AUTONOMIC MIDDLEWARE INFRASTRUCTURE FOR CONTEXT-AWARE UBIQUITOUS COMPUTING SYSTEMS

by

aCAMUS Team[1]

Technical Report V.2.0.0

RTMM, Kyung Hee University

07 July. 2006

Approved by ____
Project Supervisor

____

Date : 07 July. 2006

[1] Kamrul Hasan, Lenin Mehedy, Ngo Quoc Hung, Sungyoung Lee, Wang Jin, & Young Koo Lee. (Note: All in alphabetical order)

RTMM, Kyung Hee University

ABSTRACT

AUTONOMIC MIDDLEWARE INFRASTRUCTURE FOR CONTEXT-AWARE UBIQUITOUS COMPUTING SYSTEMS

by aCAMUS Team

Project Supervisor: Professor Sungyoung Lee
Department of Computer Engineering

Current research trend in context-aware ubiquitous systems is to make it more autonomous. The autonomic computing approach calls for the design of a system in such a way that it is aware of its constituting components and their details, it can reconfigure itself dynamically according to the change in its environment, optimize its working to achieve its goals and predict or recognize faults and problems in its work flow and rectify them.

All the autonomic concepts require that the system should be able to know the state of its software and hardware at all points. The state of the sensors and actuators which constitute the entire perception mechanism of the system is of vital importance. Knowing the state of the perception mechanism would help an autonomic system to determine the policies required for self-optimization, the current state of the perception mechanism, which quantities/events can be sensed, and in case of possible faults in the perception mechanism the system should be able to carry out isolation or self-healing policies. These requirements stress the need for a fault detection mechanism in context-aware and autonomic ubiquitous systems. This fault detection module should be able to correctly represent the system state at all time and at the same time it should be able to detect any anomaly in perceived data. Currently scenario based fault detection (to constrain the sensors and actuators to behave in a certain pre-determined fashion) using Bayesian Net is being developed in auto-CAMUS.

Recent work in self-healing largely involves reactive fault handling and healing based on feedback control i.e., the corrective steps and measures are undertaken on the basis of instantaneous need, or at most on the basis of short-term historical measurements. There are some obvious drawbacks of this approach e.g., it is static in nature, potentially large variations in performance may occur and the middleware may give slow response to change (lack of adaptability). We propose to incorporate predictive autonomic measures and approaches for self-healing in CAMUS.

Ubiquitous computing envisions the use of embedded processing in everyday objects interacting with each other and with the user by means of ad hoc wireless networking. Such ambient intelligence available almost everywhere enables new or better applications and services as well as performance improvements. In near future we can even envision self-organizing dust-like devices share their knowledge in order to make decisions and perform actions. Facilitating the cooperation between the nodes in these kinds of networks creates new requirements on programming and system technologies that can glue all these things together.

Besides, deriving the user's context from a set of collaborating smart objects distributed throughout the environment constitutes a major part of a smart object's application behavior. Information about an object's own context makes it possible to dynamically form groups of collaborating artifacts, and to form networking structures that can make collaborative context recognition more efficient.

Our initial effort in this low level sensor layer is to define a self-organizing network concept and technology that can be situated in multiple and dynamic contexts, ranging from sensor networks to virtual networks of humans. Being targeted to embedded systems, the proposed solutions also deal with many different types of design constraints. Swarm Intelligence and Ant Colony Optimization are being investigated for achieving autonomous and decentralized optimization strategies.

We also seek for the programming abstraction with collaboration group-centric to allow program developers to focus on the collaborative task in the real world, hiding the underlying mechanisms e.g. event handling, networking, etc. On top of this group-centric self-organizing network, we are developing decentralized detection and learning for collaborative context recognition using Sequential Bayesian Estimation theory.

As a future direction, we will also tackle the security and trustworthiness of this distributed communication system by embedding security and trust rules in network functionality.

Table of Contents

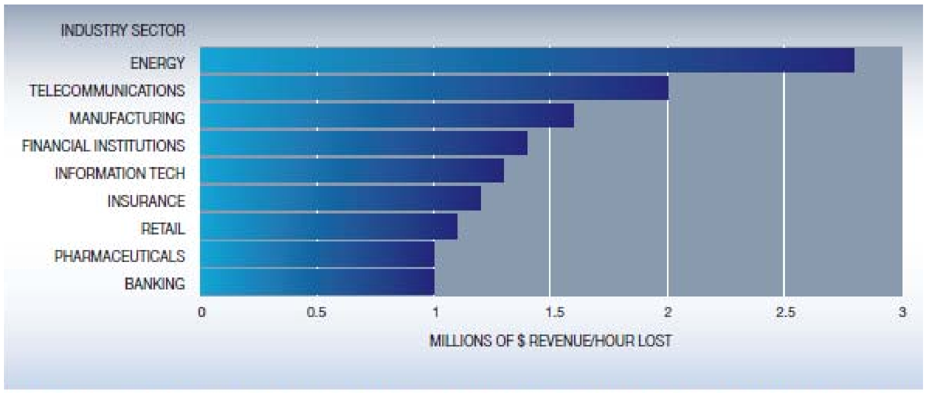
C h a p t e r 1

INTRODUCTION TO AUTONOMIC CAMUS

1 Introduction

1.1 Autonomic Computing

The essence of autonomic computing is self-management. Automating the management of computing resources is not a new problem for computer scientists. For decades, hardware and software systems have been evolving to deal with the increasing complexity of system control, resource sharing, and operational management. Autonomic computing [1] is another emerging paradigm to address this increasingly complex and distributed computing environment. Especially in the recent times, with the advent of Internet, distributed and mobile computing the system complexities has increased to a dramatically large scale.



Data from *IT Performance Engineering and Measurement Strategies: Quantifying Performance Loss*, Meta Group, Stamford, CT (October 2000).

Fig 1.1 Downtime: Average hourly impact

According to many researchers, IT industry's exploitation of the technologies in accordance with Moore's law has led to the verge of a complexity crisis. Today, complex systems require more and more skilled IT professionals to install, configure, operate, tune and maintain them. The economic motivation for autonomic systems is also obvious form the Figure 1.1.

The autonomic computing approach calls for the design of a system in such a way that it is aware of its constituting components and their details, it can reconfigure itself dynamically according to the change in its environment, optimize its working to achieve its goals and predict or recognize faults and problems in its work flow and rectify them. The inspiration of such self-managing approach has been taken from the autonomous nervous systems of biological entities and many concerted efforts are underway for the development of this field. The effect of such an autonomous system will be the reduction in the complexity and ease in management of a large system. Various ways have been proposed to achieve the fulfillment of this vision, from agent-based to policy-based self-management, from autonomic elements control loops to adaptive systems, self-stabilizing systems and many more [2], [3] [4], but the motivation and ultimate goal of all is the same.

Now, in order to integrate these characteristics in a self-managing system, the autonomic computing research community has identified various "self" properties that should be the hallmark of an autonomic system.

1.1.1 Self-Configuration

An autonomic system or component should be able to adapt automatically to dynamically changing environments. When a system or a component is able to define and redefine itself dynamically, it can be called self-configuring.

1.1.2 Self-Healing

An autonomic system of component should be able to automatically discover, diagnose, and correct faults. The objective of this property is to minimize all down-times and system crashes in order to provide maximum availability without human intervention.

Self-Configuring
Self-Healing
Self-Optimizing
Self-Protecting

Fig 1.2 Autonomic system properties

1.1.3 Self-Optimization

An autonomic system of component should be able to automatically monitor and adapt resources to ensure optimal functioning regarding the defined requirements. The objective of this property is to efficiently maximize the resource utilization of the system, even in dynamic and unpredictable environments.

1.1.4 Self-Protection

An autonomic system of component should be able to automatically anticipate, identify, and protect against arbitrary attacks. To minimize human intervention, the system should have the capability to differentiate in the usage of different security mechanism based upon the context and environment of the system.

Presently, there are no standards defined that point the path that leads towards the development of truly autonomic systems, even if the goal of a truly autonomic system based on Turing machines is contended [5]. However, researchers have already solved some problems by defining techniques and methods that help in monitoring resources and constitute autonomic elements as the building blocks of an autonomic system [6]. However, many problems involved with making management tasks autonomous are still largely unsolved. One of the biggest challenges in that regard is the development of closed control loops.

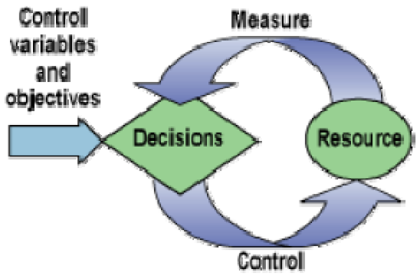


Fig 1.3 A closed control loop concept for an Autonomic Element

The fundamental idea of control loops is well known from a wide area of technical applications, for example a thermostat (consisting of a temperature sensor and a coupled flow control valve) and a car's antilock breaking system. However, modern distributed computing systems are many times more complex than these two examples.

1.2 Autonomic CAMUS

The smart office scenario can be used as the main motivation for the development of an autonomic and context-aware middleware for ubiquitous computing like auto-CAMUS. The term autonomic as related to the design of such a middleware is somewhat ambiguous. We need to consider the granularity of autonomy which should be provided to the system.

One way of providing autonomy to the system would be to provide autonomous behavior to the system considering only the interfaces which interact with the environment and the applications.

Another approach would be to provide autonomy to every member component and module. This approach would require function specific issues for autonomy. For example in the knowledge-base management module, self-healing would be needed whenever some data is missing from a specific case, and self-reconfiguration would include data-mining routines for adaptation.

The first approach in which we only provide autonomy to the external interfaces of the system, the system would exhibit autonomic behavior but only in its interaction with other applications and users. It would not be completely autonomic. This kind of approach will require a single module as the autonomic manager and implement one global policy/context repository. Similarly from a software engineering point of view the existing design can be made autonomic by just plugging in a policy database and a single module for global autonomy.

The second approach is more reliable and will be autonomic in the true sense. But for its implementation and design rigorous research and redesigning would be needed. Another issue which must be taken into consideration is that we would need different policies and different definitions for every single module and then policies for functioning with other modules as well. Thus this autonomy would be based on specific functions carried out by each individual module. Again in this respect we would also need to define the granularity with which we would like to implement the autonomic behavior of the system.

Considering these design issues we also need to define the current issues at hand. The smart office scenario (as implemented currently) would not require much reasoning and knowledge management. First the tasks supposed to be carried out by the system are very clearly defined, and secondly the users can be easily prioritized to avoid any sort of conflict in between user-preferences and user-requests. So molding and designing this scenario for a truly Autonomic CAMUS would be a reasonable goal.

In a more dynamic environment which accommodates more users (users having different priority and different interests), the need for having reasoning structures and conflict-resolution strategies becomes more important. At the same time system performance becomes a big issue, and the true autonomic behavior would aid in performance enhancement, and improvement of QoS.

So firstly, a definition of the autonomic properties is needed for each individual module, along with approaches which can be used to for accommodating these properties. The services provided by the middleware should be self-aware so that self-healing and self-optimization etc. can be easily accommodated. Similarly the system modules and components need to be aware about the status of all the utilized devices (sensors, actuators, etc.) of their interest. These would be the primary requirements for the implementation of self-healing and self-reconfiguration of the entire system. For self-optimization purposes, the scenarios corresponding to the collaboration of these devices should be identified; for example, self-optimizing behavior can correspond to the usage of the minimum number of devices to perform a desired function. Similarly self-reconfiguration can correspond to coping up with changing device locations.

Theses issues are needed to be stressed upon, and we must decide upon which approaches should be taken to implement autonomic behavior in the existing system.

1.3 Reference

[1] Ganek, A.G., Corbi, T.A.: The dawning of the autonomic computing era. IBM Systems Journal, v.42 n.1 (2003) 5-18

[2] HP Adaptive Enterprise strategy: http://www.hp.com/go/demandmore

[3] Microsoft Dynamic Systems Initiative: http://www.microsoft.com/windowsserversystem/dsi/default.mspx

[4] Bonino, D., Bosca, A., Corno, F.: An Agent Based Autonomic Semantic Platform. First International Conference on Autonomic Computing. New York (2004)

[5] R Srinivasan and H P Raghunandan, On the existence of truly autonomic computing systems and the link with quantum computing, http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:cs/0411094, 2004

[6] S.R. White, J.E. Hanson, I. Whalley, D.M. Chess and J.O. Kephart. An Architectural Approach to Autonomic Computing. Proc. of 1st International Conference on Autonomic Computing, 2004.

C h a p t e r   2

AUTONOMIC SENSING AGENT FOR COLLABORATIVE CONTEXT RECOGNITION

2 Autonomic Sensing Agent for Collaborative Context Recognition

Abstract

Ubiquitous computing envisions the use of embedded processing in everyday objects interacting with each other and with the user by means of ad hoc wireless networking. Such ambient intelligence available almost everywhere enables new or better applications and services as well as performance improvements. In near future we can even envision self-organizing dust-like devices share their knowledge in order to make decisions and perform actions. Facilitating the cooperation between the nodes in these kinds of networks creates new requirements on programming and system technologies that can glue all these things together.

Besides, deriving the user's context from a set of collaborating smart objects distributed throughout the environment constitutes a major part of a smart object's application behavior. Information about an object's own context makes it possible to dynamically form groups of collaborating artifacts, and to form networking structures that can make collaborative context recognition more efficient.

Our initial effort is to define a self-organising network concept and technology that can be situated in multiple and dynamic contexts, ranging from sensor networks to virtual networks of humans. Being targeted to embedded systems, the proposed solutions also deal with many different types of design constraints. Swarm Intelligence and Ant Colony Optimization are being investigated for achieving autonomous and decentralised optimization strategies.

We also seek for the programming abstraction with collaboration group-centric to allow program developers to focus on the collaborative task in the real world, hiding the underlying mechanisms e.g. event handling, networking, etc. On top of this group-centric self-organizing network, we are developing decentralized detection and learning for collaborative context recognition using Sequential Bayesian Estimation theory.

As a future direction, we will also tackle the security and trustworthiness of this distributed communication system by embedding security and trust rules in network functionality.

2.1 Introduction

2.1.1 Process of Context-awareness

Context services form a fabric structured into multiple levels of abstraction, as illustrated in the below figure. At the lowest level, the system's view of the world is provided by a collection of sensors that may be physical sensors such as RFID's or software sensors that probe a user's identity and platforms. The *sensing layer* generates numeric observables. To determine meaning from numeric observables, the system must perform transformations.

The *perception layer* is independent of the sensing technology and provides symbolic observables at the appropriate level of abstraction.

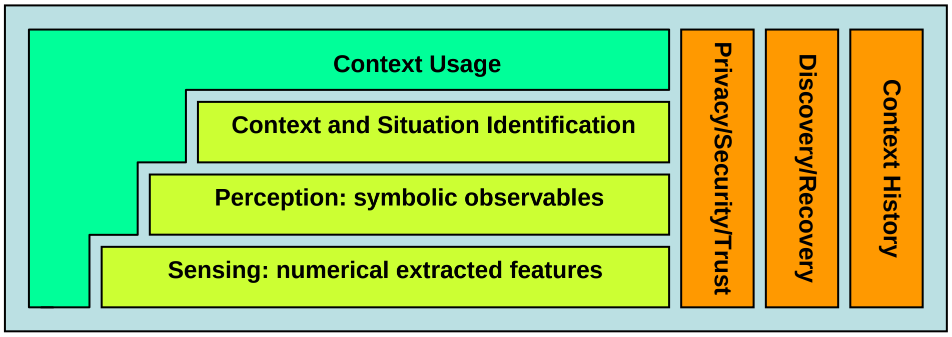

Context Usage
Context and Situation Identification
Perception: symbolic observables
Sensing: numerical extracted features
Privacy/Security/Trust
Discovery/Recovery
Context History

Fig. 1. Levels of abstraction for a general-purpose infrastructure for context-aware computing

The *situation and context identification layer* identifies the current situation and context from observables, and detects conditions for moving between situations and contexts. Services in this layer specify the appropriate entities, roles, and relations for operating within the user's activities. They can be used to predict changes in situation or in context, and thus anticipate needs of various forms (system-centric needs as well as user-centric needs).

Finally, the *context-usage layer* acts as an adapter between the application and the infrastructure. This is where applications express their requests for context services at a high level of abstraction.

These conceptual principles must be tuned and reconciled with practical considerations such as portability (for example, across operating systems), computation cost (for example, memory footprint, latency, bandwidth, and power consumption), and scalability (from microscopic to planet-wide). A variety of implementations at different scales and optimized for different targets are needed, united by common interfaces. At every level of abstraction, one must incorporate mechanisms and facilities to support privacy, trust, and security, as well as history management and discovery/recovery.

2.2 Autonomous Sensor Network

Truly intelligent, self-cognitive networks no longer act as a means to simply propagate information from one machine to the other, but become a living partner of individual and societal activities, and will play a significant role in person- and society-focused communications.

The intelligence of such networks does not lay on the nodes themselves, which have very limited resources and capabilities, but in the size and complexity of the network. Sensor and actuator nodes can be considered as simple context agents in a complex autonomous network. The architecture of the network cannot be universal for all applications, and can indeed be seen as a programming language that is used to solve a problem.

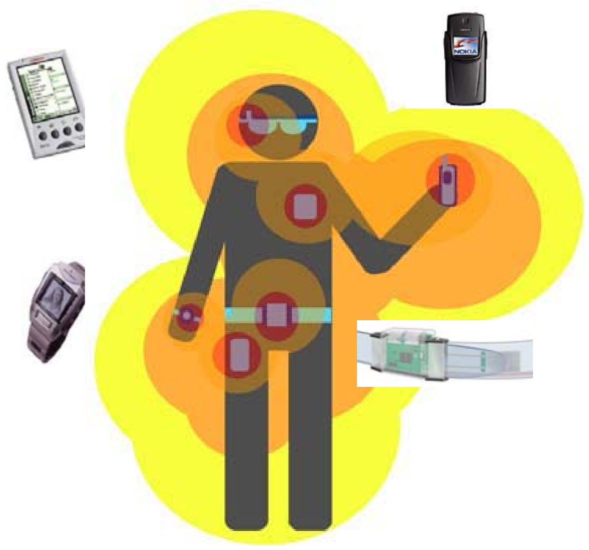


Fig. 2. An example of Wireless Body Area Network

2.2.1 Autonomic Behavior Concepts in WSN

2.2.1.1 Self-Configuration/Self-Organizing

Pre-deployment configuration in WSN is often infeasible since some configuration parameters e.g. node location, node roles, network neighborhood (e.g. node density, radio channel quality, local activity within the network, etc.) are typically unknown prior to deployment. Also, node parameters may change over time, necessitating dynamic re-configuration. Hence, a means for in-situ configuration after deployment must be developed. The term self-configuration or self-organizing is commonly used to express the fact that a sensor network should configure itself without manual intervention.

For instance, as the network and node properties change over time, role assignments must be updated to reflect these changes. Based on the assigned roles, sensor nodes may adapt their behavior accordingly, establish cooperation with other nodes, or may even download specific code for the selected role.

All the network elements in such a vision should be programmable and autonomously controlled by policies, rules and events (ECA) facilitating the desired behavior of groups of network elements (self-managing).

2.2.1.2 Self-optimization

The dynamic character will lead to a new concept of situated network, where the resources are deployed only where are needed and where are really necessary, and devices react locally on environment and context changes, balancing the load among nodes, and only communicate when they detect an interesting event.

The mostly mentioned concept is obviously load-balancing, which aims to distribute the load uniformly among different devices and avoid the bottleneck effect (like traffic congestion), and then, the network lifetime can be prolonged. For example, when one node is frequently visited (like the cluster head) or has too many neighbors, it will assign some of its tasks to the others. Some shortest path can be found so as to shorten the latency and the communication overhead can be reduced since it does not need to know the global network information. Next, some prediction mechanism and computational methods can be adopted in the routing selection scenarios so as to achieve an optimal result. Finally, some other parameters, such as the energy consumption, the sensor distribution, the bandwidth utility, the packet delivery ratio and the financial cost can be optimized based on different models.

2.2.1.3 Self-protection

Up to now, a lot of research has been done in the aspect of security in the sensor network so as to prevent the unauthorized entity from withholding, modifying and disclosing the private information. These include lots of issues, like authentication, authorization and access control, intrusion detection, security policy definition and reasoning, privacy control, resistant to fraud and persuasion, digital signature etc. By introducing these kinds of self-protection policies, not only the availability, integrity and confidentiality of information or resources can be insured, but that some system performance, like the latency, packet delivery ratio can be improved.

We can deal with security and trustworthiness of this distributed communication system by embedding security and trust rules in network functionality at modeling and design phases. By combining self-healing with self-protection, it may have less chance to fail.

Open issues: How can we achieve robustness and reliability in ubiquitous computing environments full of failure-prone tiny interacting devices (fault-tolerance, dependability)? A lot of implicit interactions of “smart-things”, sensors of limited precision, and actuators – do humans feel safe? A lot of interpretations of sensed information and implicit assumptions about human beings and interaction patterns – what if they are wrong? How can we counter the threats arising in ubiquitous computing?

2.2.1.4 Self-healing

As for the sensor network with dynamic topology, the self-healing mechanism is of utmost importance. When some node leaves the network, dies due to out of energy or malfunctions for some software bugs, the system should have the ability to detect it, then diagnose it and finally repair it, either by directly abandon it or choose other alternatives. For example, when a node waits for too long to get a reply message from its neighbor, it should have a self-healing mechanism to know whether the destination node is heavily loaded or it leaves the network or something else. And last but not the least important one, the whole system can achieve a better robustness by taking care of different levels of faults, from the basic value and timing faults, to the physical, logical and even system faults.

2.2.1.5 Self-awareness

The final step will be leveraging meaning into the network, using semantics and context to ensure both that communications technology takes full advantage of the context in which it is used, and that the context is well-served by the technology deployed. For instance, the notion of adding semantic tags to information exchanges - letting the network "know" what it is transporting - can be a key enabler for adaptive and other intelligent behavior. This can also provide some long-term stability to the whole network, as it has been observed that models and semantics are more stable than technology, as they may be re-implemented many times without changing their essential features. And this semantic layer should be implemented on top of the smart sensor network, as the reflection in virtual world of these entities in real world. Researchers at ETH have been using virtual counterparts to form augmented representations of real-world objects and encapsulate object states and object behavior in a computational entity.

2.2.2 System Challenge

2.2.2.1 Essential infrastructure

What kind of infrastructures do we need to support smart collaborating objects?

These infrastructures have to cope with a highly dynamic environment and should, among other things, provide location information to mobile objects, represent context information, and enable reliable and scalable service creation. Besides, to fill the gaps toward solutions for real-world situations, open questions arise like: How much should the system (or infrastructure) remember? When does the system try to predict the user's intentions and when are the users presented with choices?

2.2.2.2 Programming methodologies

Networked embedded systems must be able to rapidly respond to all kinds of sensing events, even though they have numerous failure-prone nodes and limited energy and bandwidth resources. As application developers, we must fundamentally rethink the organization and programming of these deeply embedded systems. What is the appropriate mental model we can use to reason about the collective behaviors of a system when programming a distributed application so that the application is portable, scalable, and robust? What are the organizational principles for developers to build large applications by mixing and matching various ad hoc communication protocols while shielded from dealing with a multitude of communication events? How does the software architecture expose the underlying system constraints so that application developers can consider important performance trade-offs?

2.2.2.3 Interaction design

As computers disappear from the perception of the users, a new set of issues is created concerning the interaction with computers embedded in everyday objects resulting in smart artifacts: How can people interact with invisible devices? How can we design implicit interaction for sensor-based interfaces? How do people migrate from traditional explicit to future implicit interaction? A major approach in this issue is hybrid worlds combined of real world and virtual world.

Consider ambient displays. These smart objects should provide services that are location- or situation-based depending on the proximity of people passing by, e.g. the Hello.Wall in the Ambient Agoras project has three different communication zones: ambient, notification, and interaction. People passing through the ambient zone contribute to and experience the ambient patterns continuously displayed on the Hello.Wall. These patterns concern, for example, general presence information. People in the notification zone are identified as individuals and agree to have the Hello.Wall enriched environment react to their personal presence. This can result in personal notification patterns being displayed on the Hello.Wall. People in the interaction zone can get directly involved with the Hello.Wall environment. The artifact reflects this by showing special interaction patterns.

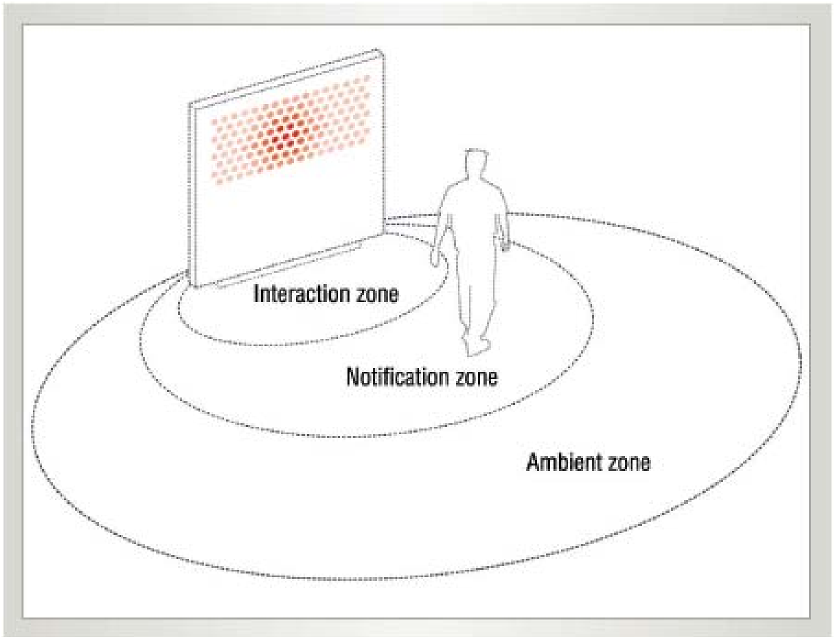


Fig. 3. Communication zones. Depending on the distance from the display, the Hello.Wall has three communication zones: ambient, notification, and interaction.

Handheld devices, which are becoming abundant nowadays, have also been considered as a means to interact with the invisible world of smart embedded devices.

2.3 Self-Organized Networks and Programming Abstraction

2.3.1 Self-Organized Networks

A Wireless Network usually consists of many nodes, each with a set of communication links connecting to other nearby nodes. By exchanging information, nodes can discover their neighbors and perform a distributed algorithm to determine how to route data according to the application needs. Although physical placement primarily determines connectivity, variables such as obstruction, interference, environmental factors, antenna orientation, and mobility make determining connectivity a priori difficult. Instead, the network discovers and adapts to whatever connectivity is present.

As an illustration of a self-organized sensor network, consider a wireless monitoring system with a mixture of light or motion sensors (constantly vigilant at low-power), and a few higher-power and higher-bandwidth sensors such as microphones or cameras. To conserve energy and bandwidth the audio sensors would be off (or not recording) at most times, except when triggered by less expensive light sensors. Instead this computation can be distributed throughout the network. Queries (user requests) are labeled with sensor type (audio or light) known to the system at design time. Queries diffuse through the network to be handled by nodes with matching sensors in the relevant geographic region. The application will hear from whatever relevant sensors respond. Moreover, the decision of one sensor triggering another can be moved into the network to be handled directly between the light and audio sensors. The alternative Internet-based architecture would have a central directory of active sensors and a central application that interrogates this database, monitors specific sensors, and then triggers others. Our goal is to eliminate the communication costs of maintaining this central information to provide more robust and long-lived networks in spite of changing communications, moving nodes, and limited battery power.

The self organization protocols can be used to establish connectivity, relative topology and position, and allow data to be disseminated and aggregated. This capability is crucial to keep the system low-maintenance, fault tolerant and able to provide in-network processing more efficiently. Cluster tree architecture that Callaway and Motorola team devised is an example of self organization in connectivity. Butler and Rus create simulation to examine how to reposition mobile sensors to where the phenomenon is, using history-free update rule or history-based algorithm. While Roedig et al, proposed an intentional delay in nodes along the route message pass through to increase probably of message aggregation and therefore power efficiency.

2.3.2 Programming Abstraction for Collaborative Group

The design methodology and software environment are needed for allowing users to write programs focused on phenomena in the world, abstracting away event handling, networking and resource awareness while still maintaining scalability and efficiency (State-Centric programming – PARC). In this section we will study on the current literature to answer such questions as: How to name and address the nodes? How to assign roles and form collaborative groups? How to propagate and maintain the groups and roles? How to define access control over the groups?

2.3.2.1 Node attributes/properties

Node attributes or properties define the elements of a node's state that are shared with its neighbors, such as available sensors (e.g. temperature, light,) their characteristics (e.g. resolution) and sensor readings, other hardware features (e.g. memory size, processing power, communication bandwidth); remaining battery power, or geographic location. Some attributes are static; some may change over the network lifetime, and dynamic properties are of particular interest as their alteration has to trigger a re-evaluation of the group membership condition. A unified interface should be provided to access attribute values, which can be encoded as name-value pairs e.g. <attr = name, val = temp>, <attr = resolution, val = 1oC>.

Naming: In order to enable a conflict-free addition of scenario-specific attributes and to make attribute implementations distinguishable, hierarchical namespaces could be used for naming attributes, e.g. sensor.type.temperature. Attributes are identified by unique keys drawn from a central authority. However in practice, in order to comply with the resource restrictions of WSN, attribute identifiers could be mapped to a more compact byte-code at compile time (see also Multi-Purpose WSN, Directed Diffusion) e.g. simple 32-bit numbers and assume out-of-band coordination of their values, just as Internet protocol numbers are assigned. The next issues are then operations defined on these attributes.

2.3.2.2 Neighborhood

The below discussion will help us answer the questions how to discover, create, evaluate, maintain, and propagate neighborhood

2.3.2.2.1 Neighbor Definition

We may define the neighbor of a node by several metrics

• N-radio hop: Nodes within N radio hops;

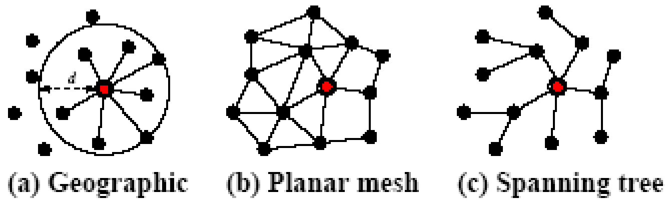
• N-radio hop with geographic filter: Nodes within N radio hops and distance d;

• k-nearest neighbor: k nearest nodes within N radio hops;

• k-best neighbor: k nodes within N radio hops with the highest link quality, as measured in fraction of packets dropped over some measurement interval;

• Approximate planar mesh: A mesh with a small number (possibly zero) crossing edges; and

• Spanning tree: A spanning tree rooted at a single node, used for aggregating values over the entire network.



2.3.2.2.2 Neighbor Discovery

Before performing other operations on a region, each node initiates the process of discovering neighboring nodes. Depending on the type of region, this may require broadcasting messages, collecting information on node locations, or estimating radio link quality [see also Abstract Region.] This is a continuous process, and each node is informed of changes in the region membership set, due to nodes joining, leaving, or moving within the network. A node may terminate this process at any time to avoid additional discovery messages. When terminated, the neighbor discovery operator returns a quality metric that measures the accuracy of the region formation, such as the percentage of candidate nodes that responded to the discovery request.

Broadcasting and filtering for radio and geographic neighborhoods

For efficiency, the broadcast/filter mechanism could be used for both data sharing and neighborhood discovery [see also Hood.] This mechanism is especially suitable for sensor networks. First, it exploits the cheap broadcast channel inherent in wireless networks. Further, it promises only the weak sharing semantics that unreliable, low-bandwidth networks can provide; the neighbors in the neighbor list and the cached values of a neighbors' attributes represent only the best or most recently observed. Any stronger guarantees about consistency, coherence or reliability are intentionally deferred to the application level.

The implementation of the radio and geographic neighborhoods is straightforward. To discover neighbors, each node broadcasts periodic advertisements containing their shared attributes. The receiving nodes "filter" the incoming attributes to determine which nodes are adequate neighbors and which of their attributes should be cached. This means that a neighborhood is fundamentally a local construction at each and every node, in contrast to groups where membership is shared a manipulated among the nodes in the group.

For example, a node might define a routing neighborhood that caches the location information of valuable routing nodes and a sensing neighborhood that caches sensor information of valuable sensing nodes. Each node's neighborhood independently decides which neighbors are valuable based on the shared attributes they are broadcasting, fills its neighbor list, and caches the attributes it deems important.

2.3.2.3 Attribute Sharing

We need to answer the questions e.g. whom to share, when to share, and how to share the node attributes. Attribute sharing can be implemented by push/pull policy. Push policy simply broadcasts the updated value each time it is set (and an indication/event should be fired.) An alternative might be to broadcast periodically, reliably under the application-level control through the push/pull interface. Pull policy simply sends a fetch message to the corresponding node.

2.3.2.3.1 Sending data

The push policy should allow the user to provide different sharing semantics for an attribute beyond the simple auto-push. Each time the attribute is written; the push policy will be notified and determines what action to take, for example, to push the value periodically in mobile networks where neighborhoods change frequently, or to push the attributes several times for robustness to packet loss for important attributes. Several attributes can be aggregated into one update message to guarantee consistency, such as light value and geographic location, and also to support membership filtering on several attributes.

2.3.2.3.2 Receiving data

When a remote attribute update is received, it will be examined by a filtering mechanism to decide 1) update the neighbor states if it is already a member, 2) remove the neighbor if it no longer meets membership criteria, or 3) add the neighbor if it meets membership criteria. In this way, discovery and data sharing are natural consequences of the broadcast/filter process. If neighboring nodes are not actively pushing attributes, then the node can issue a pull request.

2.3.2.4 Collaboration Group Forming

2.3.2.4.1 Collaboration Group

A collaboration group is a set of sensor nodes that contribute to a collaborative recognition process. Intuitively, a group encapsulates two properties: its scope defining its members, and the roles defined for each member in the group e.g. leader/follower, cluster-head/gateway/slave, source/aggregator/sink, sensor/fusion-station/actuator, etc. [see also State-Centric Programming.] A scope could be specified existentially or by a membership function (for example, all nodes within some geometric range, all nodes within a certain number of hops from an anchor node, or all nodes that are "close enough" to a temperature contour). Grouping nodes according to physical attributes rather than node addresses is an important abstraction in sensor network programming. Scope can be evaluated locally or dynamically, as long as the communication among the group members is maintained as needed. Scoping is critical to maintain scalability as sensor networks grow arbitrarily large.

The notion of roles shields programmers from addressing individual nodes by either name or address. Redundancy by having multiple members with the same role also improves application robustness over node and link failures. Roles also help implement highly optimized networking and control flow protocols at runtime to support group execution.

Geographical group

A geographical group consists of a set of nodes located within some fixed geographical region of a phenomenon, which can be specified by geographic shape such as circles, polygons, and their unions and intersections. A geographical group can be established using protocols such as Geocasting and GEAR to support communication among members.

N-hop neighborhood group

When the communication topology is more important than the geographical extent, we can define scope in terms of hop counts. We define the members of this group as the set of all nodes within n communication hops from a specified anchor node. Because local broadcasting uses hop counts rather than Euclidean distances, it can be conveniently used to determine the scope.

Publish/subscribe group

This group consists of consumers expressing interest in specific types of data or services and producers that provide those or that satisfy certain predicates over its attributes. It could be established via network protocols such as Directed Diffusion of SCADDS project.

Using multiple groups

There are various ways to compose multiple groups to provide different types of input for a single collaborative recognition task. For example, in the target-tracking example, a geographical group could be used to gather sensor measurements and a one-hop neighborhood group to select the potential next leader.

2.3.2.4.2 Scope Specification and Role Assignment

As mentioned above, a scope is basically a group of nodes that is specified through a membership condition. Considering scopes as first class objects provides an orthogonal means for structuring distributed systems and allows additional services such as access control or different communication semantics to be bound to scopes. The middleware system should provide services for handling the scope membership status of each node, and for the dissemination of scope creation requests and intra scope communication (see also Multi-Purpose WSN)

Scope specification and membership condition evaluation

A central part of a scope specification is the membership conditions. For example, the neighborhood in Hood and Abstract Regions are a special case in which scope is restricted to neighboring (geographical and radio-based) nodes. Membership condition can be evaluated on three major levels using

i) Expressions that can be evaluated locally, formed from functions (e.g. +, –, log()), predicates and comparison operators (e.g. ==, <) and Boolean expressions over constants and locally available current node properties

ii) Expressions over historic values e.g. derivations of properties over time such as velocity or acceleration, and aggregations over time windows such as average or variance

iii) Aggregations over non-local properties from multiple nodes with synchronization to select the most suitable from a set of candidate nodes.

The lifetime of a scope should be adjustable locally by the member nodes according to its usage pattern such as single-short queries or long term applications. Such local decision helps avoid communication overhead and prevents stale scope instantiations in disconnected areas.

Role assignment

Role specification is a list of role-rule pairs so that sensor nodes can take on specific functions or roles in the network without manual intervention. For each possible role, the associated rule specifies the conditions for assigning this role. Rules can be Boolean expressions that may contain predicates over the local properties of a sensor node and predicates over the attributes of well-defined sets of nodes in the neighborhood of a sensor node, similar to scope evaluation as above mentioned. Triggered by attribute and role changes on nodes in the neighborhood, the algorithm evaluates the rules contained in the role specification. If a rule evaluates to true, the associated role is assigned (see also Generic Role Assignment.)

2.3.2.4.3 Other Issues

There are several major issues with the robustness and optimization of the above mechanisms. The possible approaches to these issues described below should lead to efficient algorithms for role assignment and scope evaluation.

Consistency

The first and most important issue is the problem of how to ensure consistency, that is, in absence of property changes all nodes should eventually decide on a stable role that does not trigger role changes at any other node. In other words, we assume that rules are evaluated atomically: if the outcome of evaluation of the rules on node N depends on a sensor node M, then M must not change its role during the evaluation of the rules on node N. Time synchronization during role evaluation could ensure atomicity with high probability.

Robustness is an important issue in sensor networks, since sensor nodes and communication links are subject to frequent failures. Node failures happening during a role assignment cycle require a dynamic re-configuration for affected nodes to be triggered. This could be implemented by a failure-detection service using timeouts or by periodic reevaluation of rules.

Tuning the broadcast for self-optimization

Generally, increasing the number of messages (and hence, energy usage) improves communication reliability and the accuracy of collective operations. As a concrete example of such a tradeoff, the communication layer may tune the number and frequency of message retransmissions to obtain a given degree of reliability from the underlying radio channel. Similarly, nodes may vary the rate at which they broadcast location advertisements to neighboring nodes, affecting the quality of neighbor discovery and group formation.

We can define the quality measure as the completeness or accuracy of a given operation. In member discovery, for instance, the quality measure represents the fraction of candidate nodes that responded to the discovery request. Quality measure can be utilized for tuning low-level parameters of the neighborhood and group implementation, such as the number of message broadcasts, amount of time, or number of candidate nodes to consider when forming a neighborhood. Likewise, operations perform message retransmission and acknowledgment to increase the reliability of communication; the depth of the transmit queue and number of retransmission attempts can be tuned also. We are currently working on a resource-centric tuning mechanism that applies Swarm Intelligence and Ant Colony Optimization to enable applications to adapt to changing environments, network conditions, radio bandwidth, node failure, and energy or latency budget.

2.4 Swarm Intelligence Mechanism for Self-Optimization

2.4.1 Swarm Intelligence Algorithm

Swarm Intelligence (SI) is an Artificial Intelligence technique involving the study of collective behavior in decentralized systems. Such systems are made up by a population of simple insects/agents interacting locally with one another and with their environment, just like the relationship between autonomic elements and their managing environment. Although there is typically no centralized control dictating the behavior of the agents, local interactions among the agents often cause a global intelligent pattern to emerge. Examples of systems like this can be found in nature, including ant colonies, bird flocking, animal herding, honey bees, bacteria, and many more. Swarm-like algorithms, such as Particle Swarm Optimization (PSO) [8] and Ant Colony Optimization (ACO) [9-13], have already been applied successfully to solve real-world optimization problems in engineering and telecommunication. SI models have many features in common with Evolutionary Algorithms.

From an engineering standpoint, the principle advantages of SI system design are four-fold:

Scalability, from several to thousands of units;

Flexibility, as units can be dynamically added or removed without explicit reorganization;

Robustness, not only through unit redundancy but also through an adequate balance between explorative and exploitative behavior of the system;

Simplicity, at the individual level which also increases robustness.

The application of SI to the distributed, real-time, context-aware and embedded systems or network aims at developing robust task-solving methodologies by minimizing the complexity (including the intelligence) of the individual units and emphasizing parallelism and self-organization. So, we are once again convinced that there is an inherent accordance between SI and autonomic computing. And we can safely give some similarities between them as follows:

Aim at reducing the system complexity;

Distributed environment;

Self-organizing (including four basic characteristics of autonomic computing);

Interact in a simple, localized way and no global information is needed;

No central control.

One of the many mechanisms at work in natural swarms is the use of pheromone trails to select the shortest path between two locations, to make reinforcement learning and to reduce communication overhead etc. Pheromone intensity is a key representation of the local information through which insects interact with each other. It will dynamically evaporate or be reinforced through the interaction among the insects. Taking network routing as an example, as noted in [10], an ant selects an edge probabilistically using τ_{ij} and η_{ij} as a functional composition for P_{ij}. Here, τ_{ij} is pheromone concentration/intensity which indicates the favorability between i and j and η_{ij} is a function of the cost of edge (i, j) (which may include factors such as queue length, distance, and delay etc.). η_{ij} can be simply represented as $\eta_{ij} = 1/d_{ij}$, and P_{ij} can be given as follows [11]:

$$P_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{g \in J(i)} (\tau_{ig}^{\alpha} \cdot \eta_{ig}^{\beta})} \qquad (1)$$

In equation (1), α and β represent the respective adjustable weights of τ_{ij} and η_{ij}, and $J(i)$ is the number of node i's neighbors. Consequently, the routing preferences of ants can be altered by selecting different values of α and β. If $\alpha > \beta$, ants favor paths with higher pheromone concentrations, and if $\alpha < \beta$, the paths with more optimistic heuristic values (such as the distance) will be more likely chosen. A lower value of α is generally preferred when pheromone concentration along paths may not necessarily reflect their optimality. Examples of such situations include the initial stage after a network reboots (before the network stabilizes), and when there are frequent and abrupt changes in network status due to either link (or node) failure or introduction of new paths (nodes). However, as a network stabilizes, a higher value of α is preferred.

Another important concept is pheromone evaporation. To reduce the effect of past experience, an approach called evaporation [12] is typically used in conjunction with ACO. Evaporation prevents pheromone concentration in optimal paths from being excessively high and preventing ants from exploring other new or better alternatives. In each iteration, the pheromone values in all edges are decreased by a factor of p, namely $\tau_{ij}^{t+1} = (1-p)\tau_{ij}^{t}$. Suppose that at time t, all ants converge to a routing R and deposit a very high concentration of pheromone. Then, in the next time (t+1), the pheromone concentration along R is reduced by a factor of p. In that case, traffic congestion can be avoided and more routing candidates can be chosen.

In contrast with pheromone evaporation, there is another concept named pheromone reinforcement. Intuitively, it means the pheromone intensity can be reinforced when the route is revisited. To reduce its historical influence on the routing selection, a mechanism called aging is introduced in [13] and a pheromone upper bound τmax is adopted to avoid the bottleneck effect and to balance the load.

Besides, there are some other important research issues, like the non-linear dynamic change of pheromone, the probabilistic model of routing selection, the heuristic representation of η_{ij} etc. Fig. 5 shows the generalized procedure of pheromone based solution.

```
Procedure Pheromone-based Solution ( ) /* like shortest-path, TSP, cargo scheduling etc. */
    While (not stopping criterion)
      Activities_scheduling      /* based on different models */
          Phermone_evaporation ( ); /* re-configurable to avoid bottleneck effect*/
          Phermone_reinforcement ( ); /* refer to reinforcement learning */
          Phermone_aging ( ); /* re-configurable to reduce historical influence*/
          Phermone_...... ( );
      End Activities_scheduling
   End While
Return optimal solution
```

Fig. 5. Generalized procedure of pheromone based solutions

From the description above, we can abstract what pheromone represents as follows:

Self-organized, light weighted and context-aware;

Local simple interaction and global intelligent behavior;

Closed control loop with pheromone generation, evaporation, reinforcement and thresholding;

Distributed, dynamic property with configurable pheromone to achieve an optimal performance.

And that's why we intend to use it in our next section by incorporating its function into the autonomic agent software during the process of network routing in the sensor network.

2.4.2 Autonomic Platform

Inspired by the idea of [7], here, we present an autonomic agent (AA) based platform by introducing some autonomic functions, such as self-configuration, self-optimization, self-healing, self-protection and context-aware etc.

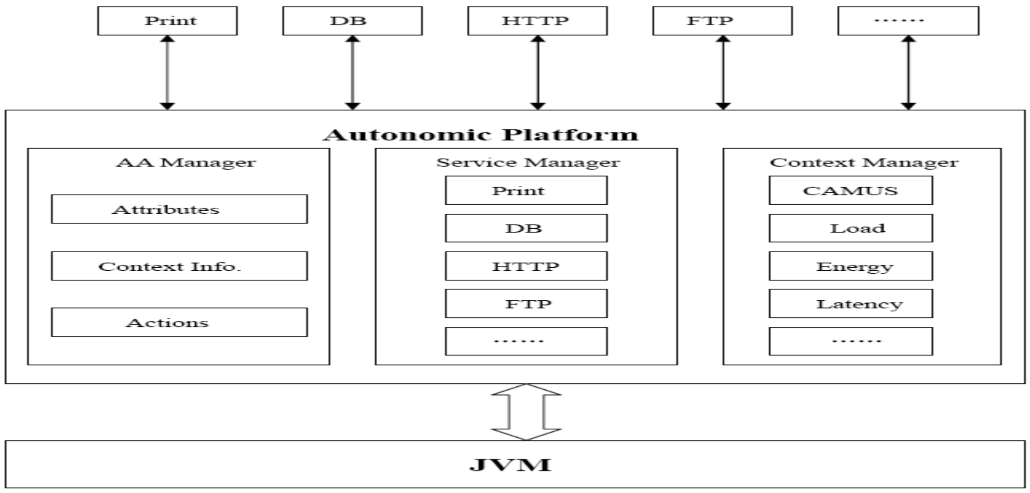


Fig.2 An agent based autonomic platform

Fig.2 shows the brief architecture of our autonomic platform. For the present, we just deal with three main modules, namely Autonomic Agent (AA) manager, Service Manager and Context Manager, and we will verify it on our "Smart Office" test bed, which has been built. Then, we will add more modules, like Security Manager, Energy Manager etc.

An AA manager is consisted of three main parts, which are attributes, context information and actions. From Table.1, we can easily understand what they include. The autonomic agents (AAs) will interactively collaborate with each other based on the pheromone and rules, and configure themselves so as to achieve an optimal system performance. While in the mean time, they will prevent the system from being attacked and malfunctioned.

Table.1 Data structure of AA manager

Attributes	Unique ID, Timestamp, Remaining energy, Service type etc.
Context Info.	Location (IP), Neighbor Information, Available service, Hop number, Pheromone distribution, Resources utility (like CUP, memory/cache, and hard disk utility), Humidity, Temperature, Lighteness, latency, etc
Actions	Generation, Migration, Communication, Service discovery, Energy exchange and storage, Relationship maintenance, Pheromone emission, Pheromone evaporation, Pheromone aging, Pheromone reinforcement, etc.

A Service Manager is in charge of different services, like HTTP, FTP, printing services. It will attach its available services with specific time duration to the AAs, and then, by the communication of AAs, different types of services can be collected and shared by various devices.

A Context manager is a key module in our autonomic platform. It is closely related with the concept of autonomic computing. The context information can be either from our original CAMUS, or from tools, such as SNMP tools or JAVA. Interested readers can refer to our [8-10] for more information.

An application scenario based on our autonomic agent is demonstrated in Fig.3. From which, we can conclude the steps as follows:

Step 1: Originally, each AA is piggybacked into the hello packet and periodically broadcasted to collect some information, such as available services, remaining energy, distance, next hop etc. Some SI mechanism is adopted with pheromone evaporation, reinforcement and aging. It acts unconsciously just like the human cells, and some global optimal performance can be achieved.

Step 2: When there is some specific services request (like printing) originated from the upper application layer, it will be directly sent to the autonomic platform, which is consisted of many closed control loop.

Step 3: And then, it will first check the Service Manager in Fig.2 to see whether this kind of service is available. If it is available, it will check the context manager so as to get the best service and then return it to the upper layer.

Step 4: If it is not available, it will contact the AA Manager. If it is attacked or malfunctioned, the platform will autonomically generate another AA with the uniform data structure, as is shown in Table.1. A self-healing mechanism and some security policies are adopted here, so, normally, it functions very well. If AA is available, then, the service discovery process will be initiated so as to find the required service. Through local interaction between AAs, the related services can be found soon, just like to find the shortest path. During this process, some actions are taken, such as Migration, Communication, Relationship maintenance, Pheromone evaporation, Pheromone aging, Pheromone reinforcement, etc.

Step 5: Once the specific service is found, the context information is sent back to the context manager. And finally, the best service is decided and provided to the user. Here, another closed control loop is formed.

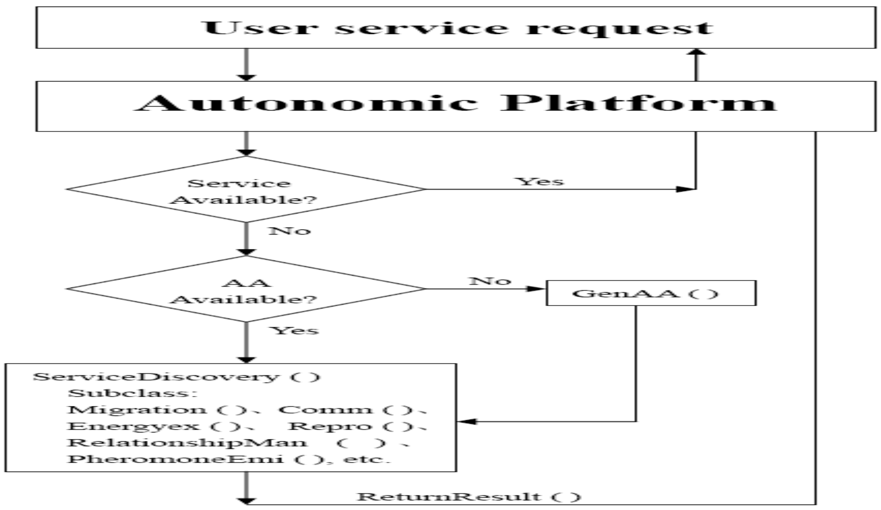


Fig.3 An autonomic agent based application scenario

2.4.3 A Swarm Intelligence inspired Localized Routing in MANET

Based on the swarm intelligence mechanism and autonomic platform, we want to further apply this self-optimizing principle to the routing problem in Mobile Ad hoc Network (MANET).

Our final objective is to optimize the energy consumption and routing overhead, while in the meantime, to maintain the packet delivery ratio. In that case, a self-optimizing scenario is achieved and our autonomic platform is validated.

2.4.3.1 Experimental Setup

According to our previous work [11], we set our simulation environment as follows. There are N nodes randomly placed within a range of 100 by 100 m2, with a uniform transmission range R of 30m. A Random Waypoint mobility model is adopted here and their velocity vary from 0 to 20 m/s. Taking N=20 as an example, we will see the deployment as in Fig.4.

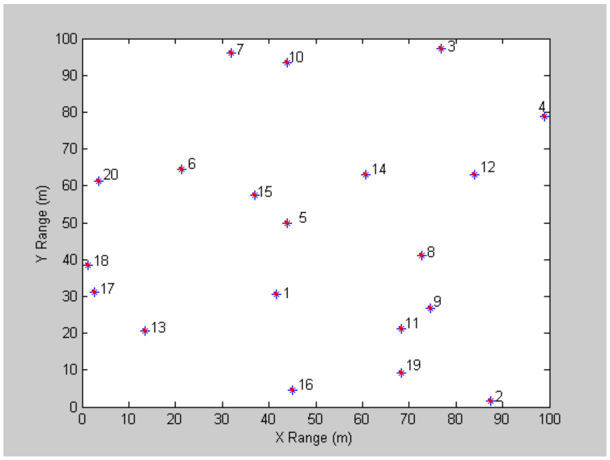


Fig. 4 N nodes random deployment

Unlike the broadcasting mechanism and (1), we set our neighbor selection criteria as follows:

$$f_{ij}(t) = e^{-\tau_i(t)} * e^{-d_{ij}/R}, \quad (d_{ij} < R) \qquad (2)$$

$$\tau_i(t+1) = \tau_i(t) + \Delta(t,t+1), \quad \tau \in [0,1] \qquad (3)$$

$$\Delta(t,t+1) = \begin{cases} K * \dfrac{1}{2 \cdot N} & reinforced \\ (\rho - 1) * \tau_i(t) & aging \end{cases} \qquad (4)$$

here, $\tau_i(t)$ is the pheromone distribution of node i at time t, d_{ij} is the distance between (i,j) and R is the transmission range. From (2), we can infer smaller d_{ij} and $\tau_i(t)$ is, higher $f_{ij}(t)$ will be and we will choose higher $f_{ij}(t)$ as the candidate node. In other words, this algorithm will prefer those nodes which are either nearer to the transmission node or being less visited with less pheromone so as to save energy. After each hop, the pheromone will automatically be updated according to (3) and (4), where the pheromone will either be reinforced according to the number of visiting agents K or aged by a factor of ρ (ρ is in the range of [0.9, 1] normally).

Through this kind of simple and localized cooperation, a great deal of global intelligence can be achieved. That is why the study of localized routing algorithms in wireless environment is listed as the first future work in [12]. In [12], some other future work such as the transmission error rate (successful routing set-up rate in our paper), localized power-efficient multicasting protocol (the critical factor is K in our paper) has also been partially done in our paper.

2.4.3.2 Study of Agent Number

So, we will first study how many autonomic agents is need for each node so as to fulfill the routing task and satisfy the response time requirement in the mean time, which is similar to the multicasting mechanism. If the agent number is too small, it will not find the related routing or the response time is too large to accept. And if the agent number is too large, the communication overhead will be a burden and it is similar to the broadcasting mechanism, which we try to avoid in this paper.

Table.2 shows the average neighbor for each node. The total node number varies from 10 to 100 and the average node neighbor increases almost linearly with it. For N=20, the individual node's neighbor varies from 2 to 5, with an average value of 4. And for N=30, it varies in the range of [2, 8], with an average value of 6.

Table .2 Av neighbor for different N

N	10	20	30	40	50	60	70	80	90	100
Neighbor	2	4	6	8	10	12	14	17	20	22

So, we tentatively set our agent number as K=min (actual neighbor, 5) and try to validate it in the next two sections.

2.4.3.3 Study of Successful Routing Set-up Rate

To verify the validity of agent number, here, we randomly generate 300 routing connections between node i and j. Taking the following routing scenario as one example:

Step 1: Node i initiates a routing request;

Step 2: If the destination node is available in its context manager, it will use the best one and end. Else, it will use the local information and select K=min (actual neighbor, 5) neighbors with higher f_{ij}.

Step 3: The intermediate node will repeat like step 2 and avoid the loop. Then the pheromone is updated.

Step 4: Algorithm terminates if the destination is found or there is no available neighbors (failed).

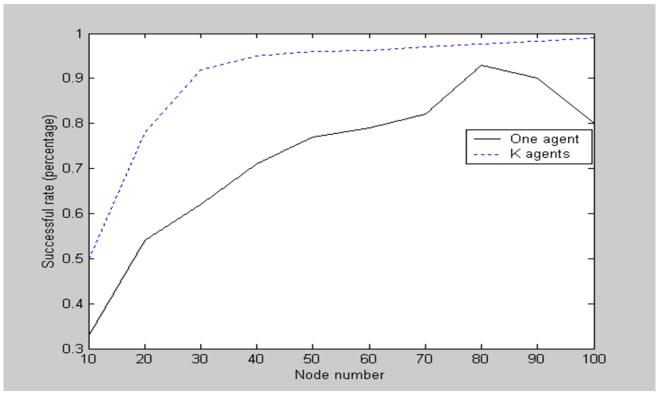


Fig. 5 Successful routing set-up rate for different approaches

In Fig. 5, we make a comparison between one agent approach and our approach in the aspect of successful routing set-up rate. From this figure, we can see that there is no reliability guarantee if the agent number is too small. For the connection of (1, 14) and the node deployment in Fig. 6, if there is only one agent, the routing sequence will be like {1, 5, 15, 6, 20, 18, 17, 13, terminate}. If there are K agents like our approach, it will first take node {5, 16} as its first hop node. Then, node 5 will detect node 14 as the destination node and return this information to the source node 1. It only takes 2 hops. And another routing sequence can also find the destination through {1, 16, 19, 18, 9, 8, 14}. So, the successfulness of finding a reliable route can be guaranteed.

2.4.3.4 Study of Energy Consumption

Finally, we will study the power consumption and make a comparison between AODV and our approach. According to our previous work in [12], we assume the power consumption is proportional to the square of distance and the data length is the same. So, the power consumption of each node is as follows:

$$E_i = k * \sum_{j=1}^{M} d_{ij}^2 + c \qquad (5)$$

Here, k and c are constants. For broadcasting mechanism in AODV, M is equal to the actual number of neighbors and for our autonomic agent based approach, M is equal to K.

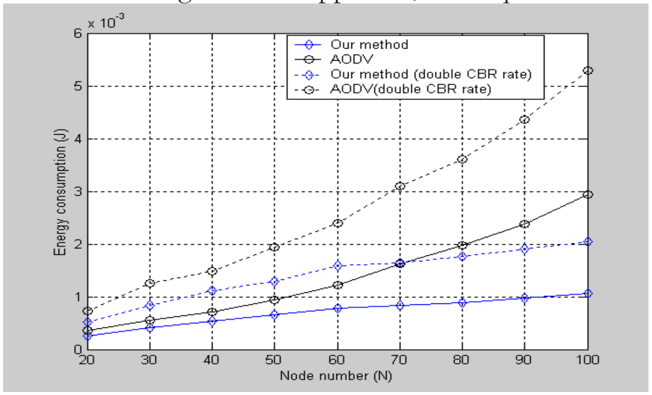


Fig. 6 Comparison of energy consumption

From Fig. 6 we can see that our K-neighbor based routing algorithm performs better over AODV. The reason lies in two aspects. First, we use a multicast mechanism rather than broadcast mechanism here. So, the energy consumed during the routing discovery and maintenance period can be greatly reduced. Secondly, since our algorithm tends to select those neighbors which are nearer to the source node, so the overall hop number is increased. In other words, the distance between each transmitter and receiver is shorter. According to the radio propagation model, the overall energy consumption over a distance of <a, c> is much larger than the summation of the energy consumed over <a, b> and <b, c>. Besides, the more traffic it is transmitted, the more energy can be reduced.

After the establishment of our autonomic platform, we further validate it through the routing scenario in mobile ad hoc network. The simulation results effectively show that our algorithm has a better performance than the traditional routing protocol AODV.

2.5 Conclusion

Ubiquitous computing envisions the use of embedded processing in everyday objects interacting with each other and with the user by means of ad hoc wireless networking. Such ambient intelligence available almost everywhere enables new or better applications and services as well as performance improvements. In near future we can even envision self-organizing dust-like devices share their knowledge in order to make decisions and perform actions such as raising an alarm, controlling a building's environment, actuating a vehicle's brakes or administering medication to a patient. Facilitating the cooperation between the nodes in these kinds of networks creates new requirements on programming and system technologies that can glue all these things together.

Deriving the user's context from a set of collaborating smart objects distributed throughout the environment constitutes a major part of a smart object's application behavior. Information about an object's own context makes it possible to dynamically form groups of collaborating artifacts, and to form networking structures that can make collaborative context recognition more efficient.

Our initial effort is to define a self-organizing network concept and technology that can be situated in multiple and dynamic contexts, ranging from sensor networks to virtual networks of humans. Being targeted to embedded systems, the proposed solutions also deal with many different types of design constraints. Swarm Intelligence and Ant Colony Optimization are being investigated for achieving autonomous and decentralized optimization strategies.

We also seek for the programming abstraction with collaboration group-centric to allow program developers to focus on the collaborative task in the real world, hiding the underlying mechanisms e.g. event handling, networking, etc. On top of this group-centric self-organizing network, we are developing decentralized detection and learning for collaborative context recognition using Sequential Bayesian Estimation theory.

For self-optimizing, we can realize all the basic characteristics of autonomic computing system through the study of "swarm intelligence inspired localized or cooperative behavior". Extensive simulation and theoretical work is undertaken. We firmly believe that our approach will outperform the others in the aspects of energy consumption, routing overhead and load-balancing, while in the mean time maintain the performance of latency and packet delivery ratio.

The proliferation of action traces (digital data originating from real-world transactions, stored in undesired places), the execution of unwanted transactions through autonomous computing agents, the misinterpretation of contextual information, and service failure due to erroneous or malicious devices and software are amongst the security threats introduced by ubiquitous computing. These threats arise in the interaction of humans with smart objects, within smart environments, where large numbers of smart devices interact, and in conjunction with novel system architectures such as wireless sensor networks, where devices with low resources are employed to monitor mission-critical environmental features.

As a future direction, we will also tackle the security and trustworthiness of this distributed communication system by embedding security and trust rules in network functionality.

2.6 Reference

[1] R. Murch, "Autonomic Computing," Upper Saddle River, NJ: Prentice Hall, Mar. 2004, ISBN: 013144025X.

[2] IBM, "Autonomic Computing Initiative," IBM Press, 2003, http://www.autonomic-computing.org.

[3] Apostolos Malatras, George Pavlou, et al, "Self-Configuring and Optimizing Mobile Ad Hoc Networks," Proceedings of the Second International Conference on Autonomic Computing (ICAC'05), Seattle, Washington, June 2005, pp. 372-373.

[4] Belecheanu, R. A., Jawaheer, G., Hoskins, A., McCann, J. and Payne, T, "Semantic Web Meets Autonomic Ubicomp," In Proceedings of The 3rd International Semantic Web Conference, Hiroshima, Japan, 2004.

[5] F. Schintke, T. Schütt, A. Reinefeld, "A Framework for Self-Optimizing Grids Using P2P Components", 14th Intl. Workshop on Database and Expert Systems Applications (DEXA'03), September 2003, pp. 689 – 693.

[6] J. Kephart and D. Chess, “The Vision of Autonomic Computing,” IEEE Computer, vol. 36, no. 1, January 2003, pp. 41–50.

[7] H. Schmeck, “Autonomic computing - vision and challenge for system design,” in Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC’04), Dresden, Germany, September 2004, pp. 3–3.

[8] J.Kennedy and R.Eberhart, “Particle swarm optimization,” Proc. IEEE International Conference on Neural Network (ICNN'95), volume 4, pp.1942-1948.

[9] Maniezzo V, Gambardella LM, De Luigi F, “Ant Colony Optimization, New Optimization Techniques in Engineering,” by Onwubolu, GC, and BV Babu, Springer-Verlag Berlin Heidelberg, 2004, pp. 101-117.

[10] M. Dorigo and T. Stutzle, “The ant colony optimization metaheuristic: Algorithms, applications, and advances,” in Handbook of Metaheuristics, F. Glover and G. Kochenberger, Eds. Norwell, MA: Kluwer.

[11] Kwang Mong Sim and Weng Hong Sun, “Ant Colony Optimization for Routing and Load-Balancing: Survey and New Directions,” IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans, Vol. 33, No. 5, September 2003, pp560-572.

[12] M. Dorigo and G. D. Caro, “The ant colony optimization metaheuristic,” in New Ideas in Optimization, D. Corne, M. Dorigo, and F. Glover, Eds. New York: McGraw-Hill, 1999.

[13] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz, “Ants for Load Balancing in Telecommunication Networks,” Hewlett Packard Lab., Bristol, U.K., Tech. Rep. HPL-96-35, 1996.

[14] Junichi Suzuki and Tatsuya Suda. “A Middleware Platform for a Biologically Inspired Network Architecture Supporting Autonomous and Adaptive Applications,” IEEE Journal on Selected Area in Communications, Vol. 23, No. 2, February 2005.

[15] Hung Q. Ngo, Anjum Shehzad, Saad Liaquat, Maria Riaz and Sungyoung Lee. Developing Context-Aware Ubiquitous Computing Systems with a Unified Middleware Framework. Proceedings of Embedded and Ubiquitous Computing: International Conference EUC 2004, LNCS Volume 3207, Springer Verlag 2004, pp. 672 – 681.

Chapter 3

CONTEXT DELIVERY MODULE FOR AUTONOMIC UBIQUITOUS MIDDLEWARE INFRASTRUCTURE

3 Context Delivery for Autonomic Ubiquitous Middleware Infrastructure

Context awareness is an inevitable issue for Ubiquitous Computing. Middlewares provide the system support, reusability and performs all the functions of context sensing and inferring that are required for developing context aware systems. But one of the major issues for these middlewares is to devise a context delivery scheme that is scalable as well as efficient. Pure unicast or pure broadcast based dissemination can not provide scalability as well as less average latency. In this report we present a scalable context delivery mechanism for context-aware middlewares based on hybrid data dissemination technique where the most requested data are broadcasted and the rest are delivered through unicast to reduce network traffic. We also provide mechanism to dynamically prioritize and classify the hot and cold context data depending on the request rate and longest waiting time. We further perform division of bandwidth depending on the hot and cold items to reduce average latency dynamically. Our solution also addresses the push popularity problem that occurs as the passive client access data without sending explicit requests. We incorporate the leasing mechanism to reduce the periodical requests (polling) for updated value of context and provide a mechanism to reduce the burst of lease renewal requests for better performance. Simulation results validate the performance of our system.

In this report we also provide a detail design of the Context Delivery Module for the middleware A-CAMUS using UML Diagrams. Our future goal is to incorporate prediction in our delivery scheme to make our current delivery scheme more efficient.

3.1 Context Delivery for Ubiquitous Computing

3.1.1 Motivation

Mark Weiser articulated a revolutionary concept of future computing known as ubiquitous computing [1], where context awareness is an inevitable issue. Context awareness is the ability to detect and sense, interpret and respond to the situation of an entity (i.e. user, application) based on local environment [2]. Often, the term "Context-aware Computing" is used synonymously to Ubiquitous Computing (ubicomp). This is because almost every ubicomp application makes use of context information. Middleware solutions provide the system support, reusability and separation of concerns that are required for developing context aware systems (see [4] for a survey). The middleware performs all the function of context sensing and inferring and smart applications utilize this contextual information to provide intelligent support to the users. Every context-aware middlewares should have the following three main phases of execution: a) Acquisition of raw sensor data, b) Context inference from the sensory data, c) Context delivery to applications Hence, the delivery of context is an indispensable part for any context aware middleware to facilitate the building up of “Context-aware Computing”.

3.1.2 Required Properties of Context Delivery Scheme

We may identify the following basic requirements regarding context delivery scheme for ubiquitous computing:

Scalability: The system should be able to deliver a vast set of context to a variety of diverse applications. It should be extensible in order to incorporate new contextual information as the context aware system matures. Furthermore it should also be adaptive with varying request rate and available network bandwidth.

Semantic Interoperability: It should define logical constrains such as ontology for more desired matching of the context advertisements and context queries. Thus it may avoid ambiguity and be more expressive.

Predictability: One of the true success of efficient context delivery lies with the ability of self-optimization. This may be achieved if the system can predict some of the future requests and delivers the contextual information accordingly to the users without getting explicit requests. Beside the use of sequence of requests for predicting next request, the system may also predict future request based on current contexts and delivers accordingly.

Secure: Secure context delivery is one of the fundamental needs for ubiquitous computing. In this regard we need to define policies for access control that should be dynamic in nature in order to incorporate changing system trust level. Access control mechanism should also be autonomous and be interactive to cater for situations in which user has incomplete knowledge about the existing policies of the system. Furthermore we need to incorporate encryption mechanism to secure the delivery of context to the appropriate client.

Real-time: The motto of ubiquitous computing is "Any where, any time". So system's support to users requires being real time in ubiquitous computing environment. As the behavior of these systems totally depends on contexts, the delivery of the contexts should also be real time. Hence the context delivery scheme of any middleware should be able to meet the hard or soft deadlines defined by the applications at the time of request.

This report focuses on the scalable and semantic context delivery mechanism for the middleware named A-CAMUS (Autonomic Context Aware Middleware for Ubiquitous System). In section 2 we describe hybrid dissemination based scalable and adaptive context delivery mechanism. We also incorporate context ontology CONTEL [12] to achieve semantic context delivery. Afterwards section 3 describes the design of Context Delivery module for our middleware A-CAMUS.

3.2 Scalable and Semantic Context Delivery

3.2.1 Motivation

To motivate the scalability issue in context delivery, let us envision a big smart environment (such as a corporate office, academic building, shopping complex etc.) where numerous context-aware applications (we interchangeably use client or receivers), running on mobile devices like PDA or stationary devices (desktop PC), frequently request various contexts to the middleware. This scenario is based on traditional client-server model with wired or wireless connection rather than ad-hoc network. Here the middleware acts as the context providing server and the smart applications are the client for various contexts. We adopted the client/server model for the context aware middleware because of several reasons: First, in a ubiquitous environment the middleware will not only give us abstraction of heterogeneous sensors and communication protocols but also it should generate complex and simple contexts based on current as well as previous data. So it requires a large repository for data. Second, a context-aware middleware has to perform various computation intensive operations such as data mining to infer the preferences, location and different complex situations of users (such as user's mood) etc. Thus the middleware has to have huge computing resources (such as memory, storage capacity, CPU speed) to perform various computation intensive context processing operation. So we assume that the middleware acts as a central server backed by several replications or mirrors (for failure recovery) to provide necessary contexts to context aware applications and thus facilitates the enterprise level ubiquitous computing. In this report we describe an efficient and scalable mechanism of delivery of context to large number of smart applications. By the term "efficient delivery" we mean that the mechanism should cause less network traffic, provide quick response time and deal with dynamic nature of ubiquitous environment (i.e. clients appear and disappear in an unpredictable manner). We essentially put emphasis on scalable delivery of context to large number of clients. For example, large number of clients (applications running on PDA of professors and students in a smart academic building) may request the middleware to provide a particular context (temperature, light, noise of class rooms and location of the user in the room) with periodic (or a-periodic) update of context information. Numerous applications may request for the same context (i.e. temperature of a room) or even some confidential context (i.e. activity in a closed door meeting) but expect quick response time and secure delivery. Some basic challenges in such a large smart environment are discussed in the following paragraphs.

First challenge is to reduce network traffic and provide quick response time for the client. If the context information of interest is the same among different clients, traditional unicast (point to point or pull based) connection-oriented data services are uneconomical because it incurs a lot of unnecessary traffic from clients to server as well as on the reverse direction. Even if the current technology allows us to have high network bandwidth and server capacity, most of it would be underutilized and wasted during non-peak periods. Broadcast (push based) is an efficient and scalable dissemination method in a connection-less mode to any number of clients with no significant performance degradation in terms of access latency[5]; but a major concern for the success of such system is broadcasting the right set of data. Because, broadcasting less important data may cause network overload. On the other hand, on-demand broadcast (pull-push) method the server aggregates the requests of clients and broadcast the data. But broadcasting the context with lowest request rate (cold item) may also increase network traffic. So hybrid approach combines the benefit of broadcasting hot context data (having higher request rate) and that of unicasting the cold context data (having lower request rate) [6]. Even with this suitable and scalable approach, we have the problem of differentiating hot and cold context data and formulate a suitable broadcast scheduling algorithm for quick response time. These challenges are also considered for web databases and mobile computing [5], [6], [7], [8], [9], [10], [11].

Besides, a smart environment is truly dynamic in nature where the context receivers may appear and disappear unpredictably. So the periodical delivery requires incorporating a leasing mechanism [15] or sending of periodical beacon from the requesters [13] so that the contexts are not delivered indefinitely if the requester disappears without prior notice.

Unavailability of a single comprehensive solution to these problems motivates us to devise a novel and scalable context delivery mechanism that resolves all these problems for context aware middlewares in ubiquitous computing domain. Authors have also published this work [22]. In brief, our delivery mechanism has the following properties:

- It uses context ontology for semantic interoperability.
- We dynamically differentiate hot and cold context items to disseminate through broadcast and unicast respectively. So, this adaptive delivery makes it more suitable for ubicomp application.
- Lease mechanism is used instead of periodic context update request (polling) and copes up with dynamic environment.

- We use request rate of an item and longest waiting time of any outstanding request for an item to prioritize as hot or cold items and also perform bandwidth division between hot and cold items for better performance in terms of average latency.
- Our delivery technique also gives solutions to push popularity problem.

Some terminologies that we use in this report are:

- **Average Waiting Time:** The amount of time on average from the instant that a client request arrives at the middleware, to the time that the data is broadcast.
- **Longest Waiting Time:** The maximum amount of time that a client request waits in the service queue before being satisfied.
- **Average Latency:** The average latency for a data item on the push channel is half of the period of the broadcast cycle if we assume that the items are broadcast sequentially. However, the latency for pulled items are totally different because if an item i of size S_i is queued at the server for transmission, the corresponding queuing delay is either $O(S_i)$ or unbounded [7].

3.2.2 Related Work

To the best of our knowledge, the researches in context delivery of the middlewares have not addressed so far the scalability issue where contexts are to be efficiently delivered to large number of mobile or static clients in ubiquitous environment. The most related research to ours is the Context Discovery Protocol (R-CDP) [13] that has been implemented and evaluated in "Reconfigurable Context Sensitive Middleware" (RCSM) [14]. The fundamental difference between R-CDP and our work is that R-CDP uses broadcast to request for a context and the middleware unicast the data to the requester, which is completely opposite to our mechanism as we use unicast for request and combination of broadcast and unicast for delivery. We use the technique of RxW algorithm [11] to prioritize the delivery where they use *Refresh Priority* based on the divergence of contexts and energy consumption of "Provider". We also perform bandwidth division [7] for hot and cold items for optimal average latency. The similarity with their work is that the motivation of their *Neighbor Validation Beacons (NVB)* is same as that of our Lease Renewal and we also have the way of specifying the update threshold for context update notification. However, they do not use context ontology for semantic interpretation. Moreover, R-CDP has not been tested for scalability [13], which we believe an important performance issue for large scale deployment of smart applications.

The idea of Hybrid data dissemination technique was first used in the Boston Community Information System [8] by combining broadcast and interactive communication to provide most updated information to an entire metropolitan area. This scheme is also adopted in [5], [6], [7], [8], [9], and [10] where the issue of mixing push and pull web documents together on a single broadcast channel was examined. But the document classification problem was introduced in [6] and later document classification along with bandwidth division was resolved in [7]. We employ these ideas of hybrid dissemination, scheduling, classification of data, bandwidth division etc. for scalable and efficient delivery of context for ubiquitous computing environment. Though our approach is very similar to [7], but we differ in calculating the popularity of an item not only on the total request but also on the longest waiting time of any outstanding request according to RxW as RxW does not suffer from starvation of request for cold item [11]. Moreover we also employ lease mechanism to reduce the periodic request (polling). Above all, we are considering the delivery of context data other than web documents and our data items are smaller in size than the average size of 8 KB [15] of web documents.

3.2.3 Proposed Scheme

Before introducing our context delivery scheme, let us assume that the middleware service is discovered by the clients through unicast or multicast similar to the look up discovery of JINI [16]. We emphasize that the clients at first discover and then authenticate to build a trust relationship with the middleware before using any middleware services (i.e. context delivery). We further assume that the clients request context information using context ontology (e.g Contel [19]).

3.2.3.1 Hybrid Dissemination Based Context Delivery

We use hybrid dissemination scheme [6] for the context delivery in ubiquitous environment. But this scheme introduces three inter-related data management problems at the middleware: First: the middleware must dynamically classify the requests between hot and cold context data and schedule the delivery according to priority or popularity (*Prioritization*). Second, the middleware should divide dynamically its bandwidth between unicast pull and multicast push for optimal use of bandwidth to ensure low latency (*Bandwidth Division*). Third, as the hot context data are broadcast, some clients may receive the information passively in the sense that they do not send any request for that data. Therefore, the middleware lacks a lot of invaluable information about the data requirement to dynamically decide the hot and cold data item (*Push Popularity Problem [7]*).

3.2.3.2 Formal Context Model and Semantics of Context

For semantic interoperability in context-aware system, a well designed formal modeling of contextual information and their interrelationship is an indispensable requirement. All the application should request a specific context expressing it in standard way based on formal context model or context ontology. The middleware as well as the application should share the same context ontology for interchanging the information. In this regard, we use Contel (more detail is in [12]) as the context ontology that has been designed for our middleware CAMUS [3], [19]. Contel is extensible and as well as reusable for any context-aware system. Contel has categorized formal modeling of context into five top level concepts such as agents, environment, device, location and time (see figure). Agent class has been further classified into SoftwareAgent, Person, Organization, and Group. Each Agent has property hasProfile associated with it whose range is AgentProfile. Also, an Agent isActorOf some Activity. Activity class, representing any Activity, can be classified based on the Actor of it e.g. SingleActivity (which has only one actor), groupActivity (which has Group as its actor and can have many SinlgeActivity instances). An Activity having some object of action on which it is done called ActivityOnObject like CookingDinner, TurnOnLight, or WatchingTV etc., while SelftActivity has no object of action e.g. Sleeping, or Bathing. The Device ontology in Contel is based on FIPA device ontology specification. The environmental context is provided by the various classes in the Environment ontology. Humidity, Sound, Light and Temperature are different environmental information we are utilizing in our framework. This sensed information is available though different sensors deployed in the smart environment, and used by the applications to adapt their behavior. Location ontology is extended from NASA Jet Propulsion Lab space ontology. We are also using the concepts from DAML-Time ontology for temporal context.

It should be noted that this report does not deal with the detail formal modeling of context for large smart environments but we emphasize the use of context ontology for better perception between middleware and applications for efficient context delivery.

3.2.3.3 Message Format

All the clients request for context information according to the context ontology. Clients specify the type of context (e.g. Temperature) and as well as the entity of which this context is related (e.g. Room). “Lease Duration” and “Update Threshold” are also to be specified if a client needs a context for a certain amount of time to avoid polling. Thus the *Request* message contains the following information: Context Type (CT), Entity Type (ET), Entity Id (EID), Lease Duration (LD) and Update Threshold (UT) (see Table 1). If any client does not need periodical update notification, it should specify the “Lease Duration” field and “Update Threshold” field as zero.

It should be noted that the middleware (context server) does not lease a client more than a predefined maximum lease period to block the indefinite delivery of context. On the contrary, the *Reply* message contains CT, ET, EID, value (V), Maximum Lease Duration (MLD), Minimum Update Threshold (MUT) and Report Probability (RP).

3.2.3.4 Prioritization

TABLE 1
MESSAGE FORMAT

Type	Content
Request	{CT, ET, EID, LD, UT}[1]
Reply	{ CT, ET, EID, V, MLD, MUT, RP }[1]

[1]All the new terms are discussed in section 3.4

To overcome the first problem stated in section 3.1, we at first enqueue the requests and categorize the requests into groups depending on the context type and resource entity information as the preprocessing step. These groups form the leaf nodes of our context hierarchy. If multiple requests for the same context are received from the same client, the system keeps only one entry (most recent one) for that request in any of the request lists (see below for different request lists: TLR, TPR). This is because the broadcast delivery of this data will satisfy all the same requests from the same client at the same time. However, this also helps to prevent the *false popularity problem* that may happen if the same client sends multiple requests for the same data intentionally to increase the popularity. To facilitate the delivery mechanism described in this report, the middleware maintains the following information for each of the groups and use of this information will be explained in the subsequent sections:

- Context Id (CID): A unique identifier is assigned to each group.
- Total Leased Request (TLR): Total number of leased requests for this specific context. This value is used in determining whether this data should be scheduled for broadcast (hot item) or unicast (cold item).
- Leased Request List (LRL): This list contains the requesters’ ids (IP address) that have been leased along with their lease duration.
- Max Lease Duration (MLD): Maximum lease duration among the leased duration. This information is also sent along with the data to let the clients know how long this data will be delivered. If any client is receiving the data passively and wants to use longer than this time, it will renew the lease with longer period.
- Total Pending Request (TPR): This field denotes the total number of requests that have been received but no delivery of the context has been done yet. This field is reset to zero after each delivery of the context and incremented after receiving of each new request for this context. The larger the value of this field, the higher the priority of delivery of this context should be.
- Pending Request List (PRL): This list is similar to LRL but contains the requesters’ ids (IP address) and requested lease duration of the pending requests. After the delivery, the

requests with lease duration greater than zero will be added to the LRL and TLR will be updated accordingly.

- First Arrival Time (FAT): This is the arrival time of the first request which is still pending. Longest waiting time (LWT) of any pending request for this context is the difference of current time and FAT. LWT is used to determine the priority of delivering this context together with TPR. FAT is reset to zero after each delivery and set to the arrival time of the first request as it is queued. The larger the value of this field, the higher the priority of delivery of this context should be.
- Last Delivery Time (LDT): The most recent time when the context was delivered. This is used to calculate the longest waiting time of the leased requests in LRL.
- Min Update Threshold (MUT): The minimum of update threshold values among the requests**.** If the context is changed by this amount, it is then scheduled for delivery.
- Candidate for Scheduling (CS): This is a binary value. If the context change exceeds the threshold MUT, the value of this field becomes one (true) and implies this data to be delivered. This field becomes zero (false) with the next delivery of the context data.
- Last Update Time (LUT): This field denotes the time of last update of this context data.
- Frequency of Update (FUT): The frequency of update of this context data.
- Value (V): The current updated value of this context. This field may contain any kind of data (i.e. character, string, double, integer, boolean etc).
- Size (S): Size of this data item.

To set the priorities of the requested context data, we use the total number of requests (R) and longest waiting time of the outstanding request (W) for that item and it is motivated by RxW algorithm [11]. Thus we prioritize a data either because it is very popular or because it has at least one long-outstanding request.

The use of RxW algorithm requires us to define appropriately the values for R (the number of outstanding requests) and W (the longest waiting time); intuitively we have two parameters TLR (Total leased request) and TPR (Total pending request) to consider for R. But TLR comes into account as soon as the change of context value exceeds the MUT (Min Update Threshold) and CS (Candidate for Scheduling) becomes one (true). Similarly, as long as CS is zero, difference of current time (CT) and FAT (First Arrival Time) is the value of W; but as soon as CS becomes one, the longest wait time is the difference of CT and LDT (Last Delivery Time) as all the leased requests have been waiting since LDT. Hence we define RxW with the following equation:

$$RxW = \left(TPR + CS * TLR\right) * \left(\left(CT - FAT\right)\left(1 - CS\right) + CS\left(CT - LDT\right)\right) \quad (1)$$

We calculate RxW value of each group (data item) and sort them in descending order. We update the list each time a request comes and use the same data structure proposed in [11] for efficient maintenance of such list. The RxW value of i th group is denoted as popularity (or priority) p_i in the following sections.

3.2.3.5 Bandwidth Division

The motivation of bandwidth division comes from the fact that the average latency (L) of a data item is less when hot items are assigned to push, cool items to unicast pull and the bandwidth is divided appropriately between the two channels. We used the bandwidth division algorithm similar to that is suggested for web database in [7]. But, we use the prioritization described in section 3.3 rather than using the prioritization based on request rate only as it is used in [15]. The bandwidth division algorithm uses the sorted list of items with decreasing order of popularity, i.e. $p_i \geq p_{i+1} \left(1 \leq i \leq n\right)$, where n is the current size of the list. It is intuitive that if item i is pushed, then $j \le i$ should also be pushed. So, the algorithm tries to partition the list at index k such that the push set $\{1, 2, ..., k\}$ minimizes the latency L given a certain bandwidth B and *pull over-provisioning factor* $\alpha > 1$. The *pull over-provisioning factor* denotes the actual bandwidth we reserve for pull is α times what an idealized estimate predicts and queuing theory asserts that $\alpha > 1$ guarantees bounded queuing delays. The optimal value k^* is found by trying all possible values of L and finally the algorithm determines the pull bandwidth $\alpha \sum_{i=k+1}^{n} \lambda p_i S_i$, which leaves bandwidth $pushBW = B - \alpha \sum_{i=k+1}^{n} \lambda p_i S_i$ for the push channel and average latency for the pushed documents is then $\sum_{i=1}^{k} \frac{S_i}{2\,pushBW}$. Here λ and S_i denote request rate and size of the item respectively. The algorithm runs in $O(n)$ as it performs binary search over all possible values of L and maintains an internal array that stores the total size of each possible partition using binary tree techniques [7].

3.2.3.6 Push Popularity Problem

As the data is broadcast, some passive clients may be satisfied with current MLD and do not need any extension. This introduces a new difficulty known as *push popularity problem*. For example, there may be a situation where a certain context (C1) is used by large number of total clients (leased as well as passive) but only a few of them have lease and most of the clients do not need any extension. On the other hand, another context (C2) may have less number of total clients but may have more leased clients than C1. Thus the middleware will be misguided to calculate higher priority for C2 than C1 based on the number of request and thus will be misguided to consider C2 as hot item.

We can not expect to solve this push popularity problem completely as it will require all the clients to send requests explicitly and hinder the benefit of broadcast. So, a portion of the passive clients should send requests even though the data is ensured to be delivered. The middleware sends a report probability (RP) with the data and a passive client submits an explicit request for this data with probability RP. It is proved in [7] that RP should be set inversely proportional to the predicted access probability for that data and the equation to calculate RP is :

$$RP_i = \begin{cases} \frac{\beta}{\lambda p_i k}, & if\ \lambda p_i k > \beta \\ 0.2, & otherwise \end{cases} \qquad (2)$$

Where β is the difference of Maximum acceptable TCP connection and request arrival rate, λ denotes aggregate request rate and p_i denotes the priority (or popularity) of group i based on total request and k denotes the current number of broadcast items. Here we notice that if $\lambda p_i k < \beta$, the probability will exceed one and hence we specified RP to be 0.2 as a default. It should also be noted that whenever the client sends a request, it sends a complete request with its desired update threshold (UT) and desired lease extension (LE). LE denotes the desired extension after the expiry of current MLD.

3.2.3.7 Lease Renewal

There are two kinds of clients in the system- one who has lease for the context data and the other who passively access the data. Both of them may need to extend the lease (the duration of the delivery) and should send request to the middleware. This request is termed as lease renewal if it is done by the leased clients and is considered as new request if it is sent by any passive client. The middleware may be overwhelmed with requests/lease renewals if all the lease renewal/requests simultaneously arrive as soon as MLD expires. Additionally, if the lease renewal/request with the highest extension arrives earlier than the expiry of MLD, all other clients are exempted from sending the request as the smaller lease duration can not change MLD.

So we incorporate a simple way to emphasize the client with higher lease extension (LE) to send the request earlier and thus avoid the necessity of any co-ordination among clients. In this technique, each client sends a lease renewal or request $\frac{LE}{LEF}$ time units (seconds) before the expiry of MLD with probability RP. Here LEF denotes the Lease Extension Fraction and it is application dependent. We use $LEF = 2$ by default, which means that the clients should send request at least half of its desired lease extension time before the expiry of MLD. In this method the client with large lease extension will request earlier. Here the use of RP is beneficial because if several clients have the same desired lease extension, all of them will not send request and thus it will cause less network traffic. This method also solves the *push popularity problem* as some of the passive clients will also send the lease extension request. Here we should note that the clients who do not want lease extension (with LE=0) will just send request with probability RP as stated in section 3.5.

3.2.4 Performance Evaluation

In order to establish the potential of our proposed context delivery mechanism, we have built a simulation model of the proposed system and evaluated using the simulation tool OMNET++ [18]. All the graphs presented here are generated using the PLOV tool of OMNET++.

3.2.4.1 Simulation Model

In our client-server model the server (our middleware) acts as a data server and delivers self identifying context data items of equal size either by broadcast or unicast upon explicit request. The clients request an item according to Zipf distribution [17] and the time of requests is exponentially distributed with mean M , where $\frac{1}{M}$ is the average request rate of each client. We present the analysis of average latency and network traffic of the proposed system in the following sub-sections.

Table 1: Simulation Parameters

Parameter	**Value**
Total Client	3000
Total Item, N	50
Size of Each Item	200 bytes
Zipf parameter Θ	1.5
System Bandwidth	512,000 bits/sec
Exponential mean M	40
System's Pull Capacity μ	150/sec
α	2
ε	0.005
Lease Duration of a client	10~ 100 ms (uniform)
Lease Renewal Probability	0.7
Lease Renewal Factor LEF	2
Update Threshold	0.5~2 unit (uniform)

Table 1 presents all the simulation parameters. Here the *pull over-provisioning factor* α and the *tolerance factor* ε are used by the bandwidth division algorithm described in [7]. *Lease Duration* and *Update Threshold* in Table 1 are uniformly distributed between the stated ranges. We took the snapshot of the number of clients' requests and replies of the server after every 500 ms in order to plot them.

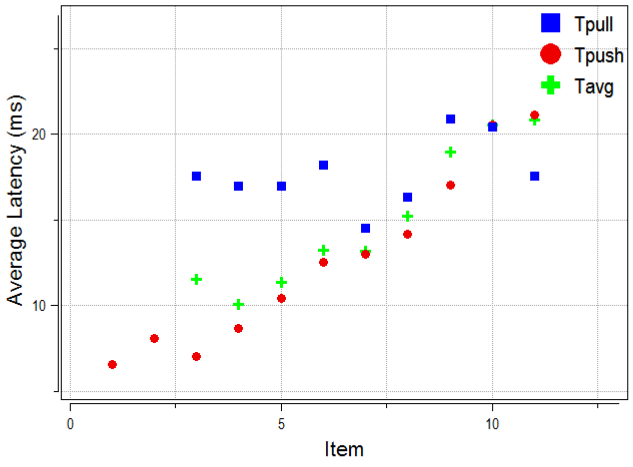


Fig 2 Relation of average latency of Push and Pull as the number of broadcast items changes according to our experiments. Here the intersection of T_{push} and T_{pull} occurs at $k = 10$ and before k=3, T_{pull} grows arbitrarily large.

3.2.4.2 Average Latency

Let $G(k)$ be the average latency (T_{avg}) if the k most popular items are broadcasted. The function $G(k)$ is a weighted average of the average latency of pushed items $T_{push} = \sum_{i=1}^{k} \frac{S_i}{2\,pushBW}$ and the average latency for the pulled items $T_{pull} = \frac{1}{\mu - \left(\sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{k} \lambda_i \right)}$, where λ_i is the Poisson request rate for each item i [6], [7]. Our result is shown in Fig 2. Notice that the minimum of $G(k)$ is to the left of the intersection (at k=10) of the push and pull curves though theoretically it should be on the right side of the intersection [6]. The minimum of $G(k)$ occurs at a relatively small value of k and precedes the intersection due to two complementary reasons. First, the most popular items are chosen for push and are also those to which a Zipf distribution gives substantially more weight. So, if an item is delivered using broadcast, it will also have the largest impact on the globally average delays. As the numbers of the most popular items are small and are broadcasted first, the overall minimum delay occurs for small values of k. Second, pull delays are actually minimized at the points k' where the pull-curve flattens out. However k' precedes the intersection in our graph, and so the overall minimum occurs before that intersection.

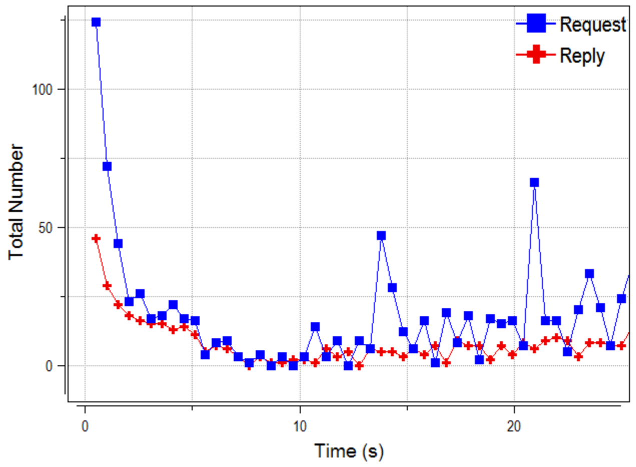


Fig 3 Number of request and reply with change of time. The number of reply denotes total of broadcast and unicast items.

3.2.4.3 Network Traffic

Fig 3 presents our simulation result regarding network traffic. Here we can see that in the beginning of time, the number of request is high. But as the server starts to deliver items, the number request decreases due to two reasons. First, the replies contain the maximum lease period and minimum threshold value for the context items and the clients do not need to send explicit request again until the lease expires. Second, as the most popular items are broadcast, the clients that also need the data do not send request but uses the data passively. But we can see some spikes in the request graph because of the lease renewal requests and the requests sent by the passive clients due to *Report Probability* as we have already discussed. Here we see that incorporation of lease mechanism and threshold reduces overall network traffic from client as well as from server. Besides, the lease renewal and Report Probability cause the generation of necessary traffic from clients to determine the hot and cold item at server. Fig 4 shows the number of broadcast and unicast items with the change of time.

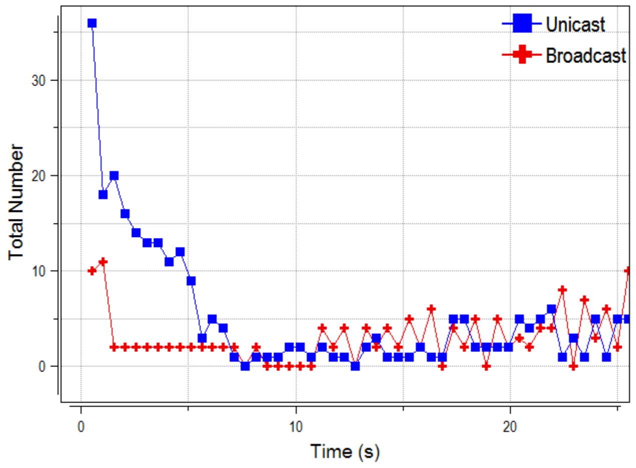


Fig 4 Number of Broadcast and Unicast items with change of time. According to our approach, some of the requested items are broadcasted while the remaining items are delivered by unicast. Total numbers of broadcast and unicast replies are shown in Fig 3.

3.2.5 Summary

In this report we present a scalable context delivery mechanism for context-aware middlewares based on hybrid data dissemination technique where the most requested data are broadcasted and the rest are delivered through unicast to reduce network traffic. Our mechanism dynamically prioritizes and classifies the hot and cold context data depending on the request rate and longest waiting time. We further perform division of bandwidth depending on the hot and cold items to reduce average latency dynamically. Our solution also addresses the push popularity problem that occurs as the passive client access data without sending explicit requests. We incorporate the leasing mechanism to reduce the periodical requests (polling) for updated value of context and provide a mechanism to reduce the burst of lease renewal requests for better performance. Currently we are engaged in developing a prototype using JINI [16] and JRMS [20] for our middleware CAMUS [3], [19].

There is a lot of interesting works to be done in the near future for efficient context delivery. We plan to investigate indexing scheme, cache invalidation report (IR) scheme, real time delivery, predictive broadcast of context and secure delivery of context in future.

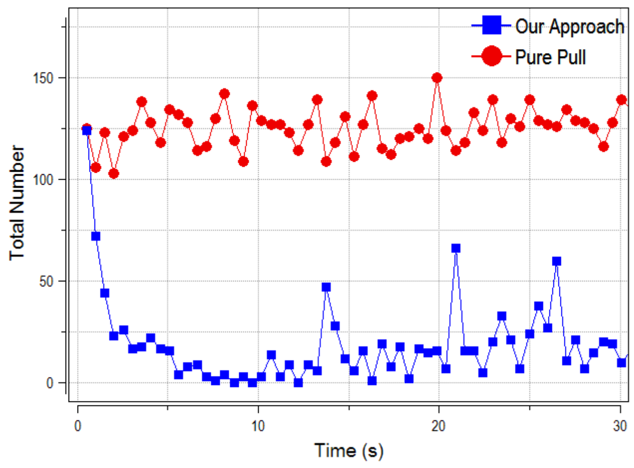


Fig 5 Number of requests in our approach and in pure pull approach. Simulation results show that our approach causes fewer requests as it avoids polling and uses lease mechanism.

3.3 Design of Context Delivery Module

3.3.1 Design Goal

We design the context delivery module (CDM) as a service based on Service Oriented Architecture (SOA) [21].

Service Oriented Architecture (SOA): SOA interaction comprise of a service provider, a service consumer and a registry (Fig 6). A service consumer can look for a service provider using the registry. A service lease specifies the amount of time or contract for which the interaction with the service is valid. The service provider supplies a service proxy to the service consumer. The service consumer executes the request by calling an API function on the proxy. It then formats the request message and executes the request on behalf of the consumer. Figure`6 shows a typical setup of a service oriented system.

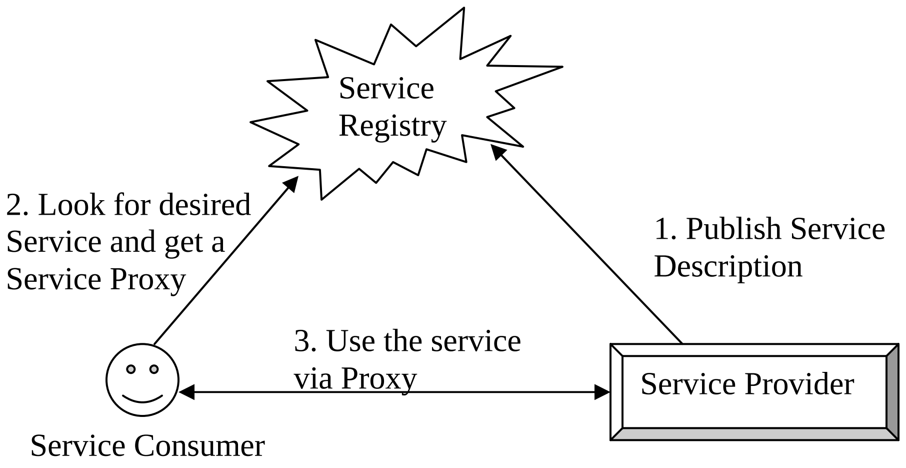

Service Registry
2. Look for desired Service and get a Service Proxy
1. Publish Service Description
3. Use the service via Proxy
Service Provider
Service Consumer

Fig 6 Service Oriented Architecture

The Context Delivery Module has a service interface because of the following promising aspect of SOA:

Dynamic Discovery: In ubiquitous computing environment it is very necessary that users get their desired service very easily and quickly. If the contextual information is provided through a static address such as URL or IP address, new users may not know that address. For example, a new customer may not know the appropriate address of context delivery service in a shopping complex and as a result, the context-aware smart application running on his/her PDA or mobile device will be unable to operate. SOA rescues us from this problem as the application may itself discover the context delivery service from the well known service registry such as Jini Lookup service [16]. We implement the context delivery module as a Jini service [16].

3.3.2 Architecture and System Work Flow

The main entities of context delivery module are Jini Service Interface, Context Delivery Manager, Request Queue, Schedule Manager, Priority Calculator, Bandwidth Allocator and Dissemination Manager (Fig 7).

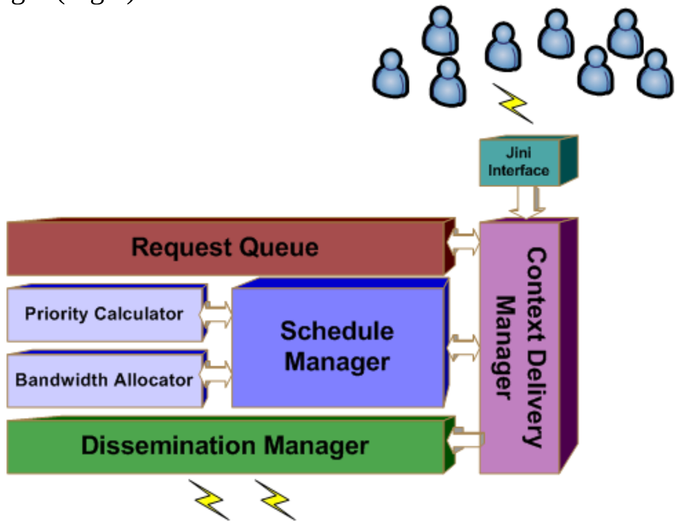


Fig 7 Architecture of Context Delivery Module

Context Delivery Module publishes a *Jini Service Interface* through which clients submit their requests. Clients at first finds out this interface using Jini Lookup service [16] and get a proxy of this interface. Then using this proxy clients submit requests and the proxy, on behalf of the clients, perform all the task of remote procedure call (RPC) to handover the request to *Context Delivery Manager*. *Context Delivery Manager* controls the dataflow among various modules. As soon as it gets a request from *Jini Interface*, it enqueues that request into the *Request Queue*. *Request Queue* maintains a tree like data structure to store the requests to optimize the searching and grouping of requests. Context Delivery Manager has two internal sub modules running as thread. As the client may take lease to get certain context during the lease period, One of the threads polls the repository to get the context values and check whether the change of value exceeds the threshold specified by the clients. The other thread in the *Context Delivery Manager* takes the pending requests and hands over to *Schedule Manager*. *Schedule Manager* prioritizes the requests using the logic inside *Priority Calculator* module and then assigns the requests to multicast or unicast delivery using the logic inside *Bandwidth Allocator*. The algorithm of *Priority Calculator* and *Bandwidth Allocator* is described in section 2. Thus Schedule Manager schedules the delivery of context according to the priority and determines the way of delivery such as using multicast or unicast. Then the actual delivery is performed by the *Dissemination Manager*. *Dissemination Manager* has the ability to deliver the context using multicast channel or using unicast (Fig 8). In our implementation, JRMS [20] is used for reliable multicast delivery.

3.3.3 Interface with Other Module

There are two interfaces between *Context Delivery Manager* and client. One is for sending request and another is to deliver context to client. Client submits request using

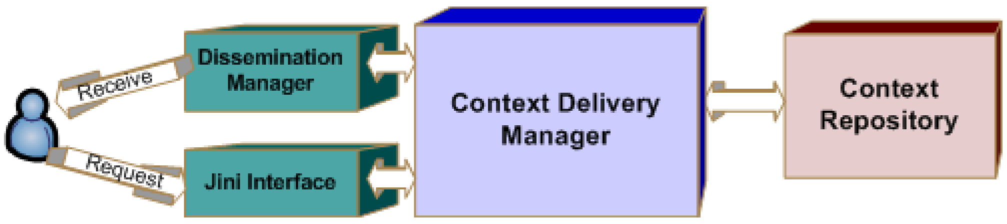


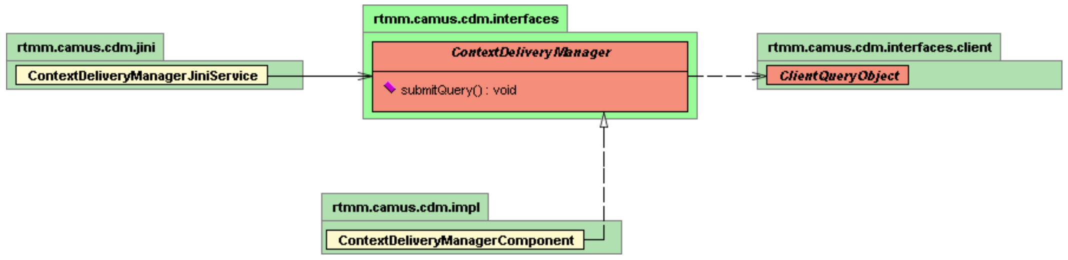
Fig 8 Interfaces of Context Delivery Manager with other modules

Jini Interface and gets contexts delivered by *Dissemination Manager* (Fig 8). On the other hand Context Delivery Manager is connected with *Context Repository* to get the context values. Context Repository should implement the *Context Provider Interface* (see section 3.4.1 for class diagram)

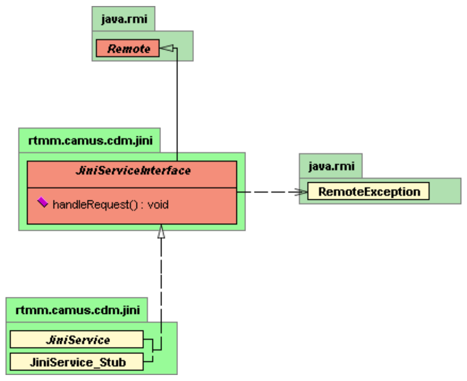
3.3.4 UML Design

Class Diagrams

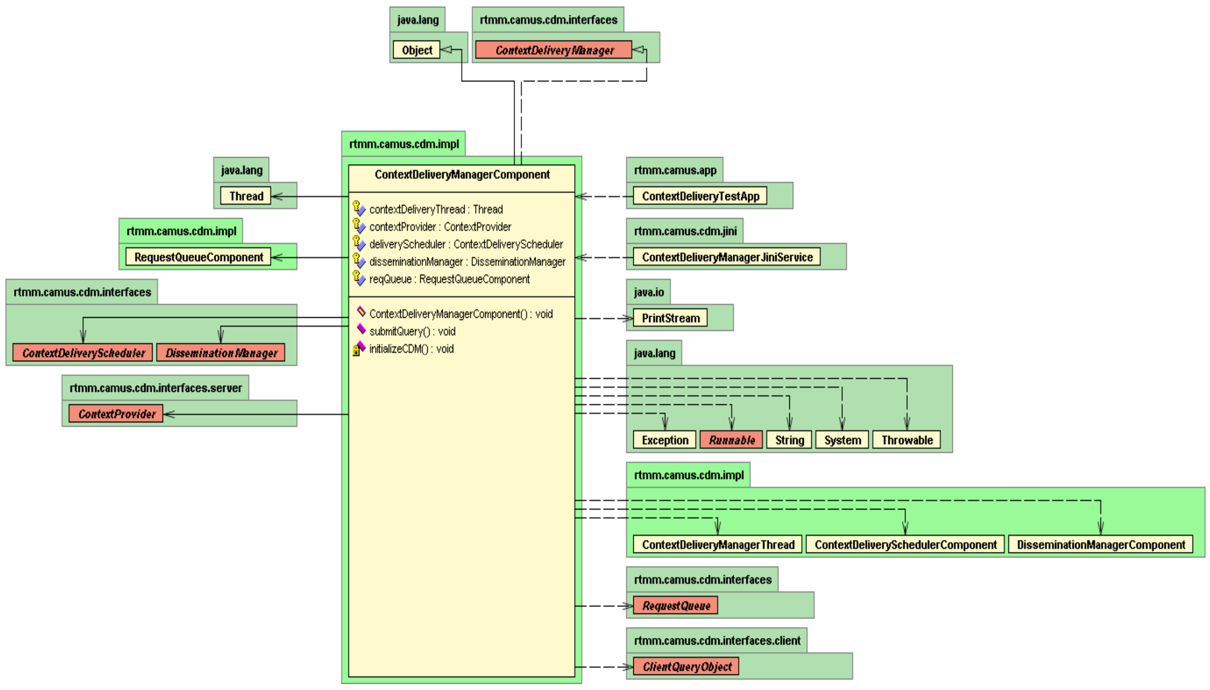
Context Delivery Manager:


rtmm.camus.cdm.jini
ContextDeliveryManagerJiniService
rtmm.camus.cdm.interfaces
ContextDeliveryManager
submitQuery() : void
rtmm.camus.cdm.interfaces.client
ClientQueryObject
rtmm.camus.cdm.impl
ContextDeliveryManagerComponent

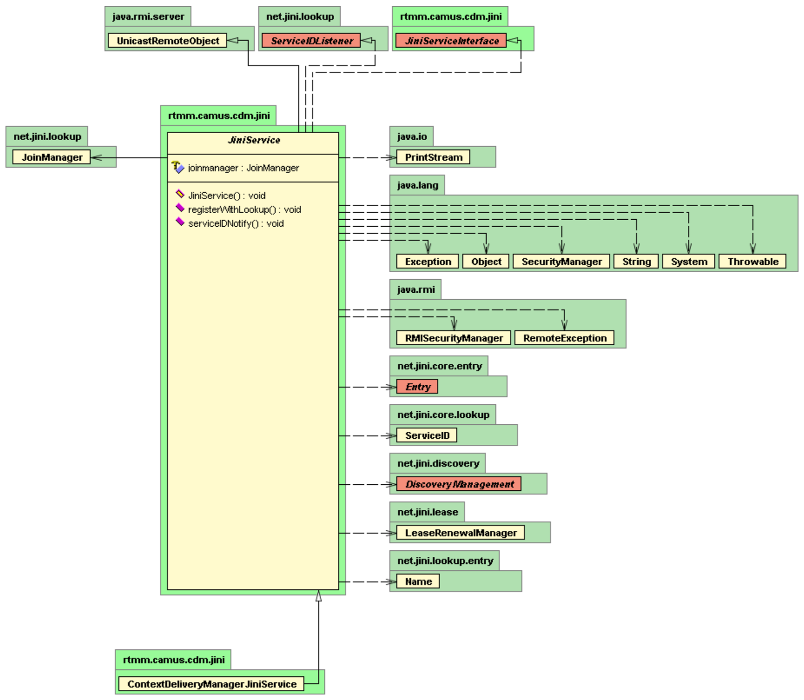
Jini Service:



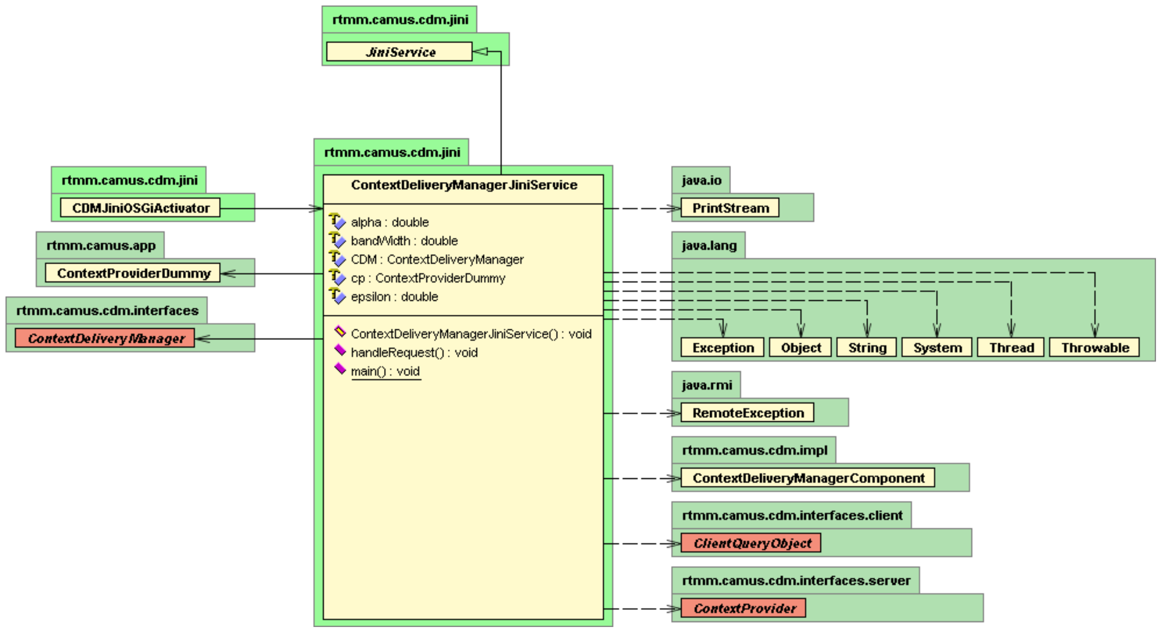
Context Delivery Manager:


java.lang
Object
rtmm.camus.cdm.interfaces
ContextDeliveryManager
rtmm.camus.cdm.impl
ContextDeliveryManagerComponent
contextDeliveryThread : Thread
contextProvider : ContextProvider
deliveryScheduler : ContextDeliveryScheduler
disseminationManager : DisseminationManager
reqQueue : RequestQueueComponent
ContextDeliveryManagerComponent() : void
submitQuery() : void
initializeCDM() : void
java.lang
Thread
rtmm.camus.cdm.impl
RequestQueueComponent
rtmm.camus.cdm.interfaces
ContextDeliveryScheduler
DisseminationManager
rtmm.camus.cdm.interfaces.server
ContextProvider
rtmm.camus.app
ContextDeliveryTestApp
rtmm.camus.cdm.jini
ContextDeliveryManagerJiniService
java.io
PrintStream
java.lang
Exception
Runnable
String
System
Throwable
rtmm.camus.cdm.impl
ContextDeliveryManagerThread
ContextDeliverySchedulerComponent
DisseminationManagerComponent
rtmm.camus.cdm.interfaces
RequestQueue
rtmm.camus.cdm.interfaces.client
ClientQueryObject

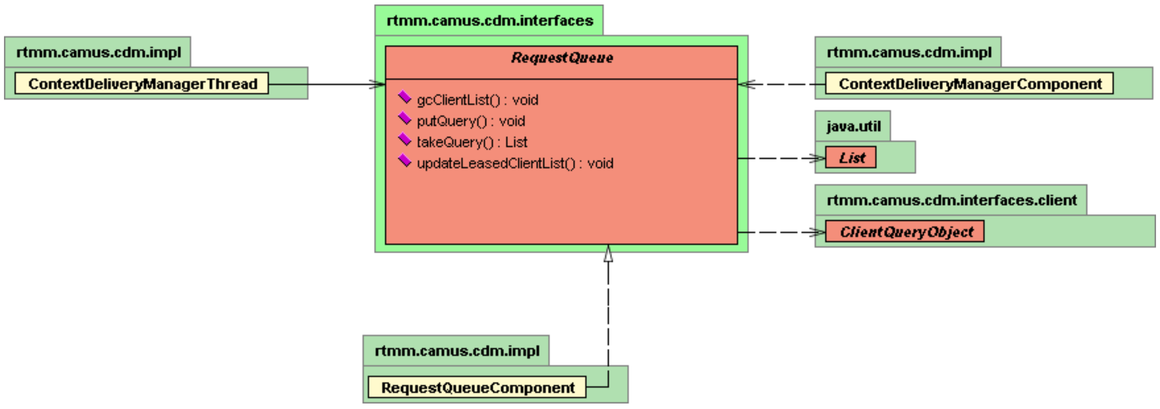
Jini Service:



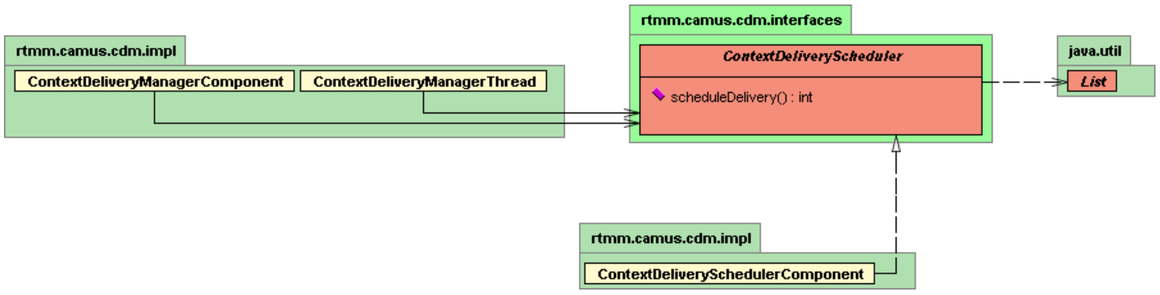
Context Delivery Manager Jini Service:



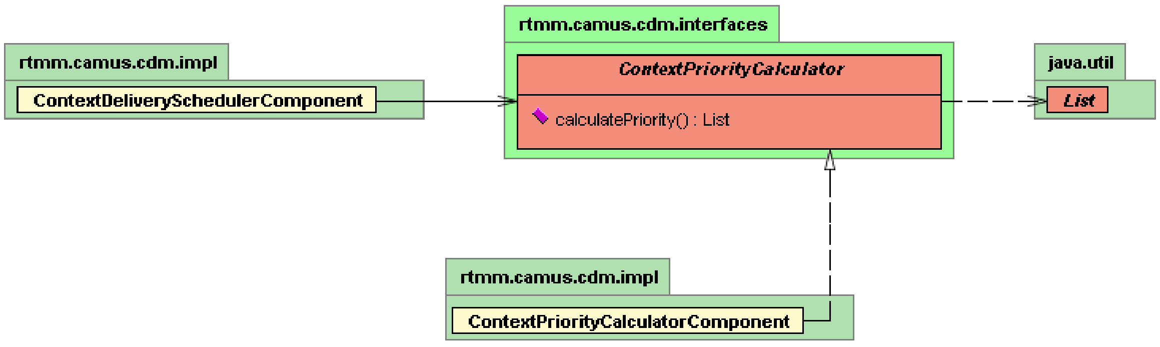
Request Queue:



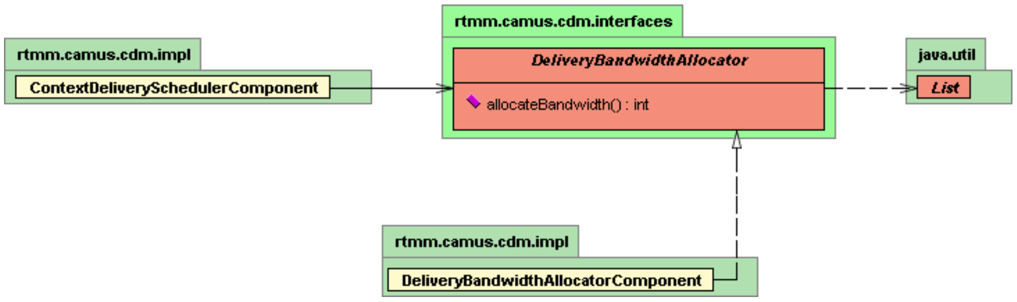
Context Delivery Scheduler:


rtmm.camus.cdm.impl
ContextDeliveryManagerComponent
ContextDeliveryManagerThread
rtmm.camus.cdm.interfaces
ContextDeliveryScheduler
scheduleDelivery() : int
java.util
List
rtmm.camus.cdm.impl
ContextDeliverySchedulerComponent

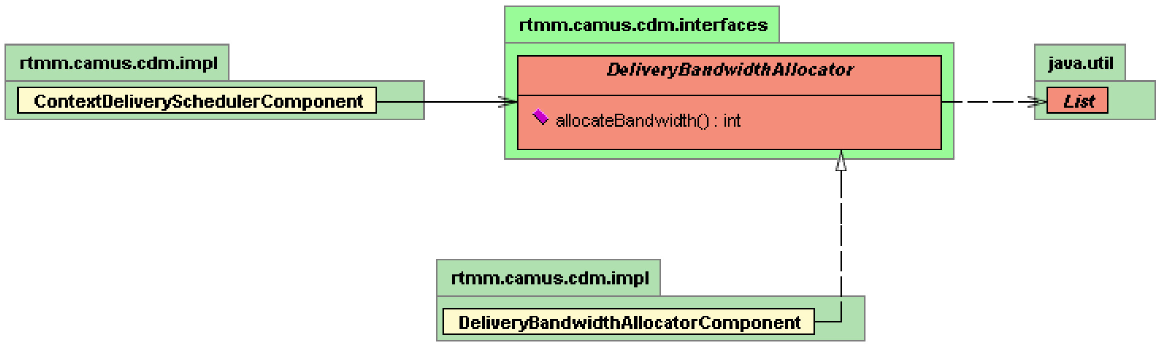
Context Priority Calculator:


rtmm.camus.cdm.impl
ContextDeliverySchedulerComponent
rtmm.camus.cdm.interfaces
ContextPriorityCalculator
calculatePriority() : List
java.util
List
rtmm.camus.cdm.impl
ContextPriorityCalculatorComponent

Bandwidth Allocator:

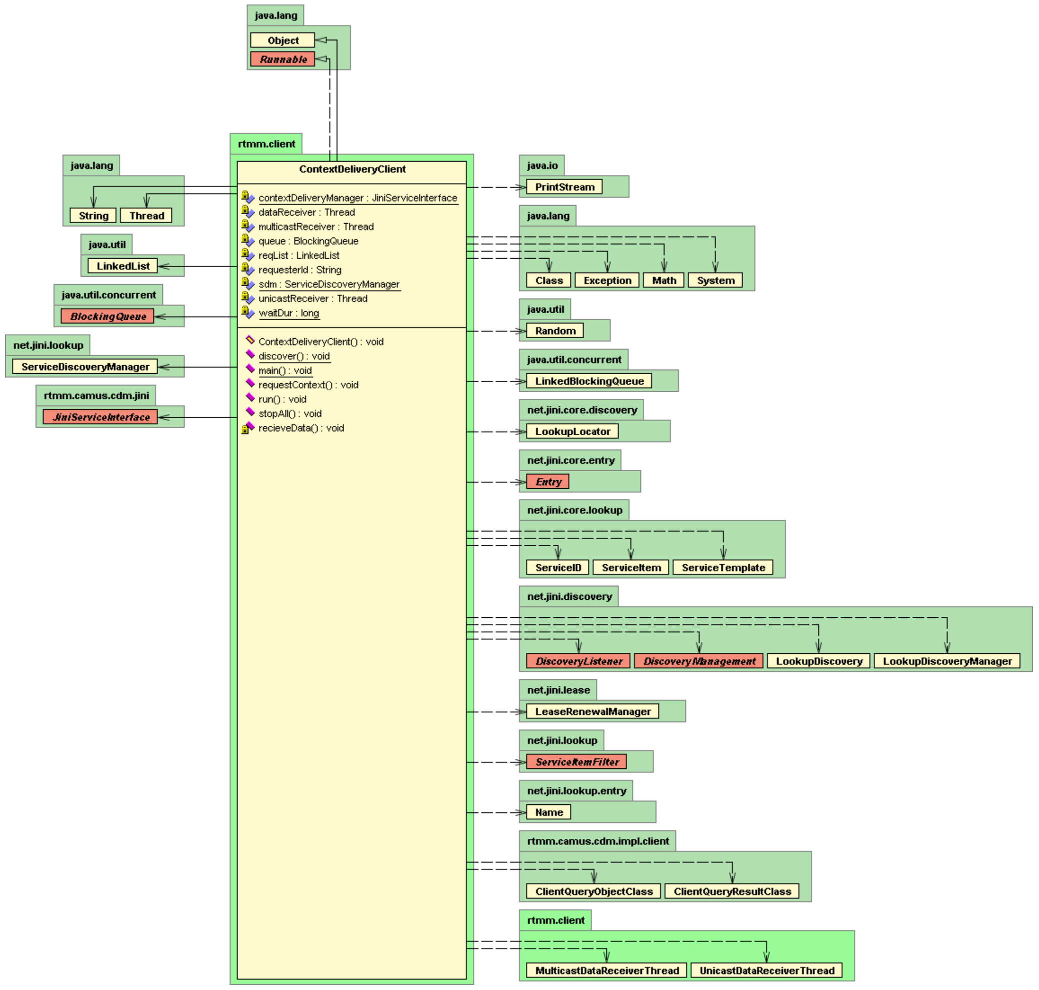

rtmm.camus.cdm.impl
ContextDeliverySchedulerComponent
rtmm.camus.cdm.interfaces
DeliveryBandwidthAllocator
allocateBandwidth() : int
java.util
List
rtmm.camus.cdm.impl
DeliveryBandwidthAllocatorComponent

Dissemination Manager:

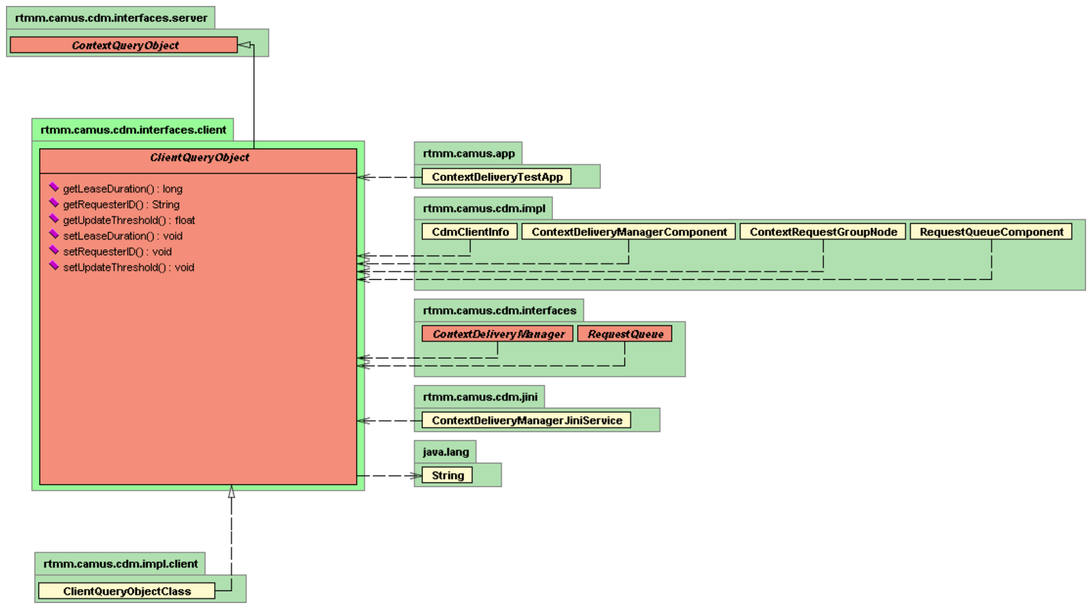


rtmm.camus.cdm.impl
ContextDeliverySchedulerComponent
rtmm.camus.cdm.interfaces
DeliveryBandwidthAllocator
allocateBandwidth() : int
java.util
List
rtmm.camus.cdm.impl
DeliveryBandwidthAllocatorComponent

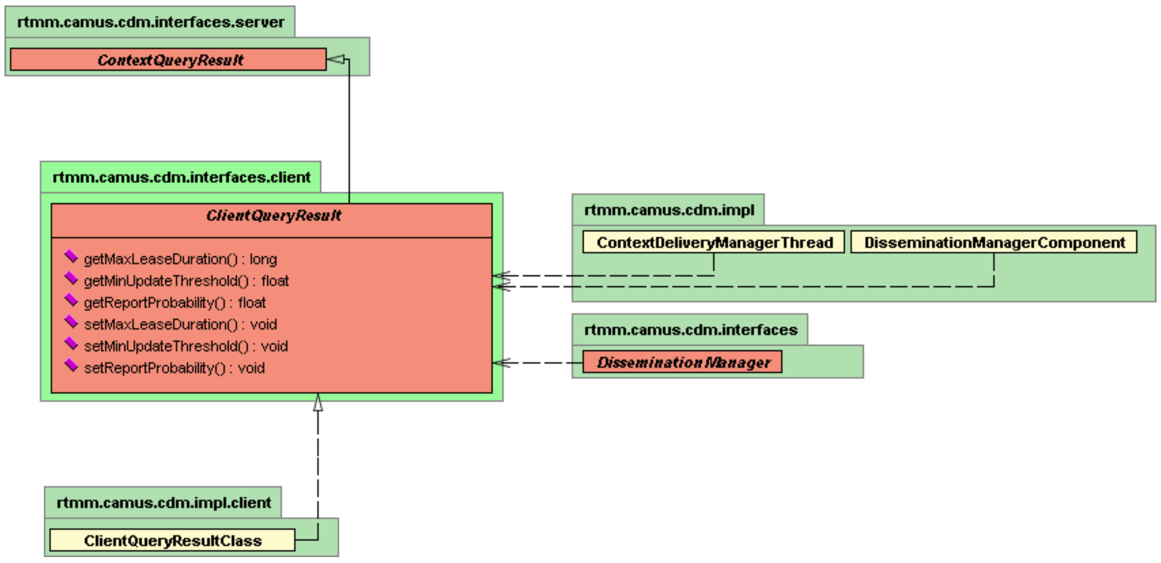
Client:



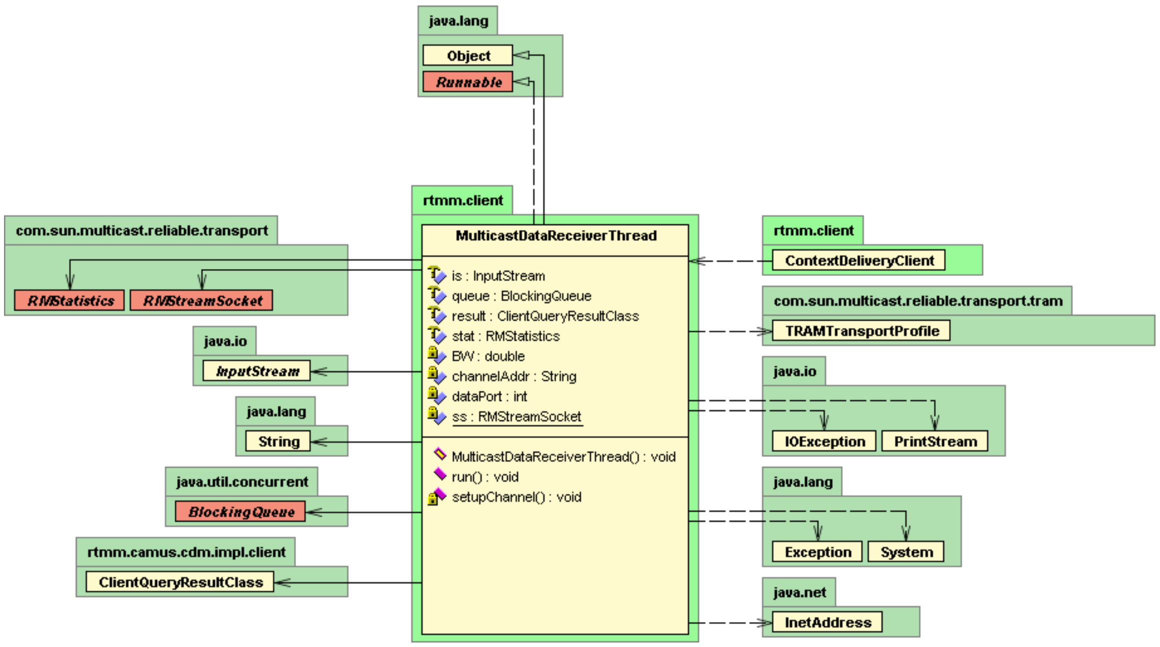
Client Query Object: Context is requested in this format by the client


rtmm.camus.cdm.interfaces.server
ContextQueryObject
rtmm.camus.cdm.interfaces.client
ClientQueryObject
getLeaseDuration() : long
getRequesterID() : String
getUpdateThreshold() : float
setLeaseDuration() : void
setRequesterID() : void
setUpdateThreshold() : void
rtmm.camus.app
ContextDeliveryTestApp
rtmm.camus.cdm.impl
CdmClientInfo
ContextDeliveryManagerComponent
ContextRequestGroupNode
RequestQueueComponent
rtmm.camus.cdm.interfaces
ContextDeliveryManager
RequestQueue
rtmm.camus.cdm.jini
ContextDeliveryManagerJiniService
java.lang
String
rtmm.camus.cdm.impl.client
ClientQueryObjectClass

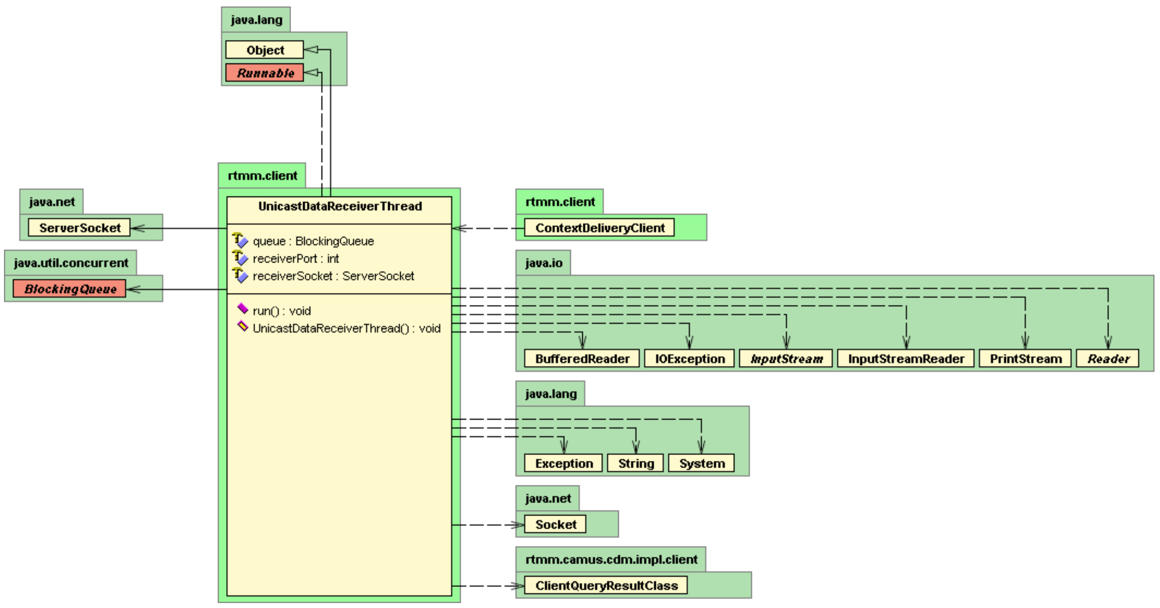
Client Query Result: Context is delivered in this format to the client


rtmm.camus.cdm.interfaces.server
ContextQueryResult
rtmm.camus.cdm.interfaces.client
ClientQueryResult
getMaxLeaseDuration() : long
getMinUpdateThreshold() : float
getReportProbability() : float
setMaxLeaseDuration() : void
setMinUpdateThreshold() : void
setReportProbability() : void
rtmm.camus.cdm.impl
ContextDeliveryManagerThread
DisseminationManagerComponent
rtmm.camus.cdm.interfaces
DisseminationManager
rtmm.camus.cdm.impl.client
ClientQueryResultClass

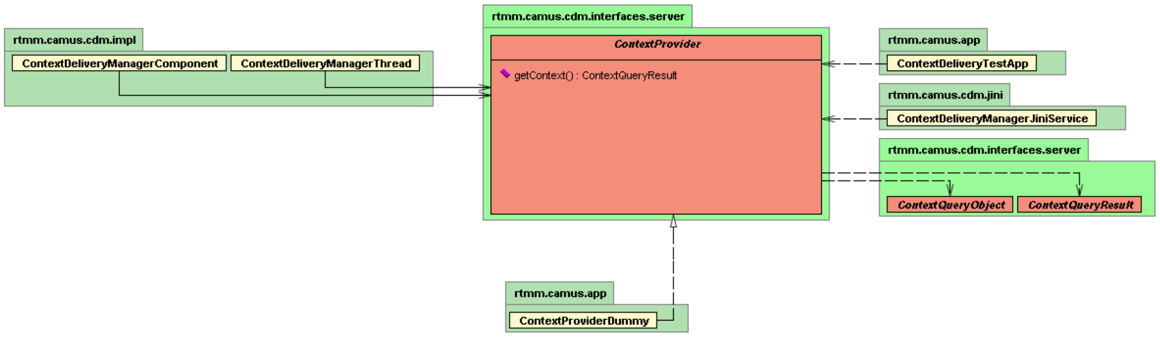
Multicast Receiver Thread: Used by the client to receive data from multicast channel. It uses JRMS [20].



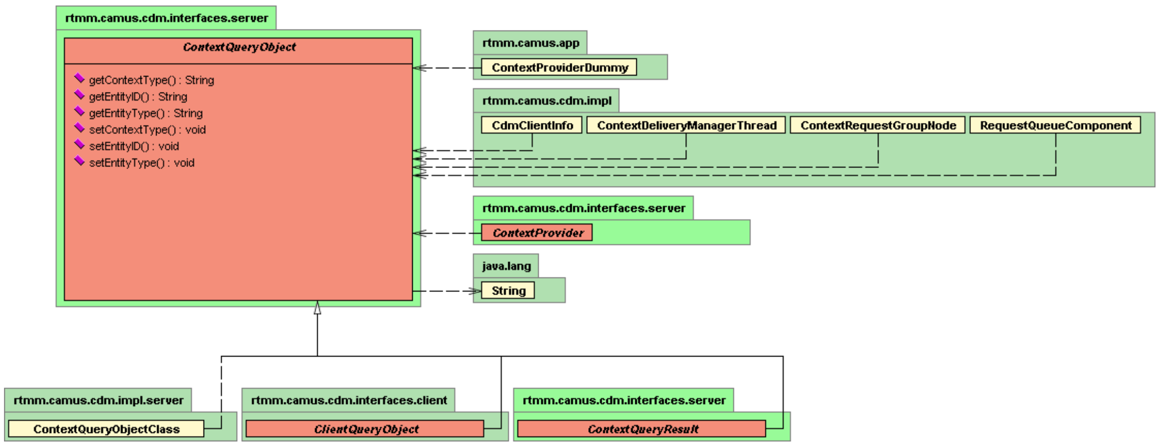
Unicast Receiver Thread: Used by the client to receive data from though unicast.



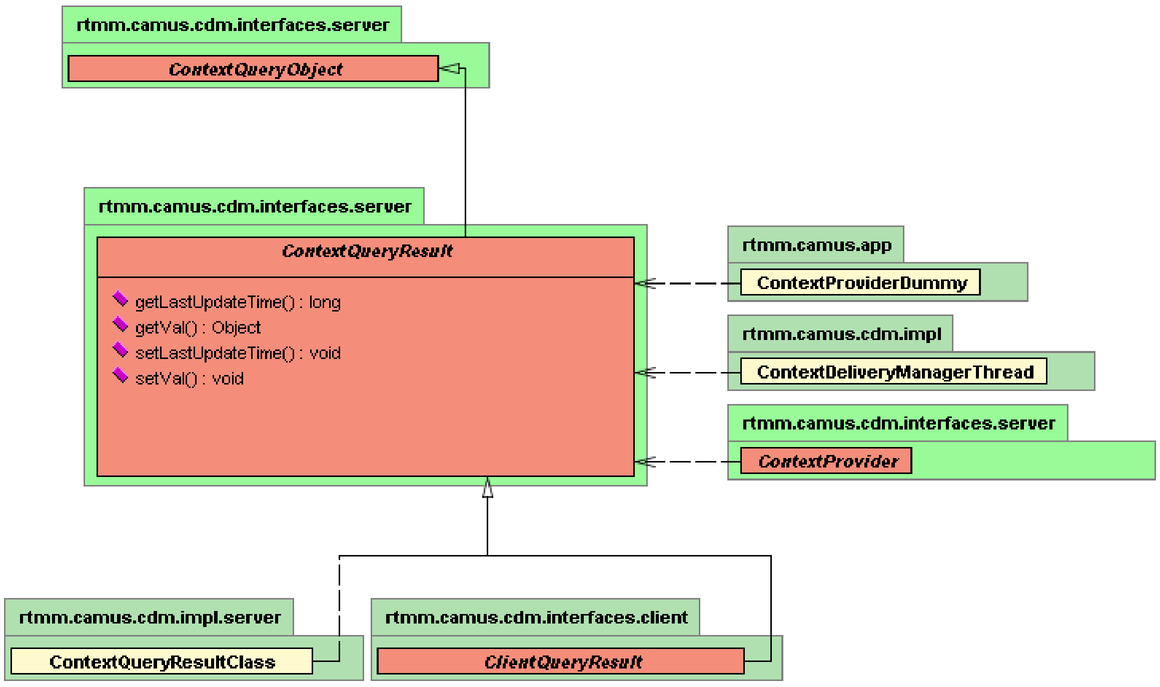
Context Provider:


rtmm.camus.cdm.impl
ContextDeliveryManagerComponent
ContextDeliveryManagerThread
rtmm.camus.cdm.interfaces.server
ContextProvider
getContext() : ContextQueryResult
rtmm.camus.app
ContextDeliveryTestApp
rtmm.camus.cdm.jini
ContextDeliveryManagerJiniService
rtmm.camus.cdm.interfaces.server
ContextQueryObject
ContextQueryResult
rtmm.camus.app
ContextProviderDummy

Context Query Object: Context is requested in this format by the *Context Delivery Manager* to the *Context Provider*

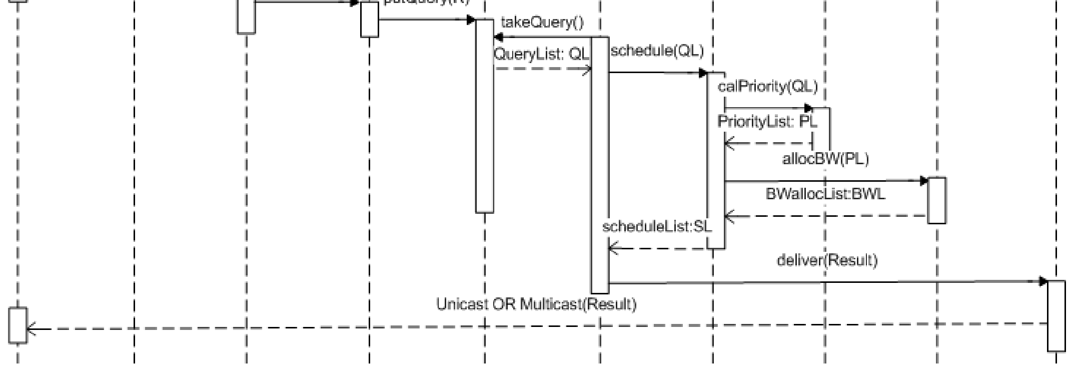


Context Query Result: Context is delivered in this format to the *Context Delivery Manager* by the *Context Provider*


rtmm.camus.cdm.interfaces.server
ContextQueryObject
rtmm.camus.cdm.interfaces.server
ContextQueryResult
getLastUpdateTime() : long
getVal() : Object
setLastUpdateTime() : void
setVal() : void
rtmm.camus.app
ContextProviderDummy
rtmm.camus.cdm.impl
ContextDeliveryManagerThread
rtmm.camus.cdm.interfaces.server
ContextProvider
rtmm.camus.cdm.impl.server
ContextQueryResultClass
rtmm.camus.cdm.interfaces.client
ClientQueryResult

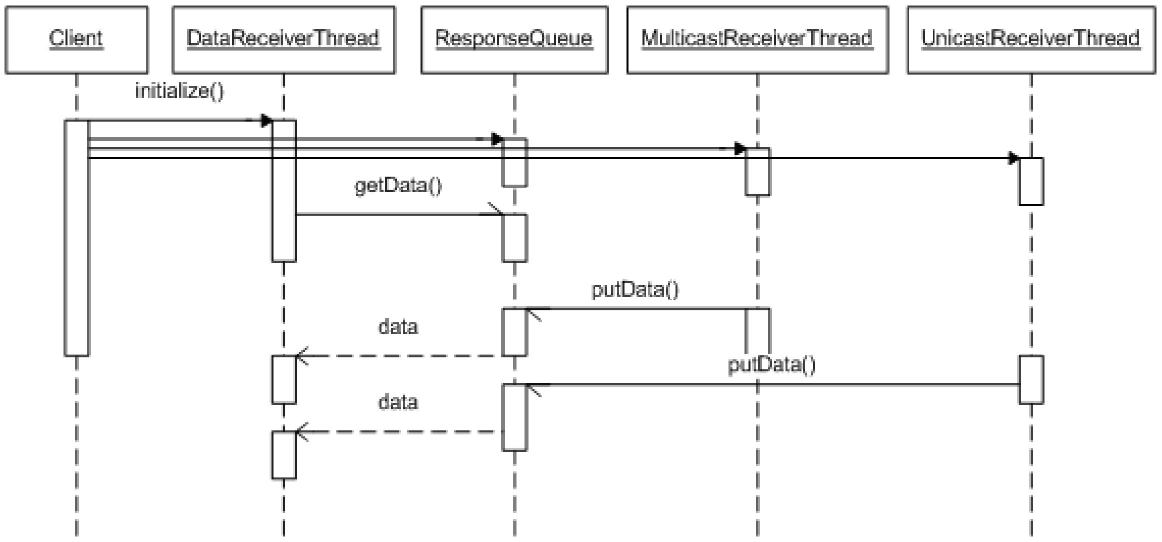
Se

Re

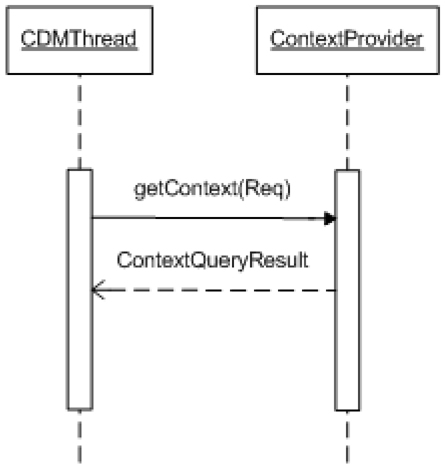


takeQuery()
QueryList: QL
schedule(QL)
calPriority(QL)
PriorityList: PL
allocBW(PL)
BWallocList:BWL
scheduleList:SL
deliver(Result)
Unicast OR Multicast(Result)

Client Receives Context:



Context Delivery Manager gets Context from Context Provider: Context Delivery Thread is responsible for polling context values.



3.4 Reference

1. Weiser, M.: The Computer for the 21st Century. In: Scientific America, Sept. 1991, pp. 94-104; reprinted in IEEE Pervasive Computing, 2002, pp. 19-25
2. Dey, A.K, Abowd, G.D.: Towards a Better Understanding of Context and Context-Awareness. In Proc. of the 2000 Conference on Human Factors in Computing Systems, The Hague, The Netherlands, (April 2000)
3. Hung, N.Q., Shehzad, A., Kiani, S.L., Riaz, M., Lee, S.Y.: Developing Context-Aware Ubiquitous Computing Systems with a Unified Middleware Framework. In: Proc. of Embedded and Ubiquitous Computing: EUC 2004, LNCS Volume 3207, Springer-Verlag (2004), pp. 672 – 681
4. Baldauf, M., Dustdar, S.: A Survey on Context-aware systems. Int. J. of Ad Hoc and Ubiquitous Computing, forthcoming
5. Acharya, S., Franklin, M., Zdonik, S.: Balancing push and pull data broadcast. In *ACM SIGMOD*, (May 1997)
6. Stanthatos, K., Roussopoulos, N., Baras, J. S.: Adaptive data broadcast in hybrid networks. In the 23rd Int. Conf. on VLDB, 30(2), (Sep. 1997)

7. Beaver, J., Morsillo, N., Pruhs, K., Chrysanthis, P. K.: Scalable Dissemination: What's Hot and What's Not. In the Seventh Int. Workshop on the Web and Databases (WebDB 2004), Paris, France, June 17-18 (2004)
8. Gifford, D.: Polychannel Systems for Mass Digital Communications. *CACM*, 33(2):141-151, (Feb. 1990)
9. Hall, A., Taubig, H.: Comparing push- and pull-based broadcasting or: Would "microsoft watches" profit from a transmitter? Lecture Notes in Computer Science, 2647 (Jan. 2003)
10. Triantafillou, P., Harpantidou, R., Paterakis, M.: High performance data broadcasting systems. Mobile Networks and Applications, 7 (2002) 279–290
11. Aksoy, D., Franklin, M.: RxW: A scheduling approach for large-scale on-demand data broadcast. IEEE/ACM Transactions On Networking, 7(6) (Dec. 1999) 846-860
12. Shehzad, A., Hung, N.Q., Pham, K.A., Lee, S.Y.: Formal Modeling in Context Aware Systems. In Proc. of Workshop on Modeling and Retrieval of Context, CEUR, ISSN 613-0073, Vol-114, Germany (2004)
13. Yau, S.S, Chandrasekar, D., Huang, D,: An Adaptive, Lightweight and Energy-Efficient Context Discovery Protocol for Ubiquitous Computing Environments. In the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04)
14. Yau, S.S, Karim, F, Wang, Y, Wang, B., Gupta, S.: Reconfigurable Context-Sensitive Middleware for Pervasive Computing. IEEE Pervasive Computing, 1(3), July-September (2002), pp.33-40.
15. Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. In IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 8, NO. 3, (JUNE 2000)
16. Sun Microsystems, Inc.: Jini™ Architecture specification. http://www.sun.com/jini/specs/
17. Wentian Li, References on Zipf's Law. URL: http://www.nslij-genetics.org/wli/zipf/
18. OMNET++, URL: http://www.omnetpp.org/index.php
19. Shehzad, A., Hung N.Q., Anh, K.P.M. , Riaz, M., Liaquat, S., Lee, Y.K, Lee, S.Y: Middleware Infrastructure for Context-aware Ubiquitous Computing Systems. CAMUS Technical Report (TR-V3.2). February 2005. http://oslab.khu.ac.kr/camus
20. Rosenzweig, P, Kadansky, M, Hanna, S. The Java Reliable Multicast Service: A Reliable Multicast Library. SMLI TR-98-68, Sun Microsystems, 1998.
21. Furmento, N., Hau, J., Lee, W., Newhouse, S.: Darlington, J. Implementations of a Service-Oriented Architecture on top of Jini, JXTA and OGSA. In: Proceedings of the UK e-Science Program All Hands Meeting 2003, Nottingham, UK. Sept., 2003
22. Lenin Mehedy, Md. Kamrul Hasan, Young Koo Lee, Sungyoung Lee, Sang Man Han, "Hybrid Dissemination Based Scalable and Adaptive Context Delivery for Ubiquitous Computing", The 2006 IFIP International Conference on Embedded And Ubiquitous Computing · (EUC' 2006), (LNCS). August 01-04, 2006, Seoul, Korea

Future plan: Adaptive, Scalable and Proactive Context Delivery for Ubiquitous Computing

Abstract: In a ubiquitous environment, numerous clients will request for different contexts. The context provider should be scalable in the sense that it should be able to satisfy the need of context information among large number of clients without performance degradation. On the other hand due to the existence of mobile devices, number of clients as well as their requests for contexts may change dramatically. Hence the system should be adaptive to the number of requests. Scalability may be achieved through broadcasting the items. But that may consume valuable bandwidth with less important data. On the contrary, pure unicast can not achieve scalability as in this case the server communicates with each of the client independently. Thus to achieve the scalability and adaptability in context delivery we have already incorporated hybrid dissemination approach [1], [3], where the hot item (most requested) are broadcasted and the cold items are unicasted. But this approach introduces the challenge of identifying hot and cold items. Though request count is a good candidate to calculate the hot item, but that may lead to starvation of delivery of cold item (with fewer requests). To cope up with this problem we need to take account of the waiting time along with the number of requests. This approach is also known as RxW algorithm [2]. Thus we give higher priority to the item with highest number of request or the item with longest waiting time. We further try to perform optimal bandwidth division between broadcast and unicast [1] to reduce average latency in receiving the context. Thus our previous work [1] proposes a scalable and adaptive context delivery mechanism for ubiquitous computing.

But still we can improve the delivery method if we can predict the future requests and disseminate before getting explicit request for them. Through this proactive approach, we can achieve better client satisfaction and reduce network traffic from the client side. Markov Tree [4], Association Rule [5] etc. are some of the approaches to predict the future requests. However, even if we incorporate different prediction mechanism, we need to find answers to the questions such as:

1. Whether we should use online or offline prediction approach. How should we gather enough information for prediction?
2. What kind of information should be used to predict future request in Ubiquitous Computing environment?
3. When should we deliver these predicted items? As these items are not requested, they have the lower priority than the items that have already been requested. So should we deliver between the delivery of broadcast and unicast or after the delivery of all the requested items? How the bandwidth division should be performed?
4. How should we deliver these items? Should we deliver through broadcast or unicast?
5. How may we know that the delivery of predicted items were successful? A delivery of predicted item is successful if a client uses the delivered data. How the client may send the feedback? What should we do if any delivery was unsuccessful? etc.

Now we are performing an extensive survey regarding predictive delivery of data in various domains such a www, mobile computing etc. We strongly believe that incorporation of prediction mechanism will surely increase the efficiency in context delivery.

3.5 Proactive Context Delivery

Motivation

Ubiquitous Computing envisions the availability of computing function “Any where, any time”. It is imagined that in a ubiquitous environment application will be smart enough to provide services to users without much users' interaction. Prediction is one of the key elements in achieving such smart environment. An application needs to be context sensitive to prove it to be smart enough in serving the users. As this need of service is real time, it is of utmost important to gather necessary contexts within a short amount of time to provide appropriate services to user. Context-aware middleware facilitate the development of context aware application by performing the computational extensive job of inferring context and delivery. In this respect, the context providing middleware needs also to predict the future need of contextual information for faster delivery of contexts to applications. Thus proactive delivery will pay an important role in ubiquitous computing environment.

Ubiquitous computing environment motivates us to predict future need of contextual information based on various parameters. One of the basic parameter is the current request pattern. Beside, the interrelationship of context and current situation also other important factor in such prediction. So our goal is to incorporate the interrelationship among contextual information which may be expressed as ontology and other available contexts to predict future need of context and deliver those contexts proactively to applications.

Related Works

To the best of our knowledge, there is no work related to predictive context delivery for ubiquitous computing. But there has been a lot of works in web domain to predict future requests for web documents. Most of the works are based on Markov model where it is assumed that the next request is strictly a function of current request. Many researchers [6]-[15] have used similar to first order Markov models for web request prediction, but this technique has also been used in many other domains (e.g. UNIX command prediction [16], hardware-base memory address prefetching [17]). Some other existing works report on better predictive accuracy using second order Markov models [18], [19] and some other with higher order models such as fourth-order [20]. Although Markov models of high order intuitively provide high accuracy, Li et al [21] argue in contrast that longest match is not always the best one.

PPM, or prediction by partial matching [22] is typically used for data compression. It works similarly to the simple Markov model-based approach and a few researchers have used this method in web prediction [23]-[26]. Curewitz et al [27] were the first to inspect the use of compression modeling techniques to trace events and pre-fetch items. They show that such techniques converge to an optimal online algorithm. They go on to test this work for memory access patterns in an object oriented database and a CAD system. Kroeger et al [28] adapts Prediction by Partial Match in a different manner. The problem domain they examine was the file systems access patterns.

Lau and Horvitz [29] have classified user queries and built a Bayesian model to predict users' next query goal or intention based on previous queries and time interval. WebWatcher system [30] makes recommendations on the future hyperlinks that the user might like to visit based on a model obtained through reinforcement learning. Other recommendation systems include the Letizia [31] system that anticipates a user's browsing actions by making forward explorations and the Syskill & Webert system [32] that learns to rate pages on the World Wide Web. Compared to these systems, [33] proposes an n-gram based prediction model to predict the next action based on statistical analysis of sequence information.

Yang et al [34] shows that latest-substring method coupled with the pessimistic-confidence based selection gives the best result in association rule based web request prediction. He also states that latest-substring rules can be considered as the union of Nth-order Markov models and hence it is more general than N-gram models or Nth order- Markov models. Zhang et al [35] also uses association rule based approach to predict web document requests.

Candidate Prediction Techniques

There are various mechanisms for predicting future events. Here we describe the techniques that we think to be potential candidates in case of proactive context delivery for ubiquitous computing.

N-gram and Markov Model

N-gram [4] is a sequence of n items. For example, the ordered pair (A, B) is an example 2-gram (or bigram) in which A appeared first, followed by B. For prediction, one has to try to match the complete prefix of length n-1 to an n-gram and predict the nth request based on the last item of the n-gram. Since there may be multiple n-grams with the same prefix of n-1 request, and n-grams do not natively provide the means to track their frequency, a mechanism is needed to determine which n-gram should be used for prediction. Markov models provide that means, by tracking the likelihood of each n-gram in a state space encoding the past. In this approach, we explicitly make the Markov assumption which says that the next request is a function strictly of the current state. Each state in a k-step Markov model corresponds to the sequence of k=n-1 requests that comprise the prefix of an n-gram. The nth element of the n-gram determines the destination of a link from this state to a future state, with a label corresponding to the last n-1 requests in the n-gram. Thus Markov model can encompass the information in multiple n-grams., but additionally tracks the likelihood of moving from one state to another.

Prediction by Partial Matching (PPM)

PPM [22] works similarly to the simple Markov model based approach, using the largest permissible context to encode the probabilities of the next item. However, it also track the probability of the next item to be something that has never been seen before in this context (called the "escape" symbol when used for compression), and thus explicitly says to use the next smaller matching context instead. There are multiple versions of PPM that correspond to different ways of calculating the escape probability: PPM-A, PPM-C, PPM-D, as well as others.

Association Rule

In brief an association rule [5] is an expression $X \Rightarrow Y$, where X and Y are sets of items. The meaning of such rules is quite intuitive. Given a database D of transaction, where each transaction $T \in D$ is a set of items, $X \Rightarrow Y$ expresses that whenever a transaction T contains X than T probably contains Y also. The idea of mining association rules originates from the analysis of market basket data where rules like "A customer who buys product X1 and X2 will also buy product Y with probability C%" are found. Their direct applicability to business decision problems such as promotions, discount etc. make association rules a popular mining method. However, we also believe that such rules also facilitate prediction of future request of context data

Proposed Scheme

We would like to extend our previous context delivery mechanism [1] through incorporation of prediction mechanism. Hence our proposed scheme is to delivery predicted context along with requested context though hybrid dissemination technique [1].

Architecture

Our architecture is similar to our previous architecture where clients sends request through the *Jini Interface*. *Context Delivery Manager* controls the data flow inside the Context Delivery module. Request Queue stores the request. Schedule Manager will prioritize the request using the Priority Calculator module. Priority Calculator Module calculates the

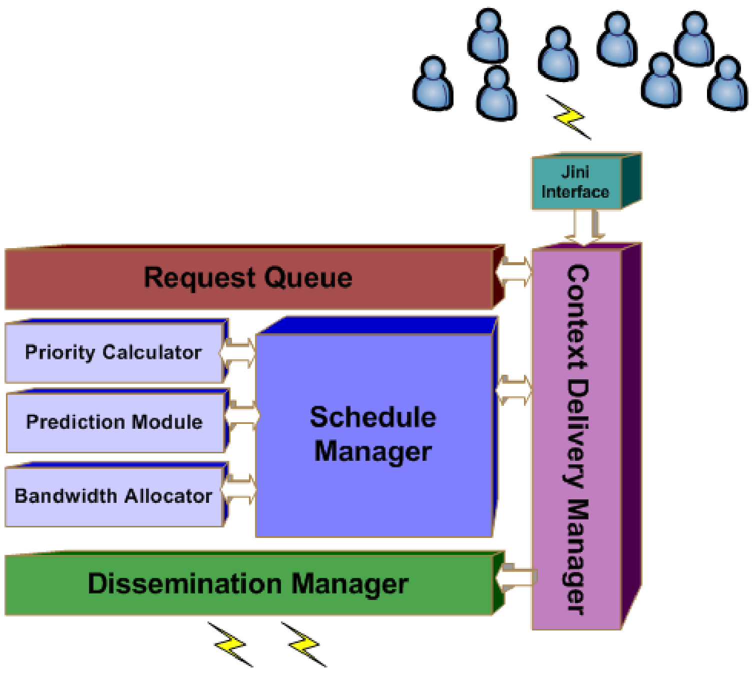


Fig 1 Architecture of Proactive Context Delivery Module

priority using RxW algorithm [1] [2]. The Prediction Module will predict future requests and add some new context in the delivery list. Then the list of context data will be allocated for multicast or unicast using Bandwidth Allocator module. Afterwards, Dissemination Manager will deliver the context using multicast channel or unicast.

System Work Flow

Requests are stored in Request queue. Prediction module gets a list of request and corresponding priority from the Priority Calculation Module. Prediction module predicts

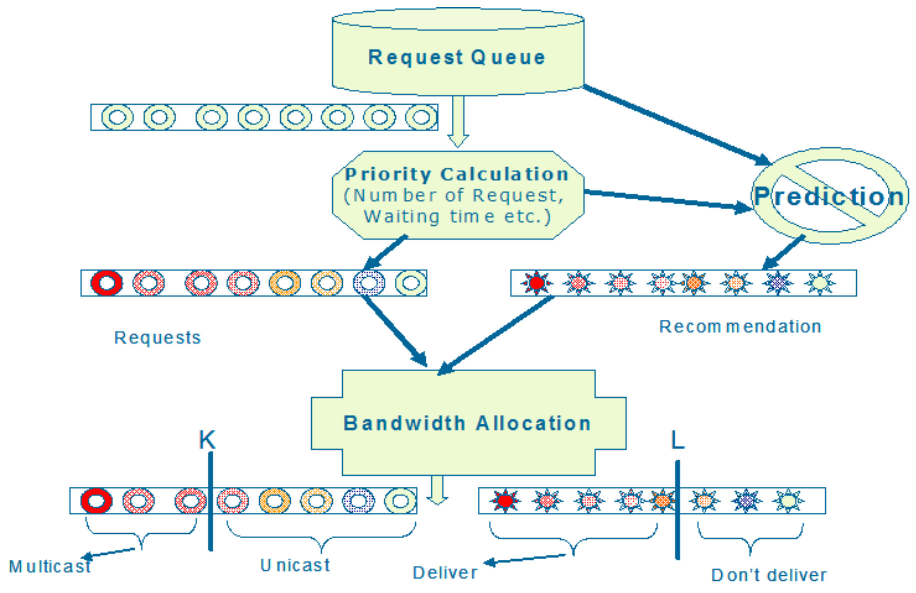


Fig 2 System Work Flow of Proactive Context Delivery

future request based on current request patter, interrelation among contexts and current situation. Then it also calculated the priorities and supplies a list of recommended context values. As the bandwidth is limited, then Bandwidth Allocation module determines which of the request are to be delivered using multicast and the rest through unicast. It also decides whether the entire recommended contexts are to be delivered or not (see Fig 2).

Summary and Future Work

Currently we are still surveying various prediction techniques appropriate for context deliver in ubiquitous environment. In future we wish to examine various prediction mechanisms and finally decide to use the best one depending on the performance in our experiment.

Reference

[1] Lenin Mehedy, Md. Kamrul Hasan, Young Koo Lee, Sungyoung Lee, Sang Man Han, “Hybrid Dissemination Based Scalable and Adaptive Context Delivery for Ubiquitous Computing”, The 2006 IFIP International Conference on Embedded And Ubiquitous Computing · (EUC' 2006), (LNCS). August 01-04, 2006, Seoul, Korea

[2] Aksoy, D., Franklin, M.: RxW: A scheduling approach for large-scale on-demand data broadcast. IEEE/ACM Transactions On Networking, 7(6) (Dec. 1999) 846-860

[3] Beaver, J., Morsillo, N., Pruhs, K., Chrysanthis, P. K.: Scalable Dissemination: What's Hot and What's Not. In the Seventh Int. Workshop on the Web and Databases (WebDB 2004), Paris, France, June 17-18 (2004)

[4] Brian D. Davison, “Learning Web Request Patterns”, In A. Poulovassilis and M. Levene (eds), Web Dynamics: Adapting to Change in Content Size, Topology and Use, PP. 435-460, Springer. 2004.

[5] Bart Goethals, “Survey on Frequent Pattern Mining”, Helsinki, 2003. http://www.cs.helsinki.fi/u/goethals/publications.html

[6] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve World Wide Web latency. Computer Communication Review, 26(3):22-36, July 1996. Proceedings of SIGCOMM '96.

[7] Azer Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time for distributed information systems. In Proceedings of the International Conference on Data Engineering (ICDE'96), New Orleans, LA, March 1996.

[8] Zhimei Jiang and Leonard Kleinrock. An adaptive network prefetch scheme. IEEE Journal on Selected Areas in Communications, 16(3):358-368, April 1998.

[9] Ann E. Nicholson, Ingrid Zukerman, and David W. Albrecht. A decision-theoretic approach for pre-sending information on the WWW. In Proceedings of the 5th Pacific Rim International Conference on Arti_cial Intelligence (PRI-CAI'98), pages 575-586, Singapore, 1998.

[10] Bin Lan, Stephane Bressan, and Beng Chin Ooi. Making Web servers pushier. In Proceedings of the Workshop on Web Usage Analysis and User Pro_ling, San Diego, CA, August 1999.

[11] Dan Duchamp. Prefetching hyperlinks. In Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS '99), Boulder, CO,October 1999.

[12] Edith Cohen, Balachander Krishnamurthy, and Jennifer Rexford. Efficient algorithms for predicting requests to Web servers. In Proceedings of IEEE INFOCOM, New York, March 1999.

[13] Dan Foygel and Dennis Strelow. Reducing Web latency with hierarchical cache-based prefetching. In Proceedings of the International Workshop on Scalable Web Services (in conjunction with ICPP'00), Toronto, August 2000.

[14] N. Swaminathan and S. V. Raghavan. Intelligent prefetching in WWW using client behavior characterization. In Proceedings of the Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2000.

[15] Michael Zhen Zhang and Qiang Yang. Model-based predictive prefetching. In Proceedings of the 2nd International Workshop on Management of Information on the Web-Web Data and Text Mining (MIW'01), Munich, Germany, September 2001.

[16] Brian D. Davison and Haym Hirsh. Predicting sequences of user actions. In Predicting the Future: AI Approaches to Time-Series Problems, pages 5{12, Madison, WI, July 1998. AAAI Press. Proceedings of AAAI-98/ICML-98 Workshop,published as Technical Report WS-98-07.

[17] Doug Joseph and Dirk Grunwald. Prefetching using Markov predictors. Transactions on Computers, 48(2), February 1999.

[18] Rituparna Sen and Mark H. Hansen. Predicting a Web user's next request based on log data. Journal of Computational and Graphical Statistics, 12(1), 2003.

[19] Ingrid Zukerman, David W. Albrecht, and Ann E. Nicholson. Predicting users' requests on the WWW. In Proceedings of the Seventh International Conference on User Modeling (UM-99), pages 275-284, Banff, Canada, June 1999.

[20] Peter L. Pirolli and James E. Pitkow. Distributions of surfers' paths through the World Wide Web: Empirical characterization. World Wide Web, 2:29-45, 1999.

[21] Ian Tianyi Li, Qiang Yang, and KeWang. Classification pruning for Web-request prediction. In Poster Proceedings of the 10th World Wide Web Conference (WWW10), Hong Kong, May 2001.

[22] Ian H. Witten, Alistair Mo_at, and Timothy C. Bell. Managing Gigabytes: Compressing and Indexing Documents and Images. Morgan Kaufmann, San Francisco, 1999. Second edition.

[23] Themistoklis Palpanas and Alberto Mendelzon. Web prefetching using par tial match prediction. In Proceedings of the Fourth International Web Caching Workshop (WCW99), San Diego, CA, March 1999. Work in progress.

[24] Li Fan, Quinn Jacobson, Pei Cao, and Wei Lin. Web prefetching between low bandwidth clients and proxies: Potential and performance. In Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '99), Atlanta, GA, May 1999.

[25] Xin Chen and Xiaodong Zhang. Coordinated data prefetching by utilizing reference information at both proxy and web servers. Performance Evaluation Review, 29(2), September 2001. Proceedings of the 2nd ACM Workshop on Performance and Architecture of Web Servers (PAWS-2001).

[26] Xin Chen and Xiaodong Zhang. A popularity-based prediction model for web prefetching. IEEE Computer, 36(3):63{70, March 2003.

[27] Curewitz K M, Krishnan. P and Vitter. J. S. 1993. Practical Prefetching via Data Compression. SIGMOD Record,22(2):257-266 . ACM, Jun. 1993

[28] Kroeger T M and Darrell D.E. 1996 Predicting Future file-System Actions From Prior Events. Proceedings of the USENIX 1996 Annual Technical Conference. Jan 1996

[29] Lau T., and Horvitz, E., (1999) Patterns of search: analyzing and modeling web query refinement. User Modeling '99, pp119-128.

[30] Joachims, T., Freitag, D., and Mitchell, T. (1997) WebWatch: A tour guild for the World Wide Web. IJCAI 97 – Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, 770-775.

[31] Lieberman, H., (1995). Letizia: An agent that assists web browsing. IJCAI95 –Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 924-929.

[32] Pazzini, M., Muramatsu, J., Billsus, D., (1996). Syskill and Webert: Identifying Interesting web Sites. Proceedings of the AAAI 1996., Portland, OR., pp54-62.

[33] Zhong Su, Qiang Yang, Ye Lu, and Hong-Jiang Zhang. WhatNext: A prediction system for Web request using n-gram sequence models. In First International Conference on Web Information Systems and Engineering Conference, pages 200-207, Hong Kong, June 2000.

[34] Qiang Yang, Tianyi Li, Ke Wang, "Building association rule based sequential classifiers for web document prediction", Journal of Data Mining and Knowledge Discovery, 8(3), 253-273, 2004

[35] W. Zhang, B. Xu, W. Song, H. Yang, Pre-Fetching Web Pages Through Data Mining Based Prediction, in Journal of Applied Systems Studies, Edition of Web Information Systems Applications, Cambridge International Science Publishing, Cambridge, England

Chapter 4

FAULT-TOLERANCE AND SELF-HEALING IN CAMUS

4 Fault-tolerance and self-healing in CAMUS

The CAMUS architecture (Fig. 4.1) provides support for gathering context information from sensors in a unified manner, incorporating different reasoning mechanisms for deducing high-level context, and delivering appropriate contexts to applications as well as notifying/activating the applications on context changes.

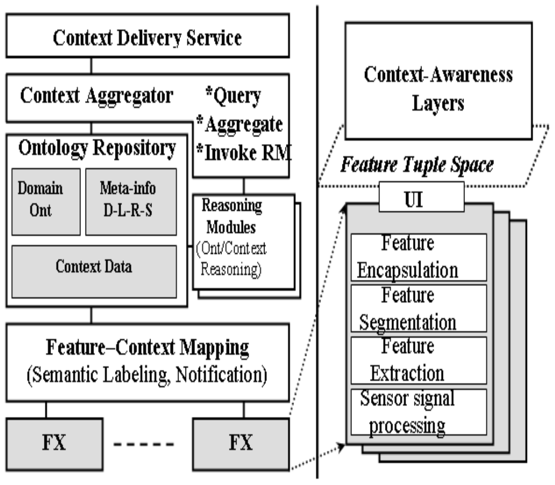


Fig 4.1 CAMUS Architecture

Here we just list the components of our middleware architecture:

Feature Extraction Agents (FXA)

Context Repository & Query module

Reasoning Module

Context Delivery & Aggregator Services

In addition of all the above features, CAMUS should also have the capability to anticipate and correct problems before they impact any aspect of its performance.

4.1 Autonomic Prediction Approach

Current work in autonomic computing and ubiquitous systems largely involves reactive fault handling and healing, based on feedback control i.e., the corrective steps and measures are undertaken on the basis of instantaneous need, or at most on the basis of short-term historical measurements. There are some obvious drawbacks of this approach e.g., it is static in nature, potentially large variations in performance may occur and the middleware may give slow response to change (lack of adaptability).

I propose to incorporate predictive autonomic measures and approaches for self-healing in CAMUS.

4.2 Prediction in CAMUS

Historically, there are three broad categories of prediction techniques that we been used in software systems in general. These are:-

- Statistical methods
- Parametric methods
- Machine Learning methods

There are many benefits of using prediction techniques in a middleware like CAMUS. Prediction can be helpful in a wide variety of responsibilities like predicting fault probability of a monitored resource, dynamically strengthening and adjusting fault-tolerance (Self-healing), improvement in performance, accounting and security management of CAMUS (Self-optimization), enabling components to predict their own behavior, load balancing of work/user/device loads (Self-configuration) and prediction of intrusion or denial-of-service attacks (Self-protection) etc.

However, constantly analyzing current performance and predicting future performance and the system's reaction to such events can be a highly dynamic and intensely computational operation. So we have to use such a prediction scheme that has the desirable characteristics of a large look-ahead time, high prediction accuracy, low overhead and robust operation. Each of these characteristics is inter-related and a suitable system-specific tradeoff will need to be determined for every component.

Fig. 4.2 illustrates a typical application scenario of CAMUS which is envisioned for the prediction of information-use of an individual in a ubiquitous computing environment like a smart office. The system should be able to handle:-

Prediction of user/device location

Optimize performance of an individual within a certain environment

Efficiently gather and sort information

Locate new information sources based on predicted services

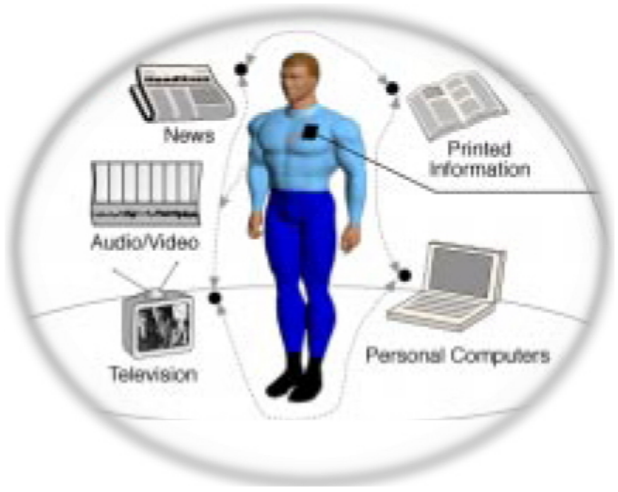


Fig 4.2 A typical smart office scenario

There are many existing prediction techniques that can be utilized and applied on a system if a sufficiently managed model of that particular system can be made. We considered the following briefly described four approaches:

4.2.1 Step-Wise Regression

This technique uses linear regression analysis to determine the relationship between two or more quantitative variables so that one variable (the Y or dependent variable) can be predicted from the others (the X or independent variables). The general form of such a prediction system is:

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n$$

It further involves finding the suitable independent variables and values for the β coefficients. It's called step-wise because each independent variable is selected for entry depending on their correlation with the dependent variable and various steps of entry and removal.

4.2.2 Rule Induction

This technique is also known as Concept Learning System (CLS). Its basic approach is to derive more general rules from specific cases in a training set. The resulting rules are organized into trees. The resulting trees are then used to classify future samples. The purpose of this technique is to discover patterns that delineate categories, assemble them into classifiers, and use them to make predictions.

There are many different algorithms for achieving this, e.g., ID3 and C5.0.

4.2.3 Case-Based Reasoning

This technique utilizes specific knowledge of previously solved cases to solve future ones. Case-based reasoning has four distinct aspects:

Characterization of cases

Storage of past cases

Retrieval of similar cases to use as analogies

Utilizing the retrieved case to solve the target case

4.2.4 Artificial Neural Nets

This is a very well-known technique, which incorporate a network of relatively simple processing elements, where the global behavior is determined by the connections between the processing elements and element parameters. For prediction purposes, generally multilayer perceptrons with a specific back propagation learning algorithms are used. Its one limitation is that the speed of convergence is slow.

4.3 Which approach for CAMUS?

After reviewing a lot of research material and considering the combination of various data set characteristics, performance issues and accuracy of predictions, I think that each member should decide which approach is best suited for his particular module, as each module may be suited for different type of prediction scheme. Some changes may have to be incorporated in CAMUS to allow for predictive autonomicity like introducing code mobility in some of the middleware components and modules and the ability to modify execution of components and parts of components within the CAMUS middleware.

This work involves identifying entities (both functional and behavioral) in the middleware whose prediction is required or is beneficial, then making their quantifiable models (by utilizing Resource and Event Modeling) for monitoring, prediction and execution and finally identifying and changing appropriate components and modules of CAMUS which should have support for code mobility or dynamic re-configuration.

4.4 Reference

[1] Ganek, A.G., Corbi, T.A.: The dawning of the autonomic computing era. IBM Systems Journal, v.42 n.1 (2003) 5-18

[2] HP Adaptive Enterprise strategy: http://www.hp.com/go/demandmore

[3] Microsoft Dynamic Systems Initiative: http://www.microsoft.com/windowsserversystem/dsi/default.mspx

[4] Bonino, D., Bosca, A., Corno, F.: An Agent Based Autonomic Semantic Platform. First International Conference on Autonomic Computing. New York (2004)

[5] R Srinivasan and H P Raghunandan, On the existence of truly autonomic computing systems and the link with quantum computing, http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:cs/0411094, 2004

[6] S.R. White, J.E. Hanson, I. Whalley, D.M. Chess and J.O. Kephart. An Architectural Approach to Autonomic Computing. Proc. of 1st International Conference on Autonomic Computing, 2004.

Chapter 5

KNOWLEDGE-BASED FAULT-DETECTION IN THE PERCEPTION MECHANISM

5 Fault-detection in the perception mechanism

5.1 Introduction

Ubiquitous systems have been designed to facilitate the interaction of humans with computers so that instead of being distinct objects in a user's environment, computers become a part of it by embedding the computations into the environment. Recent work includes making such ubiquitous devices and systems context-aware, enabling the devices or the systems react and adapt to changes which take place in their domain of concern [3]. Achieving context-awareness is not easy because the entire perception of the system is made up of disparate sensors and certain controllers responsible for controlling various environmental factors.

Requirements of context-aware ubiquitous systems include that the system maintain an intense interaction with the environment and make decisions according to the various environmental entities such as users, devices, physical quantities. These decisions are also based on the system itself including the performance of the various software modules and the hardware involved [3]. As a high degree of user ubiquity is needed in such context-aware systems, the system relies heavily on the sensors and controllers it uses to monitor and change the environment. Sensors and controllers are also physical devices which may malfunction under different circumstances. Any such malfunction in the perception mechanism of a context-aware system should not go unnoticed and undetected. These malfunctions need to be detected and reported so that the higher level context formation remains flawless, keeping the behavior of the system reliable.

Recent research has also tried to make such context-aware ubiquitous systems more autonomous [4]. A system needs to be self-healing, self-reconfigurable, be able to self-optimize and be self-protected [4]. All these concepts require that the system should be able to know the state of its software and hardware at all points. The state of the sensors and actuators which constitute the entire perception mechanism of the system is of vital importance. Knowing the state of the perception mechanism would help an autonomic system to determine the policies required for self-optimization, the current state of the perception mechanism, which quantities/events can be sensed, and in case of possible faults in the perception mechanism the system should be able to carry out isolation or self-healing policies. These requirements stress the need for a fault detection mechanism in context-aware and autonomic ubiquitous systems. This fault detection module should be able to correctly represent the system state at all time and at the same time it should be able to detect any anomaly in perceived data.

In context-aware ubiquitous systems the system possesses enough prior domain knowledge that it can anticipate the changes in the environment and reason about them [3]. This prior knowledge of the domain can be used for sensor fault detection and isolation. The need for fault-detection in the perception mechanism becomes very important for efficient system functionality. The motivation for using Bayesian networks comes from the fact that Bayesian networks not only model variables of a domain but also impose a causal ordering on them [1,2], and the beliefs of individual variables combine to form the overall belief in the entire modeled system. Scenario based fault detection constrains the sensors and actuators to behave in a certain pre-determined fashion. The beliefs of the system in light of the initially sensed data automatically give posterior beliefs about actuator settings and the resulting sensor readings. Thus Bayesian Networks help in determining the state of a certain sensor or actuator based upon acquired sensor data, this is done with the help of prior beliefs about the sensor state and the physical quantity or the event being monitored by the sensor.

5.2 Context-Aware Ubiquitous Systems

Context-aware ubiquitous systems have been designed to maintain continuous interaction with the user and his environment [3]. In doing so the system needs to know the current context in which it is supposed to function. The context is made up of various domain features gathered from sensor and actuator data [3]. These sensors and actuators are susceptible to faults, such faults need to be identified and the erroneous data discarded.

5.2.1 The Interaction Mechanism

In a context-aware ubiquitous system the entire perception mechanism of a system is composed of a number of diverse sensors deployed in the environment to monitor various physical quantities. A number of controllers or actuators are used by the system to respond to various changes which take place in the environment. The detection of such changes and the formation of context based on these changes is dependent on the data sensed from the monitored environment [3].

In any particular scenario the steps taken by the system can be defined as sensing some data from the environment and acting on its basis. The action taken in the light of the sensed data is determined through various factors such as available resources, the contextual contents, and user preferences. Every such decision step taken by the system also involves sensing data which is needed for validating if the action has indeed succeeded. The complete interaction cycle in a scenario is shown in Fig.6.1.

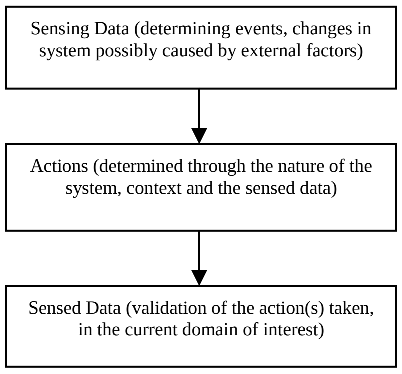

Sensing Data (determining events, changes in system possibly caused by external factors)
Actions (determined through the nature of the system, context and the sensed data)
Sensed Data (validation of the action(s) taken, in the current domain of interest)

Fig 5.1 The complete interaction cycle of a context-aware ubiquitous system.

Therefore sensors are needed both for determining the sequences of changes to be taken by the system and to sense if the desired results have been achieved. Actuators act as the effectors of the system which help it in controlling the state of affairs in the domain of concern.

For detecting anomalies in system-behavior such scenarios need to be identified so that the system behavior becomes predictable, and any deviation from the desired course of action results through some fault at any of the three levels.

5.2.2 Context Formation

In large context-aware ubiquitous systems, the formation of context plays the most important role in their functionality. Context formation is done, using some prior domain-specific knowledge and the sensed data [3]. Prior domain knowledge can be represented using any feasible knowledge representation technique such as ontology etc [3]. Context is formed by fusing together sensed data and this prior domain knowledge. As this formation of context is done solely on the basis of sensed data, if through any sequence of events the sources of such data get corrupted the context formed would be incorrect. As the contextual knowledge plays the central role in the interaction cycle of a ubiquitous system, incorrect contextual information would result in erroneous system behavior.

5.3 Fault Detection Using Bayesian Networks

This section outlines a scheme for scenario based fault detection in context-aware ubiquitous system using Bayesian networks. The section outlines the modeling of scenarios, sensors and actuators. At the end of the section the complete network representation and working are explained through an example.

5.3.1 Modeling a Scenario

The overall system behavior can be monitored for given scenarios which can occur in the domain of concern. This requires the modeling of sensors and the actuators involved in the interaction mechanism of the system. The main reason for using scenarios in modeling system behavior modeling comes from the fact that all sensors and actuators do not collaborate every time, so monitoring and reasoning about the entire set of sensors and actuators becomes computationally infeasible and absurd. Scenarios within the domain of concern pin the current focus of the system on only a subset of all the sensors and actuators, and at the same time they also define a relation between their behaviors. As explained previously sensors are used by the system at two levels in its interaction mechanism, we need to model them keeping this perspective in mind.

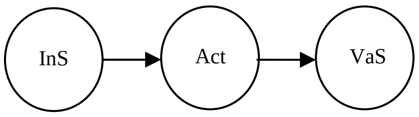


Fig 5.2 The general scheme for a Bayesian belief network representing a scenario. (InS: Input Sensors, Act: Actuators, VaS: Validating Sensors)

A general Bayesian belief network structure for modeling scenarios is shown in Fig. 6.2. As can be seen from the figure sensors have been modeled at two different levels within the Bayesian network, once for sensing the data, which triggers the response of the system through the actuators, and then again for sensing the desired changes. This structural imposition makes the system more deterministic. The behavior of all the sensors and actuators becomes inter-related given the scenario specifications. This inter-relation helps in detecting any sort of anomaly in the system and isolating any possible faulty component.

As shown in Fig. 5.2, a complete scenario is modeled as a serial connection [1,2], in which the actuator(s) form the connecting node. The model defines a conditional independence between the sensors needed for performing an action (InS) and the sensors for validating the action (VaS), by modeling the actuators as the connecting node. Once the action has been performed the behavior of the sensors in the validating phase is dependent only on the current actuator settings and the initial sensor data need not be considered in the validating phase as depicted by the network structure in Fig. 5.2

5.3.2 Modeling Sensors

For modeling a sensor it is necessary to know its current state. The state of a sensor represents its correctness and can be determined by taking into account certain factors such as the age of the sensor equipment, its reliability as provided by the vendor etc. In order to form a belief network for representing a sensor it is necessary that we take into account the quantity or the event being monitored for example temperature for a heat sensor. The main reason for including the physical quantity in the belief network is purely causal, because it is the physical quantity which causes the sensor to change its value. The belief network should also include the behavior of the sensor as represented by its actual reading, and in the end the state of the sensor should also be a part of the network. These three variables are sufficient to correctly model a sensor. Fig. 5.3 shows a Bayesian network depicting a sensor.

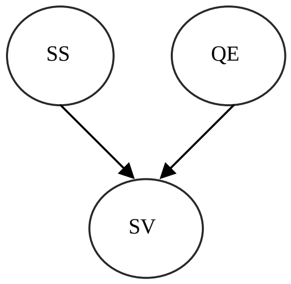


Fig 5.3 A Bayesian network representing a sensor (SS: Sensor State, QE: Physical Quantity/Event, SV: Actual Sensor Reading)

The model represents a converging connection [1,2] between the three variables with the actual sensor value being the connecting node. In a converging connection once evidence arrives at the connecting node the other two nodes become dependent. The model is very simple and facilitates the dependence only when actual sensor reading is considered. Evidence for the model comes in the form of sensor data and is absorbed at the connecting variable. This evidence renders the other two nodes dependent [1,2], such that based on their prior belief measures their posterior beliefs in light of recent evidence can be computed easily using any of the evidence propagation algorithms for Bayesian belief networks [1,2]. This means that at any instance a sensor-reading can be used for determining the state of the sensor.

$$P(SS \mid e) = \frac{\sum_{QE,SV,e} P(SS,QE,SV,e)}{P(e)} \quad ....(1)$$

5.3.3 Modeling Actuators

Actuators are used by the system for controlling various domain objects. As in the case of sensors the actuators should also have some reliability measures. In the current discussion we define actuator-state as being the variable which depicts the current belief in the correctness of the actuator. This parameter can be obtained by considering various factors such as the age of the component, the current environmental conditions, its failure rate as provided by the vendor etc. A Bayesian network designed for an actuator should contain its state, the physical quantity responsible for bringing about a change in the actuator settings, and the physical quantity or the domain object being controlled by the actuator. A Bayesian network depicting an actuator is shown in fig. 5.4.

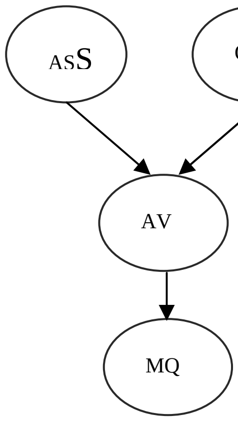


Fig 5.4 A Bayesian network representing an actuator (AS: Actuator State, QE: Physical Quantity/Event, AV: Actuator Reading, MQ: controlled quantity)

According to the model shown in figure 4, the controlled object is conditionally independent from the actuator state and the monitored physical quantity, given the actuator configuration. Similarly any evidence on the actuator setting makes the actuator state and the monitored physical quantity dependant on each other.

$$P(AS \mid e) = \frac{\sum_{QE,AV,MQ,e} P(AS, QE, AV, MQ, e)}{P(e)} \quad ....(2)$$

5.3.4 Construction of the complete Bayesian Network

These independent models for sensors and actuators can now be linked together according to the scenario model presented in figure-1. This requires that sensors at the initial and the validation level should be modeled according to the sensor model. These models are then linked to the actuator model completing the Bayesian Network required for representing a single scenario; which can be subsequently used for anomaly detection. A complete link is shown in figure-5.5.

Example

In this example we present a simple scenario and use the presented technique to come up with the results. The scenario is explained as follows:

"When there is any user in the room, the internal temperature of the room is adjusted according to the outside temperature. These adjustments or preferences have been pre-fed into the system by the user. The sensors used for this scenario consist of a movement sensor, temperature sensors for external temperature and internal temperature and a thermostat which is used to control the internal temperature of the room." Table-1 describes the variables used in the example and figure-5.5 shows the corresponding Bayesian network.

All the state variables have been given the probability distribution of (0.9,0.1) corresponding to the variable state description in table 1. Similarly the prior distribution for movement is given as (0.85,0.15) and for the external temperature it is given as (0.1,0.65,0.15,0.05,0.05) corresponding to the variable state description in table 1. Now if evidence is entered into the network in the form of sensor and actuator readings assuming that the internal temperature sensor is malfunctioning we want to see how the network behaves. Let the following set of evidences be entered onto the network:

e1: Ext_sen_read (0,0,0,1,0), e2: MD_read (0,1), e3: THERM_ACT (1,0,0), e4: In_sen_read (0,0,1,0).

TABLE I VARIABLE DESCRITPION

Variable Name	States	Description
State_MD	Correct, Incorrect	State of the movement sensor.
Movement	Yes, No	Prior probability of movement.
MD_read	Yes, No	Actual reading of the movement sensor.
State_Ext_Sensor	Correct, Incorrect	State of the external temperature sensor.
Ext_Temp	0-10,11-20,21-30,31-40,41-50	Prior probability about external temperature.
Ext_Sen_read	0-10,11-20,21-30,31-40,41-50	Actual sensor reading of the external sensor.
State_ACT	Correct, Incorrect	State of the thermostat (actuator).
THERM_ACT	11-15,16-20,21-25	Setting of the thermostat.
State_IN_Sensor	Correct, Incorrect	State of the internal temperature sensor.
Int_Temp	5-10,11-15,16-20,21-25	Actual internal temperature.
In_Sen_read	5-10,11-15,16-20,21-25	Actual reading of the internal temperature sensor.

The hypothesis variables namely the states of the sensors and the actuators are given as follows:

State_MD = (0.7129 , 0.2871).

State_Ext_Sensor = (0.9 , 0.1).

State_ACT = (0.9492 , 0.0508).

State_IN_Sensor = (0.1854 , 0.8146).

The above hypothesis variables clearly indicate that the internal temperature sensor has malfunctioned, as can be seen from the state variable description of the internal temperature sensor (State_IN_Sensor) which shows that it is incorrect with a belief of 81.46%. This example was simulated using the MSBNX™ tool.

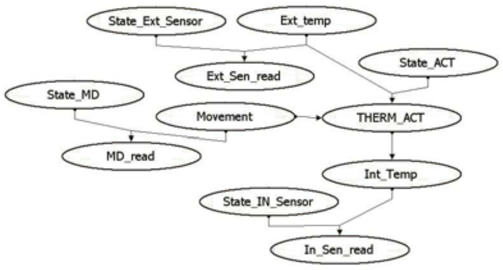


Fig 5.5 A Bayesian network for the example.

Thus, equipped with correct prior estimates and conditional probabilities the proposed scheme would be able to detect any anomaly in the system's perception mechanism.

5.4 Issues

In this technique we heavily use the knowledge already present in the system about the various scenarios it can encounter and cope up with. We would first like to implement this technique as an anomaly detection approach in which the system is able to identify any anomaly in the system and then come up with the source of this anomaly on the basis of the prior distribution attributed to each participating device or sensor. Afterwards the beliefs of the system can also be strengthened by the use of multiple scenario validation of an anomaly source.

As a ubiquitous system is supposed to interact with a number of different scenarios, it is possible that each device or sensor has a replicated role in multiple scenarios. Using this role-replication we can strengthen our beliefs about the malfunction of a certain device or sensor. For this purpose a simple Bayesian network cannot be used, as we would need multiple and structurally different networks. Hence, it is better that we use Multiply-Sectioned Bayesian Networks [5] for modeling multiple scenarios and then fusing their beliefs together.

5.5 Reference

[1] Jensen, Finn V., "Bayesian Networks and Decision Graphs", Springer-Verlag 2001, ISBN 0-387-95259-4.

[2] Pearl. Judea, "Probabilistic Reasoning in Intelligent Systems: Networks of plausible inference", Morgan Kaufmann publishers 1988, ISBN 1-55860-479-0.

[3] Hung Q. Ngo, Anjum Shehzad, Kim Anh Pham, Maria Riaz, Saad Liaquat, and S. Y. Lee, "Developing Context-Aware Ubiquitous Computing Systems with a Unified Middleware Framework".

[4] Roy Sterritt, Dave Bustard, "Autonomic Computing - a means of achieving dependability?", Proceedings of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03).

[5] Yang Xiang, "Probabilistic Reasoning in Multiagent Systems: A Graphical Models Approach", Cambridge University Press, 2002, ISBN:0521813085.

SOME AUTONOMIC AGENTS FOR COLLABORATIVE CONTEXT RECOGNITION IN **OPEN HOUSE** SCENARIO

6 Appendix

Smart wall: touch sensitive, embedded with ambient display, and interactive, which can change its colors for some purposes, display the messages, and response to the user interactions etc. It can also send private messages to each user through his mobile device.

Smart chair: embedded with pen-based computer, the user can take notes during a meeting/conversation, and then have those notes transferred to his PDA/PC for further processing. And with pressure, ID based sensors; the chair can recognize who is sitting on it, sitting pattern, etc.

Smart table: integrated with a touch-sensitive plasma display for interaction, pressure sensors for recognizing object placement patterns.

Smart floor: instrumented with force measuring load cells, the Smart Floor aims to identify and track a user around an instrumented space.

Smart door: combined and adapted all the cameras, buzzers, keypads, motion detectors, RFID readers and intercoms into a wireless integrated door station and combined it with the door handle which can integrate existing and planned infrastructure at low cost and high convenience, e.g. authorized persons no longer need a key.

Smart refrigerator: smart refrigerator is capable of tracking all of your groceries stored in it. It can track the foods you use, how often you restock your refrigerator and can let you know when that milk and other foods spoil, add milk to your grocery list, or you could program it to order these items automatically.

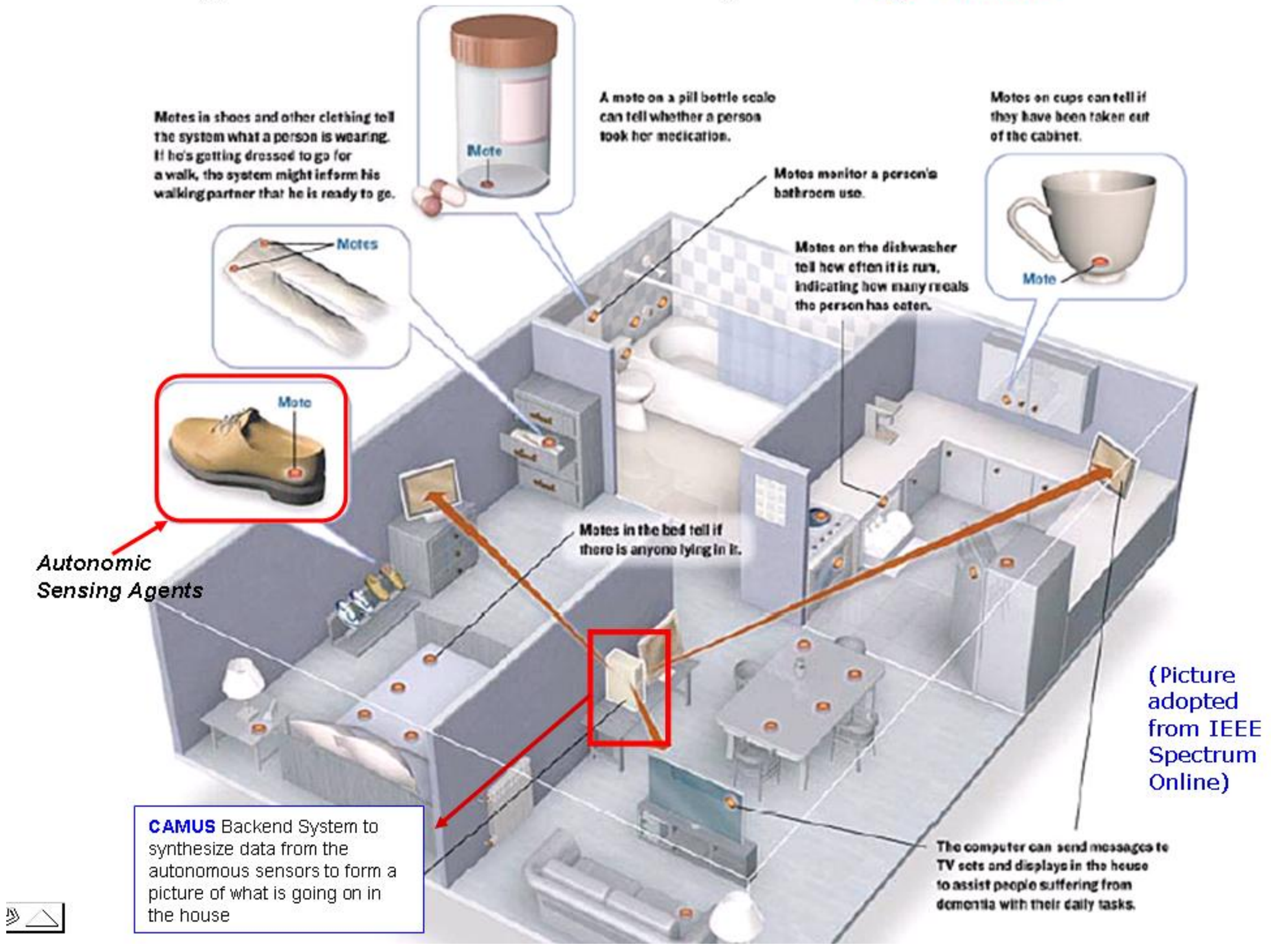
Smart bed (sometimes prof. needs to stay in office ;-): an instrumented sheet with sensors to detect human body position shifts, temperature variation and certain physiological states, such as breathing rate and heart rate, without the need to attach wired sensors to the person's body.

Smart container: sensors deployed inside containers can monitor environmental conditions for perishable goods; detect tampering or leakage of dangerous goods, or provide an RFID-based real-time inventory for tracking containers during their journey, or determining the position of a specific container.

Other stuff to think of:

Smart cup, smart picture frame (N/A), smart bookshelf, smart projector

Autonomic Agents for Collaborative Context Recognition in Open House Scenario


Motes in shoes and other clothing tell the system what a person is wearing. If he's getting dressed to go for a walk, the system might inform his walking partner that he is ready to go.
Mote
A mote on a pill bottle scale can tell whether a person took her medication.
Motes monitor a person's bathroom use.
Motes on cups can tell if they have been taken out of the cabinet.
Motes
Motes on the dishwasher tell how often it is run, indicating how many meals the person has eaten.
Mote
Mote
Autonomic Sensing Agents
Motes in the bed tell if there is anyone lying in it.
(Picture adopted from IEEE Spectrum Online)
CAMUS Backend System to synthesize data from the autonomous sensors to form a picture of what is going on in the house
The computer can send messages to TV sets and displays in the house to assist people suffering from dementia with their daily tasks.