

Context-Aware Middleware for Pervasive and Ad Hoc Environments

Hector A. Duran-Limon¹, Gordon S. Blair¹, Adrian Friday¹, Paul Grace², George Samartzidis¹,
Thirunavukkarasu Sivaharan², Maomao WU¹

Computing Department, Lancaster University, Bailrigg, Lancaster LA1 4YR, UK.

Tel: +44 1524 65201, ext: 93141

Fax: +44 1524 593608

{duranlim,gordon,adrian,gs,maomao}@comp.lancs.ac.uk¹, {p.grace,
t.sivaharan}@lancaster.ac.uk²

Abstract: *Recent advances in the area of mobile computing and pervasive computing have driven the emergence of new challenges. For example, the “Intelligent Environment” or “Smart Environment” has become one of the key research areas in the pervasive computing arena. These scenarios typically consist of intelligent rooms capable of gathering raw sensory data from embedded “invisible” sensors, then processing and analysing them to deduce high-level context and infer personal preferences (for room temperature, light intensity, type and level of music, etc.) Mobile and ad hoc scenarios also include time-critical applications such as an automatic car control systems in which cars are able to operate independently and cooperate with each other to avoid collisions. Another example is an air traffic control system whereby thousands of aircraft are proactively coordinated to keep them at safe distances from each other. In this paper, we present a context-aware middleware architecture for the support of both pervasive and ad hoc environments. The approach makes use of both reflection and component technology. A prototype of the middleware has been implemented in OpenCOM, which is a lightweight, efficient and reflective component model based on Microsoft’s COM.*

1. Introduction

Recent years have witnessed advances in the enabling technologies for mobile computing, such as the increasingly mature end-systems, various kinds of wireless communication protocols, and mobile networking technologies. As a result new challenges have been emerged in both the pervasive computing and ad hoc computing arenas. Regarding the former, one of the key research areas in recent year is the “Intelligent Environment” or “Smart Environment”. A personalized intelligent room, for example, can set the room temperature according to the occupier’s preference, play his beloved music, and make his favourite coffee, etc. By gathering raw sensory data from embedded “invisible” sensors, the intelligent room can then process and analyse them to deduce high-level context and infer personal preferences. Storing context information with personal preferences can make the system behave more intelligently. For example, the room can make black coffee for the occupier in the morning, but espresso in the afternoon; and when there are visitors, it might make cappuccino for them. Public intelligent environment should also be able to resolve conflicts between different users. If there are more than two persons in the room, the intelligent room should be able to decide what the temperature to set so that it can minimize the disturbance, what kind of music to play to maximize satisfaction, etc.

In addition, examples of ad hoc applications include air traffic control systems whereby thousands of aircraft are proactively coordinated to keep them at safe distances from each other, direct them during takeoff and landing from airports and ensure that traffic congestions are avoided. Another example is an automatic car control systems in which cars are able to operate independently and cooperate with each other to avoid collisions. This kind of systems is time-critical and needs to be provided with both timeliness assurances and adaptation mechanisms that lead the system to a safe state in case of unexpected changes in the environment.

The CORTEX Project¹ is examining fundamental issues relating to the support of such applications, including the development of middleware for this domain. The CORTEX approach is based on anonymous and asynchronous event models. Such models are well-suited to both pervasive and ad hoc environments. That is, many-to-many communication scenarios are well supported by the anonymous dissemination of information. In addition, asynchronous communication is ideal in systems whereby frequent disconnection are likely to happen as blocking conditions are avoided.

¹ This work is supported by the EC, through project IST-FET-2000-26031 (CORTEX). <http://cortex.di.fc.ul.pt>

In this paper, we present a context-aware middleware architecture which provides support for both pervasive and ad hoc computing. The middleware is structured in terms of a number of component frameworks (CF) [1]. For example, the middleware platform provides support for the dissemination of events in a timely fashion. Mechanisms for context-awareness and intelligent decision-making are also provided. Lastly, a flexible framework is offered for enabling the use of a number of diverse service discovery protocols.

The paper is structured as follows. Section 2 introduces some background on the CORTEX architecture. Section 3 then presents the associated middleware which includes a publish/subscribe CF, a resource management CF, and a service discovery CF. The implementation of the system's prototype is then described in section 4. Section 5 provides an overview of some related work and section 6 draws some concluding remarks.

2. Background on the CORTEX Architecture

Central to the CORTEX architecture is the notion of a *sentient object* [2, 3, 4] which is defined as an entity that is able to both consume and produce events as shown in figure 1. That is, sentient objects are objects that receive events as input, process them and generate further events as output. Input events are received either from sensors or from other sentient objects. Similarly, output events are sent either to actuators or other sentient objects.

Sentient objects are autonomous entities that are able to sense their environment. Interestingly, sentient objects have a proactive role in that they are capable of taking decisions and performing some actions (i.e. generate further events) based on the information sensed. Hence, sentient objects include a control logic, as shown in figure 1, which realises the decision making mechanism. Briefly, the capturing function is in charge of translating the events received to context-based information. In addition, a database of past, current and future context information is maintained. The former refers to the context history whereas the second tell us about the current state of a sentient object. The third element reflects an attempt to predict future context information based on the current state and context history.

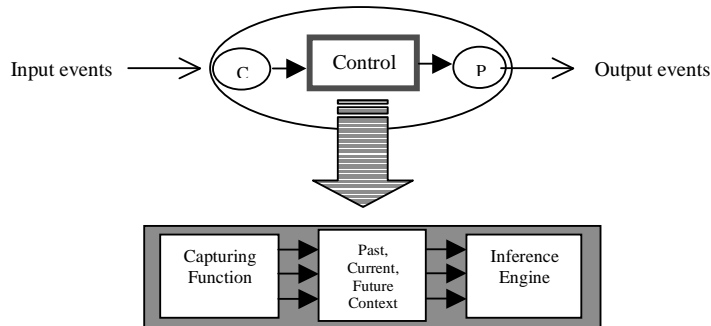


Figure 1. The Sentient Object Model

In addition, as the envisaged applications modelled with the sentient object paradigm are typically complex, the realisation of these systems based on a single sentient object would result in a difficult task. Therefore, hierarchal composition of sentient objects is supported to alleviate such complexity. That is, sentient objects can recursively include a number of cooperating sentient objects. Hence, *composite sentient objects* may include two or more sentient objects in which such objects may receive and produce internal and/or external events, as shown in figure 2. In contrast, *primitive sentient objects* are not further partitioned. As an example of sentient object composition, consider a car including a number of controllers (e.g. engine controller, transmission controller, braking controller, etc) connected to a CAN [5]. Each of the controllers may be represented by a sentient object. Furthermore, the whole car including the set of controllers and the rest of the car machinery can be modelled as a composite sentient object.

Finally, the CORTEX architecture includes an event model called STEAM [6] that allows for the dissemination of events in an ad hoc network environment. Basically, STEAM is a publish/subscribe system whereby producers publish events and consumers subscribe to event types from which consumers receive events. STEAM is an implicit event model in which there is not an event broker or mediator,

instead brokering functions are implemented at both the consumer and the producer side. This is useful in an ad hoc environment where permanent communication with a broker is unlikely to be maintained. Interestingly, event filtering is achieved by performing both subject-based and proximity filtering at the producer side. The former describes the particular event types that the consumer is interested. The latter specify the geographical area within which event types are valid. On the other hand, the consumer uses content-based filtering in which an expression is evaluated against a set of values included in an event.

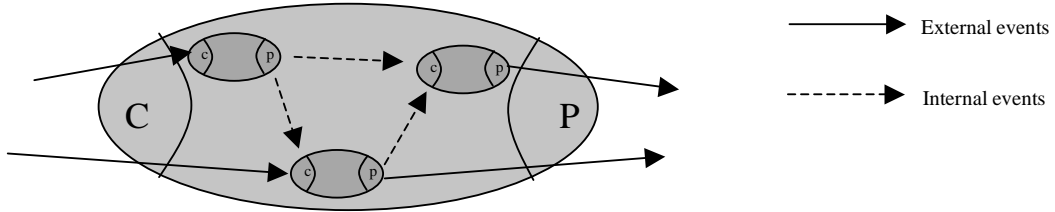


Figure 2. Composite Sentient Object

3. CORTEX Middleware

3.1 Overview

We are building a middleware platform in order to provide support for CORTEX applications. In order to tackle the requirements imposed by ad hoc environments, configuration and reconfiguration capabilities are introduced in the middleware architecture. Based on previous experience in the construction of reflective middleware [7], we make use of *reflection*, *component technology* and *component frameworks (CFs)*. In fact, the implementation of the middleware is developed in OpenCOM [8], which is a lightweight, efficient and reflective model based on Microsoft’s COM [9].

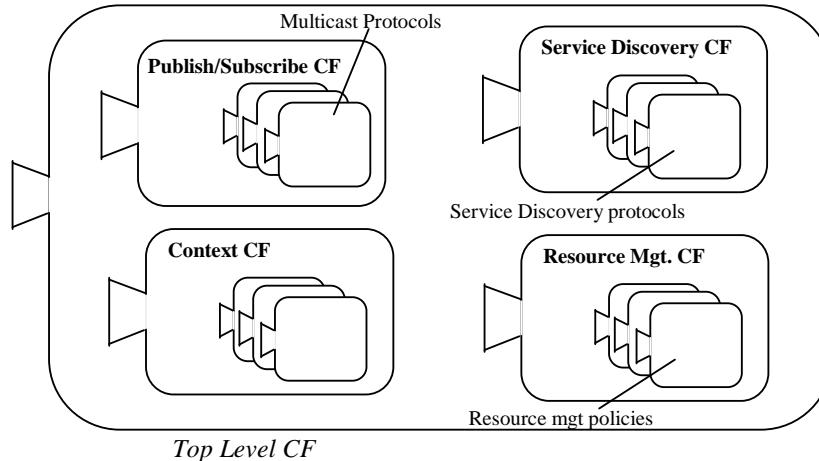


Figure 3. The Middleware Architecture of CORTEX

Reflection is a means by which a system is able to inspect and change its internals in a principled way [10]. Basically, a reflective system is able to perform both self-inspection and self-adaptation. To accomplish this, a reflective system has a representation of itself. This representation is causally connected to its domain, i.e. any change in the domain must have an effect in the system, and vice versa. A reflective system is mainly divided into two parts: the base-level and the meta-level. The former deals with the normal aspects of the system whereas the latter regards the system’s representation. The meta-level interface is often referred to as the *meta-object protocol (MOP)* [11]. On the other hand, component

technology introduces more configuration and reconfiguration capabilities into distributed applications and increases the level of reuse. Within the context of CORTEX a component is basically “a unit of composition with contractually specified interfaces and explicit dependencies only” [1]. The granularity of a component may be diverse ranging from components that realise only a part of the machinery of a single sentient object to components that encompass two or more sentient objects.

The environment support for the interaction of sentient objects is also conceptualised as a componentised middleware platform. As said before, the middleware is structured in terms of component frameworks [1] as shown in figure 3. Essentially, component frameworks are “collections of rules and interfaces that govern the interaction of components ‘plugged into’ them” [1]. In other words, a component framework is a reusable architectural design targeting a specific domain whereby desired architectural properties and invariants are enforced. Details of the various CFs are introduced in turn below.

3.1 Publish / Subscribe CF

As said before, PSMMoC is a componentised prototype of a STEAM-like publish/subscribe system as shown in figure 4. PSMMoC, like STEAM, follows an implicit event model [6] whereby mediators are not required as entities subscribe to particular event types. As mentioned earlier, this model is suitable for ad hoc environments in which periods of disconnection are unpredictable. The role of the publisher component is to push events to the event system whereas the subscriber component receives events. Users then get notified of the arrival of an event by the notify component. Hence, the application programmer only needs to implement this component to suit the preferred notification behaviour. Notably, PSMMoC provides support for subject-, content- and context-based event filtering. The functionality of the two former features was introduced in section 2. The latter provides context aware information to consumers. Distance context is currently supported. In addition, filters use a query language (or subscription language), called *Filter Event Language (FEL)* [12]. The event data model exploits XML to represent events. A flexible and general XML profile is defined to represent XML based events. The dissemination of events over the network is then achieved by using SOAP Messaging [13], which is a lightweight XML-based protocol. The SOAPtoMulticast component is in charge of mapping the SOAP messaging protocol to IP multicast.

The IP multicast protocol is exploited by the multicast component to route events from publisher to subscribers. The main idea is to send the event only towards mobile devices that have subscribed to the particular type of event. The event types are mapped to multicast group addresses. The mapping of event types to multicast groups is done using a hash algorithm [14] which generates a unique key from a sequence of characters of arbitrary length and spreads the keys evenly into multicast group addresses. The total size of the publish/subscribe system is 69.5 Kbytes.

The publish/subscribe CF is extended to provide real time event service, by collaborating it with Resource management CF (see below). In a typical application scenario various event types are published and subscribed but the priority of the event types varies, for example in an office scenario events published by fire alarm needs high priority in dispatching the event than events published by an iris scanner which publishes events about who enters a room, the idea being high priority events should be given more priority /resources in processing them than the low priority events to guarantee timely execution and to meet critical deadlines. To meet this requirement event types are mapped to *tasks* and each task have a corresponding VTM (see resource management CF). When large number of events are published by various publishing entities for example by using the Publish() method of IPublish interface, the publish method is intercepted using OpenCOM’s *method interception* facility and the event type is determined and the rest of the event processing is done by the associated VTM for that task. Similarly, at the subscriber side the Dispatcher component is responsible for ensuring priority based event dispatching. The DispatchEvent method of IDispatcher is intercepted and the priority of the event is determined and the associated VTM is used for further processing the event, for example filtering of the event and eventually pushing the event to the consumer.

In pervasive and ad-hoc environments mobile subscribers face the problem of not having a priori knowledge of what types of publishers are present in the current environment and the event types that are published by them. This shortcoming is overcome by the use of the Service discovery CF (see below). The service discovery CF plays a key role in enabling subscribers the dynamic discovery of event types and services that are present in the pervasive and ad-hoc environments.

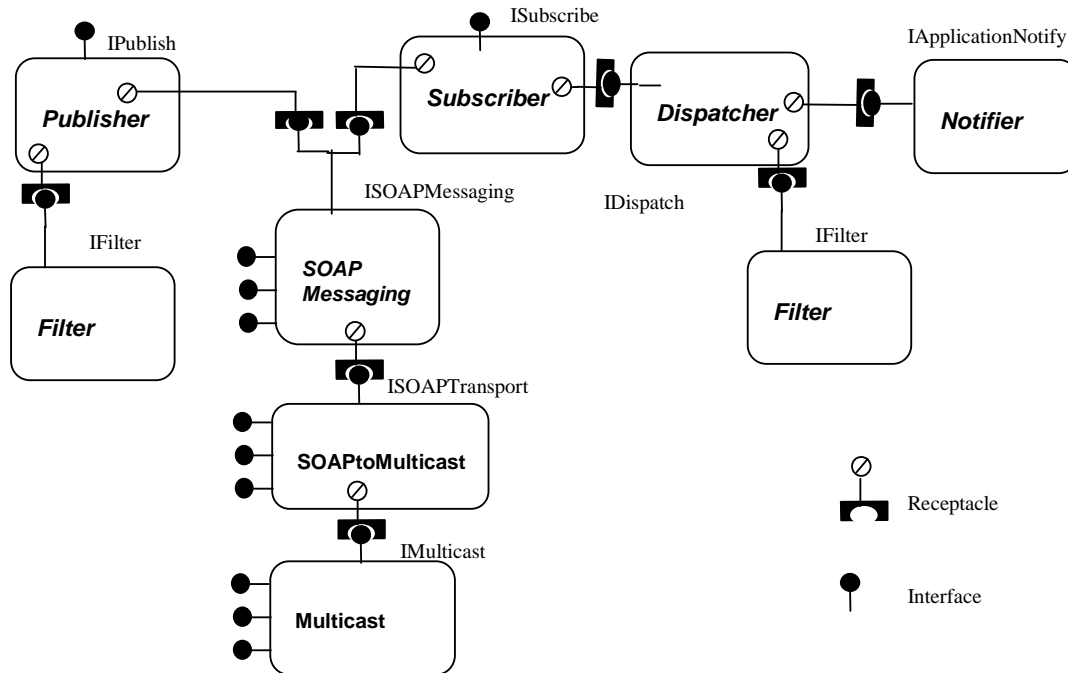


Figure 4. Component Configuration of the Publish/Subscribe System

3.2 Service Discovery CF

The principal function of the Service Discovery framework [15] is to allow services that have been advertised by different service discovery protocols to be discovered. This is performed by changing the component configuration depending on what type of discovery technology is currently used in the environment. For example, if only SLP is currently in use, the framework's configuration will be an SLP Lookup personality. However, if SLP and UPnP are both being utilised at a location then the framework's configuration will include component implementations to discover both. An application may also require services to be advertised; therefore, the personality can be changed to include service registration functionality using one or more protocols of choice. The framework allows individual components to be changed, added or deleted. This is beneficial due to the range of functionalities that service discovery technologies offer. For example, in SLP you may wish to perform lookup using just the multicast protocol if no directory agent is present, but at a later stage if a directory agent is discovered the configuration can be changed to direct requests to it, rather than send multicast requests.

The service discovery framework presents its own custom interface of the service it provides (IServiceLookup); the IDL for this interface is listed in figure 5. The interface offers two key method calls. The first makes a generic service lookup call and will return the information from any service discovery technology lookup call in a generic format. For example, if a generic lookup of a weather service is called and two discovery configurations are implemented by the framework (UPnP and SLP) then the framework will return a list of matched services, from both types in a generic format. The interface also provides methods to determine if a particular service discovery technology is available in the environment or not.

As regards timeliness requirements, we believe that resource management plays an important role in providing support for real-time applications. The resource management CF provide mechanism that allocates resources in the system to ensure that critical activities will be provided with enough resources to carry out their tasks in a predictable way.

```

Interface IServiceLookup{
    Service[] Lookup(ServiceType st, Attributes[] attrs);
    Boolean SDTinUse(SDTType sdt);
    SDTType FindSDTs();
}

```

Figure 5. IDL definition of IServiceDiscovery Interface

3.3 Resource Management CF

The most important elements of the resource model are *abstract resources*, *resource factories* and *resource managers* [16, 17]. Abstract resources explicitly represent system resources. In addition, there may be various levels of abstraction in which higher-level resources are constructed on top of lower-level resources. Resource managers are responsible for managing resources, that is, such managers either map or multiplex higher-level resources on top of lower-level resources. Furthermore, *resource schedulers* are a specialisation of managers and are in charge of managing processing resources such as threads or virtual processors (or kernel threads). Lastly, the main duty of resource factories is to create abstract resources. For this purpose, higher-level factories make use of lower-level factories to construct higher-level resources. The resource model then consists of three complementary hierarchies corresponding to the main elements of the resource model. Importantly, *virtual task machines* (VTMs) are top-level resource abstractions and they may encompass several kinds of resources (e.g. CPU, memory and network resources) allocated to a particular task.

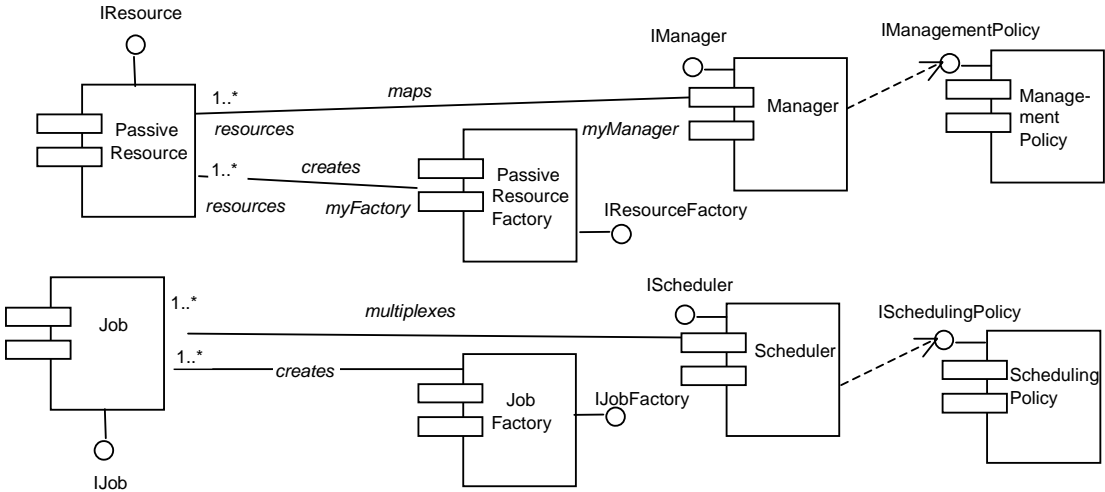


Figure 6. UML Component Diagram of the Resource Model

The component types of the resource model are shown in figure 6. Such component diagram may be used to support a particular instantiation of the resource framework. *Passive resource* components represent non-processing resources such as system memory and battery life. In contrast, *jobs* are capable of performing some activity, that is, they receive messages and process them. Both passive resources and jobs are created by factories as shown in figure 6. In addition, passive resources are managed by managers. However, since jobs are processing resources, they are managed by schedulers instead. In addition, a management policy component determines the management strategy that managers employ. Similarly, schedulers use a scheduling policy component for determining the execution order of their associated jobs.

4. Implementation

As mentioned above, the implementation has been carried out using a reflective component model developed at Lancaster University. This component model, called OpenCOM [15], uses certain core low level features of COM, namely the basic binary level interoperability standard, Microsoft's Interface definition language, COM's globally unique identifiers and the IUnknown interface for interface discovery and reference counting. The fundamental concepts of OpenCOM are then *interfaces*, *receptacles* and *connections* (bindings between interface and receptacles). An interface expresses a unit of service provision and a receptacle describes a unit of service requirement. OpenCOM deploys a standard runtime that manages the creation and deletion of components, and acts upon requests to connect/disconnect components. Interestingly, support for interception is provided whereby a list of both pre-methods and post-methods can be dynamically attached to a component interface. Furthermore, a system graph of the components currently in use is maintained to support the introspection of a platform's structure. The OpenCOM environment, which has a memory footprint of 17 Kbytes, was developed for both Windows NT platforms and Pocket PC devices with StrongARM processors running the Windows CE operating system.

A prototype of the STEAM-like publish/subscribe system, called publish/subscribe Middleware for Mobile Computing (PSMMoC) [12], has been implemented on both Windows NT and the Compaq iPaq h3870 PCs running the Windows CE 3.0 operating system. In addition, a particular instantiation of the resource framework was developed in order to extend PSMMoC with resource management capabilities. The resource framework was implemented on Windows NT and is currently being ported to Windows CE.

Finally, we have developed the service discovery CF with two service discovery protocol implementations: SLP and UPnP. Allowing us to demonstrate how to overcome the problems of the availability of multiple service discovery protocols. However, it is feasible for any discovery protocol to be integrated into the framework.

5. Related Work

The Gaia project [18] developed at the University of Illinois is a distributed middleware infrastructure that provides support for ubiquitous computing. Although Gaia shares several common design goals with CORTEX, Gaia's main intended application domain is confined to fixed intelligent environments and lacks the support for ad hoc and real-time application scenarios. The EasyLiving project [19] from Microsoft focuses on development of an architecture and technologies for intelligent environments. It identifies several research efforts required on a variety of fronts, including middleware, geometric world modelling, sensing capabilities, and service description. The middleware, named InConcert, identifies the importance of having an asynchronous communication model for the coordination of entities contained in the environment. However, this approach does not fully address the concerns of ad-hoc and real time pervasive application scenarios.

In addition, several middleware services utilising the publish/subscribe communication model have been developed such as OMG's CORBA Event service specification [20], Sun Microsystems's Java Distributed event specification [21], TIBCO's TIB/RendezvousTM, SIENA [22] (a wide area event notification service), Gryphon [23] (a distributed content based notification service), and JEDI [24] (a Java event based distributed infrastructure). However, most of them assume a fixed network infrastructure and require the use of stationary event brokers. The Java Distributed event specification does not require event brokers because it uses peer to peer event model, but it requires system wide services like naming service for the subscribing entities to obtain references of interested publishing entities. Thus making them less suitable for pervasive and specially for mobile ad-hoc computing.

Regarding reflective middleware, work at Illinois has developed the Universal Interoperable Core (UIC) [25] which is a reflective middleware platform designed for handheld devices. Interestingly, the system can adopt different middleware personalities, e.g. a SOAP server and a CORBA server. The Reflective Middleware for Mobile Computing (ReMMoC) [15] platform is a similar approach which includes asynchronous middleware styles and heterogeneous discovery protocols.

6. Conclusions

We have presented the CORTEX architecture and its associated middleware architecture for the support of both pervasive and ad hoc computing. More specifically, the sentient object model was introduced as a central concept in the CORTEX architecture. This model provides key features for allowing both context-

awareness and intelligent autonomous behaviour. In addition, the middleware is constituted by a number of component frameworks. The frameworks allow for the pervasive dissemination of both non-critical and time-critical events in fixed and ad hoc environments. Support is also provided for the dynamic use of appropriate service location personalities.

Future work is looking at the development of the context CF as well as introducing further features into the supported CFs. Regarding the publish/subscribe system, we are looking at the inclusion of support for proximity groups. An application-level multicast facility is also being developed for the support of ad hoc routing. As concerns the resource management CF, we are collaborating with the University of Lisbon to provide support for the timing failure detection in an ad hoc communication environment. In this approach, fail-safe applications switch to a safe-state as soon as a failure is detected. Hence, work is being carried out for extending a timely computing base (TCB) [26] to an ad hoc environment.

Finally, we are working on the design and implementation of two application scenarios: an intelligent room and a cooperating car application. The former consists of an application whereby the electronic appliances in a room are controlled according to the user preferences, which are learned from his previous behaviour. The latter then includes a number of car robots which are controlled by Pocket PC handheld devices running Windows CE 3.0.

References

- [1] Szyperski, C. "Component Software: Beyond Object-Oriented Programming." Harlow, England, Addison-Wesley. 1998.
- [2] CORTEX. "Preliminary Definition of the CORTEX Programming Model. CORTEX Project." IST-2000-26031, Deliberable D2, March 2002.
- [3] Verissimo, P., V. Cahill, et al. "CORTEX: Towards Supporting Autonomous and Cooperating Sentient Entities (2002)." *In Proceedings of European Wireless 2002*, Florence, Italy. 2002.
- [4] Verissimo, P. and A. Casimiro. "Event-Driven Support of Real-Time Sentient Objects." *Eighth IEEE International Workshop on Object-oriented Realtime Dependable Systems (WORDS 2003)*, Guadalajara, Mexico. January 2003.
- [5] GmbH, R. B. "CAN Specification Version 2.0." 1991.
- [6] Rene Meier, V. C. "Steam: Event-based Middleware for Wireless Ad Hoc Networks." *In Proceeding of the International Workshop on Distributed Event-Based Systems (ICDSC/DEBS'02)*, Vienna, Austria. 2002. pp. 639-644
- [7] Blair, G. S., G. Coulson, et al. "The Design and Implementation of Open ORB version 2." *IEEE Distributed Systems Online Journal* 2(6): 2001.
- [8] Clarke, M., G. Coulson, et al. "An Efficient Component Model for the Construction of Adaptive Middleware." *IFIP/ACM Middleware 2001*, Heidelberg, Germany. November 2001.
- [9] Microsoft. "COM: Delivering on the Promises of Component Technology." Microsoft Corporation. 2000. <http://www.microsoft.com/com/default.asp>
- [10] Maes, P. "Concepts and Experiments in Computational Reflection." *Proc. ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'87)*, Orlando, FL USA. October 1987. 147-155
- [11] Kiczales, G., J. des Rivieres, et al. "The Art of the Metaobject Protocol.", MIT Press. 1991.
- [12] Sivaharan, T. "Publish/Subscribe Component-based Middleware for PDAs in Wireless Ad-hoc Networks." M. Sc., Lancaster University. 2002.
- [13] W3C. "Simple Object Access Protocol (SOAP) 1.1." 2000.
- [14] Preiss, B. "Data Structures and Algorithms with Object-Oriented Design Patterns in C++.", John Wiley & Sons, Inc. 1999.
- [15] Paul Grace, G. B. "Interoperating with Services in a Mobile Environment." *Accepted in Proc. ACM/IFIP International Middleware Conference (Middleware 2003)*, Rio de Janeiro, Brazil. June 2003.
- [16] Duran-Limon, H. A. and G. S. Blair. "Reconfiguration of Resources in Middleware." *Seventh IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2002)*, San Diego, CA. January 2002.
- [17] Duran-Limon, H. A., G. S. Blair, et al. "Resource Management for the Real-Time Support of an Embedded Publish/Subscribe System." *Submitted to the 9th IEEE Real-Time/Embedded Technology and Applications Symposium (RTAS'03)*, Toronto, Canada. 2003.
- [18] Manuel Roman, C. K. H., Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. "Gaia: A Middleware Infrastructure to Enable Active Spaces." *IEEE Pervasive Computing* Oct-Dec: pp. 74-83, 2002.
- [19] Brumitt, B., B. Meyers, et al. "EasyLiving: Technologies for Intelligent environment." *Handheld and Ubiquitous Computing* September: 2000.
- [20] Object Management Group, "CORBA Services: Common Object Services Specification." 95-3-31, Dec 1998.
- [21] Sun Microsystems, Inc, "Java Distributed Event Specification." July 1998. <http://www.javasoft.com/products/javaspaces/specs>.
- [22] Carzaniga, A., Rosenblum, D. and Wolf, A. "Design and Evaluation of a Wide-Area Event Notification Service." *ACM Transactions on Computer Systems* 19(3): pp 332-383, 2001.

- [23] IBM. "Gryphon: An Information Flow Based Approach to Message Brokering." IBM Research. 1998. <http://researchweb.watson.ibm.com/gryphon/home.html>
- [24] Cugola, G., Di Nitto, E., and Fuggetta, A. "The JEDI event-based infrastructure and its application to the development of the OPSS WFMS." *IEEE Transactions on Software Engineering* **9**(27): pp 827-850, September 2001.
- [25] Roman, M., Kon, F. and Campbell, R. H. "Reflective Middleware: From Your Desk to Your Hand." *IEEE DS Online*(Special Issue on Reflective Middleware): 2001.
- [26] Verissimo, P. and A. Casimiro. "The Timely Computing Base Model and Architecture." *Transaction on Computers - Special Issue on Asynchronous Real-Time Systems* **51**(8): August 2002.