# Novel Component Middleware for Building Dependable Sentient Computing Applications

Maomao Wu, Adrian Friday, Gordon Blair,
Thirunavukkarasu Sivaharan, Paul Okanda, and Hector Duran Limon
Department of Computing, Lancaster University,
Bailrigg, Lancaster, UK, LA1 4YR.
{maomao, adrian, t.sivaharan, okanda, gordon, duranlim}@comp.lancs.ac.uk

Carl-Fredrik Sørensen
Department of Computer and Information Science,
Norwegian University of Science and Technology (NTNU),
NO-7491 Trondheim, Norway.
carlfrs@idi.ntnu.no

Gregory Biegel and René Meier
Department of Computer Science,
University of Dublin, Trinity College,
Dublin 2, Ireland.
{Greg.Biegel, Rene.Meier}@cs.tcd.ie

**Abstract**

With advances in sensor-based computing and mobile communication, people have started to explore ubiquitous or pervasive computing systems that aim to have computing devices literally available everywhere, making them disappear into the physical environment. Novel ubiquitous computing applications such as intelligent vehicles, smart buildings, and traffic management have special properties that traditional computing applications do not possess, such as context-awareness, massive decentralisation, autonomous behaviour, adaptivity, proactivity, and innate collaboration. In this paper we argue that such applications require a new computational model and middleware that can reflect the autonomy and spontaneity of cooperative entities. The EU funded CORTEX[1] project proposes the sentient object model to support the construction of such large-scale applications. We report on a flexible, run-time reconfigurable component based middleware that we have used to engineer the sentient object programming paradigm. We demonstrate the appropriateness of the novel computational model and validity of the middleware by constructing a proof of concept demonstrator based on the notion of autonomous cooperating vehicles.

**Keywords:** Context-aware, Component, Middleware, Sentient object.

[1]The CORTEX (CO-operating Real-time senTient objects: architecture and EXperimental evaluation) project is supported by the Future and Emerging Technologies programme of the Commission of the European Union under research contract IST-FET-2000-26031, http://cortex.di.fc.ul.pt/.

# 1 Introduction

With advances in sensor-based computing and mobile communication, researchers have started to explore ubiquitous computing (UbiComp) systems [26, 27] that aim to have computing devices embedded literally everywhere, while making them disappear into the physical environment (e.g in our cars, buildings, soft furnishings, appliances, clothing etc.). Novel applications are possible in these environments, but many of the scenarios we can envision require elements to operate independently of direct human control. Among the most popular examples of these kinds of application are based around intelligent vehicles, traffic management systems, and smart buildings or working/living environments. We believe that there are a number of special characteristics that differentiate these classes of application from traditional computing applications, such as:

- **Sentience:** These applications are context-aware, i.e. have the ability to perceive the state of the surrounding environment, through the fusion and interpretation of information from possibly diverse sensors.
- **Autonomy:** Components of these applications will be capable of acting in a decentralised fashion, based solely on the acquisition of information from the environment and on their own knowledge.
- **Decentralisation:** There is no single central server that does intensive computation for the clients. Typical applications consists of components that might be scattered across geographical regions, e.g., street, buildings, cities, countries, and continents.
- **Proactivity:** These applications are able to act in anticipation of future goals or problems without direct human intervention. They should have a certain degree of intelligence, and be able to decide what action to take from gathered sensor data.
- **Adaptivity:** These applications will have to cope with changing conditions during their lifetimes. Not only must the applications be designed to evolve, but their underlying support must be adaptable as well.
- **Time and Safety Criticality:** These applications interact with physical environments and are required to provide real-time services to human users. It is important to provide real-time guarantees and dependability assurance through some system or middleware modules, e.g., resource management and configuration, timing failure detection and Quality of Service (QoS) management.

These characteristics make it extremely challenging for application designers and system engineers to design and implement ubiquitous computing systems and applications. We postulate that an appropriate computational model, programming abstraction and supporting middleware are crucial for realising the vision of UbiComp and assisting in constructing these forms of novel application.

In the EU funded CORTEX [25] project, we propose a programming model that we believe contributes an important abstraction for supporting the construction of these forms of application. In our programming model, applications are constructed from large numbers of software components which accept input (construct their view of the world) via a variety of sensors, and autonomously react, acting upon the environment using a variety of actuators. These components must have a certain level of "intelligence" to allow them to act autonomously based upon this acquisition of information, and should be able to cooperate with each other using a range of different networking technologies. We named these intelligent software components "sentient objects", and define a programming model for the development of such objects.

In this paper we describe our programming model in more detail (section 2) and report on our experiences of building middleware to support this model (sections 3 and 4). Our component based middleware in particular offers a number of important features including flexible, run-time configuration and reconfiguration, and a novel unified mechanism for supporting both context-aware and QoS adaptation. To demonstrate the appropriateness of the paradigm and the validity of the middleware, we have also constructed a proof of concept demonstrator based on the notion of autonomous cooperating vehicles (section 5). Section 6 contrasts our approach

with existing approaches in the area of middleware for ubiquitous computing. Finally, we summarise our experiences in our concluding remarks and propose some future work.

## 2 Sentient Object Programming Model

In the sentient object programming model [3, 7], software entities are categorised into sensors, actuators, and sentient objects. Sensors are defined as entities that produce software events in reaction to a stimulus detected by some real-world hardware device. An actuator is defined as an entity that consumes software events and reacts by attempting to change the state of the real world in some way via some hardware device. Both of these may be a software abstraction of actual physical devices. A sentient object is then defined as an entity that can both consume and produce software events, and lies in some control path between at least one sensor and one actuator.

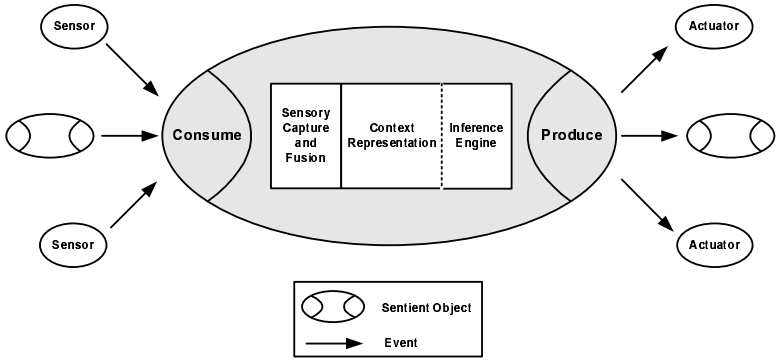


Figure 1: Sentient Object Model

Sentient objects are cooperative and communicate with each other and with sensors and actuators via an anonymous generative event based communication paradigm [17], permitting loose coupling between objects which supports component reconfiguration and application evolution. A sentient object and its internals are illustrated in Figure 1. The novel event-based communication mechanism incorporated in the model is specifically designed for mobile ad-hoc wireless environments such as those typified spontaneous device interactions and networking found in embedded systems (e.g. mobile computing, sensor networking, cooperative vehicles, etc.).

Three main components have been identified [7] in order for a sentient object to be context-aware: sensory capture, context representation, and an inference engine. The major issues in the area of sensory capture are data filtering and sensor fusion. Probabilistic sensor fusion techniques can be used to provide more accurate estimates of contexts, and dead-reckoning can be employed when there is insufficient traffic or delays in sensor data. The context representation component deals with the representation of context information in a way that is useful to the sentient object and may be easily exchanged among sentient objects. The inference engine is analogous to the brain of a sentient object, and all the intelligent behaviour of the sentient objects are attributable to the inference engine. It has the capability of generating high-level context from low-level context or sensor data, and making actuation decisions on how to react based on this contextual data.

## 3 Middleware Design Challenges

Middleware can conceal programming complexity from application developers, and provide services to facilitate communication and coordination of distributed software components. Instead of realising the sentient object programming model directly, we decided to design and implement

a flexible middleware so that it can be reused in different circumstances. Some key research challenges we aimed to address in building our middleware were as follows:

- **Context-awareness:** We follow the sentient object paradigm to handle inputs from diverse sources, e.g., different sensors or other sentient objects. Uncertainty is a major problem in sensing the environment due to the inherent limitations of sensors with respect to accuracy and precision. This has led to a crucial requirement for our middleware that it provides uncertainty management for software components whose actions are based on environmental perception. Additionally, "intelligent" software components that reason based on context are required in order to make sentient objects autonomous and proactive.
- **QoS management and fail safety:** Due to the real-time nature of ubiquitous computing applications, the middleware needs to take into account the provision of incremental real-time and reliability guarantees. QoS properties need to be expressed as a metric of predictability in terms of timeliness and reliability. For distributed objects coordinating in uncertain environments, the timing bounds for distributed actions could be violated because of the timing failures. This requires a reliable timing failure detection service for distributed operations.
- **Communication model:** Traditional communication models, such as client-server and the RPC paradigm, are not well suited to mobile ad-hoc environment, because there is no fixed infrastructure to host centralised services. Since disconnections are common in the wireless communication environment, the communication paradigm should be decoupled and asynchronous. Moreover, in novel applications with mobile or context-based elements, the scope of information dissemination is dynamically determined by spatial parameters. For example, in the cooperating cars scenario, one might wish to limit dissemination to those vehicles directly affected by an obstacle on the road, and the information is only valid in a restricted geographical area.
- **Routing in mobile ad-hoc environment:** In mobile ad-hoc networks, the senders and receivers move constantly so that the network topology frequently changes. This poses a challenge for routing packets in such dynamic environments. Multicast protocols based on proactive and reactive ad-hoc routing, using shared state kept in the forms of routes and adjacent information, is useful in environments with low node mobility. However, this shared state and topology information can quickly become outdated in the highly mobile environments. Hence, it requires a new type of routing protocol in highly dynamic ad-hoc network environments.

We believe middleware support encourages the widespread development of aforementioned novel applications. It is always important for middleware designer to bear in mind the issue of reusability, configurability and reconfigurability. Previous research experience [8] in reflective middleware also motivates us to use reflection, component technology, and component frameworks to create a flexible and dynamically configurable middleware platform. In the following section we discuss how these challenges have been addressed by the component frameworks that are used to compose our sentient objects.

# 4 Component-oriented Reflective Middleware

## 4.1 Component Model and Component Frameworks

The autonomous feature of sentient objects requires that they adapt based on contextual information, either communicated as high level contexts (e.g. as software events) or derived directly from raw sensor information. To move from low level sensing to discernable contexts requires both conditioning of the data (e.g. fusion and interpolation) and higher order reasoning (rule based inferring). Clearly the demands of a given application and its associated environment will vary over time and situation, both in terms of fine grained adaptation (tuning parameters on a particular fusion algorithm for example), and at a courser grained level (switching behaviours

or between information sources). To support this level of configuration and reconfiguration we have chosen to build our middleware based on a reflective component model called OpenCOM [8]. This technology allows use to introspect the running system and adapt any aspect of the system at run-time, permitting us ultimate flexibility.

OpenCOM is built atop a subset of Microsoft's COM. Ignoring COM's advanced features, OpenCOM keeps its core aspects including the binary level interoperability standard, Microsoft's IDL, COM's globally unique identifiers and the IUnknown interface as the basis of its implementation. The fundamental concepts of OpenCOM are interfaces, receptacles and connections (bindings between interface and receptacles). An interface expresses a unit of service provision while a receptacle describes a unit of service requirement. OpenCOM deploys a standard runtime to manage the creation and deletion of components, and to act upon requests to connect and disconnect components. Moreover, a system graph of the running components is maintained to support the introspection of a platform's structure.

Component Frameworks (CFs) were originally defined in [23] as "collections of rules and interfaces that govern the interaction of a set of components plugged into them". CFs constrain the design space and scope for evolution, so that they are targeted at a specific domain and embody "rules and interfaces" that make sense in that domain. To realise sentient objects, the OpenCOM component model has been employed to construct families of middleware, each of which is in turn created as a set of configurable CFs. We have designed and implemented a number of CFs, including a Publish-Subscribe event channel CF, a Context CF, and a Resource and a QoS CF. As a result, reflection can be used to discover the current middleware configuration and behaviour, and middleware configuration can also be changed at run-time.

## 4.2 Publish-Subscribe CF

The Publish-Subscribe (P-S) CF is a componentised prototype of the STEAM P-S system [17]. STEAM is based on an implicit event model and has been designed for mobile applications and ad hoc networks. STEAM differs from other P-S systems in that it does not rely on the presence of any separate infrastructure and supports distributed techniques for identifying and delivering events of interest based on location. STEAM supports a decentralised approach for discovering peers, for routing event notifications using a distributed addressing scheme, and for event filtering based on combining multiple filters. Filters may be applied to the subject and the content of events, and may be used to define geographical areas within which events are valid. Such proximity-based filtering represents a natural way to filter events of interest in mobile applications. The P-S CF support both publisher and subscriber side filtering, by using a query or subscription language called Filter Event Language (FEL) [22] to express their preferences. FEL can be used to create subject, content, and context filters, which are also componentised and can be dynamically reconfigured.

Publishers and subscribers are anonymous, and subscribers should be able to interpret the events without a priori knowledge. Therefore, we have the need for a generic event dialect which can be understood by all publishers and subscribers in the system. The events in the P-S CF are represented in XML, and a generic XML profile defines the generic event dialect. The XML based events provide for easy interoperability and extensibility of the event dialect.

Events are published to a notional event channel allowing one-to-many distribution of events. The event channel is supported by an underlying communication mechanism supported by a group abstraction. The Group Communication CF provides a range of group communication protocols. The Publish-Subscribe CF can flexibly select a group communication protocol from Group Communication CF, which currently supports a probabilistic ad-hoc multicast protocol, an IP multicast based protocol and a local (shared memory based) group communication protocol. The P-S CF is used to construct various event channels, which are required by the Context CF (as describe below) and inter-sentient object communication.

## 4.3 Context CF

The Context CF consists of two parts: sensor capture and fusion, and the inference engine.

### 4.3.1 Sensor Capture and Fusion

The sensory capture and fusion components are able to receive sensor data through an event channel and perform some data fusion algorithm in order to manage uncertainty of sensor data and derive higher level context information from multi-modal data sources. We have explored three different techniques for our middleware component design.

One of the widely-used methods in sensor fusion and machine learning is the Gaussian modelling and multivariate Gaussian modelling [14]. Raw sensor data samples associated with certain target values are gathered first to establish some multivariate Gaussian distribution functions [2]. We collected ultrasonic sensor' readings at certain target values, i.e. various distances from the sensors to the obstacle ahead. When new sensor readings arrive they are fed into the established Gaussian distribution functions. The outputs of these functions are compared, and the target value associated with the distribution function generating the highest output is the most probable target value. In the ultrasonic sensor example, we can obtain the most probable distance to the obstacle from the current sensor readings by feeding the current sensor readings to the various Gaussian distribution functions associated with different target values. Multivariate Gaussian is the one of the most important methods to get the best statistical target value for current sensor readings.

Another probabilistic sensor fusion scheme is also employed, based upon Bayesian networks [12], which provides a powerful mechanism for measuring the effectiveness of derivations of context from noisy sensor data. A Bayesian network is a directed acyclic graph in which nodes represent random variables and the absence of arcs represents conditional independence. A Bayesian network graph allows us to manage the exponential increase in the size of a joint probability distribution over a set of random variables, by exploiting conditional independence: $P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i|pa(X_i))$ where $pa(X_i)$ are the parents of node $X_i$ and $P(X_i|pa(X_i))$ is the conditional distribution of $X_i$ given its parents. Using Bayesian networks to model the uncertainty of sensor data and the dependencies between a set of sensors, gives us the ability to efficiently reason that a hypothesis is true, given available evidence from sensors.

Finally, dead-reckoning has been used to compensate for insufficient data and latency in transmissions. Instead of relying on the latent current sensor readings for fusion, historical data can be processed during run-time to generate more reliable and real-time values. This technique is especially useful for sensors, such as GPS, that may have unpredictable delays. In the case of GPS, objects should be able to calculate their own dead-reckoned position and update their own trajectory using extrapolation algorithms to make up for this latency. They could compute their current position based on predictive algorithms that have a basis on a history log such that there is continuity and timeliness in otherwise sporadic and delayed data from a source.

### 4.3.2 Inference Engine

The inference engine is a key part of the sentient object paradigm, and the intelligence of a sentient object is realised by the inference engine and its associated knowledge base. An inference engine in artificial intelligence terms, refers to a program that reasons about a set of rules (a knowledge base) in order to derive an output [7]. An inference engine is actually a part of an expert system, which also contains a domain-specific knowledge base. The knowledge base contains the knowledge required to solve a certain problem, encoded as a set of production rules. As a result, contexts can be represented and stored as some facts [19] within the inference engine.

We chose the well-known rule-based inference engine CLIPS — C Language Integrated Production System [19] as a base of our middleware component design. CLIPS provides a cohesive tool for handling a wide variety of knowledge with support for three different programming paradigms: rule-based, object-oriented and procedural, and the internal implementation of

[2] A Gaussian distribution is also known as a normal distribution, which has the form of

$$G(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

where $\mu$ is the mean or average vector, and $\Sigma$ refers to the covariance matrix

CLIPS is based upon the RETE net [10]. Using the rule-based programming paradigm provided by CLIPS, we can specify rules to generate high-level contexts from fused sensor data or derived low-level contexts. A CLIPS rule consists of two parts: an *if* part and a *then* part. The if part of a rule is a series of patterns which specify the facts (or data) that cause the rule to be applicable. The then part of a rule is the set of actions to be executed when the rule is applicable. Here is an example of CLIPS rule named will be triggered when obstacle distance is near. The if part of the rule contains two facts, "car-id" and "obstacle". CLIPS automatically matches facts against the patterns specified in the if part of the rules, and executes the actions (then part) of applicable rules. The sample rule activates two actions: "retract" or delete the fact referenced and call user defined function "publish". CLIPS also provides ways for the programmers to define their own functions, which can also be specified in the rules. This makes it possible to automatically perform actuation when certain contexts are derived from the inference engine. The user defined function "publish" in the sample actually makes use of the P-S CF to publish the "stop" command to the car speed control event channel. This use of inference on the basis of contextual information makes it possible to do Context-based Reasoning (CxBR) [11].

```
(defrule rule-obstacle-near "CLIPS rule for obstacle near"
    (car-id (id ?id))
    ?f1 <- (obstacle (distance near))
  =>
    (retract ?f1)
    (publish ?id stop)
)
```

One of the important features of our approach is that the paradigm facilitates uniform treatment of both context and QoS. End-to-End QoS violations are provided by the Timely Computing Base (TCB) [5]. The TCB provides the facility to monitor timeliness of event delivery on distributed event channels, thus providing estimations and awareness of timing failure probability for a given required stability coverage. The detection of timing failures by TCB can be used to trigger the timely execution of fail safe-procedures and adaptation strategies. QoS rules can be included in the inference engine that trigger adaptations based on changes in measure QoS or coverage stability from the TCB (see the following sample rule below that slows the vehicle in our demonstrator when intra-vehicle communication becomes too unpredictable).

```
(defrule rule-network-coverage-bad "CLIPS rule for network coverage bad"
    (car-id (id ?id))
    ?f1 <- (network-coverage (value bad))
  =>
    (retract ?f1)
    (publish ?id slow)
)
```

Note that this uniform approaches allows QoS to become part of context descriptions. Moreover, both changes in context and/ or changes in QoS can trigger adaptations and actuations.

We have wrapped the CLIPS into a DLL and created OpenCOM components for both WinNT and WinCE. This component can perform adaptation at the rule-level (by loading and unloading different rules at run-time).

# 5 Autonomous Cooperative Vehicles Application

To demonstrate the appropriateness of the sentient object model and validity of the componentised middleware, we constructed a proof of concept demonstrator based on the notion of autonomous cooperating vehicles. The vehicles have the objective of travelling on a given path (a virtual circuit), predefined by a set of GPS waypoints. Each vehicle needs to build a real-time

image of its surrounding environment within some bounded error to make informed decisions regarding its next move. The cooperation between vehicles is critical to avoid collisions, to follow a leading vehicle and to travel safely. The vehicles also need to obey external traffic signals and give way to pedestrians who cross the road by sensing their presence, and adapt to QoS data such as network coverage stability.

We have built a small number of autonomous cooperative vehicles by modifying remote controlled robot cars and augmenting them with multiple sensors and driven them by our software running on iPAQ PocketPC2002s. A GPS receiver is used to sense location, a digital compass is integrated to sense direction, and eight units of ultrasonic sensors are fixed to detect the presence of neighbouring physical objects. The PocketPC has two WaveLAN cards configured in ad-hoc mode. One is exclusive to the TCB control channel, and the other is for the event channels (payload) used for inter-vehicle communications. Each vehicle uses an onboard Controller Area Network (CAN) [20] to connect the hardware devices, e.g, sensors, actuators and iPAQ. The onboard network is a bespoke ring topology with single break failure resilience, and the design enables addition of further devices on to the ring in a plug and play fashion.

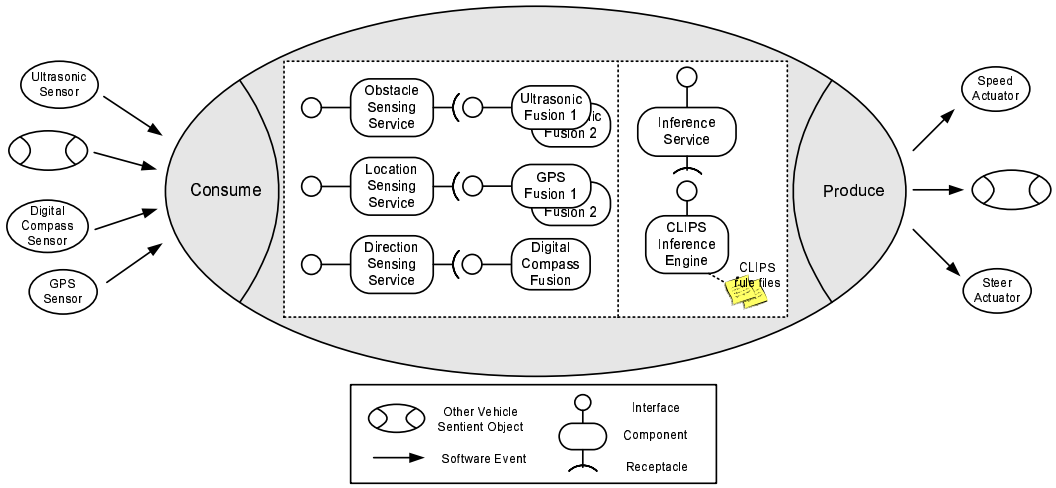


Figure 2: Middleware Configuration for Autonomous Cooperative Vehicles Application

The demonstrator application — "autonomous cooperating sentient vehicles" is implemented using instances of our middleware CFs, and the detailed configuration of the software components is illustrated in Figure 2. Different sensing service components are responsible for receiving related software events from a particular sensor and fusing the sensor data. These sensing service components make use of data fusion algorithm components, which can be plugged in and out at run-time. For example, the obstacle distance sensing service can dynamically swap between two different ultrasonic sensor fusion components: Gaussian modelling and a customised fusion algorithm. Context information is derived from these fusion components, and is fed into the inference service component. The inference service component makes use of the CLIPS inference engine to derive higher level contexts and decide actuation actions. The CLIPS rules are predefined in CLIPS script files that can be dynamically loaded into and unloaded from the inference engine. We have three sets of independent rules to do speed control, steer control of the vehicle, and derive network coverage stability context. By matching the current contexts to the actuation rules, the inference engine component decides how to actuate the vehicle by transmitting software events to actuators, which in turn interact with the hardware to fulfill the task.

The rulesets can send special adaptation "commands" using the event channel to trigger middleware component level adaptation (both swapping components to change algorithm and finer grained internal tuning of algorithms) and CLIPS rule level adaptation (which changes behaviour of the system). We have found this simple yet elegantly consistent approach to context

and QoS adaptation provides us with with a great deal of expressive power and flexibility in our demonstration.

# 6 Related Work

A tremendous amount of activity, particularly from the car manufacturers themselves, has been taken under the former category of application. In BMW's ConnectedDriv [2], research vehicles can communicate with each other by using ad-hoc network networking technology, e.g., Wireless LAN. Each vehicle acts as a sensor registering and monitoring the road traffic, and it can recognise congestion traffic and transmit a local traffic report to other vehicles in the nearby area, requiring no human intervention. Motorola laboratory in Arizona, US, Delphi Safety & Interior Systems in Michigan, US and DaimlerChrysler research centre in Ulm, Germany [13] are all trying to build very similar "smart cars" that will make the road travel safer. By monitoring the driver's physical status, such as body movements, eye-blink pattern, and respiration, as well as driving behaviour, e.g., the number and the severity of steering corrections, smart cars are proposed to be able to either alert the driver by some means, e.g., sound or vibration, or even take over some tasks from the driver, such as braking or steering. The Intelligent Transportation Systems (ITS) program at the General Motors [1] is more ambitious, which proposes to harmonize traffic flow, e.g., reducing speed fluctuations and traffic shock waves, and maximising highway capacity by not only constructing closely coordinated vehicles but also developing automated highways. Such complex real time dependable applications are difficult to engineer using traditional systems development techniques and paradigms. We believe the sentient object paradigm may highlight a useful approach that can be adopted by the developers of systems such as these. Moreover, our paradigm offers intrinsic scope for flexibility, reuse and reconfiguration to offer new behaviours and applications.

Different approaches of reflective middleware in Ubicomp have been investigated, e.g., Gaia and UIC. Gaia [21] is a metaoperating system built as a distributed middleware infrastructure that coordinates software entities and heterogeneous networked devices contained in a physical space. Gaia supports development and execution of portable applications in *active spaces*. Active spaces are programmable ubiquitous computing environments in which users interact with several devices and services simultaneously. Users, services, data, and locations are represented in the active spaces, and are manipulated dynamically and in coordination. Context is important to Gaia's applications, which have access to context via a "context file system". However, context is externalised and is not a first class entity that drives the behaviour of the active space systems. We believe that active space applications could easily be modelled as sentient objects and potentially benefit from our paradigm.

Reflective middleware has been adopted by other projects interested in adaptation and reconfigurability. UIC (Universally Interoperable Core) [24] is targeted to mobile devices as a minimal reflective middleware. Heterogeneity issues are solved by UIC in a set of pluggable components that allow developers to specialise the middleware to different devices and environments. The configuration of the middleware can be updated both at compile- and run-time.

The Mobile Platform for Actively Deployable Service (MobiPADS) system [6] is another reflective-based middleware designed to support context-aware processing by providing an execution platform to enable active service deployment and reconfiguration of the service composition in response to environments of varying contexts. MobiPADS supports dynamic adaptation at both the middleware and application layers to provide flexible configuration of resources to optimize the operations of mobile applications. To alleviate the adverse conditions of a wireless environment, services (known as mobilets) are configured as chained service objects to provide augmented services to the underlying mobile applications. The reflective model in MobiPADS provides metainterfaces to make applications able to directly participate in computation adaptation in response to the changing context. A mobile application can access contextual information, the service configuration and adaptation strategy, and examine and modify these entities to obtain optimal service provision through the metalevel object representation of the internal event system and service reconfiguration mechanism.

Also other approaches have been developed that focus on meta-data representation for services and application-dependent policies. Capra et al. [4] have e.g. developed mechanisms for dynamic adaptation and conflict resolutions. Several approaches to data-sharing middleware for mobile and ad hoc environments have been developed the last decade. Many of these are based on tuple spaces (e.g. LIME [18], TOTA [15], and $L^2imbo$ [9]). The approaches have in common that they aim to let data be de-coupled and de-centralised. Data sharing though XML-based structures have been proposed in e.g. XMIDDLE [16] to support more complex data models than what a tuple represent.

# 7 Concluding Remarks

This paper has described a novel componentised reflective middleware for engineering dependable sentient computing applications based on the sentient object model. A proof of concept demonstrator based on the notion of autonomous cooperating vehicles has been fully implemented to investigate the appropriateness of the novel computational model and validity of the middleware. Based on the experience of designing such a flexible middleware and engineering the fully-fledged demonstrator, we can identify a number of key results:

- The sentient object model has proved to be an excellent programming abstraction for the development of real-time, cooperative, context-aware applications, particularly because of its intrinsic support for decoupled event channel communication and context-awareness. The demonstrator shows it can be used to handle both context adaptation, e.g., responding to the traffic light signals, and QoS adaptation, e.g., reacting when network coverage stability is low, in a self-consistent and uniform way.
- OpenCOM component model and CFs are the two key enabling technologies underpinning the construction of the middleware. The component-oriented approach for engineering the middleware offers benefits of flexible configuration and reconfiguration of the middleware components, supporting the development of context and QoS adaptive applications. For example, the sentient object instance constructed from CFs can dynamically reconfigure its internal architecture by responding to events from the event channels, e.g, the sensor fusion service components can plug out and in different sensor fusion components at runtime.
- The middleware architecture also provides the management of non-functional concerns such as timeliness and reliability properties. The integration of these modules is greatly eased by following the sentient object model. For example, the data from TCB is distributed using the anonymous communication paradigm, and it in turn goes through the inference engine to derive high-level network coverage stability context and trigger actuation events being published to the car speed control channel.

We believe that our middleware is reusable, and we are keen to investigate the generality of our approach by applying our middleware to other application domains involving embedded autonomous components. This is a key aspect of our future work.

# Acknowledgements

We would like to thank Joe Finney and Angie Chandler for the construction of the robot cars augmented with a token ring network with an array of sensors. Trimble Component Technology Europe also supported the construction of the demonstrator by providing us free GPS development kits.

# References


[1] Steven Ashley. Smart Cars and Automated Highways. *Mechanical Engineering Magzine*, May 1998. http://www.memagazine.org/backissues/may98/features/smarter/smarter.html.

[2] Automotive Intelligence News. Talking Cars For Less Congestion — The Future Of Telematics. http://www.autointell-news.com/News-2003/October-2003/October-2003-1/October-01-03-p6.htm, 2003.

[3] Gregory Biegel and Vinny Cahill. A Framework for Developing Mobile, Context-aware Applications. In *2nd IEEE Conference on Pervasive Computing and Communications (Percom 2004)*, pages 361–365, Orlando, Florida, March 14-17 2004. IEEE Computer Society Press.

[4] Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo. A Micro-Economic Approach to Conflict Resolution in Mobile Computing. *ACM SIGSOFT Software Engineering Notes*, 27(6):31–40, 2002.

[5] Antonio Casimiro and Paulo Verissimo. Using the Timely Computing Base for Dependable QoS Adaptation. In *20th IEEE Symposium on Reliable Distributed Systems*, pages 208–217. IEEE Computer Society Press, 2001.

[6] Alvin T.S. Chan and Siu-Nam Chuang. MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing. *IEEE Transactions on Software Engineering*, 29(12):1072–1085, 2003.

[7] CORTEX. Final definition of CORTEX programming model. Technical Report CORTEX Deliverable D6 version 1.0, CORTEX, April, 2003.

[8] Geoff Coulson, Gordon S. Blair, Michael Clark, and Nikolaos Parlavantzas. The Design of a Highly Configurable and Reconfigurable Middleware Platform. *ACM Distributed Computing Journal*, 15(2):109–126, April 2002.

[9] Nigel Davies, Adrian Friday, Stephen P. Wade, and Gordon S. Blair. $L^2$imbo: A Distributed Systems Platform for Mobile Computing. *Mobile Networks and Applications – Special Issue on Protocols and Software Paradigms of Mobile Networks*, 3(2):143–156, August 1998. Internal report number MPG-97-03.

[10] Charles Lanny Forgy. RETE: A Fast Algorithm for the Many Patterns/Many Objects Pattern Match Problem. *Artificial Intelligence*, 19(1):17–37, September 1982.

[11] Avelino J. Gonzalez and Robert Ahlers. Context-Based Representation of Intelligent Behaviour in Training Simulations. *Transactions of the Society for Computer Simulation International*, 15(4):153–166, 1999.

[12] Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer Verlag, 2001.

[13] Willie D. Jones. Building Safer Cars. http://www.spectrum.ieee.org/WEBONLY/publicfeature/jan02/road.html, 2002.

[14] A. Schmidt G. Kortuem K. Van Laerhoven, N. Villar and H.-W. Gellersen. Using an Autonomous Cube for Basic Navigation and Input. In *ICMI/PUI 2003*, pages 203–211. ISBN: 1-58113-621-8, ACM Press, 2003.

[15] Marco Mamei and Franco Zambonelli. Programming Pervasive and Mobile Computing Applications with the TOTA Middleware. In *Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, pages 263–273, Orlando, Florida, March 14-17 2004. IEEE Computer Society Press.

[16] Cecilia Mascolo, Licia Capra, Stefanos Zachariadis, and Wolfgang Emmerich. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *Personal and Wireless Communications Journal*, 21(1), April 2002.

[17] René Meier and Vinny Cahill. Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications. In *4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'03)*, pages 285–296, Paris, France, 2003. Springer Verlag, Heidelberg, Germany. LNCS 2893.

[18] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. Developing Mobile Computing Applications with LIME. In *22nd International Conference on Software Engineering (ICSE'2000)*, pages 766–769, Limerick, Ireland, 2000. ACM Press.

[19] Gary Riley. CLIPS homepage. http://www.ghg.net/clips/CLIPS.html, 2002.

[20] Robert Bosch GmbH. CAN Homepage of Robert Bosch GmbH. http://www.can.bosch.com.

[21] Manuel Román, Christoher Hess, Renato Cerqueira, Anand Ranganathan, Roy T. Campbell, and Klara Nahrstedt. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–82, October–December 2002.

[22] Thirunavukkarasu Sivaharan. Publish Subscribe Component-based Middleware for PDAs in Wireless Ad-hoc Network. Technical report, Lancaster University, 2002. MSc thesis.

[23] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1997.

[24] UbiCore. Universally Interoperable Core. http://www.ubi-core.com, Last accessed August 5th 2003.

[25] Universidade de Lisboa, Lancaster University, Trinity College Dublin and Universität Ulm. CORTEX project homepage. http://cortex.di.fc.ul.pt, 2001.

[26] Mark Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36(7):75–84, July 1993.

[27] Mark Weiser. The Computer for the 21st Century. *IEEE Pervasive Computing*, 1(1):18–25, January-March 2002. Reprinted from Scientific American, 1991.