF are the `Entity`, `Context`, `Relation`, and `ContextItem` interfaces. JCAF provides default implementations of these core interfaces. For example the `GenericEntity` class implements the `Entity` interface and can be used to create concrete entities using specialization. These are illustrated in figure 4.2.

Persons, places, things, patients, beds, pill containers, etc. are examples of entities. A Hospital Context and a Office Context, each knowing specific aspects about a hospital and an office, respectively, are examples of context. Physical location, activity as revealed by a user's calendar, and the status of an operation are examples of context items. Examples of relations are 'using' or 'located'. Hence, we can model that '$person_X$ is *located* in $room.333$' where $person_X$ is the Entity, *located* is the relation, and $room.333$ is the context item.

The JCAF framework can handle the acqucition and transformation of context information in two ways – synchronously and asynchronously. A context monitor can continously supply context information (i.e. items) to an entity. For example, a location monitor can update the location of an entity when it sees it. A client requesting the context information for an entity will received the latest location information. This is called *asynchronous context management*, because the client and the monitors (in general all clients) work independent of each other. The asynchronous mode is the prevalent mode in the JCAF framework. However, some context-aware applications might want to have up-

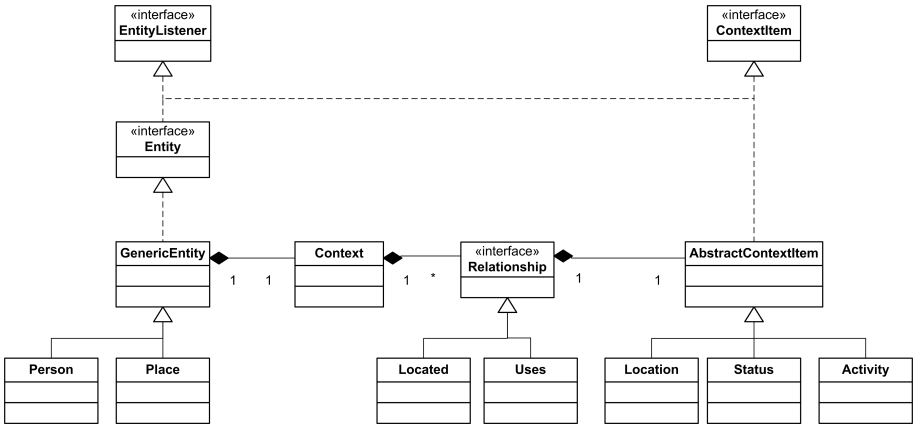

Figure 4.2: The UML model of an `Entity` with a `Context` containing a range of `ContextItem`s, each having a certain `Relationship` to the context.

to-date context information. Therefore, the JCAF framework also supports the *synchronous mode*, where a client requests the context for an entity, and the entity asks its context to refresh itself. A user's current activity according to his calendar is an example, where the activity monitor asks the calendar about the activity at the time of calling. The synchronous mode is clearly vunable for deadlocks – e.g. the client may newer return from a call if the calendar system does not answer. This mode should hence be used with caution.

The central processing part of an Entity is its `contextChanged()` method in the `EntityListener` interface. This method is garanteed to be called by the entity container whenever this entity's context is changed. This is a very powerfull way to implement functionality handling changes in the entity's context and thereby create some logic, which translates such changes into meaningful activities for users of the application. For example, we can make the TV react when a person is approaching it by adding the following code to its `contextChanged()` method:

```
public void contextChanged(ContextEvent event) {
  //TODO

}
```

The `contextChanged()` method is called if the TV's context is changed. This happens if the TV's own context is changed, or if the context of the `Person` changes. Hence, the `contextChanged()` method is called rescursively down the line of entities embedded in each other's context. In the example above ... TODO!

The `ContextItem` interface is shown below. It is important to be able to judge the quality of a context item [10]. For example, how accurate is the location estimate. The `getAccuracy()` method is used for this purpose. Implementations of a context items returns a probability between zero and one. The `isSecure()` method is used to establish whether this context information originates from a trusted and authenticated context monitor.

```
public interface ContextItem extends Serializable {
  public long getSequenceID();
  public boolean isSecure();
  public double getAccuracy();
  public boolean equals(ContextItem anotherItem);
}
```

A subtle, but rather important aspect of entities is that they themselves are context items. Hence, in JCAF it is possible to add an entity as a context item for another entity. For example, in a Bang and Olufsen Home entertainment project we needed to model that a person is using a certain A/V equipment, like a TV or Radio. In JCAF both persons as well as the A/V equipment were modeled as entities and it was hence easy to model that "$person_A$ was using TV_x" by adding TV_x to the context of $person_A$ with a *using* relation.

4.3 EntityListeners and ContextEvent

The event-based architecture of JCAF is supported by the `EntityListener` interface and the `ContextEvent` class in the programming model. By implementing the `EntityListener` interface a client can subscribe to changes in context for an entity. Entity listeners can subscribe to changes in a specific entity or can subscribe to changes in a specific type of entities. For example, an entity listener can listen to all person entities. Clients interested in listening to context changes can implement the `EntityListener` interface shown below.

```
public interface EntityListener {
   public void contextChanged(ContextEvent event);
}
```

Entities themselves are aware of changes to their context by implementing the `EntityListener` interface (see figure 4.2). The central processing part of an entity is hence its `contextChanged()` method. This method is guaranteed to be called by the entity container whenever this entity's context is changed. This is a very powerful way to implement functionality handling changes in the entity's context and thereby create logic, which translates such changes into meaningful activities for users of the application. The `ContextEvent` object is a standard `java.util.EventObject` that gives access to the entity and the context item, which caused the change. A `RemoteEntityListener` interface exists as well, enabling clients to listen on changes to Entities in a remote context service process. This remote entity listener interface is also used across context services, thereby enabling one context service to listen to changes on entities in another context service. In the example where a special 'operating context service' is deployed in a hospital, this context service would listen to changes concerning e.g. persons who are in the operating room. Hence, in the AWARE framework developed on top of JCAF (see section **??**), this operation context service would listen to changes to the context of the operating surgeon and may take appropriate actions, like revealing that he is busy operating or forward emergency calls only.

Acknowlegments

The Danish Center of Information Technology (CIT) and ISIS Katrinebjerg funded this research. Henrik Bærbak Christensen was much involved in the early discussion on contet-awareness in hospitals.

Bibliography

[1] Gregory D. Abowd. Software engineering issues for ubiquitous computing. In *Proceedings of the 21st international conference on Software engineering*, pages 75–84. IEEE Computer Society Press, 1999.

[2] Jakob E. Bardram. Applications of ContextAware Computing in Hospital Work – Examples and Design Principles. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 1574–1579. ACM Press, 2004.

[3] Jakob E. Bardram. From Desktop Task Management to Ubiquitous Activity-Based Computing. In Victor Kaptelinin and Mary Czerwinski, editors, *Integrated Digital Work Environments: Beyond the Desktop Metaphor*. MIT Press, 2005. To appear.

[4] Jakob E. Bardram. The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications. In Hans Gellersen, Roy Want, and Albrecht Schmidt, editors, *Proceedings of the 3rd International Conference on Pervasive Computing (Pervasive 2005)*, volume 3468 of *Lecture Notes in Computer Science*, pages 98–115, Munich, Germany, May 2005. Springer Verlag.

[5] Jakob E. Bardram and Thomas R. Hansen. The AWARE architecture: supporting context-mediated social awareness in mobile cooperation. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 192–201. ACM Press, 2004.

[6] L. Capra, W. Emmerich, and C. Mascolo. CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications. *IEEE Transactions on Software Engineering*, 29(10):921–945, October 2003.

[7] Anind Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16:97–166, 2001.

[8] Karen Henricksen and Jadwiga Indulska. A software engineering framework for context-aware pervasive computing. In *Proc. PerCom'04*. IEEE, 2004.

[9] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In Mahmoud Naghshineh and Friedemann Mattern, editors, *Proceedings of Pervasive 2002: Pervasive Computing : First International Conference*, volume 2414 of *Lecture Notes in Computer Science*, pages 167–180, Zrich, Switzerland, August 2002. Springer Verlag.

[10] Jeffrey Hightower, Barry Brumitt, and Gaetano Borriello. The location stack: A layered model for location in ubiquitous computing. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)*. IEEE Computer Society Press, 2002.

[11] Fritz Hohl, Lars Mehrmann, and Amen Hamdan. A context system for a mobile service platform. In H. Schmeck, T. Ungerer, and L. Wolf, editors, *Proceedings of ARCS 2002: Trends in Network and Pervasive Computing*, volume 2299 of *Lecture Notes in Computer Science*, pages 21–33, Karslruhe, Germany, March 2002. Springer Verlag.