IST IP CASCADAS

“Bringing Autonomic Services to Life "

D6.3—Proof of concept design of the application test-bed

Status and Version:	Final Version	
Date of issue:	17.05.2008	
Distribution:	Public	
Author(s):	Name	Partner
	Ricardo Lent, Antonio Di Ferdinando	ICL
	Franco Zambonelli	UNIMORE
	Benko Borbala Katalin	BUTE
	Roberto Cascella, Mauro Brunato	UNITN
	Antonio Manzalini	Telecom Italia
Checked by:	Antonio Manzalini	Telecom Italia

Abstract

The CASCADAS Toolkit will facilitate the development of autonomic, situation-aware and self-similar communication and content services with the use of autonomic components (ACEs). The toolkit will be released under an open source license to facilitate the dissemination of technological achievements to the scientific community and industry. The toolkit is platform independent and uses Java as programming language.

This deliverable describes the demonstrator, which consists of an application scenario and test-bed facilities. The demonstrator models a modern exhibition centre that is equipped with screens, which can be used to advertise products and services to people attending an event. People carrying smart devices can freely transit within the centre, which includes a large pervasive infrastructure of embedded devices. The scenario offers two distinctive features. It allows advertisers to tailor announcements to the current audience by either displaying suitable information to the people near one screen or by directing messages to selected smart devices. This application is a typical example of pervasive computing with scalability constraints. On the other hand, it allows advertisers to compete in an auction for the screen timeslots available to show the information of their interest. The latter case represents an application than can typically operate on a wide area network with hard real-time constraints. Although existing technology allows the construction of this scenario, large scale deployments would require more advanced and innovative solutions, such as the ones provided by the CASCADAS Toolkit. We expect that the proposed architecture will serve as an outstanding showcase for CASCADAS results.

Table of Contents

1 Introduction

1.1 Purpose and Scope

This document describes application scenario specifications and test-bed that will serve as the main demonstrator for key results of CASCADAS. The scenario incorporates challenging applications of pervasive computing and wide-area communications with scalability and real-time constraints that are crucial and difficult to solve with current technologies. The document explains the demonstrator software architecture that will be developed with the use of the CASCADAS autonomic toolkit.

1.2 Reference Material

1.2.1 Reference Documents

[1] D6.1—Part A: "Description of application scenarios and of the services to be provided"

[2] D6.2—Part A: "Experimental QoS Evaluation in Autonomic Network Environments"

[3] D6.2—Part B: "Distributed test bed specifications"

[4] D1.2—"Prototype Implementation"

1.2.2 Web pages

[DIET]	http://diet-agents.sourceforge.net/
[REDS]	http://zeus.elet.polimi.it/reds/
[JDREW]	http://www.jdrew.org/
[RULEML]	http://www.ruleml.org/

1.3 Document History

Version	Date	Authors	Comment
0.1	17/01/2007	Ricardo Lent	Initial document
0.2	20/04/2007	Antonio Di Ferdinando	Preliminary comments on the Auction scenario
0.3	27/05/2007	Antonio Di Ferdinando	Updated description of the demonstrator
0.4	03/06/2007	Franco Zambonelli	Contribution to the description of the demonstrator
0.5	19/06/2007	Antonio Di Ferdinando	Detailed description of the demonstrator

0.6	22/06/2007	Benko Borbala Katalin	Contribution to the Autonomic Toolkit section
0.7	26/06/2007	Roberto Cascella, Mauro Brunato	Contribution to the test-bed infrastructure
0.8	01/07/2007	Antonio Manzalini	Contribution on ACE toolkit and abstract Final Revision
0.9	14/05/2008	Ricardo Lent, Mauro Brunato	Revision

1.4 Document overview

The document is structured as follows. Section 2 contains a short description of the test-bed infrastructure that will host an implementation of the application scenario and a discussion of test-bed interconnection. Section 3 describes the autonomic toolkit that will serve to develop the application scenario. Finally, section 4 offers a detailed description of the application scenario and expected outcomes.

2 Test-bed Infrastructure

2.1 Available hardware and software

The following is a list of the equipment available to host the demonstrator. A detailed discussion of their specific use in the demonstrator will be covered in Section 4.

2.1.1 FOKUS

The Fokus test-bed consists of a single management server and 9 small devices, several of them mobile. Most of the test-bed is also portable.

The equipment in detail:

- 5 x Portable Mini-ITX PC's VIA EPIA-MII12000 with a x86 compatible processor with 1.2GHz frequency, 1GB RAM, 40GB HD, 100Mbit LAN, PCMCIA Wi-Fi
- 2 x Ultra-Mobile PC's OQO Model 1+ with 1GHz Transmeta Crusoe processor, 512MB RAM, 30GB HD, Wi-Fi, Bluetooth
- 1 x MacBook Pro with 1.86 Intel Core Duo processor, 2GB RAM, 120GB HD, Wi-Fi, Bluetooth
- 1 x Nokia 770 Internet Tablet with 250MHz TI OMAP 1710 processor, 64MB RAM, Wi-FI, Bluetooth
- 1 x Dell Rack Server with dual 64Bit Intel Xeon 2.8GHz, 2GB RAM, 2 x 250 GB HD, 2 x Network cards

2.1.2 UNITN

The UNITN test-bed consists of access points deployed in the Faculty and serve both students and professors. The authentication system has been designed and implemented by the UNITN team, and supports user localisation and other profile information, allowing for multiple authentication sources. Servers and desktops are also available. The equipment in detail:

- 6 Intel-based Servers
- 3 Desktop PCs (Dell)
- 20 Access Points (Cisco, 3Com, D-Link, LinkSys)
- 2 Routers (LinkSys)
- 3 LAN Switches (Allied Telesyn, D-Link)
- 5 Palmtops (iPAQ)
- One Bluetooth GPS receiver
- Two Java-enabled smartphones

2.1.3 ICL

The ICL test-bed consists of a number of mobile and stationary computers. The computers run a version of the Linux operating system modified by the ICL team to implement self-aware networks (CPN routing).

- 82 Rack PC Intel P4 2.4 GHz (single core, hyper-threading) 512 MB RAM, 5 - 12 Ethernet interfaces Linux (Debian)
- 3 Desktop PC Intel P4 2.4 GHz (single core, hyper-threading) 512 MB RAM, 4 Ethernet interfaces, WiFi 802.11g Linux (Debian)
- 1 Wireless access point
- 7 PDA H5500 Linux Familiar 0.8.2
- 4 Fast Ethernet LAN switch (24 ports)
- 3 Cabinets, patch panel

2.1.4 UNIMORE

UNIMORE has the availability of:

- One ALIEN mid-range RFID reader, about one hundred of RFID tags, and the associated software driver for controlling the reader. These will be used for setting up the practical demonstration of the pervasive advertisement, by using RFID tags to store and collect user profiles.
- 16 Crossbow MICAs wireless sensors, and the associated gateways. These can be used for collecting additional information about the context, to be used for adding more user-profiling features to the pervasive advertisement demonstrator, as well as to demonstrate additional capabilities of exploiting contextual information.
- Compaq IPAQ PDAs

- Bluetooth GPS receivers
- various WiFi access points

2.2 Establishing a Virtual Private Network

The introduction of a Virtual Private Network (VPN) in a multi-institutional test-bed is very important from several points of view: a VPN provides a uniform environment for all test-bed equipment regardless of its location, it helps insulating the test-bed from the ordinary network activities in the participating institutions, debugging is simplified by concentrating traffic and secure inter-institutional channels can be provided.

Uniform environment – A uniform environment is provided throughout the test-bed, so that applications that reside in different institutions can communicate without the need of being aware of specific restrictions enforced by each network administration. Such restrictions depend on the specific vulnerability issues considered by network administrators, and no uniform view is possible. Since VPN traffic between two institutions is encapsulated, the burden of getting administrative authorization for setting it up is reduced to a single request.

Test-bed insulation – In the hypothesis that not all institutions could be able to set up a separate physical LAN in order to insulate the test-bed from external traffic, setting a VPN server as the gateway for the local machines helps reducing the risk of interference with the rest of the LAN.

Network-related debugging – Since all test-bed traffic is encapsulated into specific inter-institutional channels, VPN servers concentrate all the traffic between institutions and are therefore good points where packet sniffers or throughput analysers can be located.

Channel security – VPN tunnels can easily be encrypted, and a Certification Authority can be set up for endpoint authentication purposes.

All these aspects are taken into consideration in the following sections of this document.

2.2.1 Design choices

2.2.1.1 VPN software

The CASCADAS project makes heavily use of the open-source model. The OpenVPN software[1], already introduced in D6.2 (part B), provides functionality to the test-bed with regards to the features listed above:

- It is an open-source project, so that customization is possible without license infringement.
- It provides several tunnelling models, at levels 2 and 3 of the ISO/OSI stack.
- It has native SSL/TLS support.
- Clients and servers are available for many different OSs.
- In the case of Linux, it integrates with the *netfilter* kernel mechanism, so that all necessary routing, firewalling and NATting functionalities can be implemented on top of the VPN system.

[1] `http://www.openvpn.org/`

2.2.1.2 Suggested settings

In the CASCADAS test-bed a Layer 3 VPN solution is sufficient for full operability: machines do not need to be within a single logic LAN, while the presence of Layer 3 devices enables the experimentation of more general network configurations. Layer 3 operation is easier to configure and maintain: different network configurations in the participating institutions can be masked so that a uniform view is provided to other network segments; a Layer 2 VPN requires higher uniformity in the participating networks.

The layer 3 operation implies that the VPN endpoints must be configured for routing, and machines participating in the test-bed need a small amount of custom configuration, limited to the routing table.

2.2.1.3 Main network settings

In the following we shall assume that three institutions (A, B and C) must be connected via a VPN. A set of private IP subnets that are not used in any of the hosting institutions should be detected, for instance three class C subnets in the 10.0.0.0/8 interval. Moreover, each institution shall expose a public IP address within its publicly assigned subnet as an endpoint of the VPN tunnel. For instance, suppose that the following IP addresses are determined:

Institution	**Private subnet For the test-bed**	**Private VPN endpoint**	**Public IP address for the VPN server**
A	`10.64.65.0/24`	`10.64.68.1`	`195.196.197.198`
B	`10.64.66.0/24`	`10.64.68.2`	`199.200.201.202`
C	`10.64.67.0/24`	`10.64.68.3`	`203.204.205.206`

A fourth private network, namely 10.64.68.0/24, is kept free for VPN operation, and shall be used as the internal addresses of the VPN interfaces. All VPN endpoints have an internal (virtual) interface in 10.64.68.0/24.

The basic configuration arising from the chosen settings is shown in Figure 1. The VPN equipment is represented by the three boxes at the cloud borders provided with a public and a private IP address. The top box is chosen as the VPN server endpoint; the two other boxes operate as clients and connect to it. A more symmetric system can be achieved by running both client and server software on one of the two lower boxes, so that a complete connectivity between institutions is implemented.

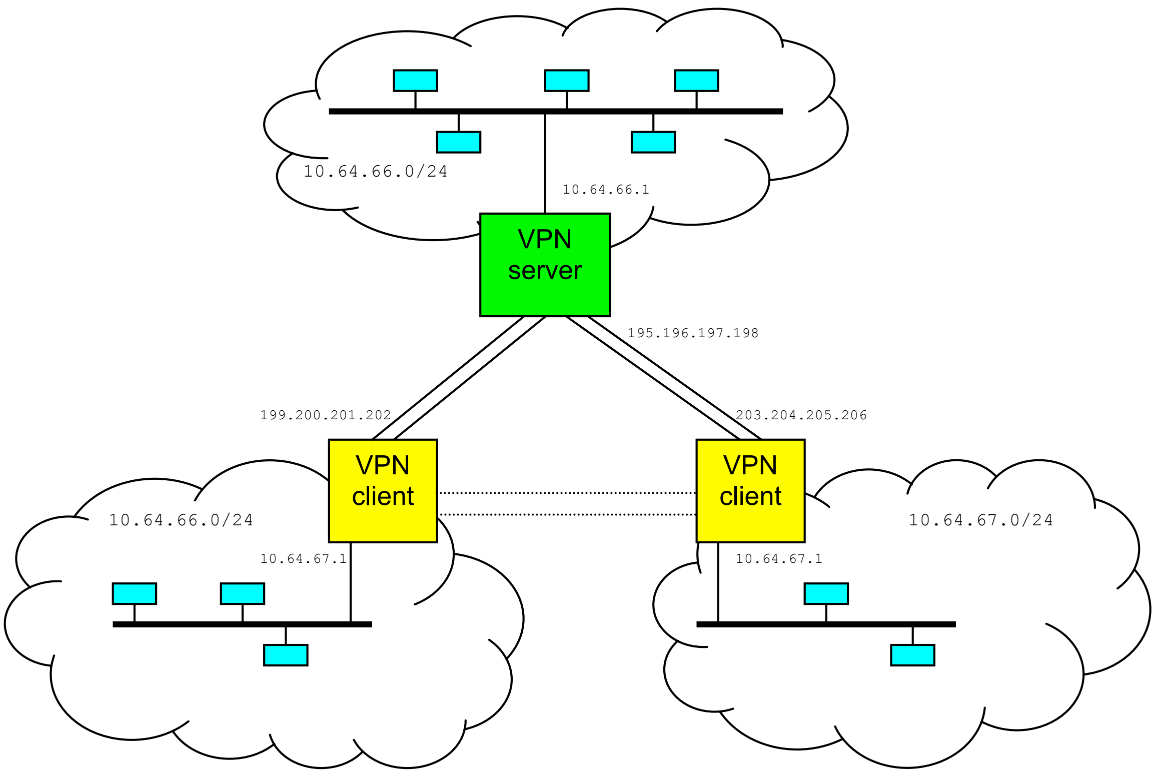

Figure 1: Basic VPN configuration; all test-bed clients (small boxes) use their VPN endpoints as gateways.

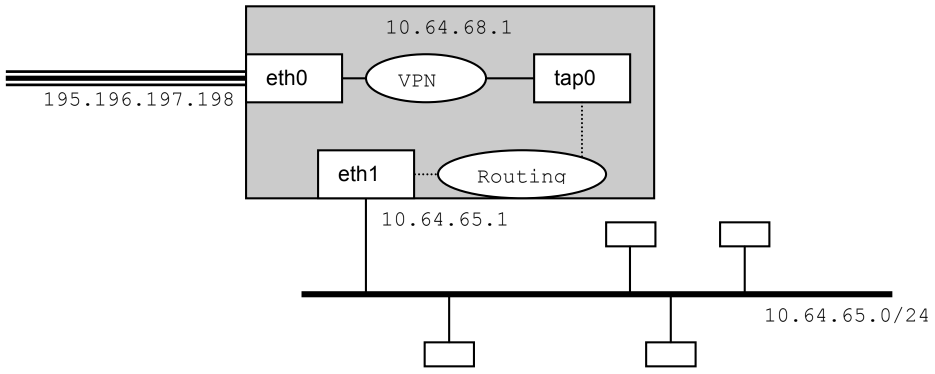

Figure 2: VPN server configuration

2.2.2 Configuration

The three main items that need to be configured are the VPN server(s), the VPN clients, and the test-bed stations (machines that participate in the test-bed without being aware of the VPN).

2.2.2.1 VPN server configuration

The VPN server is the most critical node to be configured. Its main functions include:

- Provide a tunnel endpoint, with capability of accepting multiple client connections.
- Provide the necessary routing and firewalling services.

As a basic example, shown in Figure 1, let us suppose that the machine running the server is provided with two separate network interfaces, one (`eth0`) having the public IP address `195.196.197.198`, the second (`eth1`) having private address `10.64.65.1` connected to a LAN dedicated to the test-bed (see Figure 2) corresponding to the subnet `10.64.65.0/24`. The VPN server listens to the public interface on port 1194, using the UDP protocol. The `tun0` VPN endpoint is the standard virtual interface for Layer 3 operation. The certificate of the CA, the certificate of the server and the private key of the server are needed for authentication and cryptography purposes. The VPN virtual interfaces will lie in the 10.64.68.0/24 subnet (the server will keep the 10.64.68.1 address for itself). The resulting configuration file contains the following lines:

```
local 195.196.197.198
port 1194
proto udp
dev tun
ca ca.crt
cert server.crt
key server.key
dh dh1024.pem
server 10.64.68.0 255.255.255.0
ifconfig-pool-persist ipp.txt
client-to-client
keepalive 10 120
comp-lzo
persist-key
persist-tun
```

After the openVPN server has been launched with these parameters, the machine's operating system must perform some routing operation as sketched in this table:

Destination	Gateway	Interface
10.64.68.0/24	-	tun0
10.64.66.0/24	10.64.68.2	tun0
10.64.67.0/24	10.64.68.3	tun0
10.64.65.0/24	-	eth0
Other non-VPN settings		

On Linux machines, routing is finally enabled by writing "1" on the file

```
/proc/sys/net/ipv4/ip_forward
```

2.2.2.2 VPN client configuration

The VPN client needs a similar configuration as the server; only IP addresses change.

```
client
dev tun
proto udp
remote 195.196.197.198 1194
resolv-retry infinite
nobind
ca ca.crt
cert thisclient.crt
key thisclient.key
comp-lzo
persist-key
persist-tun
```

The routing table configuration for the client is much similar to the server's one; in fact, the only distinction between server and client in this context only concerns protocol initiation. Here is the routing table for institution B:

Destination	Gateway	Interface
10.64.68.0/24	-	tun0
10.64.65.0/24	10.64.68.1	tun0
10.64.67.0/24	10.64.68.3	tun0
10.64.66.0/24	-	eth0
Other non-VPN settings		

IP forwarding must be enabled in the same way as the VPN server.

2.2.2.3 Test-bed station configuration

Machines participating in the test-bed are represented in Figure 1 as small blue boxes. The only configuration needed is the routing table; in institution A the routing table should read:

Destination	Gateway	Interface
10.64.65.0/24	-	eth0
10.64.66.0/24	10.64.65.1	eth0
10.64.67.0/24	10.64.65.1	eth0
Other routing information for Institution A		

Only relevant lines are shown; in general, nothing prevents test-bed machines to access the institution's network, so that only test-bed-related addresses need to be routed by using the VPN endpoint as a gateway.

2.2.3 Security

The institution that owns the VPN server will also operate the Certification Authority (CA) used to sign all certificate requests by the other institutions. The creation of a CA secret key, its storage and maintenance and the protocol for providing client certificates, either by generating them or by signing requests issued by such clients, shall be responsibility of the VPN server maintainer institution.

We believe that within a test-bed setting the need for strong security protocols is rather mild, so that ordinary e-mail should be an adequate channel for certificate exchange, although private keys should be kept in the VPN endpoints themselves.

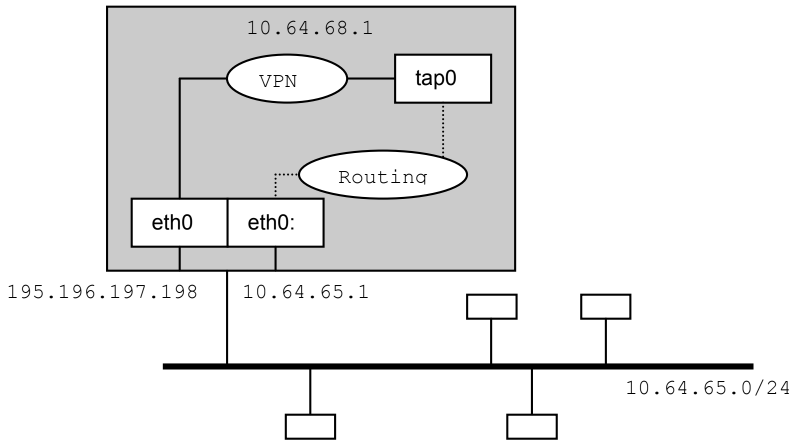

Figure 3: VPN server configuration with one interface

2.2.4 Particular networking requirements

In the hypothesis that no separate LAN is allowed for test-bed machines, the VPN can work without modifications provided that routing tables are programmed as described before: distinction between test-bed and non-test-bed machines and applications is done on the basis of IP addresses.

If a separate subnet of IP addresses cannot be obtained and only sparse IP addresses are available for test-bed use in an institution, the machine operating the openVPN software can perform Network Address and Port Translation (NAPT) by appropriately configuring the netfilter module of the Linux kernel (or the corresponding component in other operating systems).

Finally, the requirement of two separate interfaces for VPN endpoints can be relaxed as well: a single interface with two IP addresses (or two virtual interfaces sharing the physical one) are sufficient to make the system work correctly. Figure 3 shows the configuration of the VPN server that has only one interface.

2.3 Traffic monitoring tools

The open source (and multiplatform) packet-sniffing code `tcpdump` can be used for all traffic monitoring purposes. In particular, a running `tcpdump` daemon can be hooked to the virtual network interface `tun` in the VPN nodes, so that all inter-institution traffic (limited to the VPN, i.e., test-bed, part) can be recorded in the machines for later inspection and analysis.

For traffic within the institution, different policies can be adopted:

- A (dedicated) machine connected to a switch port configured as a mirror for the relevant machines.
- Copies of `tcpdump` on all machines logging test-bed-related traffic
- Utilities that inspect the proc tree (in the case of Linux systems): this would not provide information at packet detail, but overall traffic figures.

3 CASCADAS Autonomic Toolkit

The CASCADAS Toolkit shall enable autonomic, situation-aware and self-similar development of communication services based on ACEs as core components. Moreover the toolkit will be released under an open source license to facilitate the spreading of technological achievements both to the scientific community and industry; for this reason it is platform independent and adopts Java as programming language.

The other important aspect of the first prototype is the adoption of the DIET Agents platform [DIET] as the underlying unified execution environment. DIET Agents is a platform for developing agent-based applications. It was created as part of the EU-funded DIET (Decentralised Information Ecosystem Technologies) project. The reasons behind this choice are twofold: on one side the synergy between EU projects is an important result of the effectiveness of the project *per se* and on the other hand DIET tries to overcome some limitations of other agent platforms. In addition the support available by BT, the lead partner in DIET, is the other key towards the choice of DIET.

The DIET project [DIET] is concerned with the development of an ecosystem-inspired approach to the design of agent systems. In this context an ecosystem can be viewed as an entity composed of one or more communities of living organisms, in which organisms conduct frequent, flexible local interactions with each other and with the environment that they inhabit. Although the capability of each organism itself may be very simple, the collective behaviours and the overall functionality arising from their interactions exceed the capacities of any individual organism. These higher level processes can be adaptive, scalable and robust to changes in their environments. These elements are perfectly in line with the work carried in CASCADAS as the ACE interaction is seen as an emergent behaviour where the group ability is greater than the sum of each individual acting alone.

3.1 Main characteristics of the Toolkit

ACEs are defined to be autonomic, self-similar, light weighted, situation-aware and semantically self-organising communication elements. The model developed for ACEs tries to satisfy all these requirements.

To realise self-similarity, while being light-weighted, an ACE is separated into a Common Part, equally implemented for all ACEs, and a Specific Part that can be equipped with specific functionality differing from ACE to ACE. This section shortly describes how the ACE model is implemented within the CASCADAS Toolkit. A more detailed description can be found in [4].

In order to be platform independent, the ACE model is implemented using the Java programming language. The DIET Agent Platform [DIET] is utilised as the underlying unified execution environment. It provides very basic features to organise the ACE components to a complete ACE as well as communication and control principles which help to manage large systems of ACEs.

In structural detail, the ACE model consists of a set of Organs each responsible for delivering a subset of the functionality of the whole component. Figure 4 gives an overview of an ACE and its Organs.

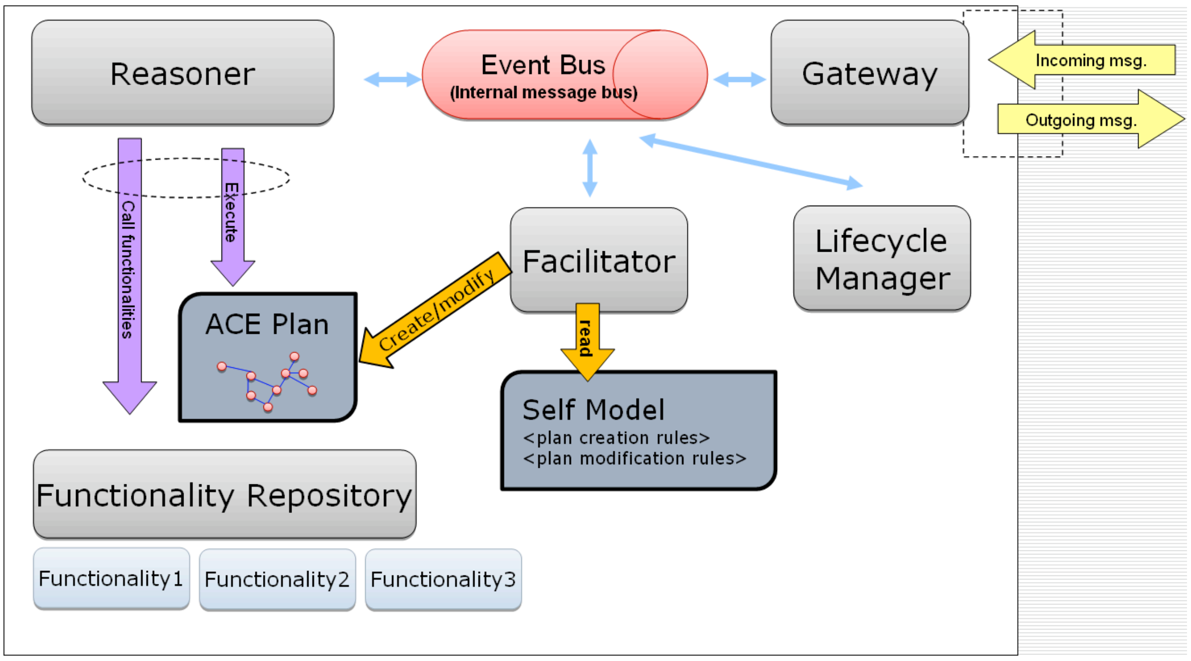

Figure 4: Organs of an ACE

Within this model, the Specific Part is the functionality which is deployed into the Functionality Repository, where an ACE developer can embed all individual services the ACE needs to fulfil its goals. The Organs form the Common Part which cannot be changed by ACE developers. It is uniform for each and every ACE and therefore builds the basis for self-similarity. Following, a list of all ACE Organs is presented explaining their functionality and the role they play within an ACE.

3.1.1 Gateway

Communication in the ACE model is based on events. The Gateway is the Organ which is responsible for managing external communication, i.e. between different ACEs. From the view of an ACE, the Gateway is the interface to everything which is not inside the ACE. External communication can occur in two different ways. First, it is possible to exchange events between ACEs in an uncoupled, session-less way. The REDS middleware [REDS] is used to implement this. Applying this communication mechanism, an ACE can subscribe to a set of events which it is interested in. Each time an according event occurs, REDS publishes it to all subscribed ACEs. REDS realises this by using clients, one for each ACE, that can send and handle events, and brokers which are responsible for managing the subscriptions and the routing of events.

The second external communication mechanism implemented in the CASCADAS Toolkit is point-to-point, session-based communication using contracts between two ACEs. Establishing a contract with another ACE requires knowing its address. A EstablishContractEvent can then be sent in order to request a contract with the desired party. Thereafter a "Mirror Agent" is created using the functionality provided by the DIET Agent Platform. This Mirror Agent then establishes a direct communication channel between the two contract parties until one of them quits the contract by sending a CancelContractEvent. For the second release of the CASCADAS Toolkit an advanced contract negotiation mechanism is planned.

3.1.2 Bus

The Event Bus (i.e., Bus) is responsible for internal communication between the subcomponents of an ACE. Like the Gateway, it is based on events. Organs can subscribe to a set of events they are interested in. Whenever an event is sent, the Bus will dispatch it by calling appropriate handle methods that have to be implemented by each subscribed Organ.

The Bus supports both synchronous and asynchronous event handling. For asynchronous communication, events need to be sent and will then be further dispatched by the Bus. Communication in a synchronous way means that after an Organ has sent an event, it is blocked by the Bus until all subscribed components finished to handle the concerned event.

3.1.3 Facilitator and Self Model

The ACE autonomic behaviour is specified by the ACE developer within the Self Model and provided by the Facilitator. The Facilitator is the ACE Organ responsible for ACE Plan creation and modification. It processes the Self Model and creates a new or modifies an existing ACE Plan accordingly. Changing the ACE plan means changing the behaviour of an ACE.

The Self Model is defined through an XML file which has to be provided by the ACE developer. It contains all different Plans an ACE can process, as well as rules for their creation and modification. An ACE Plan is a state machine where states are connected to each other via transitions. Transitions contain actions and conditions for their execution. The Self Model basically contains the specifications for states and transitions as well as rules for building and modifying the state machine (the ACE Plan). Plan creation and modification rules are formulated in standard RuleML [RULEML] syntax.

Facilitator actions are triggered via events. The Facilitator can be used to create a new ACE Plan via CreatePlanEvent. Plan modification is a continuous process, which depends on context information. In the current implementation the Facilitator subscribes to the Reasoner for receiving certain context data. Each time it gets notified about context data changes, it executes the Plan modification rules and modifies the ACE Plan accordingly. For introspection purposes, the Facilitator provides access to the Self Model of the ACE.

3.1.4 Reasoner and Plan

ACE Plans are generated by the Facilitator and executed by the Reasoner. They are state machines containing states and transitions. Each Plan has an initial state and a goal to achieve which is specified within the transition that contains the *goal achieved* description. Executing the ACE Plan means executing actions specified within transitions. Plan execution always starts at the initial state and ends after reaching a final state.

The Reasoner executes the Plan in two cases: first, immediately after it receives the new Plan from the Facilitator and second, when the ACE service is requested by another ACE. In case the ACE Plan contains a loop back transition between two states, the Plan will be continuously executed. In this current implementation the Reasoner also takes care of gathering the context information.

The Reasoner relies upon the OOjDREW rule engine [JDREW]. It translates the ACE Plan into standard RuleML syntax, executes the rules in the rule engine, evaluates the outcome and performs actions accordingly.

3.1.5 Lifecycle Manager

The Lifecycle Manager is responsible for starting and stopping an ACE. It is triggered by the Reasoner which may request any lifecycle action according to the ACE Plan. The Lifecycle Manager will then prepare the requested action by informing all other ACE Organs. These have to prepare themselves for the announced action. Having done this, each Organ has to confirm to be ready by answering with a LifeCycleEventResult. The Lifecycle Manager collects and evaluates all results and finally executes the requested action when all Organs are ready.

The first version of the CASCADAS Toolkit implements the lifecycle actions *start* and *stop*. For the next release, it is planned to include cloning and moving of ACEs as well. These features will then enable ACEs to be mobile and will provide them with the ability to change their working environment if necessary.

3.1.6 Functionality Repository

The Functionality Repository is an Organ which is responsible for deploying and executing individual services of an ACE. It is used as a container and access point for the features, an ACE is equipped with. Arbitrary code, specifically programmed as well as already existing libraries, can be added to the Functionality Repository. Using the concepts of modularity and reusability optimises programming expenses for creating systems of ACEs.

To deploy a service to the Functionality Repository of an ACE, a library implementing the service and an XML descriptor describing it are necessary. The descriptor includes an identification (ID) and a name for the functionality that should be deployed. Furthermore, it lists the required input and output parameters.

Any service which is deployed to the Functionality Repository can be accessed using events. ACE internal functionality calls are dispatched by the Bus. If an Organ likes to invoke functionality from the Repository, a FunctionalityCallEvent has to be sent. Thereafter, the requested functionality is executed and the results are returned. In the case that an ACE likes to call a service provided by another ACE, the requester has to send a ServiceCallEvent to the provider. This results in the execution of the desired functionality and the delivery of the results. Of course, all service calls have to contain values for all parameters specified in the XML descriptor of the invoked service.

Looking at the Organs comprising the ACE model, each of them contributes to fulfil the ACE platform requirements. A Common Part containing all Organs and being implemented equally guarantees self-similarity. The features and properties the Common Part provides are granted for each and every ACE. Keeping this common functionality restricted to what is absolutely necessary for all ACEs ensures their light weighted character. Anyway, the possibility to deploy any individual service to the Functionality Repository enables a practically unlimited diversity of functionality to be offered. Autonomicity and situation-awareness result from the ability of the Facilitator to react on the context an ACE encounters. Its behaviour, even its main goal, can be adapted accordingly. The GN-GA (Goal Needed – Goal Achievable) protocol is used to advertise an ACE's own services and find those which an ACE cannot perform itself. Communicating and interacting this way enables the creation of a complex autonomic system.

3.2 Toolkit Dependencies

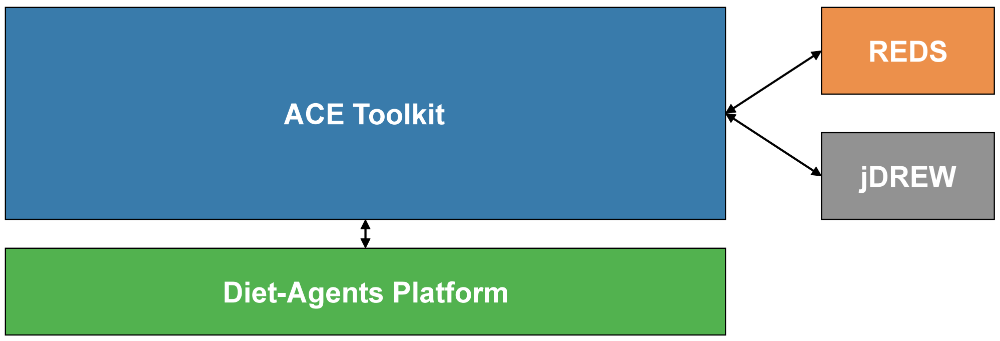

Figure 5: The ACE Toolkit and its dependencies

The ACE Toolkit is the synergy between the new original solutions of the CASCADAS project and three previously existing, external software components: Diet-Agents, REDS and jDREW (Figure 5).

- The ACE toolkit has been built upon a well-known agent platform: the toolkit uses the Diet facilities for the execution and migration of ACEs, and for the dedicated one-to-one communication.
- REDS (Reconfigurable Dispatching System) is a distributed publish-subscribe system for large and dynamic networks [REDS]. REDS has been used as the basic information propagation mechanism for unaddressed messages (when the recipient of the message is not known), namely for the GN-GA protocol.
- jDREW (Java Deductive Reasoning Engine for the Web) is a powerful deductive reasoning engine for first-order logic [JDREW]. jDREW is used in the Facilitator for plan creation and in the Reasoner for plan execution.

The ACE Toolkit has been implemented in Java (version 1.6) in order to ensure platform independence. The three external components are attached to the toolkit in form of libraries, making it easy to differentiate between the toolkit code and the integrated external software.

4 Demonstrator

To illustrate the feasibility and usefulness of key research results of the project, a demonstrator will be implemented on a test-bed setting. This section describes the software architecture of the demonstrator. The software relies on the ACE model described in the previous section for essential communication facilities.

4.1 Overview of scenario

Overall, the scenario that was selected for demonstration purposes considers a modern exhibition centre, such as a big museum or a stadium. In contexts of this kind, it is realistic to assume the presence of a pervasive infrastructure of embedded devices such as sensors of various types, PDAs, RFID tags and other location systems. In fact, exhibition centres may afford the costs of deploying such infrastructures if this enables them to provide good services to visitors (and accordingly, to attract a higher number of persons) and get higher revenues. Also, we consider realistic to assume that visitors to the exhibition centre may have a PDA or a smart phone, or that they can be provided with an RFID-based ticket, which can store information about entrance fees and additional information about the user.

We consider the presence in the exhibition centre of a number of advertising screens that can be used to display to visitors information about the exhibition itself as well as commercial advertisements. Similar advertising screens are increasingly deployed, for example, in public areas, such as airports, rail stations and exhibition centres. However, current advertising screens only display generic information by simply cycling over a sequence of advertisements and independently of the situation (i.e., independently of who is actually close to the screen). A "smart" service devoted to decide what information to display could exploit the availability of contextual information to adaptively decide what information to show on the basis of the people around and of their activities and interests. This would increase the value of the displayed advertisement both to users and to advertising companies.

To make the above adaptive decision possible, it is necessary, on the one hand, to properly exploit the pervasive infrastructure to gather and analyse all the information necessary to *(i)* make it possible to evaluate what advertisements would be better to show at a given time and on a given screen, and *(ii)* provide means for someone interested in buying a screen time slot to show a specific advertisement to actually get access to the proper time slot. The above two issues are particularly challenging, for several reasons:

First, in a large exhibition centre, there may be hundreds of screens and dozens of thousands of users, each with peculiar preferences and profiles. Thus, it is necessary to continuously collect and organise large amounts of data that can be of some use to evaluate what kind of advertisements, for each of the many screens, would be worth showing at any given time. At the simplest level, this may consists in collecting (anonymously) from users' PDAs/smart phones or from their RFID tickets, the profiles of all the users nearby a given screen. However, one could also think of a more elaborated way to profile users and to track their activities (anonymously), such as to recognise, for instance, that a group of users having already seen elsewhere a specific advertisement may obtain little value in showing such advertisement again.

Second, in the presence of a large number of screens and of a large number of parties interested in buying screen time slots, solutions must be identified for allocating such time slots and for enabling interested parties in getting the best out of them (i.e., by being given the possibility of displaying advertisement in the presence of people potentially interested in them, and of paying for this a balanced bill). Auctions are one of the best solutions for allocating screen time slots. However, executing auctions in the envisioned scenario is particularly challenging due to the system dynamics. On the one hand, for each screen, and assuming commercial advertisements of 20 seconds, there will be three auctions per minute. On the other hand, interested parties (which, in the end, will be software agents in charge of participating to screen auctions) cannot decide in advance how to bid in such auctions. The price to pay for a screen time slot may continuously change for each screen and depending on the people currently surrounding the screen.

The demonstrator that will be put in place considers (see Figure 6):

- one real screen devoted to actually displaying advertisements in an adaptive way, depending on the surrounding users and properly controlled by an associated PC
- real users, each with either a RFID or PDA, that can get close to that screen and whose profiles can be tracked and analysed (anonymously) for the sake of evaluating what advertisements to show. An RFID reader on the screen PC will make it possible to access RFID data. In addition, data on users' mobile devices could be accessed by the PC via Bluetooth as well as via WiFi.
- The execution of distributed auctions to sell the screen time slots. These auctions will take place by executing a number of ACEs on the CASCADAS distributed test-bed, which will be connected via VPN to the local PC.

More detail on the envisioned overall software architecture is described in the following.

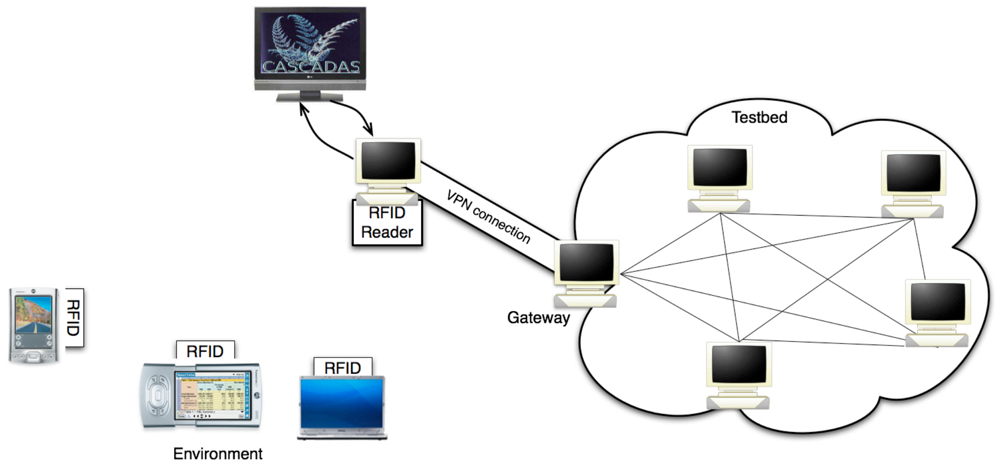

Figure 6: Demonstrator scenario hardware configuration.

4.2 Software architecture

The software architecture, will be fully ACE-based, and will consider the following. An ACE-based implementation of the knowledge network will be allocated on the screen PC to provide up-to-date information to any requesting ACE, whether local or remote. An ACE will be associated to the screen PC and will act as a "seller" for the time slots of the screen. Such ACE can advertise the auction in a (possibly remote) auction centre, again realised via ACE. A variety of other ACEs will be instantiated on the remote CASCADAS test-bed to act as bidders and to participate in auctions related to the selling of the screen time slots.

Because of the nature of the application, the implementation of security mechanisms (e.g. to offer message encryption) are key factors to take into account. Within the project, we are investigating efficient ways to offer basic security features with the ACE model. Furthermore, we are also investigating advanced security features that could be offered by ACEs to diminish denial-of-service attacks that will be integrated to the demonstrator at a later stage of the project.

4.2.1 Pervasive Advertisement

The architecture for dealing with the specific part on pervasive advertisement considers the following.

RFID tickets

The RFID tickets will include the following information:

- user ID (it is assumed that the same user ID will also characterise the mobile devices of the user)
- user interest, in the simple form of a 2-field tuple *(age range, subject of interest)*.

The user interest will be stored in the RFID tag in the simple form of two single-byte numerical values. When this data is transferred, via the RFID reader, to the screen PC, and will feed the knowledge network, it will be properly and automatically translated into the XML format for Knowledge Atoms (see Figure 7).

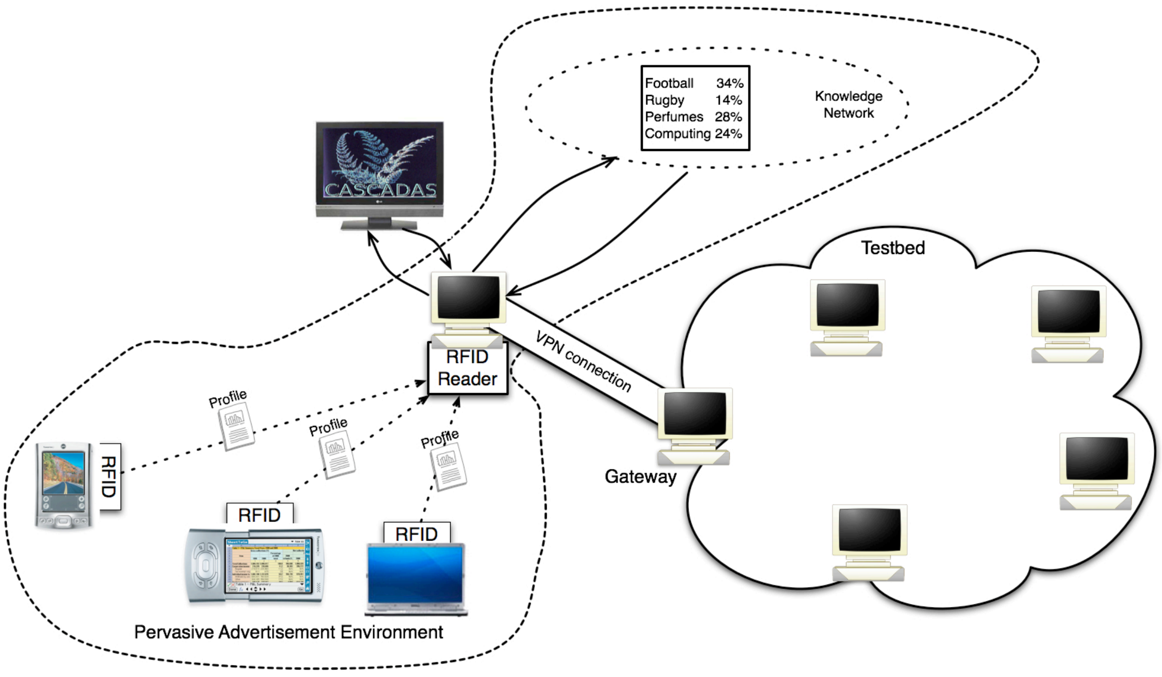

Figure 7: Acquiring user profiles and elaborating them within the knowledge network.

We will consider the following values for age ranges and subjects of interest:

AGE RANGE AND GENDER

Numerical Value in RFID	XML Value
0	Male Child
1	Male Teen
2	Male Young
3	Male Middle Age
4	Male Elder
128	Female Child
129	Female Teen
130	Female Young
131	Female Middle Age
132	Female Elder

SUBJECTS OF INTEREST (more can be added if needed)

Numerical Value in RFID	XML Value
0	Football
1	Basket
2	Trekking
3	Fashion
4	Cosmetics
5	Movies
6	Books
7	Toys

Knowledge Network

The information above will be dynamically gathered by the screen PC, and will enter a knowledge network locally allocated on the PC. At any given time, the knowledge network will contain one knowledge atom for any user currently in range of the reader.

Within the knowledge network, at least one knowledge container will be present to refer to all such knowledge atom, gather information from them, and aggregate such information to produce statistical information about who is around. In the simplest form, the provided information will be a list of the percentages for each class of persons around and for each subject of interest. For example, a possible answer at a given moment could be as follows:

Age Ranges and Gender	Percentage
Female Child	21%
Male Young	32%
Female Middle Age	47%
Interests	**Percentage**

Toys	24%
Movies	56%
Books	66%

The above information can be made available by the knowledge network to any requesting ACE. Also, please note that access to individual knowledge atoms is always made possible for any ACE, thanks to the specific mechanisms integrated in knowledge networks.

As the users also carry on a mobile device, we will also consider the possibility of having such users connect their devices to the laptop directly (e.g., via Bluetooth or WiFi) to provide more information. In fact, the mobile devices held by users can also store and dynamically gather more information about their own user (e.g., what room it has already visited, and what advertisement it has already seen. Thus, one can think that once a user has been recognised via the RFID, some connection can be established with his mobile device.

4.2.2 Advertisement Auction System

In the scenario, sequential time slots become available over time, each of them more or less suitable for a particular type of audience. The pervasive computing component described above allows the system to learn the suitability of potential viewers to receive different types of advertisements. Such time slots become valuable to sellers interested in advertising their products. In the scenario, time slots are sold through an auction system. As a general rule, each time slot will need to be sold within stringent time constraints (e.g., while another advertisement is being displayed), so that real-time communications and high standard QoS networks become critical for the operation of the system.

The auction system is a typical example of a wide-area computing application exhibiting real-time requirements and QoS of communications (e.g. low delay and high packet delivery ratio), which could directly impact the revenue of participants. For example, given that auction transactions need to be completed within a short time period, the selling price of a time slot could not reach high values if slow or error prone communications are used. On the other hand, the wide-area nature of the communications could also influence a bidders' winning rate and their possibility of advertising products to the right audience, will the consequently loss in selling revenues.

Although, time slots could be sold ahead in time in a real system possibly allowing the creation of a complex market for such slots (e.g. reselling), we will concentrate on a single-slot selling to simplify the demonstrator. Therefore, the seller will always advertise the auction for the slot of time starting after the current one, and therefore auction duration will not have to exceed the duration of a slot.

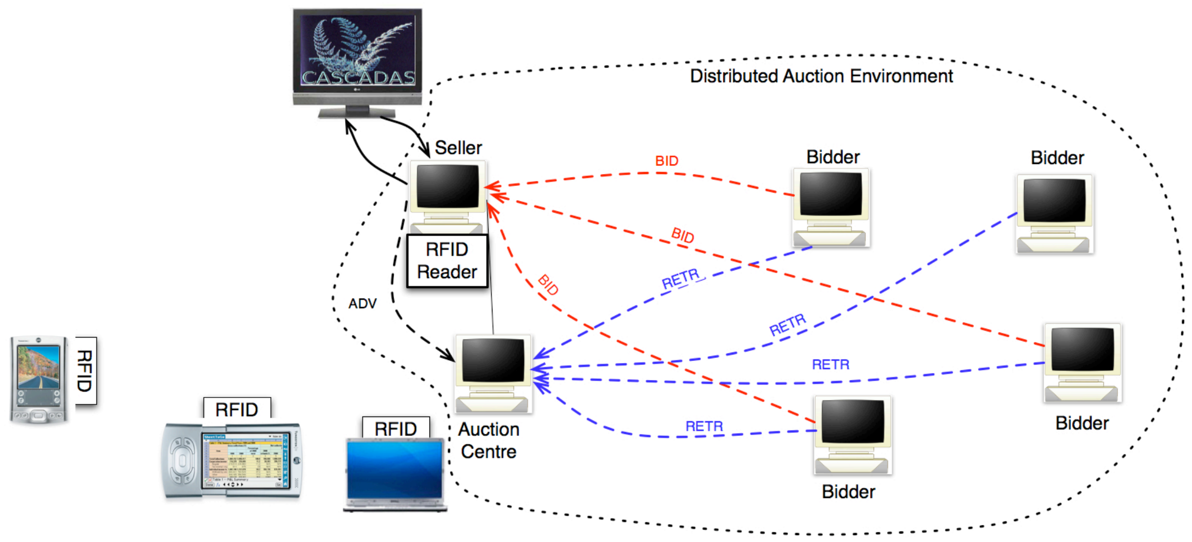

Figure 8: Auction System in the demonstrator scenario.

As for the auction component, three main roles are foreseen:

- *Auctioneer*: the entity owning the item, and willing to sell it (thus acting as a *seller*) under a specified set of rules collectively defined as the *auction model*. In our scenario, the referring auction model is the *English Auction* model, characterised by the fact that bids aim to increase the item's initial (minimum) price stated by the seller.
- *Bidder*: the entity willing to compete to the extent of buying one or more items.
- *Auction Centre* (AC): provides a meeting point for auctioneers and bidders. Through an AC, auctioneers advertise their will to auction goods, and bidders come acquainted of what goods are currently being auctioned. ACs list currently ongoing auctions through an *Auction Web Page* (AWP), which is kept consistent by direct communication with sellers.

In the scenario introduced as demonstrator, the machine communicating with the main display will act as a seller. The item under auction is the concession to advertise, on the main screen, for a slot of time of fixed length and the machines on the test-bed will act as bidders competing for the advertisement slot.

4.2.2.1 Auction Messages

Parties involved in an auction communicate by exchanging *Auction Messages* (AMs). The structure of such messages is shown in Figure 9.

Sender	Type	Data	Destination	ID	Max_no	Seq_no

Figure 9: Auction Message Format

The `Sender` field allows specifying the sender of the message. The `Type` field defines the purpose of the message that, in turn, allows destinations to process it accordingly. A more detailed description of message types will follow in this section. The `Data` field encapsulates the objects to be processed and, together with the Type field, allow destinations to perform actions with respect to the auction transaction. A more detailed discussion on the objects used will be provided later on in this section. The `Destination` field contains the list of addresses that specify the destinations the message. When the list contains more than one destination, such as for instance when the seller notifies termination of the auction, the message is transmitted by a set of concurrent unicasts. The `ID` field provides the message with a unique identification number, while the `Max_no` and `Seq_no` allows handling cases where the message needs to be fragmented. In fact, lower-level communication facilities pose a limit on the size of packets to be sent. When the size is exceeded, as for instance when the message is transmitted by an AC and contains a list of items of interest for a bidder, the message is fragmented and transmitted sequentially. Then, the Max_no field identifies the total number of packets the message is composed by, while the Seq_no specifies the position if a packet in the original message.

The purpose of a certain packet, in the context of the auction transaction, is specified by its type. The following types have been individuated as sufficient to carry out the auction transaction:

- `ADV`: the message has the purpose of advertising an auction. Typically this message is produced by the auctioneer that notifies its will to auction a certain item under a certain auction model. The Data field will thus contain an object, the `AWPElem`, containing all information needed by a bidder to decide whether to participate in the auction or not.
- `BID`: the message contains a bid for a specified auction. The producer will thus be a bidder, while the consumer will be the auctioneer.
- `SELL`: the message is to signal that a specific auction has terminated and a winner has been declared. The message is produced by the auctioneer and directed to all auction competitors. The auction the message refers to, is specified in the Data field, along with information on who is the winner and the final price paid.
- `UPDATE`: with this message, the auctioneer asks the AC to update details of a specific auction. This message will typically transmitted when the auctioneer receives a bid for an item under auction, and the update ad the AC allows new bidders, eventually joining the auction at a later stage, to come acquainted of the current price for the item.
- `HIGHEST`: notifies auction competitors that a new highest price for an item under auction has been offered. The auctioneer typically transmits this message to all competitors upon reception of a new bid. This message is received by all competitors. In particular, reception of this message allows the previous highest bidder to be notified that it has been outbid, while acknowledging the new highest bidder as such.
- `RETRIEVE`: this message is sent to the AC by the bidder, which asks the former to retrieve auctions of interest based on information contained in the Data field. Auctions of interest are specified, in the Data field, by describing characteristics such as price.

- `LIST`: this message contains the AC's reply to the `RETRIEVE` message. It is therefore used by the AC to reply to the bidder, and it will typically contain a list of objects the bidder finds of interest.
- `DELETE`: when a specific auction is terminated, the auctioneer sends this message to the AC to the extent of asking deletion of that specific auction from the AWP.

Figure 10 summarises message types, also highlighting producer, consumer, purpose and object contained.

Message Type	Producer	Consumer
`ADV`	Auctioneer	AC
`BID`	Bidder	Auctioneer
`SELL`	Auctioneer	Bidders
`UPDATE`	Auctioneer	AC
`HIGHEST`	Auctioneer	Bidders
`RETRIEVE`	Bidder	AC
`LIST`	AC	Bidder
`DELETE`	Auctioneer	AC

Figure 10: Summary table for message types.

4.2.2.2 Auction Transaction

The auction transaction starts with the seller, i.e. the machine connected to the main display, advertising its will to sell an advertisement slot of time through an English auction. To this extent it sends an `ADV` message to the AC, as shown in Figure 11 and waits for bids to arrive.

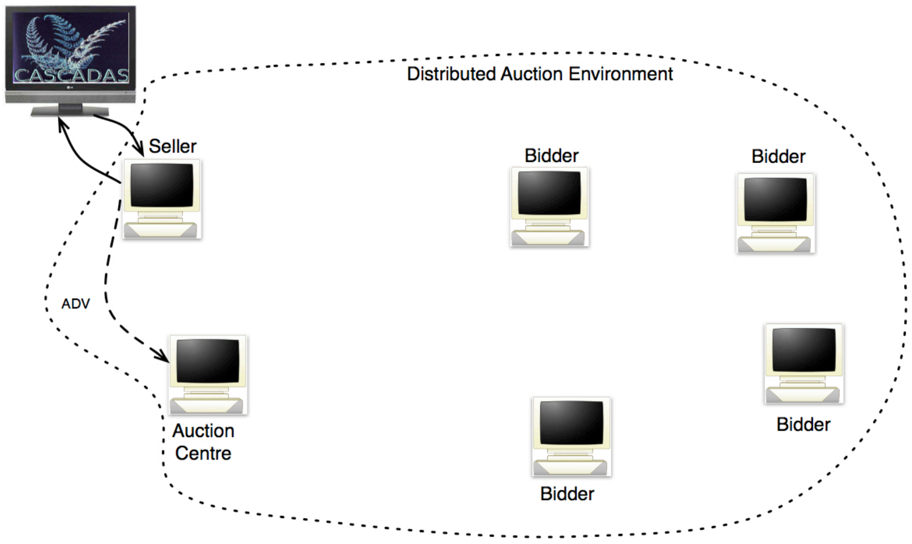

Figure 11: Advertisement on an Auction Centre

The AC receives the message, extracts the object contained and inserts it into the AWP. Sometimes later, one or more bidders will ask the AC for retrieval of a specific good. The AC replies, after inspection of the AWP, with the list of goods of interest for the bidder. This phase is depicted in Figure 12.

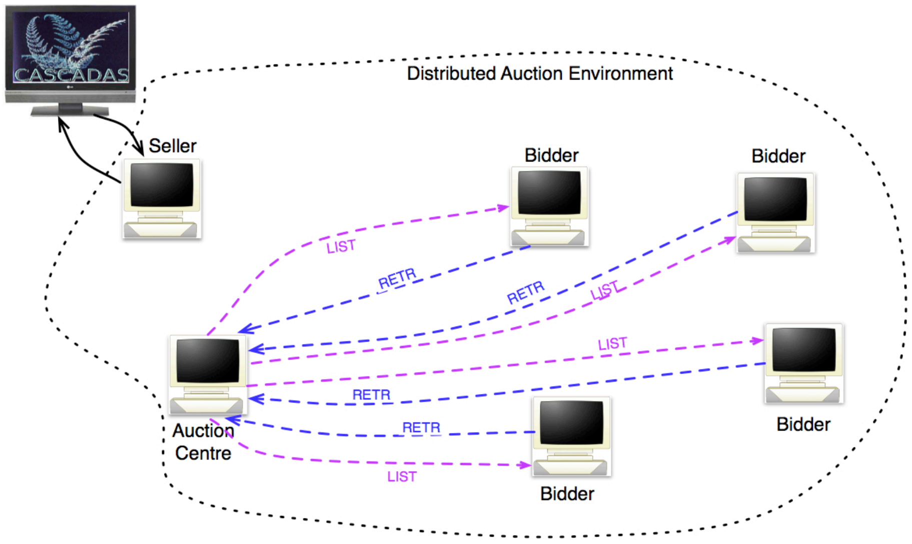

Figure 12: Item Retrieval phase.

Bidders then receive a list of items of interest and chose the most suitable one, if any. Placement of a bid is realised by direct point-to-point communication with the auctioneer. This latter receives bids, for a specific auction, and notifies all competitors of a new highest bid. This message has several purposes:

- It notifies the previous highest bidder that it has been outbid;

- It acknowledges the new highest bidder that the bid has been received and is the current highest;
- It notifies competitors (other than the highest bidder) that eventual new bids cannot have a price smaller than the new current one, also notifying them that previous bids placed for lower prices were found to be insufficient.

In addition, as Figure 13 shows, upon reception of a bid, the seller contacts the AC for update of the auction status. This is done through an `UPDATE` message containing details, i.e. the new price, to change in the stale advertisement. The purpose of the update is to keep consistency between the advertisement at the AC and the current evolvement of the auction, so as to allow eventual new bidders joining the auction at a later stage to know the up to date price of the item.

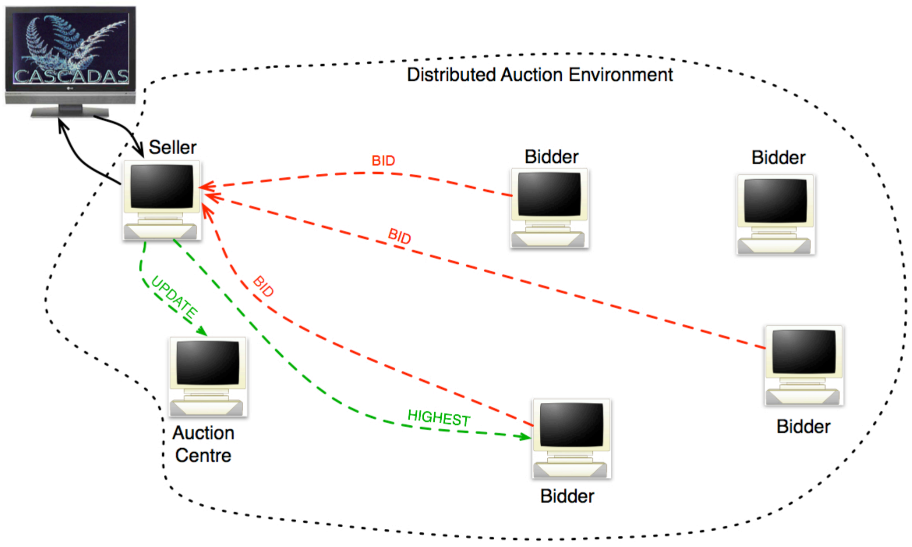

Figure 13: Auction bidding phase.

4.2.2.3 Bidding Decisions

Bidding decisions are taken, in the auction system, according to a *policy*. A policy defines what and how parameters need to be considered when taking decisions. These are then evaluated mathematically through an aggregate function, and the policy contains facilities to map the value obtained by the function to the final decision to be taken.

Bidding policies are composed by three fundamental objects:

- *Aspects*: is a set of parameters that need to be considered when taking decisions. Typical aspects will be the current budget, the price to pay, etc. Elementary aspects will be numbers, such as the aforementioned budget, and more sophisticated aspects can be modelled as a function of other parameters. For instance, the available budget, i.e. the current budget minus the price eventually committed in other auctions, can be defined as a function of the initial budget and used to model the relevance of the item on the current supply. Aspects so modelled are then evaluated through an aggregate function, to the extent of obtaining a number to be mapped onto a final decision. The aggregation function can be as simple as a multiplication of the evaluation of each parameter.

- *Consideration Factors*: each aspect is "weighted" by means of a consideration factor. The purpose of such weighting is to quantify the impact of a certain aspect based on its relevance in the exact moment the decision needs to be taken. For instance, if it is detected that the impact of a bid on the current budget is reduced, the budget aspect in the policy can be assigned a small factor.
- *Thresholds*: provide a mapping between the value generated by the aggregated function and the final decision to be taken. Providing that such function generates a number *E* such that $0<E\leq 1$, the use of threshold foresees the interval <0,1] to be divided into as many *buckets* as are the possible decisions. Then, the decision corresponding to the bucket where the value *E* falls into will be the final one. In the simplest case, the decision will be either "bid" or "do not bid".

The widths of the buckets, as well as the actual value of the consideration factor for each of the aspects defined, are assigned dynamically. Assignment will be based on current environmental conditions, i.e., context-awareness, and local conditions, such as, the financial status.

Bidders can be individual users, enterprises or a *coalition*, i.e., a number of single entities acting jointly. In this latter case, a leader acts on behalf of all the entities included in the coalition. Members democratically (for instance by means of a fair consensus protocol) take a set of decisions that will regulate the coalition behaviour. Decisions to be taken include, in the case of a coalition of bidders, leadership, maximum payable price for a good, bid increase etc. Coalitions will be able to revise own strategies based on past experiences, for instance increasing the maximum price and/or bid for a good if bids for a good of the same time have always been outbid in past.

4.2.2.4 Architecture

The software architecture of the auction system has been organised in packages whose structure is depicted in Figure 14.

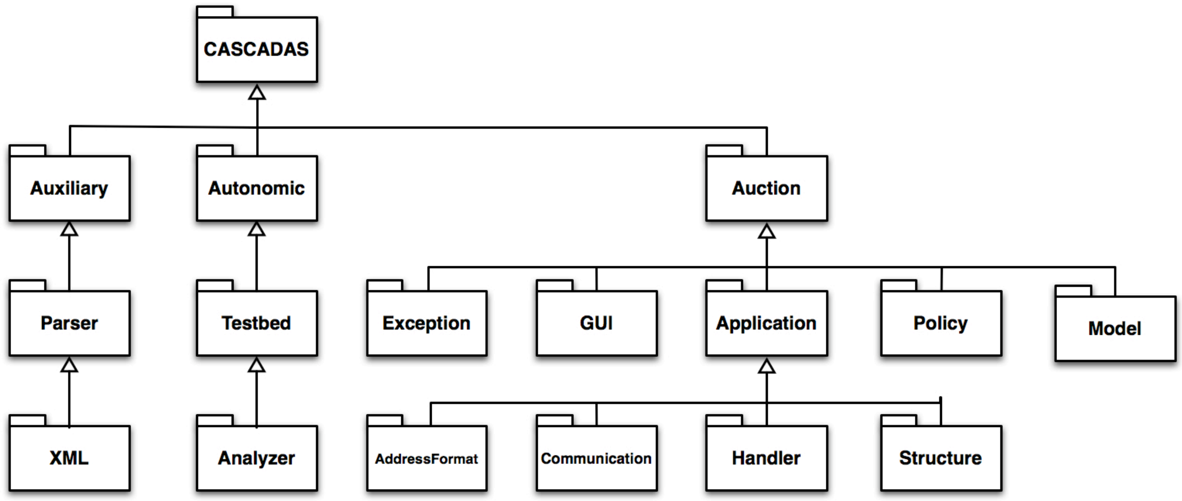

Figure 14: Package Structure.

The main `CASCADAS` package contains three sub-packages. The first, named `Auxiliary` contains facilities that do not influence the auction directly but might be helpful in extending the I/O system. At present, this package contains a simple XML parser.

The `Autonomic` package contains objects to provide the system with the *Cognitive Packet network* (CPN). The original technology, coded in C++, has been integrated in the (Java-based) auction system through the *Java Native Interface* (JNI) technology. The interface has the purpose of translating Java objects into objects that the CPN object underneath can understand. The actual communication is then realised through the CPN, and the eventual output is then converted back into a Java objects for the consumption in the auction system.

The fundamental objects for the software architecture of the auction system are contained in the `Auction` sub-package. Besides the `Exception` and `GUI` sub-packages, which are self-commenting in their names, the `Application`, `Policy` and `Model` sub-packages can be found, containing application fundamental objects, policy-based decision files and definition of auction models respectively.

The `Application` sub-package is specialised by the `AddressFormat`, `Communication`, `Structure` and `Handler` sub-packages. `AddressFormat` contains definition of communication-specific addressing schemes. `Communication`, on the other hand, defines general-purpose objects for sending and receiving AMs while `Structure` contains data structures to support the auction transaction.

The logic for the role-based behaviour, i.e. the logic for behaving as auctioneer, bidder or AC role, is contained in the `Handler`. Each of the roles is defined by specialising a `Handler` interface contained in the sub-package. As a consequence, the auction transaction is regulated by handlers.

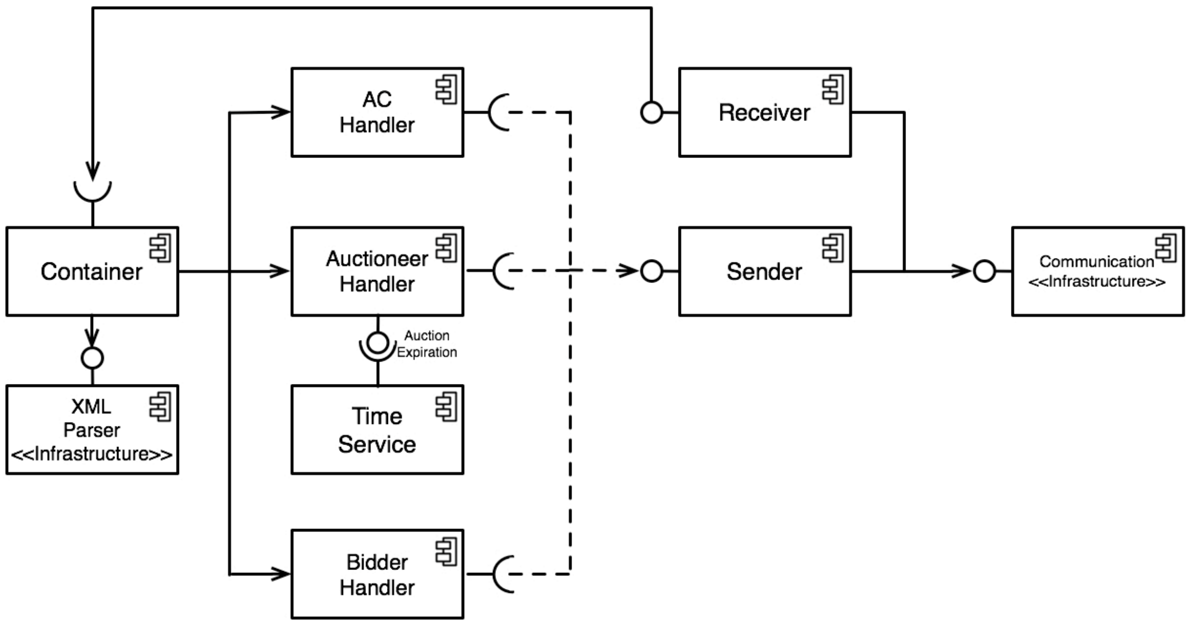

Figure 15: UML Component Diagram showing the software architecture.

Figure 15 contains a UML Component Diagram showing how handlers make realise the auction transaction. The `Container` is a special handler whose purpose is to dispatch messages received, by the `Receiver` object through the CPN, to the right handler. The use of the `XMLParser` object, shown in the figure, aims to extend the system with capabilities to interpret configuration files written in XML format.

The selection of the right handler is made according to the type of the AM received. More precisely, the message is delivered to the consumer specified. Once determined the right handler, the message is consumed, according to the behaviour specified by the role, and the corresponding action is taken. When such action involves communication with other parties in the auction, handlers use the lower level `Sender` and `Receiver` which, in turn, use the corresponding CPN facilities to carry out the actual communication. As a part of its logic, the `AuctioneerHandler` is required to handle timeouts to regulate duration of auctions. This functionality is asynchronous through the use of a `TimeService` object. Each time the auctioneer needs to setup a timeout, it creates a listener and deposits it into the list (of listeners) contained in the `TimeService` object. A thread "ticks", then, time to check whether the current time has exceeded the deadline in the listener. If such a deadline has passed, the timeout is expired and the auctioneer is notified through the method associated to expiration.

4.2.2.5 Requirements

Most requirements of the auction application were identified previously and reported in Deliverable 6.1 part A. The main requirements are:

- Determination of the delegated node needs QoS-sensitive information;
- Communication among parties involved in an auction must provide QoS guarantees;
- Means to transfer the state of an auction object, when migrating;
- Communication between aggregated entities and/or entities united into a coalition must be exclusive, secure and protected;
- Broadcast communication primitives are needed inside coalitions;
- Channel links between users and its Virtual Locations must be made stable;
- Fairness must be guaranteed among entities engaged in a coalition;
- Monitoring for malicious behaviour must be provided when coalitions are formed
- Fairness must be guaranteed when there is competition in the context of an auction;
- Identity of bidders and sellers must be validated;
- Trust must be provided when a seller and many bidders engage in an auction;
- Non-repudiation needs to be provided when auctions are conducted;
- Access control is needed to regulate the per-auction role-based behaviour;
- Load-balancing techniques must be provided in order to make auctions scalable in terms of bidders capable to be handled;
- Bidders and sellers must have a way to seamlessly discover the presence of ACs;

4.2.3 Knowledge Networks

Knowledge networks will support the demonstrator by making available an ACE-based software to access information about users' interests. As already described previously in this document, one instance of the knowledge network will run in the screen PC, and will dynamically manage the knowledge atoms connected to RFID describing users' interest.

The knowledge network will be realised in terms of ACEs, and will be accessed by other ACEs via a specific instantiation of the GA/GN protocol, that is, a "knowledge needed / knowledge available" protocol. Via this protocol, application ACEs (i.e., potential bidders interested in evaluating whether to bid or not) can "query" the knowledge network to get aware about the current situation of users, i.e., their characteristics and interests.

Although the specific format of the GA/GN messages that will be used to query the knowledge network will be decided at a later stage, what it is already decided is that knowledge network will make it possible for ACEs to query about both aggregated information and individual information. Aggregated information will provided to ACEs a synthetic indication of the overall interests of users around (see also Subsection 4.3), and will enable potential bidders to get a very quick understanding of the usefulness of bidding. Further, special-purpose, aggregation algorithms can be later integrated in the knowledge networks to help potential bidders in understanding whether and how to form coalition, and to help AC to properly advertise auctions. In addition, the knowledge network will also make it possible for any ACE to access, via proper queries, to atomic information about individual users.

At a later stage, the knowledge network will incorporate information collected by sensor networks in the environment that, if properly mixed with RFID user information, can be exploited by potential bidders to get more informed decisions. Also, it is planned that the knowledge network will integrate algorithms to evaluate consistency of knowledge and avoid errors due to partial/inconsistent information.

4.3 Expected demonstration and results

The demonstration will show how the CASCADAS technology can be used to realise an auction-based pervasive advertisement application. A main screen, located at the demonstrator site, will project advertisement whose topic will change based on interests of the crowd currently on site. The advertising time will be divided into one-minute slots, and several companies will compete to acquire the rights to advertise for a certain slot by participating to an auction.

As anticipated in Subsection 4.1, this scenario will be realised as follows. At the demonstrator's location, a laptop computer will be connected with the main screen projecting advertisements. In addition, a series of mobile devices will be available, always at the demonstrator location. Bearing in mind the hardware setup, a laptop computer will be equipped with an RFID reader, for gathering information about surrounding users, each of which will have an RFID ticket, with some basic user information (age range, subjects of interests), to feed the local knowledge networks. Upon recognition of the presence of a user, the laptop computer can contact the mobile device of that user for negotiating the provisioning of further user details. All the data eventually obtained will be gathered by the laptop computer, and processed/organised at the local knowledge network, and made available to whatever ACE is interested in. At the more elementary level, the result of knowledge network processing can be a statistical real-time enumeration of interests as specified by the users, which will be used as a source of information in the auction for the one-minute advertisement time slot to be carried out soon thereafter. The rationale for including the statistical information to the slot of time is that bidders will use such data to decide whether it is worth to place a bid for that particular slot or not. For instance, the statistical data might specify that the environment around the main screen is composed by people whose interests are divided as shown in Figure 16:

Football	36%
Cosmetics	32%
Trekking	19%
Computing	13%

Figure 16: Example of statistical enumeration of interests.

Bidders might use this information to decide whether to get involved in the auction for the slot of time associated with this table. Thus, a company selling sport clothing in general might be interested in getting involved in the auction, since it might join interests for football and trekking, while a company specialised in home furniture would not find any relevance in the crowd's interests.

The actual auction will be conducted internally an ad-hoc test-bed made available by the partners of the CASCADAS consortium specifically for this purpose, and will be carried out as follows. The laptop computer connected to the main display will aggregate to a machine internal to the test-bed, and to this latter it will delegate auction responsibilities. This counterpart will thus act as a seller and start the auction by advertising the item a likewise internal AC.

Machines internal to the test-bed, other than the seller and the AC, will simulate the businesses participating on the market where the item is advertised. Each one will include a characterisation of the business of the participating company. This will be used to decide whether to place a bid or not on the auction selling an advertisement slot of time. The actual decision will be taken by matching the statistical relevance of the crowd's interests to the interests of the business as specified in their policy.

Once an auction terminates, and thus a machine acquires the rights to advertise on the main screen for the next minute, the winner can instruct the seller what advertisement to show on the screen. This request will then be redirected to the aggregate machine at the demonstrator's site, where the advertisement can be shown.

4.3.1 Auction Visualisation

The demonstrator will also include a visualisation system by means of which it will be possible to observe auctions remotely. The visualisation system will consist of graphical and textual information about auctions.

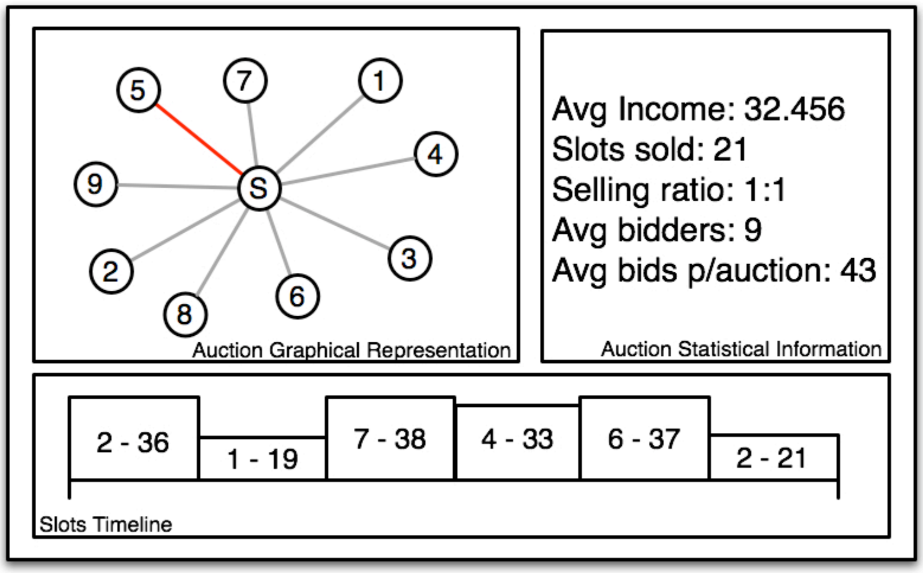

Figure 17: Sketch of the auction visual observation interface.

In particular, the visualisation system will be composed of three main sections, as Figure 19 shows:

- **Auction graphical representation**: will allow to graphically observing the evolution of the auction at runtime. An auction will be represented in as a graph, with the seller in the middle of a star-like structure and bidders at the edge. Involvement of a machine in the auction process, i.e. placement of a bid, will be represented with a line connecting the actual bidder with the seller. The colour of the line will determine the status of the bidder, and different colour will define the highest bidder (shows as a red line in Figure 17), outbid bidders (grey lines in figure), and bidders currently bidding (green lines in figure). In addition, links will display network information, such as for instance the network delay towards the seller.
- **Slots timeline**: will provide a graphical visualisation of the outcome of past auctions. The visual representation will be a sequence of boxes. Each of these will represent a slot auctioned in past, whose height will be determined by the price achieved at termination of the auction. Internally, the box will show auction information such as name of the winner and the final price paid.
- **Auction Statistical Information**: will provide statistical textual information on a per auction basis. The information shown might include, but not limited to, seller's average income per slot, number of slots sold, ratio between slots auctioned and sold, average number of bidders per slot and average number of bids per auction (i.e. per slot).

The auction observation interface will be made available through a Java applet. Thus, anyone interested in observing how auctions are being carried out will be able to connect to a specific website and visualise the applet.

5 Conclusions

This document has described the essential characteristics of a demonstrator that will serve to illustrate key research results of CASCADAS. The demonstrator considers a modern exhibition centre equipped with screens that can be used to advertise products and services to people attending an event. The exhibition includes a large pervasive infrastructure of embedded devices where people with smart devices can transit freely. The scenario offers two distinct features. It allows advertisers to target announcements to the current audience by either displaying proper information to the people near a screen or by directing messages to selected smart devices. This application is a typical example of pervasive computing with scalability constraints. On the other hand, it also allows advertisers to compete, with auction rules, for screen time slots to show information of their interest. The latter case represents a wide area network application with hard real-time constraints. Although current technology may allow the implementation of this scenario, its complexity would limit its scalability so that larger deployments would require more advanced and innovative solutions. We expect that the proposed architecture will serve as an outstanding showcase for CASCADAS results.