The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications

Jakob E. Bardram

Centre for Pervasive Healthcare,
Department of Computer Science, University of Aarhus,
Aabogade 34, DK–8200 Aarhus N., Denmark
bardram@daimi.au.dk

Abstract. Context-awareness is a key concept in ubiquitous computing. But to avoid developing dedicated context-awareness sub-systems for specific application areas there is a need for more generic programming frameworks. Such frameworks can help the programmer develop and deploy context-aware applications faster. This paper describes the *Java Context-Awareness Framework* – JCAF, which is a Java-based context-awareness infrastructure and programming API for creating context-aware computer applications. The paper presents the design goals of JCAF, its runtime architecture, and its programming model. The paper presents some applications of using JCAF in three different applications and discusses lessons learned from using JCAF.

1 Introduction

The idea of *context-aware computing* was one of the early concepts introduced in some of the pioneering work on ubiquitous computing research [28, 27, 13] and has been subject to extensive research since. 'Context' refers to the physical and social situation in which computational devices are embedded. The goal of context-aware computing is to acquire and utilize information about this context of a device to provide services that are appropriate to the particular setting. For example, a cell phone will always vibrate and never ring in a concert, if it somehow has knowledge about its current location and the activity going on (i.e. the concert) [22].

This paper presents the *Java Context-Awareness Framework* – JCAF. The goal of JCAF is to provide a Java-based, lightweight framework with an expressive, compact and small set of interfaces. The purpose is to have a simple and robust framework, which programmers can extend to more specialized context-awareness support in the creation of context-aware applications. Several projects have already been undertaken using JCAF, as discussed in section 6.

The paper starts by motivating JCAF and introduces the main design principles. Section 3 presents the JCAF runtime infrastructure and section 4 presents the programming model. Section 5 discusses the current implementation status of JCAF and the ongoing work based on the lessons learned so far. Section 6 presents how JCAF has been used and evaluated and presents two specific projects, discussing in detail how JCAF was

H.W. Gellersen et al. (Eds.): PERVASIVE 2005, LNCS 3468, pp. 98–115, 2005.

used and conceived by programmers using JCAF. Section 7 discusses related work and section 8 concludes the paper.

2 Motivation and Design

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 [11, 17, 15, 14, 16, 1, 8, 3].

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 [3, 4]. In section 7 we shall discuss in more details how JCAF differs from the other related middleware support for context-aware applications.

2.1 Federated Context Services

The infrastructure of JCAF relies on having a set of distributed context services that cooperate in a loosely coupled peer-to-peer fashion. A context service is often dedicated to a specific purpose. For example, a context service might run in an operating room, handling specific context information in this setting, like who is there, what are they doing, who is the patient, and what is the status of the operation. This context service cooperates with context services in other parts of a hospital. Most context management is specific for the operating theatre, but occasionally it might become relevant to contact services running in the rest of the hospital. Therefore, the JCAF infrastructure consists of a network of loosely coupled context services, which cooperate in a peer-to-peer or hierarchical fashion. The exact topology of the context services are designed to fit the specific deployment of JCAF in a certain application.

2.2 Modifiable, Event-Based and Secure Architecture

A core software architecture quality [6] of JCAF is *modifiability*, i.e. support for adding, deleting, modifying, or varying functionality, capacity, or platform. The JCAF framework is designed to be highly modifiable and extensible at runtime – not at design and compile time as many other frameworks are. JCAF services, monitors, actuators, and clients can be added to the JCAF runtime infrastructure while running. The design principle of deploying a federated set of cooperating context services enables users of the framework to add special designed context services and register them in the infrastructure. For example, in a hospital where a set of JCAF context services may run as the

core context-awareness infrastructure, a new context service responsible for context-aware application in the emergency department can be deployed at a later stage. This specialized 'emergency context service' serves context-aware application in the emergency department while cooperating and exchanging context information with the rest of the context services running in the hospital.

Many context-aware applications may only be interested in being notified about changes of context. Therefore, JCAF is based on an *event-based* infrastructure [9, 12], thereby ensuring a decoupling in space, time, and thread synchronization. The context service in JCAF has a publish-subscribe-notify interface notifying subscribers about changes in context. For example, a context-aware application showing relevant medical images during surgery would subscribe to changes in the context of an operating theatre. This application would be notified on the enterance of the patient and the surgeon, and is able to display appropiate images for the patient tailored to the preferences of the surgeon.

Context data, used e.g. in a medical setting, should be protected, subject to access control, and not revealed to unauthorized clients [7, 21]. Furthermore, establishing the credibility and origin of context information is key for some type of context-aware applications. Such cases may require an authentication mechanism for clients, and even a secure communication link between clients and services. However, in line with [20] we argue for supporting *adequate security* in a ubicomp environment. Hence, eavesdropping sensor information like temperature and location is seldom a major security issue – often it is easier to measure the temperature than listening in on low-power radio communication. JCAF supports a minimal set of default set of security mechanisms for access control and authentication. Additional support for security, authentication, access control, and encryption can be added to the JCAF framework by using the Java Security API.

2.3 Minimal Java API

The main goal of the programming model of JCAF is to provide the programmer with a framework that is extensible in a way that it helps him or her implement application-specific functionality by extending the JCAF framework. Hence, in the design of JCAF we have put special emphasis on providing only a minimal set of interfaces and classes that provides generic support for context modeling and handling while ensuring that these interfaces and classes (constructs of the programming model) are as expressive as possible. As we shall discuss in section 6 the constructs of the JCAF programming model have evolved over long period of time and incorporated experience in developing several types of context-aware applications for different settings.

Furthermore, applications are concerned with the quality of context information, including uncertainty [16, 24]. Therefore, the JCAF programming model encourage the programmer of context-aware applications to consider saving, revealing, and using quality measures for context information. Hence, the JCAF interface requires the programmer to implement methods on context information quality. For example, a clinical application trying to find relevant patient data during an operation might suggest to show more than one piece of medical data, if the sensing uncertainty is too high. Quality measures for context information are preserved from measurement, through any transformation to its use by applications.

3 The JCAF Runtime Infrastructure

The JCAF Runtime Infrastructure is illustrated in figure 1. Figure 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 1 are remote and hence all components in JCAF can be distributed in a network.

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.

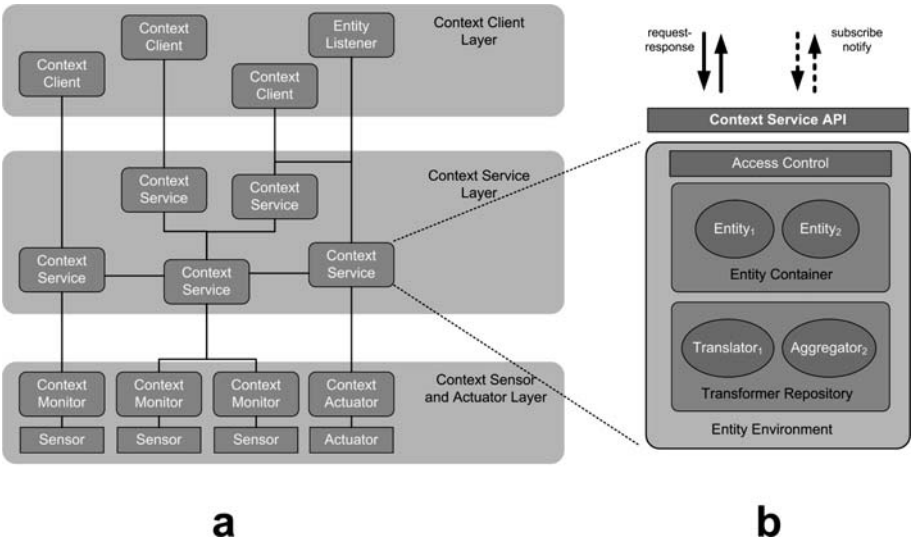

Fig. 1. The Runtime Architecture of the JCAF Framework. a – An example of a deployment situation of a set of context monitors, context actuators, and a set of cooperating context services. b – Details of a context service

3.2 Context Service Layer

Figure 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 with other components to accomplish their tasks. This is accomplished via the *Entity Environment*, which all Entities have a handle to when executing. The Entity Environment provides access to general resources like initialization parameters and logging facilities, and to user-specific resources, like databases, RMI stubs, shared objects, and other resources which are maintained across entities. Furthermore, the Entity Environment holds *Context Transformers*, which are small application-specific Java programs that a developer can write and add to the *Transformer Repository*. The Transformer Repository can be queried for appropriate transformers on runtime.

Access to a Context Service is controlled through the *Access Control* component, 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 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

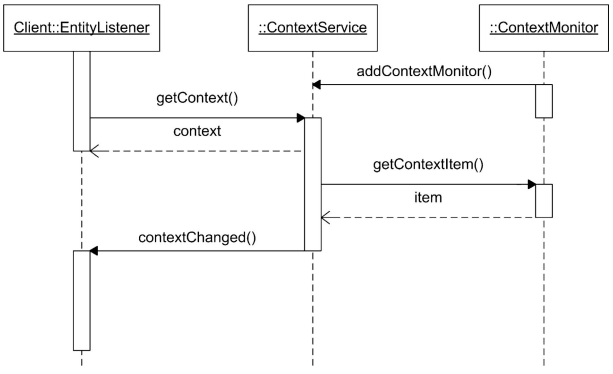

Fig. 2. Interaction Diagram for asynchronous context acqucition using Context Monitors registered at the Context Service

are notified using the `contextChanged()` method in the `EntityListener` interface.

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.

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

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 trans-

formers, 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.

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 3.

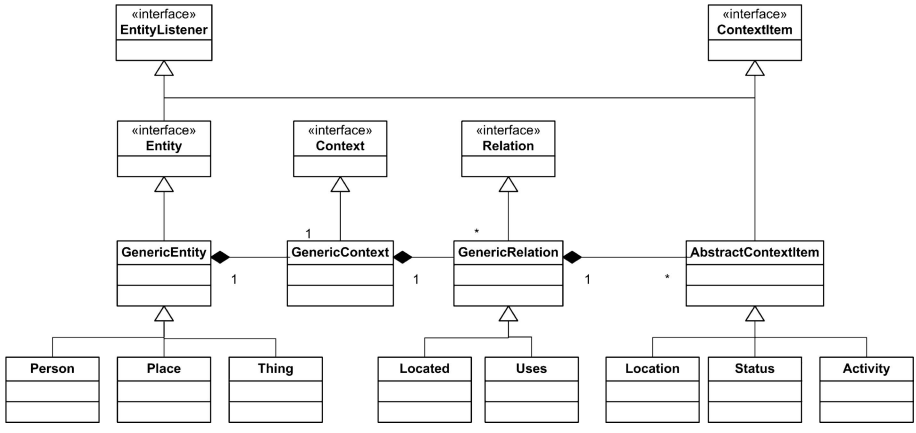

Fig. 3. The UML model of an `Entity` with a `Context` containing a range of `ContextItems`, each having a certain `Relation` to the context. Note that the an entity is also a context item

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 `ContextItem` interface is shown below. It is important to be able to judge the quality of a context item [16]. 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 3). 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 6.1), 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.

5 Implementation and Ongoing Work

JCAF is currently in a version 1.5 and is implemented using J2SE 1.4. The core functionality of JCAF as described above is implemented and working. Remote communication is currently implemented using Java RMI. A Context Service is looked up using the Java RMI Registry and accessed using RMI invocation. The lookup of entities in associated context services (using the `lookupEntity()` method) is also done using RMI. A configuration file contains information about known peers. Hence, there is no automatic discovery of other context services.

Security is implemented using an authentication mechanism based on a digital signature using the Java Authentication and Authorization Service (JAAS), which is a part of J2SE. This is currently used for context clients (including context monitors and actuators) and the authentication mechanism is part of the `ContextClientHandler` interface. Context information from authenticated monitors is labeled 'secure'. This security mechanism could be extended to include other types of context clients, like entity listeners and transformers added to the JCAF while running. Finally, security might be enhanced by using encrypted communication between a context service and a client, especially if sensitive (medical) data is transmitted. However, as discussed in section 2 we are very cautious about providing 'adequate security' and we are not sure if these latter security mechanisms are necessary. As for access control, a simple role-based access control mechanism is used currently: monitors can add context items (secure monitors can add secure items), and clients can query context information. From a privacy perspective, this access control mechanism is clearly a coarse-grained mechanism and we plan to extend it to real access control lists, which have a fine-grained specification of the rights of each client.

The projects that have been using JCAF have implemented a range of monitors for monitoring location based on RFID, WLAN, Bluetooth, and IrDA. Furthermore, monitors for monitoring activity in an online calendar and status information in an Instance Messaging system have been implemented. JCAF also contains several implementations of common entities (person, place, thing) and context items (location, status, activity, network capacity) as well as generic implementations of context clients and monitors.

6 Evaluation

A central research question is how to evaluate a programming framework. Our approach has been to use the JCAF framework in different situations – in different types of projects, including students and research projects, and in different types of application areas and for different types of applications within each area. Table 1 contains an overview of these projects. In this section we will discuss how JCAF was used in two of these projects: Proximity-Based User Authentication and the AWARE Framework. Both of these projects highlight different parts of the JCAF framework[1]. Furthermore,

[1] Each of these projects is motivated in our work on developing pervasive computer technology for the hospital of the future [2]. The design and evaluation of this technology have been done in cooperation with a range of clinicians, applying user-centered design methods like observations, design workshops, and prototyping.

Table 1. The use of JCAF in different projects, ranging from research projects (R) to students projects (S) in class

Project Title	Type	Description
Proximity-Based User Authentication	R	Enables a user to log in to a computer by physically approaching it.
Context-Aware Hospital Bed	R	A hospital bed that adjust itself and react according to entities in its physical environment, like patient, medicine, and medical equipment.
Bang & Olufsen AV Home	S	Using context-awareness to make B&O AV appliances adjust themselves according to the location of people and things.
AWARE Framework	R	A system that distributes context information about users, thereby facilitating a social, peripheral awareness, which helps users coordinate their cooperation.
Wearable Computers for Emergency Personnel	S	A wearable system for emergency workers, like ambulance personnel. Helps them react to changes in the work context.

in the end of this section we will discuss the feedback from programmers who have been using JCAF.

6.1 The AWARE Framework for Social Awareness

When people need to engage in a cooperative effort there is a risk of interrupting each other. For example, when calling people using a mobile phone or accessing them directly in their offices. People hence often tries to maintain a 'social awareness' of each other in order to align their cooperation to the work context of their colleagues. This social awareness relies on having access to the work context and when people are not co-located this access can be mediated using networked computers (including very small portable ones). The main purpose of the AWARE platform is to provide such social awareness by notifying and informing users about the working context of their fellow colleagues [4]. For this purpose JCAF is used to monitor the context of people.

Figure 4 shows the deployment of services, including context services, in the AWARE architecture. It illustrates the federation of JCAF context services into special-purpose context services. The `AwareContextService` in the AWARE architecture is responsible for managing context information for the users of the AWARE system. By using the lookup method and registrering as a (remote) entity listener, this `AwareContextService` replicates context information for users residing in other context services and maintains context information that is specific to the AWARE architecture. In our current implementation this includes location, status, and calendar information. In a typical deployment (as illustrated in figure 4), status monitors and calendar monitors add status and calendar context information directly to the `AwareContextService`, whereas location information is available in other context services in the network of federated JCAF context services.

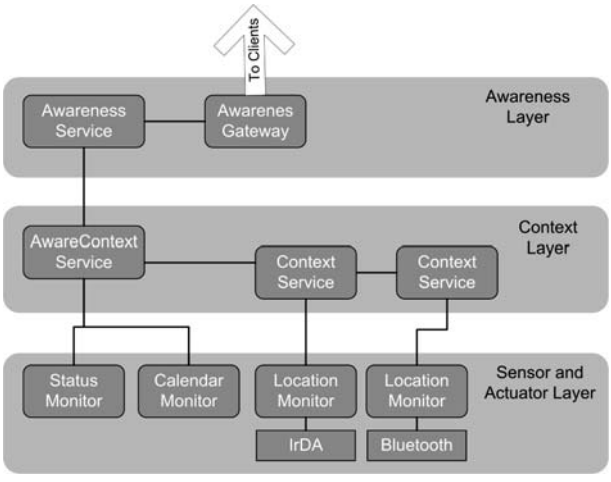

Fig. 4. The services in the AWARE Architecture, including the `AwareContextService` responsible for managing context information by cooperating with other Context Services

We have built two types of AWARE clients – a simple browser interface where a user can see context information about his fellow colleagues, and the AWAREPhone, which is a mobile phone client. The AWAREPhone implements a location monitor, using its built-in Bluetooth capabilities. Via the list of contact persons, a user can see the working context of a colleague and based on this information choose an appropriate cooperation strategy, like calling, sending a message, or not to disturb.

6.2 Proximity-Based User Authentication

Proximity-based user authentication [5] is a mechanism that allows users to log in to a computer just by approaching it and start using it. The system consists of two independent mechanisms. The first mechanism is a personal token with enough processing power to do public key cryptography. This token can be some jewelry (e.g. a ring, necklace, or earring), or it can be a personal pen used on the various touch screen embedded in a hospital. Currently we are using Java Smartcard technology as the personal token. When the user approaches a computer, this token can authenticate the user using public key cryptography. This is however not secure enough for use in hospitals – this token might be lost or stolen. Hence, when using e.g. smartcards in hospitals today, users are also required to enter a password or a PIN code. To avoid this, the second mechanism in our setup is to track the user's location via the context-awareness infrastructure. If the infrastructure can verify the location of the user in the same place as the token (and hence the computer) s/he is authorized. The location of the user can apply various methods based on e.g. something the user wear or trying to recognize the voice. Currently we monitor RFID tags woven into the clinicians whitecoats (see [5] for details).

In this application of the JCAF framework, two aspects become important. The first one is the security of the framework. If the context-awareness framework is used to verify the location of the users, it is of crucial importance that any adversary trying to gain illegal access cannot send a false "I'm here" message to the systems. Hence, we need to trust and hence authenticate the context monitors reporting on the location of users. A context monitor is authenticated to a context service using a public-key infrastructure. An example of a secure context monitor is shown below:

```
public class SecureLocationMonitor extends AbstractContextClient {

 SecureContextService scs

 public SecureLocationMonitor() {
  super();

  try {
   PrivateKey key = ... // holds this client's private key
   byte[] data = this.getClass().getName().getBytes();
   Signature sig = Signature.getInstance("DSA");
   sig.initSign(key);
   sig.update(data);
   byte[] signature = sig.sign();

   // tries to authenticate at the server.
   scs = getContextService().authenticate(this.getClass().getName(), data, signature);

   // If successful, then a secure context service is returned.
   if (scs!= null) {
    // Now use this secure service to provide some location information
    scs.setContextItem("1732745-3872", new Location("loc://daimi.au.dk/hopper.333"));
   }
  } catch (Exception e) {...}
 }
}
```

The only way to access a secure context service is through the authenticate method. When the `setContextItem()` method on the secure service is used, the context item is marked as secure. Hence, a client using this context information can ask if this item is secure by using the `isSecure()` method on the `ContextItem` interface.

The second aspect concerns the quality of the context data. It is of equal importance that the user authentication mechanism can judge the quality of the location data and decide whether the quality is sufficient to trust as a verification of the location of the user. Hence, the aggregation of quality (or uncertainty) measures is important in this application of context-aware computing, and thus relies on the `getAccuracy()` methods of a Context Item. In our current implementation of the `Location` context item, accuracy decreases by 1% pr. minute since the last measurement. In the User Authentication protocol there is a threshold, which determines how accurate the location estimation needs to be to verify the user.

6.3 Discussion

After each project we interviewed programmers about their thoughts on using JCAF. These interviews were often informal and were typically done while going through the code of their applications to see how they were using JCAF. Some of the things that

was mostly appreciated in JCAF includes its event-based architecture, its modifiability, and its Java-based programming model.

The event-based architecture made programmers develop context-aware applications that were very loosely coupled and were reactive to changes in context. This style of application design and implementation was conceived by all programmers as a central benefit from JCAF.

The extensibility of the JCAF infrastructure and programming model was also highly appreciated by all programmers. The support for adding and removing context services, monitors, and actuators in a running infrastructure was used extensively. Often the event mechanism in JCAF was used to develop and deploy new modules in the infrastructure. For example, in the Bang and Olufsen project a 'Context-Aware Triggering' component were added, which were notified about changes in the environment and could trigger certain actions. For example, disallow children to watch PG-rated 16 movies on a B&O television. In another project, a history module was added. This module was listening in on relevant entities and a history update were triggered when an entity changed. This way of implementing new components as plug-ins to the existing infrastructure worked out well. In addition, such new components were working asynchronously with the rest of the framework by having the history or other modules run in their own threads (potentially on separate host machines), thereby not adding significant response latency to the rest of JCAF.

As for extending the programming model, all the projects implemented new types of entities, context, relations, and context items, which was a simple task for most programmers. Also new kinds of transformers were made and deployed. For example, filter transformers in the B&O project.

The Java programming model was also appreciated by most programmers. It was seen as simple and it was easy to start using. Also the programming model of JCAF conforms to the programming models of many other Java frameworks and experienced Java programmers hence find JCAF easy to understand and use. The Java programming model also helps integrate JCAF in other Java frameworks, like the Java Authentication and Authorization Service (JAAS) or Jini.

Some of the things that the programmers found problematic concern the deployment of context services in a network topology. To establish how to divide the infrastructure into different cooperating services was not always trivial. Furthermore, problems arose when an entity, like a person, had a representation and context information, which were distributed across several services. This introduced synchronization problems, which could be solved by having the distributed version of the entity in the different services subscribed to changes on each other. However, this still introduced some headache. A related problem was that information relevant for an entity – like a patients name and id number – often was in other systems, like a hospital information system. Hence, synchronizing such data was also problematic. This is, however, a more general software engineering challenges in many distributed systems and is hence not specific to JCAF.

Some programmers also found the modeling of context information in Java to be an overhead compared to just putting this information in a relational database and use SQL queries and triggers. Relational databases are clearly inherently good at handling

relational data on the type of entity-relationships. However, many programmers also appriciated the pure-Java approach in JCAF and we have decided to stay within this model. We are, however, looking into how to address some of these challenges in the modeling capabilities of JCAF.

7 Related Work

As discussed in section 2, JCAF tries to incorporate the contributions from a wide range of related work within software frameworks for context-awareness. We shall hence concentrate on work specifically related to the core design principles in JCAF as described in section 2.

Creating support for context-awareness by having one server or infrastructure component is common in many context-awareness systems, like Schilit's mobile application customization system [25], the Contextual Information Service (CIS) [23], the Trivial Context System (TCoS) [17], and the Secure Context Service (SCS) [7, 21]. All of these act as the middleware that acquires raw contextual information from sensors and provides interpreted context to applications via a standard API. They can also monitor the context changes and send events to interested applications. This approach is often the simplest and most efficient way to ensure data integrity, aggregation, filtering, enrichment, etc. From a technical point of view, the use of one central server has its drawbacks in terms of having a single point of failure, scalability issues, and extensibility concerns. From a functional point of view, the collection of everything in one place makes a context-aware infrastructure hard to maintain while it grows in size and complexity. There is no way to separate responsibilities and concerns.

More distributed architectures have been proposed. For example, the Context Toolkit [11] has a range of loosely coupled distributed components and the architecture proposed by Spreitzer and Theimer [26] is based on multicasting context information to all members of a domain's multicast group. The Context Toolkit is distributed on a very low level of details and there is no way of collecting related context data and services into logical bundles. The disadvantage of multi-casting context information around is increasing computation and communication thereby paying a scalability penalty.

The federating context services approach in JCAF in a hybrid between having one server or a totally distributed infrastructure. This design principle is based on our different application areas. One the one side, the JCAF framework enables us to have a context-awareness infrastructure deploying in an organization (e.g. a hospital) where applications can discover and utilize this infrastructure when needed. On the other hand, the JCAF framework enables us to partition our infrastructure into separate but cooperating context services, each responsible for acquiring, handling, different kind of context information often in different localized settings. Hence, there is support for separation of concerns.

Many context-awareness systems incorporate some notification functionality. Infrastructures based on relational databases (e.g. [14, 17]) often use the triggering mechanisms in such databases and e.g. the Context Toolkit [10] supports subscriptions to state changes in a Context Widget. These approaches require their own specification language (e.g. in XML) to specify a subscription and only support subscribing for changes

in low-level context information represented as native types are supported. In contrast to these approaches where subscriptions, notifications, and events are represented outside the programming environment, the event mechanism in JCAF is a part of the programming model. Hence, Java APIs for subscription and events exist and can be extended. The Rome system developed at Stanford [18] is based on the concept of a context trigger, much like context events in JCAF. However, Rome's decentralized evaluation of triggers embedded in devices does not allow context sharing and requires the device to have the capability to sense and process all of the necessary raw contextual information. In JCAF entities residing in a context service are notified on context events and can access each other locally and look up remotely located entities, without involving any clients. Hence, the JCAF event structure is not only used for client notification but also for triggering actions in the entities residing in the context services' entity container.

Despite the importance of security and privacy in ubiquitous computing [20] little work have been done here, with the Secure Context Service (SCS) [7, 21] as a notable exception. However, SCS is based on a Role-Based Access Control (RBAC) mechanism and is a closed system where the identity of all clients must be known to the system a priori. JCAF, on the other hand, supports a more relaxed security strategy where unknown context clients can access and provide context information, but this context information is labeled insecure. This strategy is more aligned with the basic purpose of JCAF, i.e. to provide the basic building blocks for experimenting with context-awareness.

Even though Java has been used as a programming language in many context-awareness systems, there is to our knowledge no Java Framework or API available for context-awareness. As a toolkit to be programmed in Java, the Context Toolkit [11] is what come closest. However, in the Context Toolkit, Java's basic abstractions for TCP/IP networking and hashtables of string-based context information is used. There is no object-oriented modeling of context information, nor any use of Java serialization of complex context data or the use of Java RMI. JCAF is an attempt to suggest a Java API for context-awareness, analogue to the APIs for e.g. service discovery in JINI. Some may argue that the use of Java in itself is not a virtue of context-awareness infrastructure that has to exist in a heterogeneous execution and application environment. However, we would argue – as also supported by our users (i.e. the programmers using JCAF) – that a pure Java-based framework is valuable because there is no need for dealing with special context modeling or markup languages.

8 Conclusion

Runtime infrastructures and programming models for creating context-aware applications and services are central in pervasive computing. This paper has presented the Java Context-Awareness Framework (JCAF), including its core design principles, its runtime infrastructure, and its programming model. JCAF has been used and hence evaluated in several research and student projects and we presented in some detail how JCAF was used in two projects in our research on creating ubiquitous computing support in a hospital setting. We discussed our experience from these two projects and more general experience from interviewing programmer, who have been using JCAF in different

projects. When looking at related work, JCAF shares similarities with much of the research already done within creating generic support for the creation of context-aware applications. Distinct features of JCAF are, however, its support for distributed cooperating context services, its event-based middleware architecture, its support for a relaxed security model for authenticating context clients, and its Java-based modeling of context information.

The core feature of a framework is its extensibility and support for being used in several applications areas [19]. In our design and use of JCAF we have demonstrated that it is extensible both with respect to its runtime infrastructure and to its programming model. Hence, we believe that JCAF provides a comprehensive set of Java APIs and generic implementations which allow researchers, students, and programmers to start extending the framework and begin experimenting with context-awareness as a concept and as a technology. More information on JCAF, including a downloadable release, is available at `http://www.daimi.au.dk/~bardram/jcaf`.

Acknowledgments

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

References

1. G. 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. J. E. Bardram. Hospitals of the Future – Ubiquitous Computing support for Medical Work in Hospitals. In J. E. Bardram, I. Korhonen, A. Mihailidis, and D. Wan, editors, *UbiHealth 2003: The 2nd International Workshop on Ubiquitous Computing for Pervasive Healthcare Applications*. http://www.pervasivehealthcare.dk/ubicomp2003, Seattle, WA, USA, Oct. 2003.
3. J. 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.
4. J. E. Bardram and T. 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.
5. J. E. Bardram, R. E. Kjær, and M. Ø. Pedersen. Context-Aware User Authentication – Supporting Proximity-Based Login in Pervasive Computing. In A. Dey, J. McCarthy, and A. Schmidt, editors, *Proceedings of UbiComp 2003*, volume 2864 of *Lecture Notes in Computer Science*, pages 107–123, Seattle, Washington, USA, Oct. 2003. Springer Verlag.
6. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, second edition, 2003.
7. C. Bisdikian, J. Christensen, J. Davis, II, M. R. Ebling, G. Hunt, W. Jerome, H. Lei, S. Maes, and D. Sow. Enabling location-based applications. In *Proceedings of the 1st international workshop on Mobile commerce*, pages 38–42. ACM Press, 2001.

8. 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, Oct. 2003.
9. G. Cugola, E. D. Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *Proceedings of the 20th international conference on Software engineering*, pages 261–270. IEEE Computer Society, 1998.
10. A. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Department of Computer Science, Georgia Institute of Technology, USA, 2000.
11. A. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16:97–166, 2001.
12. P. T. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.
13. A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. *Wireless Networks*, 8(2/3):187–197, 2002.
14. K. Henricksen and J. Indulska. A software engineering framework for context-aware pervasive computing. In *Proc. PerCom'04*. IEEE, 2004.
15. K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. In M. Naghshineh and F. Mattern, editors, *Proceedings of Pervasive 2002: Pervasive Computing : First International Conference*, volume 2414 of *Lecture Notes in Computer Science*, pages 167–180, Zürich, Switzerland, Aug. 2002. Springer Verlag.
16. J. Hightower, B. Brumitt, and G. 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.
17. F. Hohl, L. Mehrmann, and A. 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, Mar. 2002. Springer Verlag.
18. A. C. Huang, B. C. Ling, S. Ponnekanti, and A. Fox. Pervasive computing: What is it good for? In *In Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 84–91. ACM Press, Aug. 1999.
19. R. Johnson. Documenting frameworks using patterns. In *OOPSLA ´92*, pages 63–76, Vancouver, Canada, 1992. ACM.
20. M. Langheinrich. Privacy by Design – Principles of Privacy-Aware Ubiquitous Systems. In G. D. Abowd, B. Brumitt, and S. Shafer, editors, *Proceedings of Ubicomp 2001: Ubiquitous Computing*, volume 2201 of *Lecture Notes in Computer Science*, pages 273–291, Atlanta, Georgia, USA, Sept. 2001. Springer Verlag.
21. H. Lei, D. M. Sow, I. John S. Davis, G. Banavar, and M. R. Ebling. The design and applications of a context service. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):45–55, 2002.
22. T. Moran and P. Dourish. Introduction to this speical issue on context-aware computing. *Human-Computer Interaction*, 16:87–95, 2001.
23. J. Pascoe. Adding generic contextual capabilities to wearable computers. In *In Proceedings of the Second International Symposium on Wearable Computers*, pages 129–138. IEEE Computer Society Press, Oct. 1998.
24. M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, Oct. 2002.
25. B. N. Schilit, M. M. Theimer, and B. B. Welch. Customizing mobile applications. In *Proceedings of USENIX Mobile and Location-Independent Computing Symposium*, pages 129–138. USENIX Association, Aug. 1993.

26. M. Spreitzer and M. Theimer. Providing location information in a ubiquitous computing environment (panel session). In *Proceedings of the fourteenth ACM symposium on Operating systems principles*, pages 270–283. ACM Press, 1993.
27. R. Want, B. N. Schilit, N. I. Adams, R. Gold, K. Petersen, D. Goldberg, J. R. Ellis, and M. Weiser. An overview of the parctab ubiquitous computing environment. *IEEE Personal Communications*, 2(6):28–43, 1995.
28. M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):66–75, September 1991.