UNIVERSITY OF MAURITIUS RESEARCH JOURNAL – Volume 16 – 2010
University of Mauritius, Réduit, Mauritius

Research Week 2008

C-Arc: A Novel Architecture for Next Generation Context-Aware Systems

Anuja Meetoo*

Kavi Kumar Khedo

Faculty of Engineering, University of Mauritius
anuja.meetoo@uom.ac.mu, k.khedo@uom.ac.mu

Abstract

Computing is becoming increasingly mobile and ubiquitous. This implies that applications and services must be aware and adapt to highly dynamic environments. However, building context-aware mobile services is currently a complex and time consuming task. The emergence of truly ubiquitous computing, enabled by the availability of mobile and heterogeneous devices and an increasing number of commercial off-the-shelf sensing technologies, is hampered by the lack of standard architectural support for the development of context-aware systems. In this paper, the common architecture principles of context-aware systems are presented and the crucial context-aware architecture issues to support the next generation context-aware systems which will enable seamless service provisioning in heterogeneous, dynamically varying computing and communication environments are identified and discussed. Furthermore, a novel architecture, C-Arc, is proposed to aid in the development of the next generation context-aware systems. A prototype implemented of C-Arc is also presented to demonstrate the architecture. C-Arc provides support for most of the tasks involved in dealing with context, namely acquiring context from various sources, interpreting context and disseminating context.

Keywords: Context-aware architecture, context-aware systems, context-aware mobile services, mobile and ubiquitous computing.

* to whom correspondence should be addressed

1. INTRODUCTION

Context-aware systems offer entirely new opportunities for application developers and end users by gathering context data and adapting systems behaviors accordingly (Dey & Abowd, 1999). Especially in combination with mobile devices, such mechanisms are particularly of high value and claim to increase usability tremendously. Context-aware systems can adapt their operations to the current context without explicit user intervention and thus aim at increasing usability and effectiveness by taking the environmental context into account. Many researchers have made claims about the potential benefits of context-sensitive mobile computing since its first appearance more than a decade ago (Want *et al.*, 1992; Abowd *et al.*, 1997; Schilit *et al.*, 1994). Such applications however remain difficult to develop and deploy due to a lack of architectural support. Programmers are often required to write large amounts of code and interact with sensor and actuator devices at a low level in order to develop relatively simple applications.

In order to achieve context awareness for mobile computing, it is redundant and inefficient for mobile applications to maintain the required contexts independently. It is also not appropriate for application developers to check the statuses of all related contexts. Instead, a more generalized and abstracted description of the current context would be adequate for a mobile application to adapt effectively. Another problem is how mobile applications utilize the underlying system service to adapt to the current context. This is a challenging task for application developers since mobile applications have to implement their own adaptation action down to the system level. In the light of all the problems faced by context aware computing, a standard architecture is needed to support future applications that will enable seamless service provisioning in heterogeneous, dynamically varying computing and communication environments (Dey, 2000). In this paper, a novel context-aware architecture is proposed that will ease the development of context-aware systems.

Section two presents the motivation for an infrastructure approach to context-aware computing. Section three gives an overview of some recent architectures developed for context-aware systems. Section four discusses the challenges for implementing such architectures and gives some research directions for novel context-aware architectures. Section five presents C-Arc, the proposed novel architecture for next generation context-aware systems. Section six gives an evaluation of C-Arc through a prototype implementation. Section seven evaluates the C-Arc architecture and section eight concludes the paper.

2. AN INFRASTRUCTURE APPROACH TO CONTEXT-AWARE COMPUTING

It is observed that the sheer diversity of exploitable context and the plethora of sensing technologies are actually working against the deployment of context-aware systems (Chen & Finin, 2003). It is necessary to decouple the application and the actual context sensing part because of the large development overhead involved in interacting with a variety of sensors to capture context, interpreting it to the desired format and using it in a meaningful way. To separate the low-level sensor data processing from high-level applications, it is necessary to introduce a middleware layer whose functionalities are to collect raw sensor information, translate it to an application-understandable format and disseminate it to interested applications (Gu *et al.*, 2004).

In the earlier stage of research in this field, many researchers focused on building application-specific context-aware systems such as the Active Badge (Want *et al.*, 1992) project which provided the phone redirection service based on the location of a person in an office and the Cyberguide (Abowd *et al.*, 1997) project which provided a context-aware tour guide to visitors. In these systems it is difficult to obtain and process context information due to the "ad-hoc" approach they employ. Furthermore, they may be dependent on the underlying hardware and operating system. Some researchers take a framework-based approach (Biegel & Cahill, 2004) to provide basic structures and reusable components for common functionalities, and hence to enable easy creation of context-aware applications. The ParcTab (Shilit, 1995) system was the earliest attempt made for a general context-aware framework. By using an object-oriented approach, the Context Toolkit (Dey *et al.*, 1999) provides a framework and a number of reusable components to support rapid prototyping of sensor-based context-aware applications. However, these systems do not provide a common context model to enable context knowledge sharing and context reasoning.

Recent research works focus on providing architectural support for context-aware systems. The advantage of using a standard architecture for development of context-aware systems has been pointed out by Hong and Landay (2001). In the Context Fabric architecture, Hong and Landay (2001) took a database-oriented approach to provide context abstraction by defining a Context Specification Language and a set of core services. In the Context Broker Architecture (CoBrA) (Chen and Finin, 2003) project, Chen proposed an agent-oriented architecture for context representation, context sharing and user's privacy control.

The development of future context-aware applications will require tools that are based on clearly defined models of context (Dey *et al.*, 2005) and software architecture. So, a standard architecture that supports the gathering of context information from different sensors and the delivery of appropriate context information to applications is needed. In order to greatly simplify the tasks of creating and maintaining context-aware systems, as much of the weight of context-aware computing as possible must be shifted onto network-accessible middleware architectures (Musolesi, 2004). By providing uniform abstractions and reliable services for common operations, service architectures can make it easier to develop robust applications even on a diverse and constantly changing set of devices and sensors. A service architecture would also make it easier to incrementally deploy new sensors, new devices and new services as they appear in the future, as well as scale these up to serve large numbers of people. Lastly, a service architecture would make it easier for sensors and devices to share sensor and context data, placing the burden of acquisition, processing and interoperability on the infrastructure rather than on individual devices and applications (Hong & Landay, 2001).

3. RELATED WORKS

The main research focus in context-aware computing is on context-aware and reconfigurable architectures. Context-awareness specifies discovery and processing of variable environment conditions in software systems. It can be used to automatically recognize user requirements from the application situation and accordingly adapt functionality and interaction patterns of the system (Anagnostopoulos *et al.*, 2004). One difficulty faced by context-aware application designers is the lack of generic infrastructure for developing context-aware applications. In addition, existing applications have used a limited range of context types. They have also been limited in context-aware features. A number of architectures that facilitate the building of context-aware services have been developed. Several researchers have proposed and prototyped general architectures to support context-aware applications. Schilit's infrastructure for ubiquitous computing (Shilit, 1995) is probably the earliest attempt in this direction.

Dey's Context Toolkit (Dey *et al.*, 1999) aims at facilitating the development of context-aware applications by offering a small set of generic 'base' classes from which an application developer can derive application specific classes. These classes are organized around high level context-aware application functions, namely collecting sensor data, combining data from multiple sensors and translating sensor data into alternate formats. Context Toolkit delivers a standardized way of implementing the syntactic part of context-aware systems. However, the reasoning about contextual information must be implemented for each domain and application. This makes it flexible only in the design and implementation phase, but and not during run-time.

Chen and Finin (2003) proposed CoBrA, in which an intelligent broker agent maintains a shared model of the context data available. This model assumes the responsibility for aggregating contextual data and making it available. The main difference between this model and the Context Toolkit is that CoBrA also includes a knowledge model of the contextual information available, and hence enables more distributed reasoning capabilities for agents wishing to use CoBrA.
The Service-oriented Context-Aware Middleware (SOCAM) project (Gu *et al.*, 2004) is another architecture for the development and the rapid prototyping of context-aware mobile services. It uses a central server, called context interpreter, which acquires context data from distributed context providers and offers it in mostly processed form to clients. The context-aware mobile services are located on top of the architecture and thus make use of the different levels of context to adapt their behavior accordingly. In SOCAM an ontology-based approach to model various contexts is proposed and the architecture supports semantic representation, context reasoning and context knowledge sharing.

Hong & Landay (2001) advocate a centralized service infrastructure approach. The infrastructure would dictate architectural rules to ensure that new components added would interoperate with others that abide to the same rules. These rules would include how data and functional responsibilities are allocated to components, how those components relate to each other, and how their interfaces are defined. Biegel & Cahill (2004) have developed an architecture which significantly eases the development of mobile, context-aware applications. The architecture allows developers to fuse data from disparate sensors, represent application context and reason efficiently about context without the need to write complex code. An event based communication paradigm designed specifically for ad-hoc wireless environments is incorporated which supports loose coupling between sensors, actuators and application components.

The CORTEX architecture (Veríssimo *et al.*, 2002) is a middleware for context-aware distributed applications. It has the required functionality for messaging, service discovery and resource management. However, the system is not capable of controlling the network resources and provides no adaptable user interface for applications. Furthermore, it does not support multimedia applications. CASS (Fahy & Clarke, 2004) is a server based middleware intended to support context-aware applications on hand-held and other small mobile computers. It enables developers to overcome the memory and processor constraints of small mobile computer platforms while supporting a large number of low-level sensor and other context inputs. A key feature of CASS is its support for high-level context data abstraction and the separation of context based inferences and behaviors from application code. This separation opens the way to making context-aware applications configurable by users.

4. CHALLENGES AND RESEARCH DIRECTIONS

4.1 Challenges for implementing context-aware architecture

Defining an architecture to support context aware applications explicitly implies a scalable description of how to represent contextual information and which abstraction models are capable of handling it (Dey, 2000). Using sensors to retrieve contextual information leads to a sensor network scheme that provides services to upper levels of applications. Operations for capturing, collating, storing and disseminating contextual information at the lowest level and aggregating it into increasingly more abstract models qualifies context-aware systems. Thus, in order to support context aware systems, a context model should be defined. The context model (Dey *et al.*, 2005) should be structured around a set of abstract entities, each describing a physical or conceptual object. System behavior in context-aware computing should take into account the adaptability of a system towards dynamic changes of the contextual information.

Existing context-aware architectures support requirements of providing distributed storage and subscription services for context data, but they still place an unnecessary burden on context-aware application developers. Using current architectures would require an application to manage a host of subscriptions to remote repositories, maintain multiple network connections, and perform all intermediate data processing. Duplicating this level of work for all desired context-aware applications is simply unacceptable. What is needed is a standard architecture supporting general, adaptive, and decentralized services that both scale to wide-level deployment and simplify context-aware application design. An architecture supporting context-sensitive applications should provide some basic components which are not easy to design and implement (Kuo *et al.*, 2004). Often the same context can be derived from different sensors. This raises difficulties for developers when the underlying sensors are changed. A component is needed to decouple the application and the actual context sensing part. The benefit of using this component is to minimize the impact of sensor change.

There are many advantages to an infrastructure approach for supporting context-aware applications, but yet there are many technical challenges that must be overcome as pointed out by Hong and Landay (2001). The first challenge lies with designing the data formats and protocols used by the infrastructure. These standards will be the glue that allows the separate pieces of the infrastructure to interoperate. For this reason, they need to be simple enough so that they can be implemented for practically any device and used by any application. The second challenge to building a context-aware infrastructure lies with designing the services. Some of these context services will be highly application specific and will have to be designed and implemented on a case-by-case basis. However, other context services will be basic enough such that they will be an integral part of the infrastructure itself.

A third and very difficult challenge to building a context infrastructure lies with scoping and access of both sensor and context data. In other words, who has access to what data? Clearly, the infrastructure needs to be secure against unauthorized access, but it also needs ways to let people introspect, so that they can understand what is being done with the data and by whom. Furthermore, the infrastructure needs to be designed such that privacy concerns are legitimately and adequately addressed. A fourth challenge is that of scale. The sheer number of sensors, services, and devices envisioned poses some fundamental engineering challenges. The infrastructure needs to work for large numbers of sensors, services, devices and people. It must also require a minimal amount of administrative effort. However, considerable progress is still needed in these areas before a context infrastructure can be deployed across a wide area.

4.2 Research Directions

In order to reduce the cost and difficulties of building context-aware applications, we must develop an architecture to enable distributed software components to contribute to and access a shared model of context and to allow users to control access to their personal information in a context-aware environment. Future architectures for context-awareness should be able to acquiring context from heterogeneous sources. In order to support the functions of advanced context-aware behaviors, it is necessary for systems to consider information from a wide range of sources, not just from sensors that are embedded in the local environment. The architecture should also maintain consistent contextual knowledge. In order to prevent applications from making inaccurate decisions due to inconsistent knowledge, the architecture should detect and resolve any contextual knowledge that may be inconsistent or ambiguous. Moreover, future context-aware architectures should enable knowledge sharing among applications entities. When acting in isolation, the knowledge of a single entity is limited. In a dynamic environment, it is more cost-effective for entities to share their knowledge.

Among the critical research issues in developing context-aware systems are context modeling, context reasoning, knowledge sharing and user privacy protection. A long-term goal of system architectures is to make sensors and context platforms flexible and scalable enough to be widely adopted in various context-aware applications. Aiming at Human-Computer Interaction, context-sensing requirements in context-aware computing applications take into account the fact that sensors are highly distributed and their configuration is highly dynamic. Contextual information may in addition be incorrect, incomplete, or inconsistent. As a consequence, different approaches have emphasized that a context model needs to represent the quality characteristics of context data (Henricksen et al., 2002). However, although the quality of context clearly needs to be taken into account, the divergence of context quality results from the specific characteristics and the imperfectness of context sensors.

Designing contextual data format and network protocols to allow interoperability by supporting different types of sensors and finding the right balance of developing a universal context model and smart infrastructures are the challenges for future context services. Many systems should also take into consideration the quality of context represented into a model supported by a meta-model scheme of context. The research goal is to find the right level of description, which abstracts away from implementation details, but is still specific enough to serve the purpose of inferring appropriate intent from context-assuming interactions. Context-awareness and mobile scenarios require an architecture that supports interoperability among components that have not been specifically designed to work together. Ideally, there should be a universally agreed architecture for service discovery and interaction. There are many competing approaches to this problem, and a good deal of research and experimentation will be required before broadly usable architectures and standards appear.

5. C-ARC: THE PROPOSED CONTEXT-AWARE SYSTEM ARCHITECTURE

This section gives an overview of C-Arc and describes how it provides mobile context-aware application designers with support for high-level context derivation. C-Arc addresses the issue of separation of context-aware application code from high-level context reasoning and behaviors. More precisely, this allows an application's context reasoning and resulting behaviors to be changed without re-compilation. C-Arc applications need not communicate with each individual source of context directly but only with the context service layers and therefore they do not need to store low-level details of context sources. An efficient model for handling, sharing and storing context data is essential for a working context-aware system. C-Arc maintains a model of the current context that can be shared by all devices and services in the same smart space. The shared model is a repository of knowledge that describes the context associated with an environment. As this repository is always accessible within an associated space, resource limited devices will be able to offload the burden of maintaining context knowledge. When this model is coupled with a reasoning facility, it can provide additional services, such as detecting and resolving inconsistent knowledge and reasoning with knowledge acquired from the space. Figure 1 depicts the overall architecture of C-Arc and a brief description of its components follows.

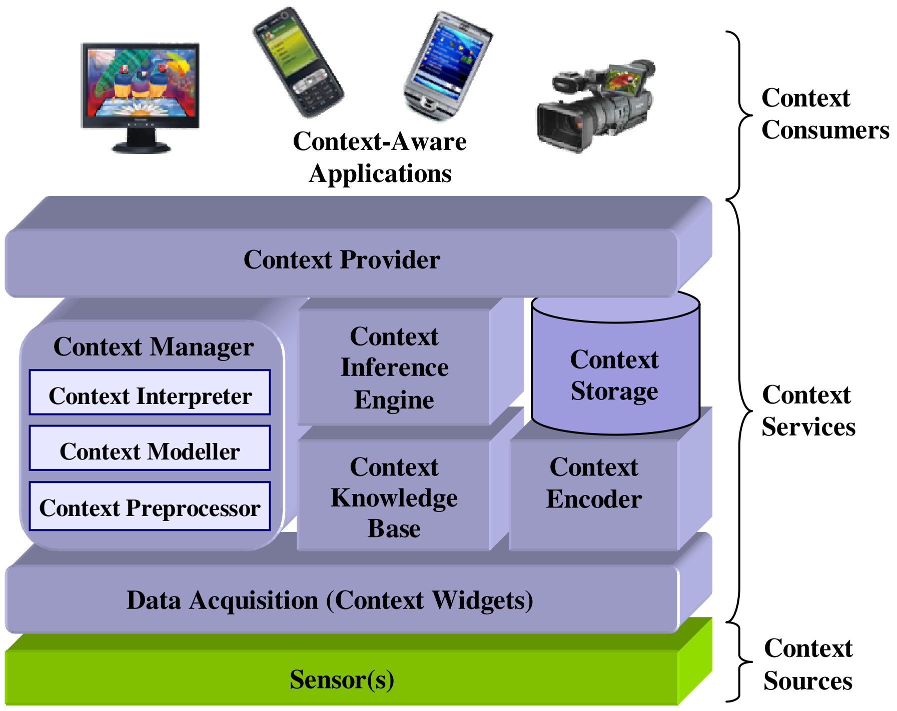

Figure 1: Overview of the C-Arc Architecture

5.1 Context Acquisition
Specific widgets are developed at the data acquisition layer to capture different kinds of contextual information. The context widget acts as a mediator between an application and its operating environment, separating context acquisition from context use. It relieves applications from context sensing issues by wrapping the sensor with a uniform interface. This makes applications design independent of the method of context sensing employed. The context widget encapsulates details of context acquisition thus leading to the flexibility of exchanging a context widget with another widget providing the same type of context information without affecting applications using it. Irrespective of the type of context a widget provides, it stores its latest sensed context value and allows applications to query. A context widget is independent of all executing applications and persistent. It continuously sends updated context information to the context manager for context encoding, storage and further processing when required. It can also be shared among applications and is reusable.

Raw context data acquired from sensors has to be processed as its customers are interested mostly in interpreted and aggregated information compared to raw, fine-grained data. Context aggregation refers to the composition of context atoms either to collect all context data concerning a specific entity or to build higher-level context objects. Context interpretation, on the other hand, refers to the transformation of context data including special knowledge. These forms of context data abstraction ease the application designer's work tremendously.

5. 2 Context Manager
The context manager consists of three subcomponents that are described here.

5.2.1 Context interpreter: The context interpreter performs context processing using logic reasoning. The tasks for context processing may include deriving high-level contexts from low-level contexts, querying context knowledge, maintaining the consistency of context knowledge and resolving context conflicts. It also acts as a context provider as it can provide deduced contexts.

5.2.2 Context Modeller: The context modeler converts context into a form that can be processed by the computer, e.g. it can converts location data to a local coordinate format.

5.2.3 Context Preprocessor: The context preprocessor may not be used in every context-aware system but may offer useful information if the raw data are too coarse grained. It is responsible for reasoning and interpreting contextual information. The sensors queried in the underlying layer most often return technical data that are not appropriate for use by application designers. Hence this layer raises data from the acquisition layer to a higher abstraction level. The transformations include extraction and quantization operations. For example, the exact GPS position of a person might not be of value for an application but the name of the room the person is in may be. Hence, the context preprocessor will convert the location from physical to symbolic.

5.3 Context Storage
Applications may be dependent on current context as well as past context to adapt their behavior. Thus, context acquired from sensors are embedded and stored in a database along with a

timestamp even when no applications currently require them. Since all context events have a well-determined structure, it is relatively simple to automatically develop schemas for storing them into a database. An ontology based approach, that entails several benefits discussed in (Chen *et al.,* 2003), is used. More precisely, context is encoded in Web Ontology Language (OWL) (Smith *et al.,* 2006), which is more expressive than other ontology languages. In particular, OWL DL, one of the three sublanguages of OWL, namely OWL Lite, OWL DL and OWL Full that are layered according to increasing expressiveness, is employed. Context storage can be queried for past context by applications. Moreover, past context information can be used to learn rules about the user and application behavior through data mining.

5.4 Context Encoder

The context encoder encodes context information acquired from sensors using OWL and saves in the context storage.

5.5 Context Knowledge Base

The context knowledge base contains rules used by the inference engine. C-Arc stores application knowledge bases that contain high-level context states and their corresponding context behavior. The context knowledge base provides a set of API's for other service components to query, add, delete or modify context knowledge. Furthermore, it contains context ontology in a sub-domain and its instances. These instances may be specified by users in case of defined contexts or context acquired from various context providers in the case of sensed contexts. The context ontology and its instances of defined contexts are pre-loaded into the context knowledge base during system initiation while the instances of sensed contexts are loaded at runtime. To ensure freshness of context information, an event triggering mechanism is deployed to allow updating of a particular context ontology or instance. Different information requires different update frequency. Having a separate knowledge base means that changes can be made to context inferences and goals relevant to an application without necessitating changes in the application code.

5.6 Inference Engine

The inference engine in C-Arc is used to find a matching goal or goals when a change in context is detected. It uses a technique called forward chaining, where known facts are used to infer other facts that can in turn be used to infer further facts. Moreover, it employs inference rules to perform context reasoning over stored facts. Forward chaining is a search technique useful for situations where the search space is wide with many potential goals, which is the case with context-aware systems. Based on the explicit description of context information provided by context modeling and semantic interoperability provided by context mediation, context inference, which ultimately incorporates contextual information into the knowledge process, plays a key role in realizing the context aware knowledge management. The inference engine works in conjunction with a knowledge base that stores facts and rules, and a context storage that stores past and current context as facts. For instance, given a certain set of conditions or parameters, the inference engine consults its knowledge base and context storage to find a matching set of conditions or parameters and a corresponding goal or solution to a problem.

5.7 Context Provider

The context provider broadcasts context information on context change. Thus, context consumers listen for events that are sent by the context provider. Moreover, they can also query the context provider for context information.

5.8 Context Consumers

The context consumers consume various types of contexts and adapt their behavior according to the current context. They obtain contexts either by querying the Context Provider or by listening for events that are sent by the Context Provider. Hence, applications can easily acquire the contexts they need to take decisions. The architecture also provides mechanisms for developers to specify context-sensitive adaptation behavior using rules. As such, C-Arc makes it very easy to develop and deploy context aware applications.

6. IMPLEMENTATION AND EXPERIMENTATION

6.1 Communication in C-Arc

All the components of C-Arc are autonomous in execution. They are instantiated and execute independently of each other. These components as well as the context-aware applications are usually distributed on several computers for better performance and efficiency. C-Arc allows a peer-to-peer communication among them using HyperText Transfer Protocol (HTTP) and eXtensible Markup Language (XML). Messages are encoded in XML and wrapped with HTTP. These allow for the lightweight integration of distributed components, enable the architecture to benefit from the platform- and language-independence features of XML and enable to build more interoperable context-aware services. Other alternatives considered include CORBA, RMI and SOAP. CORBA and RMI are too heavyweight and require additional components. In addition, RMI would dictate the use of Java. SOAP, while being lightweight would require the use of a web server that supports the protocol. However, the default communication protocol can be modified to account for other protocols, e.g. Simple Mail Transfer Protocol (SMTP) simply by creating an object that speaks the SMTP protocol for outgoing communications and one for incoming communications. Thus, C-Arc supports transparent distributed communications.

Moreover, resource discovery mechanisms are rarely used in existing frameworks. However, such dynamic mechanisms are important, particularly in pervasive environment, where available sensors and the context sources change rapidly. C-Arc exploits Jini service discovery protocol (Marin-Perianu *et al.,* 2005) to allow service providers to advertise their capabilities and information one must know to access their services and consumers to locate available services.

6.2 C-Arc Implementation details

All the components of C-Arc are implemented in Java and are multithreaded. Thus, C-Arc is platform-independent and its components are capable of handling multiple incoming messages. C-Arc provides two classes, namely CARCclient and CARCserver to send and receive messages respectively. Application programmers need not write these classes, but merely create instances of them to use them. For instance, applications as well as C-Arc components create an instance of the CARCclient class to send requests and an instance of the CARCserver class to receive replies. An instance of the CARCserver class acts as a super tiny web server that intercepts messages intended for the application or component that instantiated it. C-Arc provides a fundamental abstract class ‘Widget’ that caters for features common to all context widgets, namely (1) receiving and servicing requests from context consumers, (2) notifying the context manager when a context change is detected, and (3) sending a list of all context attributes a particular widget can provide. Classes of all types of widgets subclass from this class and thus inherit all its attributes and behaviours. The inference engine used by C-Arc is Bossam version 0.9b5 (Meditskos and Bassiliades, 2005), developed by M. Jang, as it seems to best fit the architecture. It supports ontologies, can be embedded in Java programs and makes use of URI to refer to files storing facts and rules, thus making it apt for use with the distributed components of C-Arc.

6.3 Prototype Implementation and Experimentations

A prototype has been implemented on C-Arc to demonstrate the architecture. The prototype was tested with Laptop Pentium 4 computers, Compaq iPAQ PDA and Nokia 3G mobile phones. The programming environment of the device includes NSICom CrEme virtual machine and J2ME. The devices are located by Ekahau WLAN positioning engine.

Experimentation was carried out with a prototype for routine learning which is capable of learning user’s habits and support location-based reminders. It consists of four applications, namely a profile manager, a reminder, a calendar and a personal assistant. The calendar application is used to store appointments and meetings. The context component gets events from the calendar via the reminder application. The profile manager detects the routines from the context component and the personal assistant application is used to browse the applications on the mobile device. According to the prototype scenario, the user buys his phone and starts using it. It learns that the user has a habit of switching the phone on the silent mode during weekly meetings. After it has detected a routine, the system offers to switch the phone on silent mode automatically if the user has a meeting.

Furthermore, the prototype supports location-based reminders. Thus, the user attaches a reminder to a location and then the system recognizes the place through a GPS enabled device and notifies the user by popping up a message. The capabilities and requirements of the components in the C-Arc architecture are presented in the form of XML descriptions. Such descriptions include both optional and mandatory characteristics of the components, like names of the component’s interfaces, unique ID numbers, URLs of the component’s code and so on.

The C-Arc environment supports different kind of events. The context-related events are produced and collected in a push model. Whenever the event occurs it is forwarded to the context component, which sends it to the component responsible for that event. The events are delivered in a publish-and-subscribe fashion. The event suppliers have to register their interest in some event type from a certain component event producer. The architecture supports internal events, which are delivered directly from one component to another. The internal events are the responsibility of the messaging component.

C-Arc uses a hybrid location model that supports both physical as well as symbolic location. As such, location-sensitive applications can express location in terms of either coordinates with respect to a certain axis of reference, e.g. geodetic coordinates, or symbolic names, such as room number. C-Arc also allows easy conversion between the two types of location representation through the use of interpreters. Figure 2 depicts the local coordinate format used by C-Arc, whereby the area being considered by the context-aware application has its own origin and x and y axis, obtained by adjusting the origin of the Earth Centred Earth Fixed (ECEF) Cartesian coordinate.

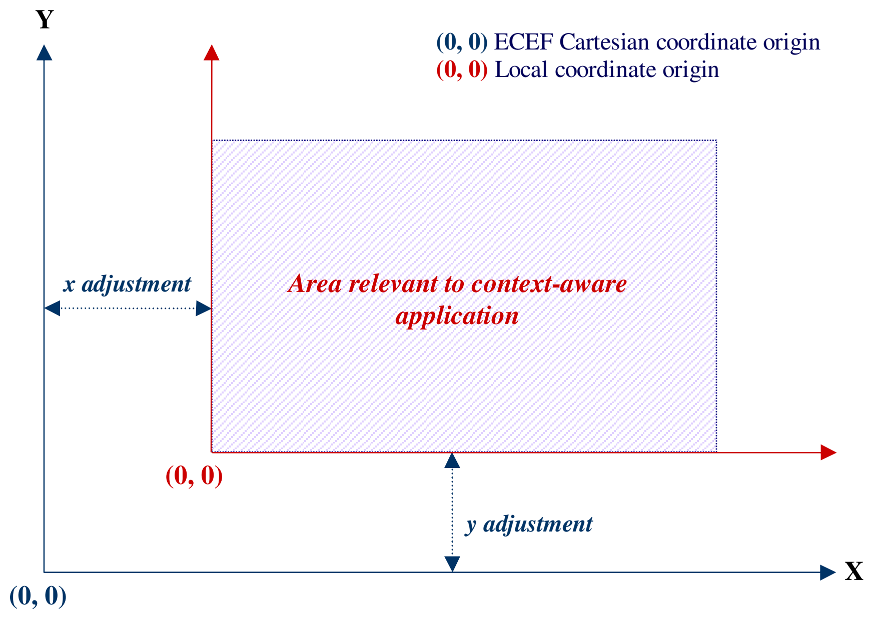
Y
(0, 0) ECEF Cartesian coordinate origin
(0, 0) Local coordinate origin
x adjustment
Area relevant to context-aware application
(0, 0)
y adjustment
X
(0, 0)

Figure 2: Local coordinate format of C-Arc

7. EVALUATION OF C-ARC

Building context-aware applications can be difficult and costly without the support of a computing infrastructure. C-Arc reduces the cost and difficulties of building context-aware applications. It enables distributed software components to access a shared model of contexts. C-Arc can also acquire context from heterogeneous sources. In order to support the functions of advanced context-aware behaviors, C-Arc considers information from a wide range of sources rather than only sensors that are embedded in the local environment.

Moreover, the C-Arc architecture enables knowledge sharing among applications entities. When acting in isolation, the knowledge of a single entity is limited. In a dynamic environment, it is more cost-effective for entities to share their knowledge. Finding the right balance of developing a universal context model and smart infrastructures is a challenge for C-Arc architecture. The architecture should also take into consideration the quality of context represented into a model supported by a meta-model scheme of context. The research goal is to find the right level of description, which abstracts away from implementation details, but is still specific enough to serve the purpose of inferring appropriate intent from context-assuming interactions.

Finally, an open issue in C-Arc is the definition of a context prediction method supporting the proactivity of context services in a proactive environment which associates 'similar' context models. The hard part will be coming up with structures that are broad enough to handle all of the different kinds of context, sophisticated enough to make the needed distinctions, and simple enough to provide a practical base for programming. The goal of this research is to provide an architecture that makes it easier for application developers to use context. The architecture will enable developers to add context to applications that are not context-aware and to increase the use of context in applications that are already context-aware.

7.1 Comparison with existing work

The Context Toolkit differs from our architecture, in that our work also supports the separation of context-awareness rules from application code. Moreover, C-Arc enables distributed software components to access a shared model of contexts.

SOCAM supports rules for specifying which methods should be invoked on events. The rules are predefined and loaded into the Context Reasoner. Whereas in C-Arc in order to prevent applications from making inaccurate decisions due to inconsistent knowledge, the architecture detects and resolves any contextual knowledge that may be inconsistent or ambiguous.
Resource discovery mechanisms are currently rarely used in the existing architectures. Such dynamic mechanisms are important, especially in a pervasive environment, where available sensors, the context sources, change rapidly. C-Arc can acquire context from changing and heterogeneous sources.

Every system and framework uses its own format to describe context and its own communications mechanisms. We believe that standardised formats and protocols are important for the enhancements of context-aware systems to make the development of context services the focus rather than the communication between context sources and users. C-Arc is an attempt to provide standardised methods for service description and access.

8. CONCLUSION

Context-aware systems are a fast growing research area with a lot of diverse research. Various existing context-aware architectures and supporting infrastructures which ease the development of context-aware applications have been discussed in this paper. It is found that the research of accurately discovering context, efficiently disseminating contextual information and making use of the available context is still at the early stage. It is believed, however, that context awareness is a key factor for new applications in ubiquitous computing. Probably the main problem in the existing architectures is the variety of used context encodings and ways to find and access context sources. Every architecture uses its own format to describe context and its own communications mechanisms. It is believed that standardized formats and protocols are important for the enhancements of context-aware systems to make the development of context services the focus rather than the communication between context sources and users. Therefore, context-aware service environments require a well designed standardized architecture for the management of context information. Unfortunately, while good design alternatives for context architectures already exist, it will be a long time before standard, interoperable context information management becomes a reality.

In this paper, the issues that arise in supporting an emerging class of applications that operate independently of direct human control have been explored. The necessary support mechanisms in terms of a system architecture has been presented. C-Arc provides supports for most of the tasks involved in dealing with context, namely acquiring context from various sources, interpreting context and disseminating context. The main feature of the C-Arc architecture is that it supports context reasoning. Through context reasoning, high-level, implicit contexts can be derived from low-level, explicit contexts and applications can be given a notion of the confidence of different contexts before acting on it. Context-aware mobile services can be easily built by using various types of contexts with different levels of complexity.

A long-term goal of C-Arc is to make sensors and context platforms flexible and scalable enough to be widely adopted in various context-aware applications. Future work in this area will investigate the use of service-oriented and autonomic computing concepts for building context-aware service frameworks. Sensing fusion can be used to improve the quality of context (QoC). Another issue that merit further investigation is to enable the architecture to detect failure of a component and automatically restart it as well as restore it to its last working state. Moreover, context prediction can also be used to support pro-activity of context services.

REFERENCES

- ABOWD, G..D., ATKESON, C.G., HONG, J., LONG, S., KOOPER, R. & PINKERTON, M. (1997). Cyberguide: A Mobile Context-Aware Tour Guide. *ACM Wireless Networks*, 3, 421-433.
- ANAGNOSTOPOULOS, C., TSOUNIS, A. & HADJIEFTHYMIADES, S. (2004). Context Awareness in Mobile Computing: A Survey. *Proceedings of Mobile and Ubiquitous Information Access Workshop*, Mobile HCI '04, Glasgow, UK.
- BIEGEL, G. & CAHILL, V. (2004). A Framework for Developing Mobile, Context-aware Applications. *Proceedings of 2nd IEEE Conference on Pervasive Computing and Communications*, (Percom) 2004, Orlando, FL.
- CHEN, H. & FININ, T. (2003). An Ontology for a Context Aware Pervasive Computing Environment. *IJCAI Workshop on ontologies and distributed systems*, Acapulco MX.
- DEY, A. (2000). Providing Architectural Support for Building Context-Aware Applications. Ph.D. Thesis Dissertation, College of Computing, Georgia Tech.
- DEY, A. & ABOWD, G.D. (1999). Towards a Better Understanding of Context and Context Awareness. Technical Report, GIT-GVU-99-22, Georgia Institute of Technology.
- DEY, A., KOKINOV, B.N., LEAKE, D.B. & TURNER, R.M. (2005). Modeling and Using Context. *5th International and Interdisciplinary Conference*, CONTEXT 2005, Paris, France.
- DEY, A., SALBER, D., & ABOWD, G.D. (1999). The Context Toolkit: Aiding the Development of Context-Enabled Applications. *In the Proceedings of the 1999 Conference on Human Factors in Computing Systems*, CHI 1999, Pittsburgh, PA, 434-441.
- FAHY, P., CLARKE, S. (2004). CASS: Middleware for Mobile, Context-Aware Applications. *Workshop on Context Awareness at MobiSys 2004*, Boston, USA.

- GU, T., PUNG, K. K. & ZHANG, D. Q. (2004). A middleware for building context-aware mobile services. *In Proceedings of IEEE Vehicular Technology Conference (VTC),* Milan, Italy.
- HENRICKSEN, K., INDULSKA, J. & RAKOTONIRAINY, A. (2002). Modeling Context Information in Pervasive Computing Systems. *Proceedings of 1st International Conference on Pervasive Computing,* Zurich, Switzerland. Berlin, Springer, 167-180.
- HONG J. & LANDAY, J. (2001). An infrastructure approach to context-aware computing', *Human Computer Interaction*, 16(2).
- KUO, Y., CHENG, K., HSU, J., CHU, H. & HUANG, P. (2004). Architectural Support for Context-Sensitive Interaction in Ubiquitous Computing Environment. A proposal submitted to the IIS, Academia Sinica.
- MARIN-PERIANU, R., HARTEL, P., & SCHOLTEN, H. (2005). A Classification of Service Discovery Protocols. Technical Report TR-CTIT-05-25 Centre for Telematics and Information Technology (University of Twente, Netherlands).
- MEDITSKOS, G., & BASSILLIADES, N. (2005). Towards an Object-Oriented Reasoning System for OWL. *International Workshop on OWL Experiences and Directions,* B. Cuenca Grau, I. Horrocks, B. Parsia, P. Patel-Schneider (Ed.) (Ireland).
- MUSOLESI, M. (2004). Designing a Context-aware Middleware for Asynchronous Communication in Mobile Ad Hoc Environments. *In Middleware 2004 Companion Proceedings*, 304-308, ACM Press.
- SHILIT, B. N. (1995). A Context-Aware System Architecture for Mobile Distributed Computing. Ph.D. thesis, Dept of Computer Science, Columbia University.
- SMITH, M. K., WELTY, C., VOLZ, R., & MCGUINESS, D. (2006). OWL Web Ontology Language Guide. W3C Recommendation February 2004. Online: http://www.w3.org/TR/owl-guide/.
- VERÍSSIMO, P., CAHILL, V., CASIMIRO, A., CHEVERST, K., FRIDAY A. & KAISER, J. (2002). CORTEX: Towards Supporting Autonomous and Cooperating Sentient Entities. *Proceedings of European Wireless 2002*, Florence, Italy.
- WANT, R., HOPPER, A., FALCAO, V. & GIBBONS, J. (1992). The active badge location system. *ACM Transactions on Information Systems*, 10(1), 91-102.