A Mobile Agent Architecture to Provide Virtual Home Environment Services to Nomadic Users

Final Report

Written by: Alan Treadway
Supervised by: Prof. Morris Sloman
Second Marker: Dan Chalmers

Project home page: http://www.doc.ic.ac.uk/~ajt6/project.html

Acknowledgements

I would like to thank my supervisor Professor Morris Sloman and second marker Dan Chalmers for their invaluable assistance through the course of this project, providing reference sources, encouragement and inspiration.

I would also like to thank Rebbecca Montanari for her assessment of the original system design, and providing access to a number of relevant papers.

Abstract

The ability to access coherent information services such as email and address books between multiple devices has so far been a result of static implementations that favour fast reliable connections and large-footprint web browser software. As a result, information access on the move via mobile devices is far from satisfactory, despite the fact that this is where access to coherent information services such as email come into their own.

The primary aim of the project is to investigate the way in which Mobile Agent Technologies can be used in order to improve this situation through development of a Virtual Home Environment based upon these technologies. The Virtual Home Environment provides user interfaces suited to the capabilities of the devices being used, as well as ensuring maximum effective use is made of available resources on the device and the network to which it is attached.

Contents

Glossary of Terms

A number of acronyms are used throughout out this report, the table below highlights the commonly used terms.

Term	Definition
ACC	Agent Communication Channel – FIPA defined Agent that acts as a gateway between LAP and RAP for Agents.
ACL	Agent Communication Language – FIPA defined grammar that all FIPA compliant Agents use for inter-agent communications.
AI	Artificial Intelligence.
AMS	Agent Management System – FIPA defined Agent that controls an AP.
AP	Agent Platform – The software entity required for Agents to exist. May span several nodes.
API	Application Programmer Interface – A software entity that is a well-defined interface to some software for third-party developer to use when interacting with that software.
ASCII	A numerical encoding for alphanumeric data that has been widely adopted across all manner of technologies.
AUV	Autonomous Underwater Vehicle – An autonomous, un-tethered vehicle that can navigate the seas carrying out missions without human intervention.
AWT	Abstract Windowing Toolkit – A Java API providing the ability to create window-based application without any knowledge of host systems' windowing system (e.g. Windows, X, MacOS).
CAMELEON	Communication Agents for Mobility Enhancements in a Logical Environment of Open Networks – European project investigating VHE concepts based upon UMTS specification.
CORBA	Common ORB (Object Request Broker) Architecture – a standard that provides interoperable mechanisms for distributed object method invocations.
DBA	Database Agent – An Agent that provides access to some common store of persistent information, which can be queried and updated.
DCOM	Distributed Common Object Model – Microsoft's DOT for the Windows platform.
DF	Directory Facilitator – FIPA defined Agent that provides yellow-page directory for Agents offering services.
DOT	Distributed Object Technologies – Describes a technology that allows Objects' to be distributed about a network, which can have methods invoked on them. CORBA is one such technology. Similar to Remote Procedure Call for functional languages.
FIPA	Foundation for Intelligent Physical Agents – A group that specifies Agent communication standards based upon KQML-style messaging.
FIPA97	FIPA's 1997 specification for Agent interoperability.
FIPA99	FIPA's 1999 specification for Agent interoperability (incomplete).
GSM	Global System for Mobile communications – Standard used by many digital mobile phone service providers as the basis for wireless communications over their network.
GUI	Graphical User Interface.
HTML	Hyper-Text Mark-up Language – A mark-up scheme which provides the ability to embed links to other documents within documents.
HTTP	Hyper-Text Transfer Protocol – A protocol that was originally designed for transmitting HTML documents between hosts, but is now used more widely to transfer any type of data between web-server and web-browser software on different hosts.
IN	Intelligent Network.
JDK	Java Developer Kit.
JNDI	Java Naming and Directory Interface – Java technology that provides a uniform interface to various naming and directory services.

Term	Definition
JNI	Java Native Interface – provides a mechanism for integrating “native” code into Java applications.
JPEG	Joint Picture Experts Group – The group which devised the JPEG image compression algorithms, which can compress natural images with ratio 20:1 without noticeable difference between the original image and compressed image. The compression is “lossy”, i.e. information is lost in the compression process (provided information lost is redundant, no image degradation occurs).
JRE	Java Runtime Environment – A cut down version of the JDK, intended for deployment on end-users machines who do not require the ability to write and compile Java code.
JVM	Java Virtual Machine – The framework within which Java applications are executed, providing a sandbox around the application - preventing direct access to the resources of the host system.
KQML	Knowledge Query Mark-up Language – A grammar for knowledge exchange, which is intended for use in AI Agent systems.
LAP	Local Agent Platform.
MASIF	Mobile Agent System Interoperability Standard – MAT standard defined by the OMG that describes how compliant platforms can interoperate.
MAT	Mobile Agent Technologies – Describes a technology which allows Agent to move from host to host within a network, hence becoming a Mobile Agent
OMG	Object Management Group – The group responsible for specification of CORBA as well as a number of other DOT’s.
ORB	Object Request Broker – The middle-ware within DOT that provides the necessary functionality to enable object location and invocation.
PCS	Personal Communication Support – A specification within the UMTS framework.
PDA	Personal Data Assistant – Generally a pocket size device that keeps track of users calendar and contact details.
PID	Persistent Identifier – Used within the produced system to uniquely identify Agents across life-cycles.
POP3	Post Office Protocol v3 – A protocol that allows email retrieval from a POP3 compliant mail server.
PSTN	Public Switched Telephone Network – The wire-line telephone network.
RAP	Remote Agent Platform.
RMI	Remote Method Invocation – Java based DOT, more efficient than CORBA but not interoperable with other programming languages.
RPC	Remote Procedure Call – Equivalent to calling a function in a program, except that the function call is executed on a remote host, and the result passed back to the originating host.
SA	Service Agent – An Agent with specific knowledge of how to provide a particular service.
SMS	Short Message Service – A service that operates over GSM networks, and allows short messages (160 characters approx.) to be sent to mobile phones and pagers.
SMTP	Simple Mail Transfer Protocol – A protocol that allows email to be sent to a SMTP compliant email server, for forwarding to the destination host.
SP	Service Provider – An entity (i.e. a business) that provides some service to a group of users.
Swing	An extension of the Java AWT API, providing extended functionality and advanced GUI components.
SysAdmin	Short form of System Administrator
TA	Terminal Agent – An Agent that provides the ability to allow a user to interaction with a system of Agents via some mechanism specific to the Agent (e.g. HTML over HTTP, local GUI, WML over HTTP etc…).

Term	Definition
TCP/IP	Transfer Control Protocol / Internet Protocol – The protocol stack which is the basis for Internet communications. It includes connection and connection-less communication capabilities.
UMTS	Universal Mobile Telephone System – The 3rd generation mobile telephone specification.
URL	Uniform Resource Locator – Textual description of the location of a “resource” on the Internet.
VAB	Virtual Address Book – VHE based address book service, as defined in CAMELEON.
VHE	Virtual Home Environment – A system that provides users with coherent information access to a number of services, such as an address book or email account, and which has the ability to be accessed from a number of hosts.
VHEA	VHE Agent – An Agent that embodies a users’ VHE.
WAE	Wireless Application Environment – Part of the WAP specification. Includes a micro-browser that supports WML, WMLS, WTA and various content types.
WAP	Wireless Application Protocol – Specifies communication protocols that are optimised for low-bandwidth, high latency, wireless, unreliable networks.
WML	Wireless Mark-up Language – Equivalent to HTML for WAP mini-browsers. Optimised for describing information to be displayed on small display devices.
WMLS	Wireless Mark-up Language Script – Equivalent to JavaScript for WML.
WTA	Wireless Telephone Application – Part of the WAE. Provides access to telephone functionality from WAP mini-browser.
WWW	World Wide Web – Description given to the collection of inter-linked HTML documents on the Internet.

Introduction

In this modern age the typical view of a user having a single desktop machine and only having access to information services from that machine has become completely inappropriate given the widespread adoption of mobile devices such as cellular phones, pagers, laptop computers and the possibility that a user may have a machine at home as well as work.

At the moment, these devices provide services which, although possibly not completely disjoint, are specific to the type of device being used (e.g. email, web-access, telephony & video-conferencing from a desktop machine or laptop; SMS, fax and voice from a mobile phone), and even where commonality between devices exist, it is not certain that the information available will be coherent across devices (e.g. personal address book on a laptop computer and the phone book on a mobile phone. More subtly, the address book information on a laptop or desktop at home may not be coherent with the information in a users desktop machine at the office). Even the level of service will differ, from the reliable high-bandwidth network connection of a desktop machine to the low-bandwidth unreliable connection of a mobile phone/laptop.

This project aims to explore the use of a concept known as the Virtual Home Environment (VHE) combined with Mobile Agent Technologies (MAT) in order to provide a solution to this problem. The VHE concept provides the notion of an environment that is available to the user no matter the device the user is currently using or where they are, in order to provide uniform service access across a wide range of devices [1]. The only difference between the devices being used are that the user interface and content of the services themselves will be “filtered” appropriately for the capabilities of the device in use [2], [5] & [6] (e.g. An email could be read on a mobile phone, but due to the limited bandwidth & capabilities available, attachments wouldn’t be forwarded to the phone. However, when the user accesses their VHE from a desktop computer, the attachments are available. A more advanced use would be filtering the content of a videoconference – if a user were to join the conference via a mobile-phone, they would only receive audio, however if they were at their desktop/laptop they would receive audio and video).

It is envisaged that through the use of MAT, a VHE would be able to bring the services that the user requires as close as possible, if not onto, the device the user is currently using in order to increase the usability and responsiveness of the services, as well as reducing and distributing network load required by the system [1].

Chapter 1: Specification

This chapter aims to specify the goals of the project, and the nature of the system that produced along with the requirements placed upon its implementation

1.1 Project Goals

The goals of the project are:

- The primary goal of this project is to produce a VHE system for use by multiple users who may interact with their VHE from a number of devices of differing capabilities. The users VHE will aim to provide *uniform service access* and *coherent information* to its user despite these inherent differences, as well as an *appropriate user interface* to the VHE for the device.

- The secondary goal is to explore how the use of MAT enables the VHE to reduce *user interaction overhead* and *network resource utilisation* compared to traditional static implementations of similar systems.

In order to better understand these goals and be able to measure how closely the finished product meets them, here are quantifiable definitions of a number of the terms used to define the goals:

- Uniform service access – This is a measure of the proportion of functionality available across all services that a user is subscribed to, across all of their devices, given that it is impossible for all devices to allow all types of functionality. For a single service *s* (where each service is numbered 0..*num_of_services*-1) on a single device *d*, this can be calculated as follows:

  $$usa(d,s) = \frac{functions\_available(d,s)}{functions\_should\_be\_available(d,s)}$$

  where *functions_available(d,s)* is the number of functions from service *s* available on the given device *d* (from implementation), and *functions_should_be_available(d,s)* is the number of functions from service *s* which device *d* is capable of presenting (theoretical, and based upon capabilities of the device *d*).

  Given this definition of *usa(d,s)*, it is possible to calculate the proportion of uniform service access for all services on a given device *d* (where each device is numbered 0..*num_of_devices*-1):

  $$usa\_dev(d) = \sum_{s=0}^{num\_of\_services-1} usa(d,s)$$

  Given this definition of *usa_dev(d)*, the uniform service access for all devices that a user has can be calculated:

  $$usa\_all\_devs = \sum_{d=0}^{num\_of\_users\_devices-1} usa\_dev(d)$$

- Coherent information – All information available from a given device is always available from all other devices, if they support presentation of that information. The following logic equations given a precise definition.

$$data\_coherent \Leftrightarrow \forall d_1 \in devices[\forall i \in all\_data(d_1)[always\_available(i)]]$$
$$always\_available(i) \Leftrightarrow \forall d_2 \in devices[can\_display(d_2, i) \Rightarrow i \in all\_data(d_2)]$$

data_coherent: true if all information is coherent
devices: set of all user devices
all_data(device): set of all information available from given device
can_display(device, info): true if the given device can display the given information

- Appropriate user interface – A user interface must meet a number of criteria in order to be considered appropriate for a given device, however quantifying these is a difficult problem. Good user interfaces are generally considered to have these properties: user friendly; uncluttered; follow standard look and feel; and be responsive. Since these properties cannot be quantified easily, the best way to measure this is to conduct user trials of the user interface and collect statistics on the views of the users with regards to these properties. The user interface should also provide access to the same range of functionality (providing the user-interface type can deal with it) across devices. This may be possible by describing the user interface in a generic fashion [4].

- User interaction overhead – This is the average amount of time it takes between a user interacting with the user interface on a device, and confirmation that appropriate action is taking place, or the result of the interaction are posted. This should be measured in milliseconds where possible.

- Network resource utilisation – This is primarily a measure of bandwidth usage, which takes into consideration the distance which information must travel. This value is calculated as follows:

  $$u = h \times b$$

  u: network resource utilisation at a given moment, in bytes a second
  h: hops between source and destination host
  b: bandwidth consumed by stream

  The justification for this is that for a stream of information being received at a fixed rate, less network resources are being used if the source host is only a small number of hops away compared with when it is many hops away. In each individual hop the stream must be consuming approximately the same amount of bandwidth, hence overall the amount of network resources being used is proportional to the number of hops and the bisection-bandwidth being used. An alternative method is:

  $$u = h \times s$$

  u: total network resource utilisation, in bytes
  h: hops between source and destination host
  s: total number of bytes transferred

  This highlights how much data has been transmitted overall throughout the network.

The VHE implementation will need to consider a number of issues given the above goals:

- The wide variation in types of devices that a user might use to interact with their VHE, based upon trends of current nomadic users [15] & [5]:
  - Office and/or home PC

  - Laptop[1]
  - PDA[1]
  - WAP enabled mobile phone[1]
  - Standard mobile phone[2]
  - Pager[2]

- The variation in device capabilities/resources to be considered when interacting with the VHE:

  - Network connection (slow/fast, reliable/unreliable)
  - Graphical display (small/large, bitmap graphics/text only)
  - Audio interaction (playback/record)
  - Input devices (keypad/keyboard/mouse/stylus/touch screen)
  - Processing power/memory

  Due to the inherent difficulty of automatically discovering these capabilities, device profiles will need to be considered as a means of discovering this information at runtime [9].

- Demonstration services that the user may[3] be able to subscribe to via the VHE:

  - Email access (send/receive emails)
  - Update notification (inform user of service updates e.g. new emails via SMS)
  - Address book access (add/update/remove contacts)
  - News & local weather information

  Each VHE will have its own profile that describes what services its user is subscribed to as well as configuration information for those services. Services may be able to interact - for example, the "Update Notification" service would monitor other services for events of interest (email from a certain user, local weather update etc).

- The system and services provided will be maintained and controlled by system administrators, who will be permitted to carry out and enforce the following:

  - Add and remove users to the system
  - Add, and remove services available to users via their VHE
  - Provide ability to configure service-provider specific (and non-user visible) service details.

[1] Either these devices may have to be emulated, or constraints imposed in order to simulate their behaviour depending upon available hardware.
[2] These devices may only be able to receive information from the VHE due to limitations of hardware available for the project implementation.
[3] Services are to be considered extensions to the core VHE system, thus the services implemented will be dependant upon the amount of time left once the core system has been implemented.

Emphasis should be placed on the fact that the services available and their configuration can change dynamically, so that the system is long-lived. Restarting the whole system should be considered only when no other course of action is available.

- The efficient use of MAT and Distributed Object Technology (DOT), in order to provide advantages over existing static systems supporting the services [1]:
    - Improved responsiveness of services
    - Decreased network load
    - Decentralisation of services, resulting in improved robustness
    - Disconnected interaction with the system is possible
    - System operates on users behalf whist disconnected, without user supervision
    - Improved utilisation of bandwidth through binary compression of data, rather than existing (often inefficient) encoding [3] (e.g. HTTP and POP3).

1.2 Use Cases

Given the aim of this project, a number of use cases for the VHE systems can be found which highlight the desired functionality of the VHE and all necessary user-interactions with the system. Where necessary, specific scenarios are given as examples to clarify the exact nature of user interactions in specific circumstances. These specific scenarios will not necessarily match the functionality of services within the final system.

Use Case 1	**User connects / logs in to system**
Description	The user activates a device that has the ability to interact with their VHE, and connects or logs into the VHE by a means appropriate for that device. The users VHE receives notification that the user has connected / logged in. It will stop what it is doing and then temporarily sets aside information from the services it is providing which will not be viewable by the device the user is using in order to minimise the amount of redundant data it is carrying. The VHE then moves itself either to a location on the network the device is connected to which minimises the number of hops between itself and the device, or directly onto the device [8]. The user interface is then created by the VHE and presented on the users device.
Actors(s)	Nomadic user
Pre-conditions	System is up and running; Nomadic users VHE is within its home network.
Post-conditions	Nomadic users' VHE is ready to interact with the user; VHE is either on the nomadic users' device or on a node within a hop of it.

Use Case 2	**User disconnects / logs out of system**
Description	The user indicates that they have finished interacting with the VHE. The VHE will then move back into the network, collecting any information it set aside when the user connected / logged in. It will resume any task(s) it was doing previously, as well as starting any new task(s) the user has assigned it.
Actor(s)	Nomadic user
Pre-conditions	Nomadic users' VHE is ready to interact with the user; VHE is either on the nomadic users' device or on a node within a hop of it.
Post-conditions	Nomadic users' VHE is within its home network.

Use Case 3	User subscribes to a service
Description	The user requests a list of available services from the VHE to which they may subscribe from the VHE user interface. The VHE provides a list of all of the services it can support from which the user may choose. If the user selects a service, the VHE will request any necessary information that may be required by the service. If the user provides all necessary information, this service and its configuration details will be added to the VHE's profile.
Actor(s)	Nomadic user
Pre-conditions	Nomadic users' VHE is ready to interact with the user; VHE is either on the nomadic users' device or on a node within a hop of it.
Post-conditions	VHE contains an extra set of service configuration details; Otherwise, as per pre-conditions.
Scenario	The user selects to subscribe to an email service, which provides uniform access to a POP3 based email account. The user is prompted for details of: • POP3 server host name • POP3 username and password • Email address • How often this account should be checked If all of this information is provided, the email service and this information is added to the VHE profile.

Use Case 4	User un-subscribes from a service
Description	The user requests a list of services they are subscribed to in order to un-subscribe from one of them from the VHE user interface. If the user selects to remove a service, the VHE will discard that services' information and the service configuration information contained within its profile immediately. Once the user disconnects / logs out from the VHE, it will also discard any information it set aside at user login related to the service.
Actor(s)	Nomadic user.
Pre-conditions	Nomadic users' VHE is ready to interact with the user; VHE is either on the nomadic users' device or on a node within a hop of it.
Post-conditions	Details of service removed from VHE profile, along with any information service contained; Otherwise, as per pre-conditions.
Scenario	The user selects to un-subscribe from an email service. If the VHE contains un-read emails then a confirmation prompt is displayed. If the user accepts, all emails stored within the VHE are discarded. Upon disconnection / user log out the VHE discards any email attachments it set aside previously.

Use Case 5	User configures a service
Description	The user requests that they wish to configure a service from the VHE interface. The VHE provides a list of subscribed services from which the user can choose. If the user selects a service, a service specific configuration “form” is displayed by the VHE interface. This “form” may include the original configuration information provided by the user when the service was first subscribed, and may include additional configuration options. If the user applies the changes, the VHE adjusts its profile to reflect these changes.
Actor(s)	Nomadic user.
Pre-conditions	Nomadic users’ VHE is ready to interact with the user; VHE is either on the nomadic users’ device or on a node within a hop of it.
Post-conditions	VHE’s profile is adjusted to match configuration changes; Otherwise, as per pre-conditions.
Scenario	The user selects to configure an email service. A form is presented with user-configurable attributes of the email service for the user to modify, which includes: • Email account name • POP3 server host name • POP3 username and password • Email address • Reply-to address • How often this account should be checked • Permission to delete mail on POP3 server • Incoming email filtering rules • Attachment filtering rules (on a per device type setting) • SMTP server for outgoing messages Ordering for service-inbox (e.g. chronological, by subject)

Use Case 6	User views information from a service
Description	The user requests to access the service inbox from the VHE interface. The user is then provided with a list of service inboxes that contain entries. If the user selects an inbox, its contents are listed, possibly in a user specified order (perhaps defined in the service configuration). Entries can be deleted from this list by selecting an appropriate option, or the user may then select an entry to view its contents. Whilst viewing the content, service-specific options may be provided.
Actor(s)	Nomadic user.
Pre-conditions	Nomadic users’ VHE is ready to interact with the user; VHE is either on the nomadic users’ device or on a node within a hop of it.
Post-conditions	Dependant on service used; Otherwise, as per pre-conditions.
Scenario	The user selects an email service-inbox, a list of incoming emails is presented in the VHE interface. Upon selecting one, the email is displayed, and the user is given the option of replying, forwarding, deleting or storing for later viewing, and depending on the device the VHE interface is being presented on, attachments may be available for viewing or saving to the local file-store.

Use Case 7	**User requests that a service-specific task is performed**
Description	The user requests that a task is initiated from the VHE interface. The VHE provides a list of service-specific tasks that can be carried out. If the user selects a task, a service-task specific "form" will be presented in the VHE interface for the user to fill in. If the user completes the "form", its details will be stored in the VHE until the user logs off / disconnects, so that when the VHE re-commences its normal processing it will attempt to carry out the task.
Actor(s)	Nomadic user.
Pre-conditions	Nomadic users' VHE is ready to interact with the user; VHE is either on the nomadic users' device or on a node within a hop of it.
Post-conditions	Dependant on service-specific task selected; Otherwise, as per pre-conditions.
Scenario	The user selects that they wish to send an email. The VHE interface then provides a "form" for the user to complete which contains the usual To, CC and Subject fields for the email, and an area to enter the email itself. When the user has finished writing the email, it is stored within the VHE. When the user disconnects / logs off and the VHE resumes its previous tasks, it will take care of sending the email.

Use Case 8	**User configures device capabilities profiles**
Description	The user requests a list of their devices that the VHE is aware of. The VHE interface provides a list of known devices that the user interacts with the VHE via. From this list, devices can be added, modified or deleted. When adding or modifying device capabilities, a form is displayed that contains a number of configurable options: • Device name • Network connection speed / type • Display type • Audio playback capabilities • Input capabilities • Processing power / memory Once the user has finished, the updated details are stored in the VHE's profile.
Actor(s)	Nomadic user.
Pre-conditions	Nomadic users' VHE is ready to interact with the user; VHE is either on the nomadic users' device or on a node within a hop of it.
Post-conditions	New/updated/removed user device profiles; Otherwise, as per pre-conditions.

Use Case 9	**System administrator adds a user**
Description	The system administrator enters the new users details into the system within their interface to the system, including a default device from which the user can configure their VHE. The system will store the users details and then create a VHE for that user, which the user can then interact with as normal.
Actor(s)	System administrator.
Pre-conditions	System administrator is connected to system;
Post-conditions	A VHE for the user is created, along with default device profile; Otherwise, as per pre-conditions.

Use Case 10	System administrator removes a user
Description	The system administrator selects the user to delete from a list of current users within their interface to the system. The users details will be removed from the system, and their VHE terminated immediately.
Actor(s)	System administrator.
Pre-conditions	System administrator is connected to system;
Post-conditions	User is removed from system records, and their VHE is discarded; Otherwise, as per pre-conditions.

Use Case 11	System administrator adds a service
Description	The system administrator adds a new service to the system by entering the new services name, default configuration details and provider-specific details within their interface to the system. Once the information has been entered, the service is added to the list of available services within the system, and each VHE is notified. Any necessary support programs are activated.
Actor(s)	System administrator.
Pre-conditions	System administrator is connected to system;
Post-conditions	New service support programs are running; VHE's are aware of new service; System is aware of new service; Otherwise, as per pre-conditions.

Use Case 12	System administrator removes a service
Description	The system administrator removes a service by selecting to remove the service within their interface to the system. This causes any active support programs to be terminated, the service removed from the list of services within the system, and notification sent to all VHE's indicating that the service has been removed.
Actor(s)	System administrator.
Pre-conditions	System administrator is connected to system; At least one service is available within the system;
Post-conditions	Service support programs are now stopped; VHE's are no longer aware of service; System is unaware of the service; System administrator is connected to the system;

Use Case 13	System administrator configures provider-specific details
Description	The system administrator selects to modify a services' provider-specific configuration within their interface to the system. The system administrator is presented with a service-specific configuration "form" for them to modify. Once the modifications have been made, the changes are sent to all necessary support programs of the service and possibly each VHE in order to update the VHE's profile with these service configuration changes.
Actor(s)	System administrator.
Pre-conditions	System administrator is connected to system;
Post-conditions	Service support programs are aware of changes; VHE's are aware of changes; Otherwise, as per pre-conditions.
Scenario	The administrator changes the outgoing mail server to be used by the email service. This causes notification to be sent to all back-end email-service programs running in the service-providers network so they send email to this server.

1.3 System Requirements

Given the goals specified above and the use-cases specified, a number of requirements can be captured with regard to the design, implementation and functionality of this project. They are listed below in no particular order:

1. The system will support multiple users, any number of which can interact with the system at a given time [Primary Goal].
2. There will be support for each user to interact with the system via multiple devices. A number of these may be emulated due to project limitations, and only one may be used by a user at one time [Primary Goal].
3. The system will be developed using DOT and MAT wherever appropriate in order to minimise *user interaction overhead* and *network resource utilisation* compared to systems based on client-server solutions [Secondary Goal].
4. Each user has their own VHE Agent that lives within the system, and acts on their behalf. All user interactions with the system will be carried out via their VHE.
5. The VHE will provide access to a number of services. These can be subscribed to, un-subscribed from and interacted with via the VHE.
6. The system will be designed with future extension of services in mind, which could be deployed at run-time without stopping the system.
7. The VHE will provide the ability to configure subscribed services to the users preferences.
8. The VHE will provide a high degree of *uniform service access* across devices [Primary Goal].
9. The VHE will provide *coherent information* within services to users between devices [Primary Goal].
10. The system will persistently store user information. In the event that the system crashes, user information will not be lost.
11. Ensure an *appropriate user interface* is provided for whatever device the user will be interacting with the system via [Primary Goal].
12. The VHE will base decisions on what and how to present information based upon the capabilities of the device and its network connection, which are part of a devices profile.
13. When the user connects to the system, their VHE will migrate to a host close to the users' device / onto the users device, in order to uphold requirement 3 during user interactions.
14. Before migrating to a node closer to the user (requirement 13) the VHE will set aside / filter / adapt any information it is holding which cannot be presented to the user given the capabilities of the device they are interacting with the system using. This will uphold requirement 3 during migration.
15. The user can request that their VHE carry out service-specific tasks, which can be modified or deleted before they are carried out.
16. The VHE will attempt to carry out tasks assigned to it by its user within the system until they are complete.
17. If the VHE is able to migrate directly to the users device, the network connection can be closed until the user is finished interacting with the VHE.
18. The VHE will provide the ability for users to modify the profiles of their devices to meet their needs.
19. The system will allow services to interact with one another in order to accomplish tasks that are impossible to complete alone.
20. The system will allow the system administrator to add, remove and configure users details.
21. The system will allow the system administrator to add, remove and configure services.
22. The system will attempt to keep track of the device the user is currently using, such that services may attempt to contact the user at that device if necessary.

1.4 Report Layout

The rest of the report covers the theory behind the system to be produced, its design, and evaluation of how well the produced system meets the goals of this project.

Chapter 2: Discusses the background theory behind the project, highlighting related work and systems.
Chapter 3: Introduces a high-level design of the system to be produced.
Chapter 4: Describes in detail the system to be produced
Chapter 5: Evaluates how well the created system meets the requirements and goals of the project
Chapter 6: Concludes the report, discussing achievements made, any major problems overcome, where this project fits into the bigger picture, and potential future work.

Chapter 2: Background Information

This chapter provides background information regarding the application domain of the project, which includes details of previous research in the area, applications or platforms that have been considered for use in the implementation of the project and existing applications which attempt to provide a solution to the same or a similar problem.

2.1 Review of Relevant Papers

The following papers reflect ideas and concepts that are crucial to the design and development of the system described in the specification (see Chapter 1: Specification).

2.1.1 Impacts of Mobile Agent Technology on Mobile Communication System Evolution [1]

Authored by Lars Hagen, Markus Breugst and Thomas Magedanz of IKV++ GmbH/Technical University of Berlin.

This paper discusses the potential use of MAT and DOT (Distributed Object Technologies) to provide a VHE for nomadic users using third generation wireless devices, hence some of the ideas presented fit closely within the domain of this project.

The papers starts by highlighting the relevant advantages that mobile agents provide, and highlighting the work that the OMG has carried out on the MASIF standard, as well as describing how MAT and DOT technologies should be used together in order to produce optimal solutions. The view is taken that both technologies have advantages that complement each other in such a way that they overcome their individual disadvantages.

A brief introduction to UMTS (Universal Mobile Telephone System, the third generation wireless technologies referred to by this paper) is given, including a description of how the VHE concept would provide mobile services for the user. The ability to filter incoming calls based upon the capabilities of the device a user is currently using is also described, and is provided by the PCS (Personal Communication Support) within the UMTS specification.

It continues by discussing the advantages of using MAT in conjunction with DOT to implement the VHE and PCS described. This includes notion of a mobile agent that provides VHE services to the user, including the concept that it follows the user (at the network provider level) from place to place providing uniform services to that user either through filtering or perhaps via deployment of appropriate mobile application agents whilst considering the limitations of the wireless connection.

The last half of the paper then focuses in detail as to how the VHE, PCS and other systems required by UMTS would interact in order to provide transparent service provision to the user.

Although the paper focuses on handling real-time audio calls primarily, many of the concepts proposed have been considered when approaching the design for this project.

2.1.2 Multimedia Nomadic Services on Today's Hardware [2]

Authored by Chris Schmandt.

This paper focuses on the use of existing hardware such as pagers, fax machines, mobile phones and the services they provide to allow nomadic access to information services based upon the capabilities of the device the user is currently using. It is important to note that all of the systems described were available and in real-world use when the paper was written, so this paper provides a good insight into services which real-world users make use of.

The paper starts by highlighting the fact that the commonly available type of media within the situations it describes is speech/audio, and from there describes how uniform access is provided for services such as email, calendar, rolodex, and weather updates given the ability of text to speech conversion. A system named Phoneshell based upon this is described which provides access to these services by fax, pager and phone, and had been deployed into three user-bases.

The paper continues by describing how Phoneshell provides these services and the ways in which users can interact with it, including its ability to filter information based upon what device the user is currently using and asynchronous messaging facilities (i.e. receiving email via pager). The notion of actively tracking a user is also briefly discussed in order that Phoneshell can intelligently decide the method of delivery of email or voice messages for example.

Overall this paper highlights the types of services that a nomadic user may require access to, and provides an idea of how uniform service provisioning can be accomplished through a combination of data-filtering and translation (i.e. from text to audio).

2.1.3 Mowgli: Improvements for Internet Applications Using Slow Wireless Links [3]

Authored by Timo Alanko, Markku Kojo, Mika Liljeberg and Kimmo Raatikainen of the University of Helsinki.

This paper focuses on the use of intelligent agents to provide better utilisation of wireless links for nomadic users. It is included here to support the claim that traditional protocols (such as TCP/IP) are too bulky for wireless links, and that using improved protocols & intelligent filtering results in better utilisation of the links.

The paper refers to a prototype system called Mowgli that provides client (mobile host) and server (stationary receiver host) agents that communicate over the wireless link using a specialised protocol. The aim is that the client-side agents are hidden behind a traditional socket API, so that traditional client side applications can work without re-compilation, and that the server-side agents act as proxies to fixed hosts which communicate using TCP/IP.

The Mowgli WWW agent/proxy is given as an example agent, and provides intelligent filtering and compression of information required at the server proxy for the client agent. An example is given that images could be compressed using a lossy algorithm such as JPEG automatically, or that advertisements could be removed from web pages before being sent to the client agent.

In conclusion, the importance of intelligent filtering and translation is highlighted above the network protocol level as ways of optimising service content to the type of device or connection a user is utilising.

2.1.4 Integrating PDA’s into Distributed Systems: 2K and PalmORB [4]

Authored by Manuel Roman, Ashish Singhai, Dulcineia Carvalho, Christopher Hess and Roy H. Campbell, of the University of Illinois.

This paper explains work carried out by the authors to create a seamless, device independent image of a distributed system which offers device independent services to end users. The aim is to integrate the existing 2K distributed system with new software to enable PDA’s (the PalmPilot specifically) to access its functionality.

2K provides a user-centric distributed system based upon CORBA, within which users can dynamically specify what resources and services they want to use. In effect, the network becomes a single pool of resources that is managed by the 2K operating system. Since 2K is CORBA based, a mini-ORB implementation called PalmORB was developed for use with PalmOS that enables access to the features of 2K. PalmORB is a cut down version of Sun's freely available ORB implementation that includes only relevant client functionality in order to keep a small footprint, and was implemented especially to solve this integration problem.

The paper introduces the notion of object oriented user interface definitions that are platform independent. The user interface required for interaction with a service / resource is described in a generic format, and as a result the interface can be presented in a manner suitable for the device it is to be presented on (the paper also highlights that the interface may not even be graphical, but instead be audio based).

Use of proxies to offload computationally intensive work is described in one section as a solution to the limitation of computational power available to PDA's. The idea is that these proxies can be activated "on demand" whenever the client device is not powerful enough to deal with a user request. A concrete example is the PalMPEG proxy software that decodes incoming MPEG data (a computationally intensive task) and optimises the resulting image data for the client device (i.e. 2-bit greyscale, 80x60 image resolution) hence reducing the amount of redundant data sent to it.

Both the use of object orientated interface descriptions and processing proxies are important concepts that are considered in the implementation of this project.

2.1.5 Adapting to Network and Client Variation Using Infrastructure Proxies: Lessons and Perspectives [5]

Authored by Armando Fox, Steven D. Gribble, Yatin Chawathe and Eric A. Brewer of the University of California.

This paper examines the use of proxies as a solution to adapting content from Internet servers to the abilities of the host that it is being delivered to. The concept is that due to the enormous installed infrastructure already present on the Internet, and the base of existent content it would be an impossible task to adapt existing systems to accommodate this. Instead, it is suggested that it is preferable to deploy an incremental solution that can be extended to deal with new content formats, hence the use of proxies to adapt content types of existing servers.

The adaptation process described is based upon data-type distillation, where each content-type is dealt with separately, and unknown data-types are simply passed straight on to the client. Lossy distillation is suggested as the best way to achieve this adaptation, and is applied to image data as well as text documents, where it is suggested that formatting information is stripped to decrease its size. A good example of this is post-script documents, where a large proportion of the content format is formatting information. Numerical evidence is provided to support the notion that the use of these proxies reduces the overall latency of retrieving content from a server on the Internet due to the removal of redundant data over a low-bandwidth connection.

Due to the requirement that these proxies are scalable, it is suggested that clusters of scalable servers be used. As more content adaptation requests arrive at a cluster, more servers within the cluster are used to process the requests. Numerical evidence is provided to support this from the demonstration application developed.

2.1.6 Context-Mediated Behaviour for Intelligent Agents [8]

Authored by R. M. Turner.

The focus of this paper is upon the advantages that context-awareness brings in deciding the behavior of an Agent, including humans. The argument is presented that through knowing the current context within an Agent is operating, it is possible for automatic, reflex like actions to be taken by an Agent given an event within that context.

The paper examines a prototype system, ECHO (Embedded Context-Handling Object), that is currently for use within Autonomous Underwater Vehicles (AUV's), although other Agents could take advantage of it.

ECHO uses definitions of different contexts' an Agent is likely to find itself in, and is able to decide which is most likely to be the current context. It is even able to recognized "novel-contexts", those that are composed of already defined contexts but are not explicitly defined, and decide appropriate behavior for the Agent. Provided that the current context can be determined, the Agent can now be provided with: predictive information about the world around it; adjusted concepts (e.g. nominal depth is different for an AUV in the ocean compared with in a harbor); some degree of automatic event-based behavior (e.g. in the event of a low-power warning, an AUV at sea would attempt to surface, whereas in a harbor it would sink to the bottom).

2.1.7 Agile Application-Aware Adaptation for Mobility [6]

Authored by Noble, Satyanarayanan, Narayanan, Tilton, Flinn and Walker.

This paper details a prototype system called Odyssey that provides the ability to adapt applications to the resources available to a mobile host. The aim is that applications can adapt their use of resources based upon the context of the mobile host, and this is based upon the notion that "adaptation is the key to mobility". The concept of agility is introduced as a means of measuring how well a system adapts to its current context, and a quantifiable definition is given.

The Odyssey system itself is based upon NetBSD, and monitors the mobile systems resources. In the event that the resources of the mobiles system change (e.g. network connection becomes available, bandwidth of wireless links changes), the applications on the mobile system change the way they use these resources. In order to accommodate legacy applications that do not directly interact with the Odyssey system, hooks are placed into the operating system such that when files / URL's of certain data-types are opened, the handling of these content streams is handed over to Odyssey so that lossy compression of these data types can be made based upon available resources (such as network bandwidth, memory available, processing capabilities, battery power).

The concept of fidelity is introduced, which describes the degree to which information at a client machine matches that at the server. Due to the use of lossy compression to improve performance of some file operations over (wireless) network connections, fidelity of the information within the file is lost. It is stressed that performance is often the only consideration when filtering content in this manner, but this paper introduces the idea that the fidelity of the information and the performance of the system should be considered equally when deciding to what degree lossy encoding should be used.

The paper also provides evidence supporting claims that the Odyssey system increased performance of 3 applications by a factor of 5, lending some weight to their arguments.

2.1.8 An Architecture for Exporting Environment Awareness to Mobile Computing Applications [7]

Authored by Girish Welling and B.R. Badrinath.

This paper also stresses the importance of adaptation when using mobile systems. It introduces an event-based architecture that informs applications of changes in available resources, in order that they can adapt their behaviour to suit these changes.

Numerous examples are given:

- Applications that switch to text-only interfaces on battery-low events, allowing the processor to move into a low-power mode due to the less demanding user interface.

- Applications that buffer outgoing messages until a stable network connection event occurs.
- Applications that release memory when low-memory events occur.

The only downside to the system proposed is that the applications have to explicitly be able to deal with these events, otherwise they will not adapt to the changing environment of the mobile system. A number of prototype systems are described, which are based upon existing software that has been modified to listen for resource events and adapt their behaviour appropriately.

2.1.9 Wireless Application Protocol Architecture Specification [15]

Authored by the WAP Forum.

Wireless Application Protocol (WAP) aims to provide a means for connecting wireless data networks with the Internet. The WAP specification itself is based heavily on existing Internet protocols including HTTP, but is optimised for deployment on "mass-market" wireless devices (devices with low-processing power, small power supply, unreliable low bandwidth connections, small displays and limited user input ability) such as mobile phones.

WAP is designed to be used over a number of network and transport types, including SMS. WAP enabled devices communicate with a WAP gateway on their service providers network that provides a bridge between the wireless network and the Internet. The WAP gateway is more that a standard Internet gateway however, since it deals with the encoding and decoding of information as its passes through it. Due to the fact that many Internet protocols are not bit-wise efficient, the WAP gateway encodes data sent to wireless devices in a bit-wise efficient manner. This improves the efficiency of wireless communications with Internet based hosts.

The WAP specification also defines the Wireless Application Environment (WAE) which is based upon WWW and mobile telephony technologies. WAE provides a standard interoperable environment for WAP devices, that allows operators and service providers to produce applications and services that are suitable for deployment on a wide and varied range of devices. WAE includes a micro-browser that understands Wireless Markup Language (WML – equivalent to HTML), Wireless Markup Language Script (WMLS – equivalent to JavaScript), Wireless Telephony Application (WTA) and various content types (based upon standard WWW content types). As a result, the WAE is a powerful and versatile environment that can be used to harness the functionality of a WAP enabled mobile device.

In conclusion, WAP provides an efficient and uniform method for communications between wireless devices and the Internet across a wide-range of potential WAP enabled devices.

2.1.10 Summary

The common themes found within these papers include the following concepts that are considered within this project:

- Use of content adaptation/translation or filtering based upon device capabilities and resources in order to achieve some benefit:
    - Reduced bandwidth usage due to reduced size/quality of content, enabling better support for wireless connections.

  - Reduced computational load on the target device due to adaptation of content to better meet the requirements of the device.

  - An increase range of information that can be presented on a target device (e.g. text to audio conversion or vice-versa, for devices that can only present audio or text respectively).

  This concept is key to the notion of reducing network load, and catering for the abilities of the device information is to be presented on.

- Use of device profiles to determine device capabilities. This is an important concept since automated discovery of device capabilities is infeasible in most circumstances.

- Use of generalised user-interface descriptions. This is a key concept to enable the ability to represent a user-interface on a very wide variety of devices independent of the mechanism used to present that user-interface.

- Use of mobile Agents to enable seamless service provision within a VHE.

However, there are other common concepts presented that are not considered:

- Use of "proxies" to allow seamless integration of traditional system at the socket level – This project is inherently Agent-based by definition, so this type of integration is not required.

2.2 Static Agent Platforms

This section examines a traditional static agent framework with the aim of deciding how suitable it would be to use such a framework in order to implement the project.

These types of platforms are generally designed for use with intelligent Agent implementations that make use of heavy-duty AI in order to carry out their tasks. It is generally perceived that these types of Agents are too cumbersome to be mobile, so the emphasis of the platforms is on the way that agent interactions and knowledge exchange take place takes place over anything else. Adaptation of this type of platform may be possible in order to reach some middle ground in terms of Agent mobility and intelligent agent communications.

2.2.1 FIPA-OS (v1.03) [24]

This platform is available from Nortel Networks as Open Source, and adheres to the FIPA97 specification. FIPA97 compliant Agents communicate with each other using ACL (Agent Communication Language) strings, the grammar of which is based upon KQML (Knowledge Query Markup Language). KQML was originally designed for information exchange between knowledge bases, and as such, the ACL grammar provides a powerful framework for knowledge-based (intelligent) Agent interactions. Messages themselves are considered a one-way communication, delivery is not guaranteed.

FIPA97 specifies the communication protocols that Agents should use at both the ACL level (where a series of ACL messages on the same subject between two Agents is called a conversation, and may follow protocols defined by FIPA) and the message transport level where CORBA is the primary mechanism for message transport (although Agents are allowed to use other non-standard transports provided they support the primary mechanism – although FIPA99 introduces the notion of other standard transports including WAP based transports).

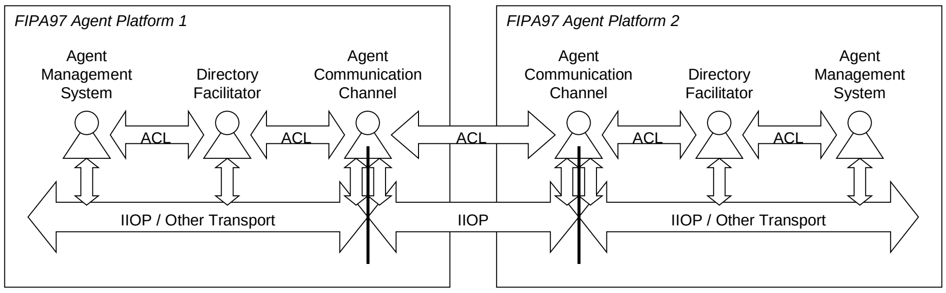


Figure 1 - Two FIPA 97 Platforms

An Agent community within FIPA97 is called an Agent Platform (AP), and each AP contains a number of standard Agents:

- Agent Management System (AMS) – This Agent has overall control over what happens within an AP, and includes white pages listings of Agents on the AP as well as authentication services.

- Directory Facilitator (DF) – This Agent provides a yellow pages directory of Agents advertising their services on the Local AP (LAP), as well as the addresses of DF's on Remote AP's (RAP) which can be searched for remote Agents.

- Agent Communication Channel (ACC) – This Agent acts as a gateway between the LAP and RAP for ACL messages. (Note: In FIPA99 the ACC has become a low-level service entity as opposed to an actual Agent, although there is still debate over whether this is the desired implementation of the ACC).

Figure 1 is a diagram of two AP's along with underlying transport mechanisms.

The platform is written in Java and contains a complex API for Agent developers to use when developing intelligent Agents, however there is no direct support for intelligent Agents over than a framework for Agent message exchange and some conversation management that follows the FIPA standards.

Platform Advantages

- Support for intelligent Agent communications via ACL

- Ability to use any transport mechanism for intra-platform communications

- Inter-operable with other FIPA compliant platforms

Platform Disadvantages

- No direct support for mobile Agents

- No straightforward way to incorporate mobility into the platform

- Use of ACL for non-intelligent communications is overkill due to parsing overhead and complexity of describing ontology's for use within ACL

- Just a FIPA based framework for Agent communications

2.2.2 Summary

Due to the relative in-efficiency of ACL based communications, and that interoperability is not of prime concern of this project, enhancing FIPA-OS to enable mobile Agents was not considered a worth-while use of time and effort, considering the other Agent platforms available (see section 2.3).

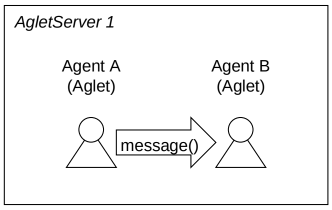


Figure 2 - Local Agent Messaging with Aglets

2.3 Mobile Agent Platforms

This section examines mobile agent frameworks with the aim of deciding how suitable it would be to use such frameworks in order to implement the project.

Mobile Agent platforms are generally designed for use with lightweight, non-AI Agent implementations and provide a lightweight “messaging” facility that acts like RPC and they allow Agents to migrate from one physical host to another. The advantages of being able to move Agents from host to host include reduced network load compared to a static Agent implementation, reduced time to complete the given task should the Agent move closer to a data-source (local messaging can be >1000 times faster than remote messaging) and the ability to deploy new “functionality” to a host without the need for a human to manually install the “functionality”.

2.3.1 Aglets (v1.0.3) [27]

IBM’s Java Aglets framework provides a basis for simple message handling and mobility. It is written using Java 1.1 and has been designed with Java developers in mind given the implementation of the platform. Messages can be sent between Agents, and these consist of a number of key value pairs that can be any Java primitive type including strings. The aim is that this minimises any necessary parsing in Agent implementations since Agents can easily interrogate the messages for necessary information.

Messages can be sent synchronously or asynchronously, and a result is produced by the target Agent and passed back to the invoking Agent. A message method can be invoked on either the target Agent itself if it is local, or its proxy object which is local and forwards invocations to the real Agent via RPC. Once the message has been received and acted upon, the sending Agent receives the result of the action – hence this is closer to direct RPC than other type of Agent messaging. There is also the option of one-way messaging – asynchronously sending a message without requiring the result to be sent back to the originating Agent. Due to the mobility of Agents running on the Aglets system, the underlying platform also automatically and transparently takes care of any transmission of Java byte-code necessary when the destination host for a migrating Agent does not already have the Agent’s code, or the code of any classes it requires.

The underlying platform consists of a number of AgletServer’s that provide an execution environment for the Agents, which are free to move from server to server. There is no logical grouping to the servers (which could be a problem for Agents that are required to dynamically move from server to server without prior knowledge about the configuration of the Aglets platform), the servers can be viewed as a “flat” structure of places that Agents can migrate to.

Figure 2 shows local messaging within a single AgletServer, and Figure 3 highlights remote messaging (although from a developers point of view there is really no difference).

Platform Advantages

- Message passing using Java Objects simplifies message handling
- Automatic migration of Java byte-code with Agents

- JDK 1.1 compliant implementation – stable versions of this JDK are available for a variety of platforms now

Platform Disadvantages

- No logical structure to AgletServers – prior knowledge of platform configuration is required by Agents

- Messages can only contain classes that are present from the target Agent's JVM

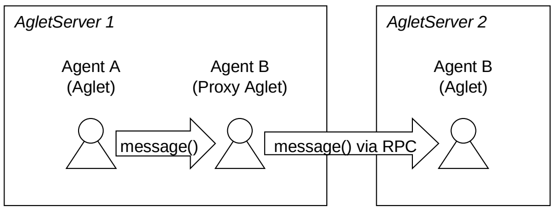


Figure 3 - Remote Agent Messaging with Aglets

2.3.2 Grasshopper (v2.0) [25]

IKV's Grasshopper mobile Agent framework written using Java 2 supports OMG's MASIF (Mobile Agent System Interoperability Standard), which defines an underlying mechanism for interoperable Agent communications and migration between compliant platforms. Rather than taking the approach of message passing for Agent communications, Grasshopper advocates the use of RPC between Agents for "communication" purposes. Once a target Agent (or one of its proxies) has been located and one of its possible interfaces ascertained, methods of the interface can be invoked on the Agent as if they were local Java method calls. This provides a very simple, efficient and developer friendly mechanism for Agent communication.

The Grasshopper platform consists of a number of logical entities which each provide services to Agents within the platform:

- *Place* – These are the entities in which any number of Agents can exist, and a mobile Agent can move between any two places (assuming security restrictions don't apply).

- *Agency* – Contains any number of places and provides services for Agents to use, including the communication service that provides low-level support for the RPC used for Agent messaging. An agency only exists on one physical host.

- *Region* – Contains any number of agencies and provides services for looking up agencies, places and Agents that are contained within the region (i.e. a yellow pages directory for everything encapsulated within the region). A region can span multiple hosts since the agencies it contains may span multiple hosts.

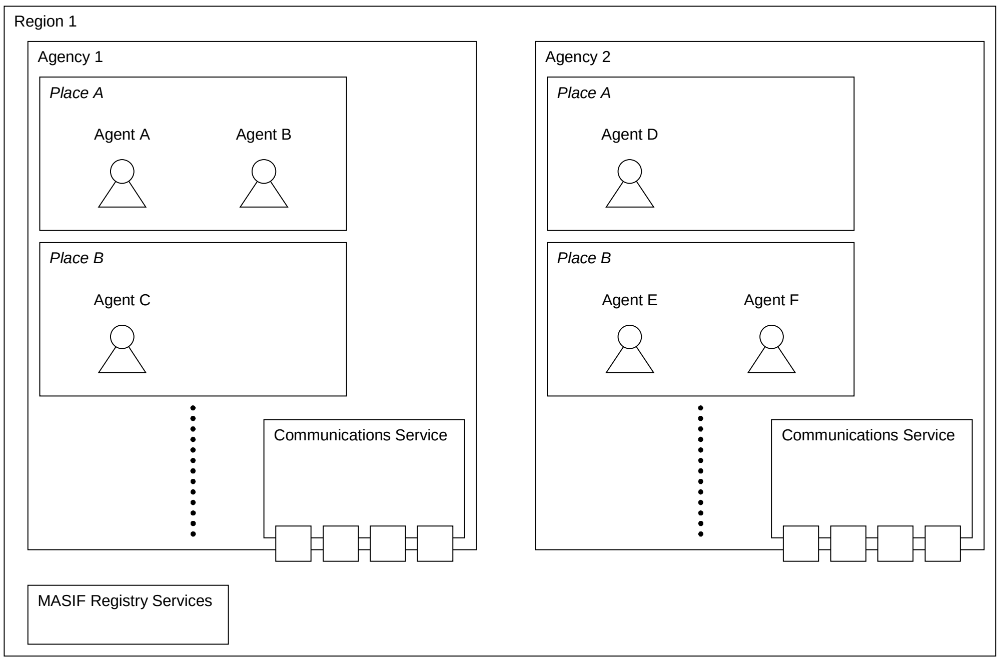


Figure 4 - Grasshopper Platform Logical Structure

Figure 4 indicates how these entities are related in a graphical manner.

As with the Aglets framework, Grasshopper automatically ensures that all necessary Java byte code travels with Agents as they migrate if the target place doesn't have access to the Agents classes.

<u>Platform Advantages</u>

- RPC style method invocation provides Agent communication mechanisms, which are both developer friendly and efficient.
- Method invocation is the same for remote & local Agents
- Logical hierarchy of OMG MASIF specification provides better structure for a dynamic platform since Agents can locate remote places/agencies/registries at run-time.
- Java byte code automatically travels with migrating Agents when necessary.

<u>Platform Disadvantages</u>

- Developed for JDK1.2, this has limited support on platforms other than Windows or Solaris (although Linux support is in the pipeline).

2.3.3 Voyager (v3.11) [26]

Objectstore's Voyager bills itself as having the ability to support mobile Agents, however after close examination it is apparent that it supports little more than the notion of code mobility due to the lack of services available or Agent shell. The code mobility it does posses is rather limited since Java byte-code is not automatically moved from host to host as required, it is down to the developer to take care of this.

Voyager supports a range of communication mechanisms including Java RMI, CORBA, DCOM and its own proprietary mechanism for RPC which are all available via its simplified API which removes some of the intricacies of the particular mechanism compared with other single mechanism implementations, and includes the ability to dynamically assign interfaces to objects at runtime. It also supports JNDI via a simplified interface.

Platform Advantages

- Wide range of simplified communication mechanism support
- Supports JDK1.1 and JDK1.2

Platform Disadvantages

- No "real" support for Agents
- Code mobility is restricted by the fact that Java byte-code is not automatically transported with mobile code when necessary.

2.3.4 Summary

Based upon these reviews, the key abilities of the mobile-Agent platforms described that are used within the project are:

- Use of RPC for communication. This improves the efficiency of Agent communications, and hence is of concern to this project.
- Use of automated code-mobility to simplify implementation of mobile Agents.
- Use of centralised directory of Agents/places within the system. Provides the ability to dynamically locate other Agents and places Agents can inhabit.

2.4 VHE Systems

Although VHE systems are not yet commercially available, there has been research into this field and prototype systems produced.

2.4.1 CAMELEON (European Project) [22]

The CAMELEON (Communication Agents for Mobility Enhancements in a Logical Environment of Open Networks) project aims to investigate the practical advantages of VHE's within IN's (Intelligent Networks) through development and trials of service roaming systems via a VHE as defined by the UMTS specification.

The trial system being produced supports three services, although due to the nature of a VHE, provision is being made for additional services that can easily be deployed along side these:

- Adaptive Profile Manager – Manages user, terminal and service profiles. No other details are given.
- Virtual Address Book – This service provides the user with an address book that is coherent across devices that the user may use, such as home PC, office PC, laptop and PDA. This service makes use of FIPA-OS, and as such the agents used are static – ACL communications are used between multiple instances of a users VAB agent in order to maintain consistency between VAB's.
- Flexible Financial Service – In effect a type of remote banking services. No other details are given.

Through prototyping, user trials and simulations the project aims to provide supporting evidence for the use of agent-based middleware for VHE implementation. Initial evidence has been presented [14] which supports claims that MAT provides a more efficient solution than RPC when used to implement the given services through a combination of real-world test data and mathematical evaluation of the behaviour of the agents.

Despite the lack of a comprehensive practical implementation, much work appears to have been put into research for this project. The CAMELEON Concept Description [9] includes detailed reasoning behind the ways the group has decided to deal with these topics:

- Roaming service access
- Information consistency
- User, terminal (device) and service profiles
- Service registration and lookup
- Security considerations

These topics are relevant to this project, although the implementation details assume a UMTS network, the concepts presented are still valid.

Overall, despite the lack of a fully working product, the ideas raised and evidence supporting the use of MAT in this application domain produced make up for it.

2.4.2 Summary

The CAMELEON project highlights the main areas of concern for a VHE implementation, including highlighting the perceived advantage of using MAT. The concepts of profiles for a number of details within the system provided the starting point for the design of the implemented system, as well as the concept of Terminal Agents to enable the ability to interact with a wide variety of devices.

2.5 Static Systems Supporting Nomadic Users

This section aims to highlight some of the existing static systems that are available which provide solutions to similar problems. Hence, this section concentrates on systems providing services to nomadic users.

In the past few years there has been an increased interest in services which provide users with "on the go" access to their information from any machine they happen to be using, however these services tend to be static web-based systems which are only really suitable for use from a desktop computer or possibly a laptop. Due to the use of web-based interfaces to the services which are inherently centralised, this solution not only places stress on a particular point of a network leading to network congestion, but this requires web-browsers on the machines being used to access these services (which have an inherently large foot print measure in tens of megabytes) and a reasonably quick and reliable network connection due to the nature of the interfaces and the HTTP protocol used to transfer the interface [3].

2.5.1 Yahoo! Web-based Services [23]

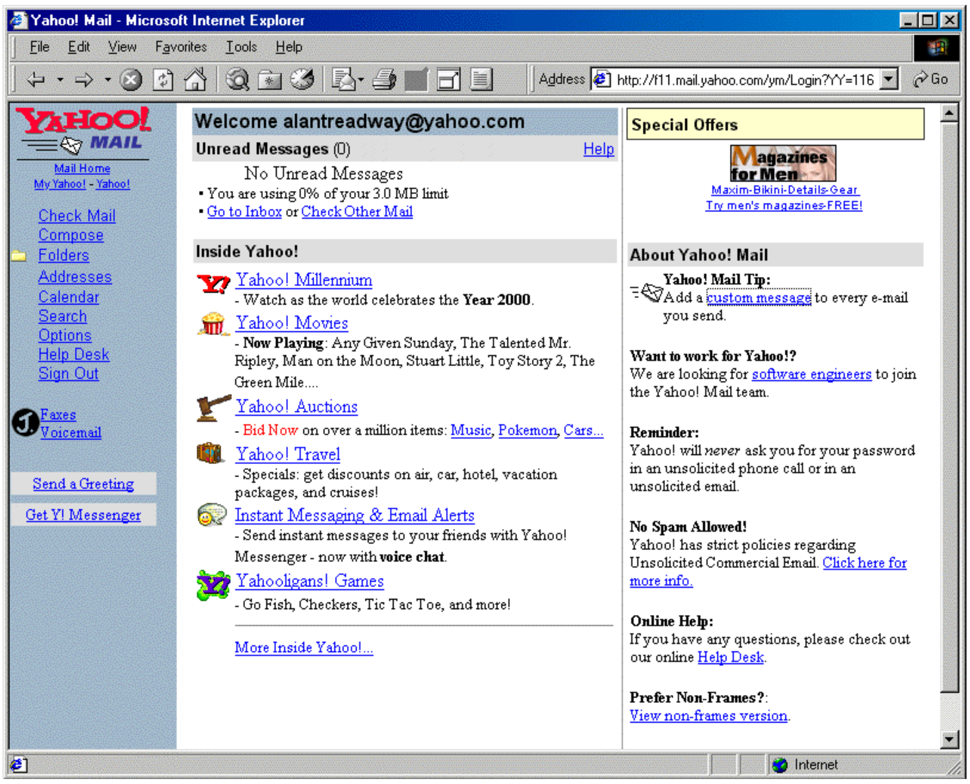
Yahoo! is just one such commercial service which aims to allow nomadic users access to various types of information services. All that the user requires is a machine equipped with a web-browser and they have uniform access to these services. Due to the increasing number of people using more than one device to access their email and other services, this service and others like it have become popular recently. Just some of the services offered include:

- E-mail – The user is given an email address of the form <username>@yahoo.com, the inbox for which is accessible from the URL http://mail.yahoo.com/. Mail sent to this address can be called up via the web-based GUI, browsed, replied to and deleted. The user may also specify other POP3 accounts that can be checked for mail, which is transferred into the users inbox within this service when the user indicates it should do so. This provides an excellent mechanism for keeping track of other email accounts, and it should be noted that the ability to download messages via POP3 from the inbox is also provided so that a permanent copy of email may be kept on the users primary machine. Figure 5 highlights the main Yahoo! Mail page from where the user can change their personal configuration, Figure 6 shows the remote POP3 mail retrieval page, from where the user can move mail into the Yahoo! Mail inbox. Figure 7 highlights the Yahoo! Mail inbox from where mail can be read, deleted, forwarded and replied to.

- Contact/Address book – This ties in with the users email account since it not only provides the usual contact information, but also takes the user to the web-based email editor just by clicking on a contacts name. Figure 8 is a screenshot of this service – as can be seen other contact information can also be included, and there is the possibility of creating distribution lists from a tab at the top of the page.

- Briefcase – Provides a temporary area for holding public or private files that are accessible wherever the user is accessing the service from. Figure 9 is a screenshot of this service.

- News & Weather information – Supplies tailored news and weather information to the user based upon their preferences and default location. Figure 10 is a screenshot of this service.



Welcome alantreadway@yahoo.com

Unread Messages (0) Help

No Unread Messages

- You are using 0% of your 3.0 MB limit
- Go to Inbox or Check Other Mail

Inside Yahoo!

Yahoo! Millennium
- Watch as the world celebrates the **Year 2000**.

Yahoo! Movies
- **Now Playing**: Any Given Sunday, The Talented Mr. Ripley, Man on the Moon, Stuart Little, Toy Story 2, The Green Mile....

Yahoo! Auctions
- Bid Now on over a million items: Music, Pokemon, Cars...

Yahoo! Travel
- Specials: get discounts on air, car, hotel, vacation packages, and cruises!

Instant Messaging & Email Alerts
- Send instant messages to your friends with Yahoo! Messenger - now with **voice chat**.

Yahooligans! Games
- Go Fish, Checkers, Tic Tac Toe, and more!

More Inside Yahoo!...

Special Offers

Magazines for Men
Maxim-Bikini-Details-Gear
Try men's magazines-FREE!

About Yahoo! Mail

Yahoo! Mail Tip:
Add a custom message to every e-mail you send.

Want to work for Yahoo!?
We are looking for software engineers to join the Yahoo! Mail team.

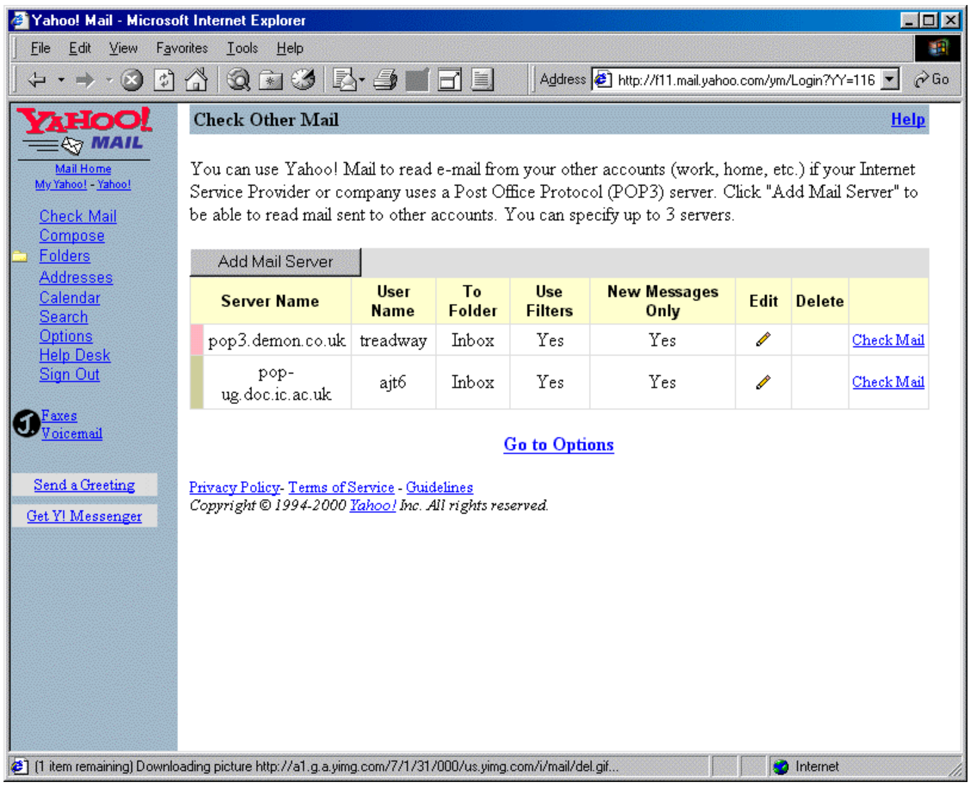
Reminder:
Yahoo! will *never* ask you for your password in an unsolicited phone call or in an unsolicited email.

No Spam Allowed!
Yahoo! has strict policies regarding Unsolicited Commercial Email. Click here for more info.

Online Help:
If you have any questions, please check out our online Help Desk.

Prefer Non-Frames?:
View non-frames version.

Figure 5 - Yahoo! Mail Main Page



Check Other Mail **Help**

You can use Yahoo! Mail to read e-mail from your other accounts (work, home, etc.) if your Internet Service Provider or company uses a Post Office Protocol (POP3) server. Click "Add Mail Server" to be able to read mail sent to other accounts. You can specify up to 3 servers.

Add Mail Server

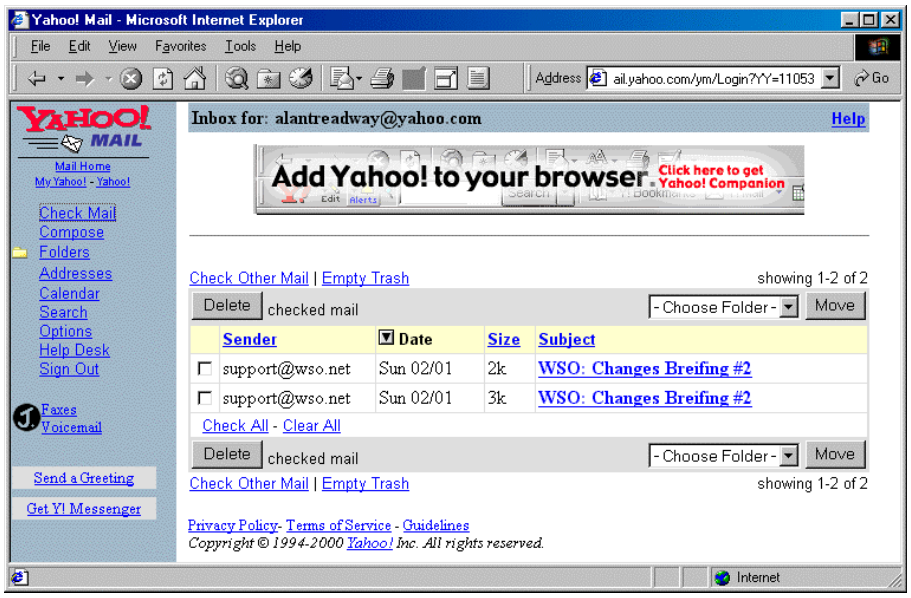
Server Name	User Name	To Folder	Use Filters	New Messages Only	Edit	Delete	
pop3.demon.co.uk	treadway	Inbox	Yes	Yes	✎		Check Mail
pop-ug.doc.ic.ac.uk	ajt6	Inbox	Yes	Yes	✎		Check Mail

Go to Options

Privacy Policy - Terms of Service - Guidelines
Copyright © 1994-2000 Yahoo! Inc. All rights reserved.

(1 item remaining) Downloading picture http://a1.g.a.yimg.com/7/1/31/000/us.yimg.com/i/mail/del.gif...

Figure 6 – Yahoo! Mail Remote Mail Retrieval Page


Yahoo! Mail - Microsoft Internet Explorer

Inbox for: alantreadway@yahoo.com

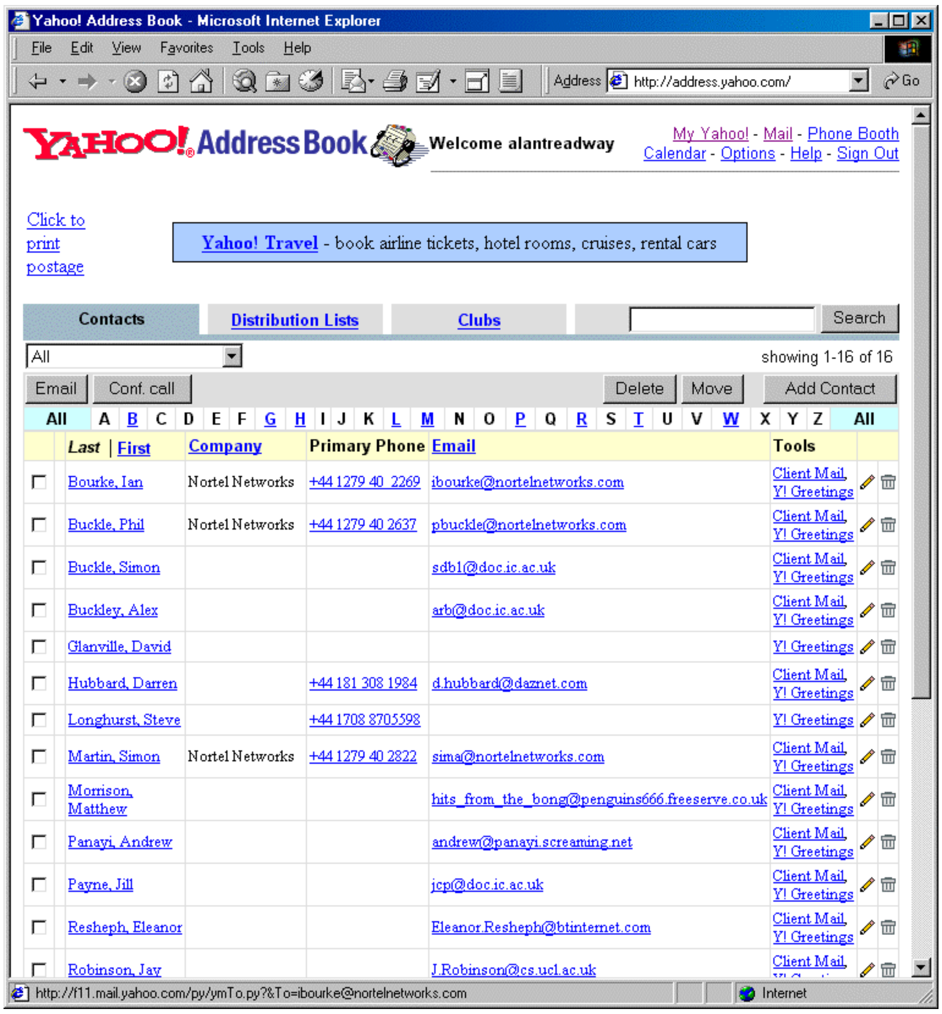
Check Other Mail | Empty Trash — showing 1-2 of 2

	Sender	Date	Size	Subject
☐	support@wso.net	Sun 02/01	2k	WSO: Changes Breifing #2
☐	support@wso.net	Sun 02/01	3k	WSO: Changes Breifing #2

Check All - Clear All

Privacy Policy- Terms of Service - Guidelines
Copyright © 1994-2000 Yahoo! Inc. All rights reserved.

Figure 7 – Yahoo! Mail Inbox


Yahoo! Address Book - Microsoft Internet Explorer

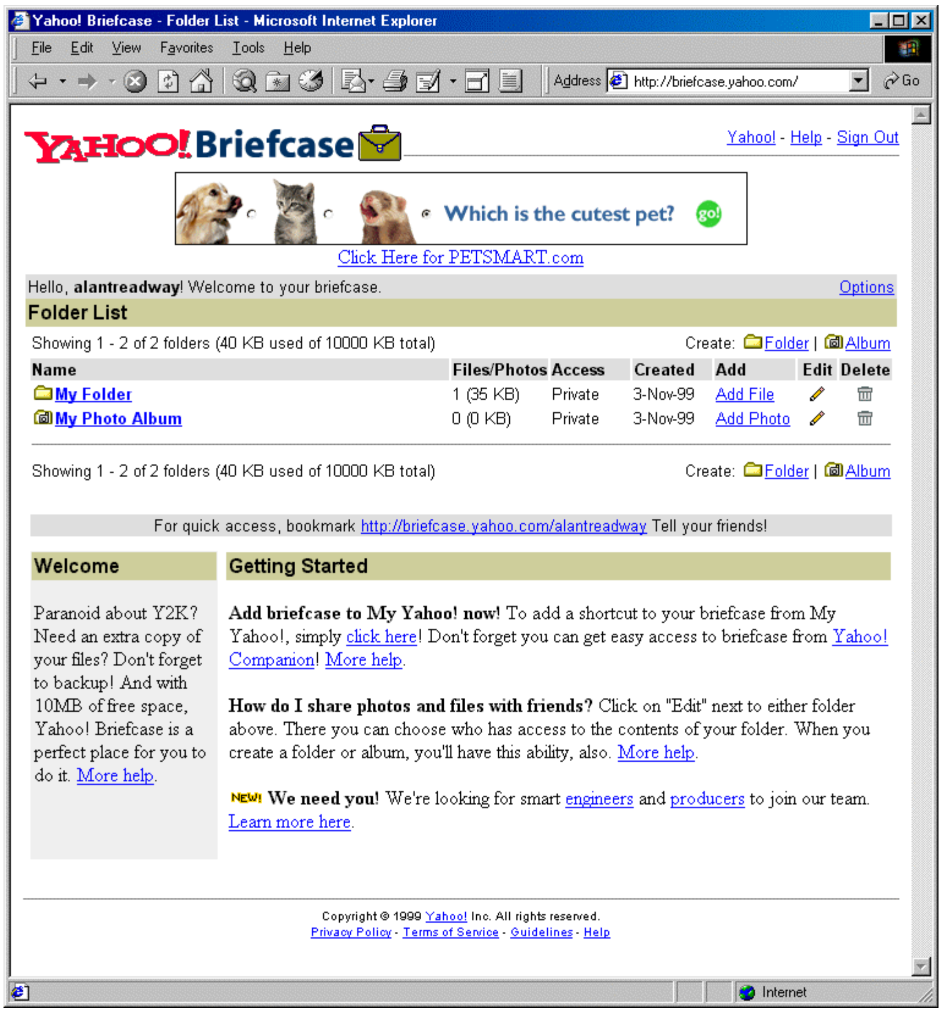
Welcome alantreadway

showing 1-16 of 16

	Last | First	Company	Primary Phone	Email	Tools
☐	Bourke, Ian	Nortel Networks	+44 1279 40 2269	ibourke@nortelnetworks.com	Client Mail, Y! Greetings
☐	Buckle, Phil	Nortel Networks	+44 1279 40 2637	pbuckle@nortelnetworks.com	Client Mail, Y! Greetings
☐	Buckle, Simon			sdb1@doc.ic.ac.uk	Client Mail, Y! Greetings
☐	Buckley, Alex			arb@doc.ic.ac.uk	Client Mail, Y! Greetings
☐	Glanville, David				Y! Greetings
☐	Hubbard, Darren		+44 181 308 1984	d.hubbard@daznet.com	Client Mail, Y! Greetings
☐	Longhurst, Steve		+44 1708 8705598		Y! Greetings
☐	Martin, Simon	Nortel Networks	+44 1279 40 2822	sima@nortelnetworks.com	Client Mail, Y! Greetings
☐	Morrison, Matthew			hits_from_the_bong@penguins666.freeserve.co.uk	Client Mail, Y! Greetings
☐	Panayi, Andrew			andrew@panayi.screaming.net	Client Mail, Y! Greetings
☐	Payne, Jill			jcp@doc.ic.ac.uk	Client Mail, Y! Greetings
☐	Resheph, Eleanor			Eleanor.Resheph@btinternet.com	Client Mail, Y! Greetings
☐	Robinson, Jay			J.Robinson@cs.ucl.ac.uk	Client Mail

http://f11.mail.yahoo.com/py/ymTo.py?&To=ibourke@nortelnetworks.com

Figure 8 - Yahoo! Address Book


Yahoo! Briefcase - Folder List - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://briefcase.yahoo.com/
Go
Yahoo! - Help - Sign Out
Which is the cutest pet?
go!
Internet

Click Here for PETSMART.com

Hello, **alantreadway**! Welcome to your briefcase. Options

Folder List

Showing 1 - 2 of 2 folders (40 KB used of 10000 KB total) Create: Folder | Album

Name	Files/Photos	Access	Created	Add	Edit	Delete
My Folder	1 (35 KB)	Private	3-Nov-99	Add File		
My Photo Album	0 (0 KB)	Private	3-Nov-99	Add Photo		

Showing 1 - 2 of 2 folders (40 KB used of 10000 KB total) Create: Folder | Album

For quick access, bookmark http://briefcase.yahoo.com/alantreadway Tell your friends!

Welcome

Paranoid about Y2K? Need an extra copy of your files? Don't forget to backup! And with 10MB of free space, Yahoo! Briefcase is a perfect place for you to do it. More help.

Getting Started

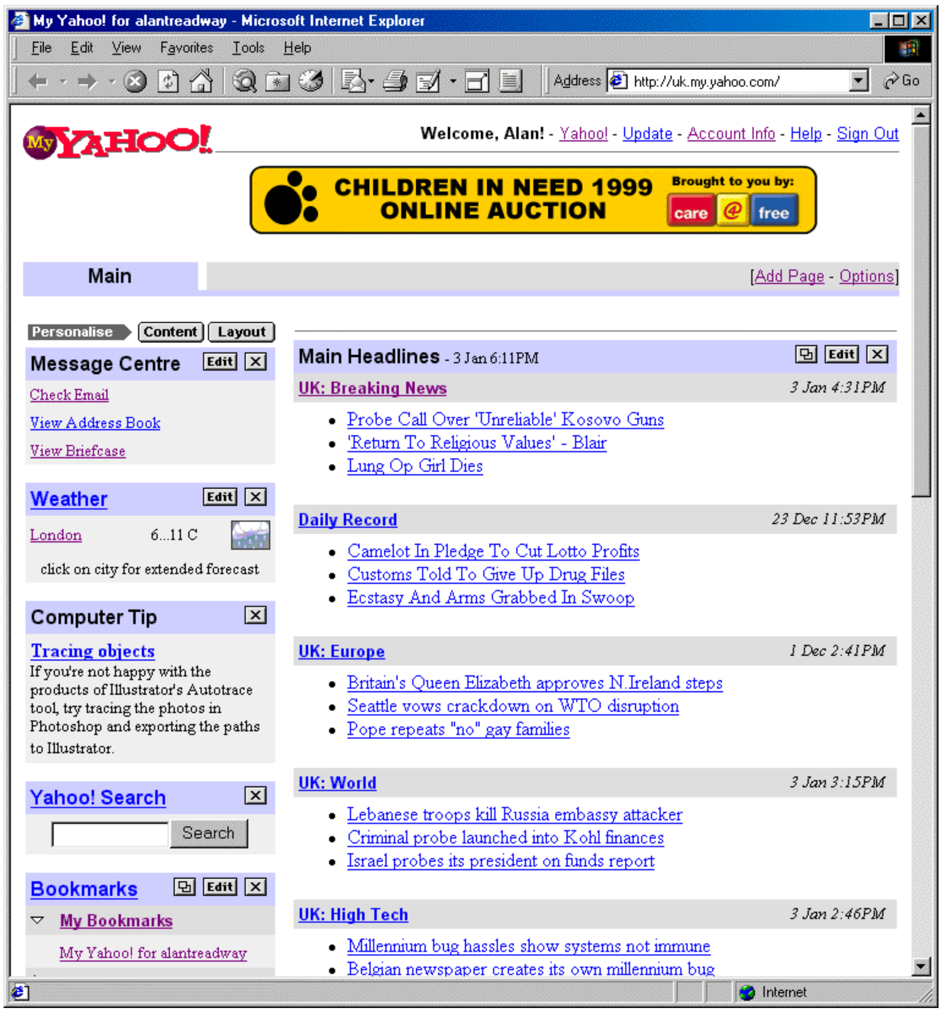
Add briefcase to My Yahoo! now! To add a shortcut to your briefcase from My Yahoo!, simply click here! Don't forget you can get easy access to briefcase from Yahoo! Companion! More help.

How do I share photos and files with friends? Click on "Edit" next to either folder above. There you can choose who has access to the contents of your folder. When you create a folder or album, you'll have this ability, also. More help.

NEW! **We need you!** We're looking for smart engineers and producers to join our team. Learn more here.

Copyright © 1999 Yahoo! Inc. All rights reserved.
Privacy Policy - Terms of Service - Guidelines - Help

Figure 9 - Yahoo! Briefcase


My Yahoo! for alantreadway - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://uk.my.yahoo.com/
Go

Welcome, Alan! - Yahoo! - Update - Account Info - Help - Sign Out

CHILDREN IN NEED 1999 ONLINE AUCTION
Brought to you by: care @ free

Main [Add Page - Options]

Personalise Content Layout

Message Centre Edit
- Check Email
- View Address Book
- View Briefcase

Weather Edit
London 6...11 C
click on city for extended forecast

Computer Tip
Tracing objects
If you're not happy with the products of Illustrator's Autotrace tool, try tracing the photos in Photoshop and exporting the paths to Illustrator.

Yahoo! Search Search

Bookmarks Edit
My Bookmarks
My Yahoo! for alantreadway

Main Headlines - 3 Jan 6:11PM Edit

UK: Breaking News 3 Jan 4:31PM
- Probe Call Over 'Unreliable' Kosovo Guns
- 'Return To Religious Values' - Blair
- Lung Op Girl Dies

Daily Record 23 Dec 11:53PM
- Camelot In Pledge To Cut Lotto Profits
- Customs Told To Give Up Drug Files
- Ecstasy And Arms Grabbed In Swoop

UK: Europe 1 Dec 2:41PM
- Britain's Queen Elizabeth approves N.Ireland steps
- Seattle vows crackdown on WTO disruption
- Pope repeats "no" gay families

UK: World 3 Jan 3:15PM
- Lebanese troops kill Russia embassy attacker
- Criminal probe launched into Kohl finances
- Israel probes its president on funds report

UK: High Tech 3 Jan 2:46PM
- Millennium bug hassles show systems not immune
- Belgian newspaper creates its own millennium bug

Figure 10 - Yahoo! Tailored News & Weather

Advantages

- Services are available from any browser equipped desktop/laptop computer
- Uniform user interface across machines
- Uniform information & services across machines

Disadvantages

- Web-browsers are generally unsuitable for mobile devices with limited screen display, memory and bandwidth
- Is user information (such as that found in the contact/address book) really private/secure?

2.5.2 IPulse [28]

iPulse is a system which aims to overcome the limitations of the individual PSTN, GSM and IP based networks by providing services which allow communications between users on these different types of network. The premise is that the user should not have to worry about underlying network technologies when they wish to communicate with someone else. As a result, the iPulse system provides the following types of communication across the given networks and hardware:

- Text messaging (PC -> PC; PC -> SMS)
- Voice chat (PC -> PC; PC -> PSTN / GSM)
- Text chat (PC -> PC)
- Web conference / web-browser application sharing (PC -> PC)

The only client software available for the system appears to be PC based, and no mention is given of PSTN / GSM to PC voice communication or text messaging support at the moment, although given that the system is designed to be expandable to incorporate more advanced service, these services could be added in the future.

The iPulse system allows the user to specify via profiles how to deal with incoming communications based upon the user's context. For example, when the user is offline (not using their PC) they can specify that text messages be forwarded to them via SMS and voice chats forwarded to their mobile phone / fixed-line phone.

The system relies on the use of clusters of servers that act in essence as gateways between the different types of network. The services provided by the iPulse system operate within these clusters, and each user is assigned a cluster with which they should interact. This provides a "semi-centralised" approach since although there is still a bottleneck within the network at the users' cluster, since there is provision for multiple clusters the user-base of the system can be spread across a number of clusters.

2.5.3 Other Web-based Services

There are many other websites that provide similar services including Microsoft's Hotmail, BT Cellnet's Genie (which offers SMS notification) and many others. These are not mentioned here since they offer similar services to those offered by Yahoo!, although it is important to recognise that there is a broad choice of providers in this area.

2.5.4 Summary

The variety of services available on the Internet provides a good benchmark against which to test the implemented system, as well as a reference point from which to construct services for a VHE. The common uptake of HTML as the content language for these services shows that a form of generalised user-interface (although now potentially very specific in nature, given the last few revisions) can be used across a variety of devices. HTML is heavyweight in nature, provides no scope for adaptation of content and is not suited to wireless devices, especially those with limited memory and processing power. A number of alternative delivery mechanisms suited to the device being used is considered by this project to overcome this limitation imposed by current Internet services.

The iPulse system highlights some useful services that use types of content filtering to achieve a type of unified messaging/telephony system, and the notion of user context to enhance the filtering mechanism based upon what devices are available to a user. If time permitted, some degree of context sensitivity could have been incorporated into the project.

Chapter 3: System Design Overview

This chapter provides an overview of the design for the system that aims to meet the specification provided for this project (see Chapter 1:Specification). The design provided here was the basis for the implementation of the system produced.

Based upon my previous experience of developing Agent-based systems and the requirements of the system, the design for the system is detailed below.

3.1 Agent Overview

The system is decomposed into several Agents, some of which have already been specified within the requirements and use-cases (e.g. the notion of a VHE Agent), some of which have not. A description of each of these Agent types follows below. Figure 11 depicts a possible scenario for these Agents and their interactions.

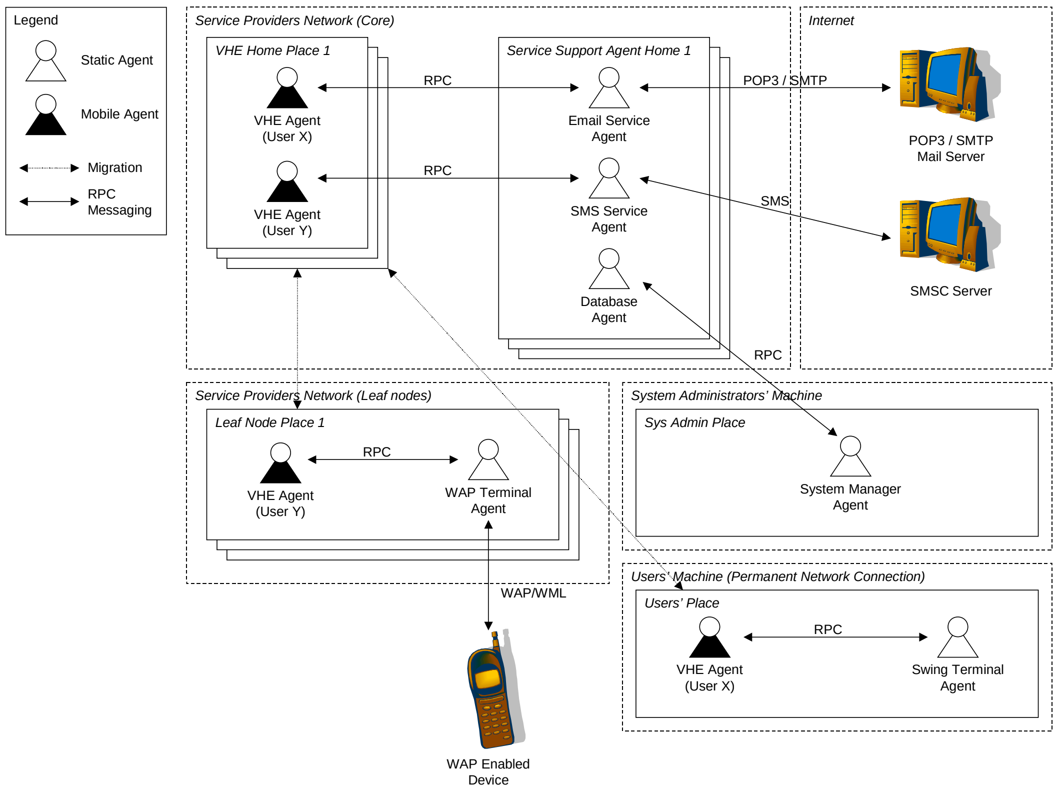


Figure 11 - Agent-Oriented System Overview

3.1.1 Virtual Home Environment Agent (VHEA)

This type of Agent represents the user within the Service Providers' (SP) network. One instance of the VHEA exists per user registered with the system (requirement 4), and carries out tasks on behalf of their user.

The VHEA operates in two main modes:

- **Carrying out tasks on behalf of its user within the network:** In this mode, the VHEA interacts with other Agents within the system including specific types of Service Agent (SA) that provide the means necessary for the VHEA to carry out the tasks it has been assigned by its user (requirement 4). For example, a VHEA might have been instructed to deliver an email via Internet email. The VHEA itself has no notion of how to accomplish this task itself, but there may be an appropriate SA that can be ask to carry out the task. The VHEA can obtain a list of all SA's within the system, and ask one to send the email. The SA always has the right to refuse[4] this task, but in this event, the VHEA can simply ask another SA to carry out the task. In the event that no SA can deal with the request, the VHEA will not attempt to carry out the task for a duration of time, but will eventually try again (requirement 16).

The VHEA interacts with the Database Agent (DBA) to ensure that the information it contains (i.e. profiles, tasks to be carried out etc…) is persistently stored, so that in the event that the VHEA falls over, this information can be easily retrieved (requirement 10).

- **Interacting with its user:** When a user summons their VHEA via a Terminal Agent (TA), the VHEA migrates[5] itself, or a duplicate of itself to the location of the TA. The VHEA filters and adapt the information it contains before migration. This ensures that no bandwidth is wasted during the course of its migration (requirement 14).

  Once the VHEA has successfully migrated to the TA's location, the VHEA presents a series of user interfaces to the user via the TA. These user interfaces are the primary means for the user to interact with the system (requirement 4), and they include:

  - The ability to add/remove/configure subscribed services to/from/within the VHEA (requirements 5 & 7)
  - The ability to add/remove/configure device profiles to/from/within the VHEA (requirement 18).
  - The ability to view service-specific information from subscribed services.
  - The ability to instruct the VHEA to carry out service-specific tasks from subscribed services, and view, modify and delete pending tasks (requirement 15).
  - Only content which is relevant to the device that the user is interacting with the TA via (a combination of the device profile used to filter and adapt information before migration, and the specific TA's ability to present different content types suitable for that device) (requirement 12)

3.1.2 System Manager Agent (SMA)

The System Manager Agent is responsible for both maintaining the system whilst a human system administrator (SysAdmin) is unavailable, providing the SysAdmin with the ability to review the status of the system as well as the ability to modify the parameters of the services within the system.

System maintenance

A SysAdmin cannot be available to monitor the state of the system 24-hours a day, hence it is imperative that the SMA automatically takes care of certain problems whilst the SysAdmin is unavailable.

- Populating the system with Agents – Upon system start-up, or new Agent places being added or removed from the system, the SMA populates the places within the system with appropriate Agents, attempting to balance the load of these Agents across all of the available places. For example if the system had 10,000 users and 500 VHE places, the SMA would ensure that each VHE place would contain at most 20 VHEAgent's each at any one time. If 100 of these places were

[4] This is one of the principal differences between Agents and distributed objects
[5] The act of moving an Agent from one physical or logical place to another

to be shutdown, the SMA would migrate these Agents to other VHE places, ensuring that a maximum of 25 VHE Agents were at each VHE place.

- Restarting Agents/Services if they fall over – Inevitably Agents within the system fall over or are terminated for no apparent reason (e.g. a machine is accidentally reset or is disconnected, or the Agent, or a Service it contains in the case of the VHEAgent, is faulty). The SMA recreates any Agents which appear to have terminated.

System Administrator (SysAdmin) Interactions

The SMA is also the SysAdmin's main way to interact with the system at runtime, and provides a suitable GUI through which they can view the current sate of the system and issue commands to the SMA in order to effect changes in the system.

- Providing an overview of the state of the system – The GUI for the SMA contains a Windows-Explorer like interface that allows the Agents within each place in the region that the SMA resides in to be examined and queried. Places and ServiceAgent's can be added or removed from this GUI to simplify the process of starting the system up. Only Agencies and the initial place containing the SMA need to be manually started by the SysAdmin.

- Adding/removing Services/Users as per SysAdmin instructions (requirements 20 & 21) – The SysAdmin has the ability to add, and remove, services and users via the Explorer-like GUI mentioned above.

- Modifying Service/User details (requirements 20 & 21)– The SysAdmin is able to configure and reconfigure the services within the system via a service GUI available from the Explorer-like GUI. New users can also be added in a similar GUI, as well as the ability to recall and change the users device profiles.

3.1.3 Database Agents (DBA)

This Agent provides a persistent store of objects to other Agents within the system (provides the ability to meet requirement 10). Multiple instances of this Agent can exist within the system, although if so they should each provide access to the different persistent stores of information due to the complexities of distributed databases[6]. Any Agent can create a new data store, and read/write restrictions can be imposed for other Agents requiring access (such as a VHEAgent only having access to persistent data relevant to its user).

3.1.4 Service Agents (SA)

This type of Agent provides the ability to integrate the system with virtually any type of existing service (requirement 6). They act as a type of gateway between the Agents within the system and legacy[7] systems, since they have knowledge of how to interact with these legacy systems to carry out specific tasks. Since they are co-operative[8] Agents, any other Agent can ask them to carry out a task on their behalf, hence any Agent can (indirectly) carry out tasks by delegating the tasks to these Agents.

Multiple instances of specific SA's may exist in the system at any one time, and may be distributed around the system in any fashion. All of these instances provide the same services, so there is nothing to differentiate between them (other than their location and willingness to carry out a task). Thus no single SA falling over results in inability of other Agents to delegate specific tasks, due to the redundancy provided by having multiple instances of different SA's distribute across multiple hosts and even multiple geographic locations.

[6] It was originally intended that each DBA would provide access to a set of distributed databases, but due to the inherent complexities this has been dropped from the design.
[7] In relation to the designed system, any centralised system is considered a "legacy" system.
[8] As opposed to being competitive. All Agents in the system are assumed to be co-operative.

An example of a type of SA is the Email SA (ESA). The ESA has the ability to check POP3 mailboxes on arbitrary hosts, retrieve email from POP3 mailboxes, and send email via SMTP. Thus, other Agents within the system could delegate the task of sending and receiving email to this Agent, without having to know how to accomplish these tasks.

In summary:

- Service access can be distributed across multiple instances of specific service-access Agents, improving the robustness and scalability of the system. VHEA's can locate and arbitrarily select the service agent they wish to interact with based upon the service-type they require.

- VHE migration size is reduced, since all non-VHE-essential service specific code is located within the service-access Agents' (this helps to meet requirement 3)

- System modularity is increased through enforced splitting of functionality. Each Agent can be developed and tested in isolation.

3.1.5 Terminal Agents (TA)

These Agents provide the ability for VHEA's to interact with a wide variety of devices without any specific knowledge of how to interact with those devices (allowing requirement 2 to be met). TA's interact with the user via a specific mechanism such as HTML over HTTP, WML over HTTP or a local GUI. The VHEA interacts with the TA by presenting generic user-interface descriptions to it, and awaiting details of how the user has interacted with those user-interfaces.

The TA's task is to present the user-interfaces to a users device using the specific mechanism it knows about, and collect the results of the users interactions. Thus, the user indirectly interacts with the VHEA through a mechanism suitable for the device being used (requirement 11), whilst allowing the VHEA to only have to consider interactions in terms of generalised UI descriptions. This provides the ability for the same information to be presented across a wide variety of devices since the information the VHEA is providing remains constant, only the mechanism of presenting the information changes (requirement 9).

An example of a TA could be a WAP TA. The user would interact with the TA via a WAP enabled device (such as a mobile phone). Upon connection to the TA, the users' VHEA would provide a generalised user-interface description to the TA, from which the TA would produce a WML page representing that user-interface. Once the user has interacted with that user-interface, details of these interactions are sent back to the TA. The TA then populates the original generalised user-interface description with the results of these interactions, and passes it back to the VHEA. The VHEA then processes the interactions, and can respond with another user-interface, or close the session.

This approach provides a number of advantages:

- VHEA does not require extra code to know how to interact with each of the users' devices, just TA's in general.

- New TA' can be produced whenever new types of device become available, making the range of devices available for interaction extendable.

- System modularity is increased. Each type of TA can be implemented and tested in isolation, and created as necessary at runtime

The following constraints were placed upon the design as a result however:

- In order to maintain requirement 3, the VHE Agent must attempt to migrate to the location of the TA with which it interacts in order to prevent the increased *network resource utilisation* and *user interaction overhead* that would be introduced if VHEA's and TA's communicated via RPC over a network (Local RPC invocations require minimum overhead and consume no network bandwidth)

- A common model for UI's is required such that the VHEA can describe the UI it wants to present to the user to a TA, in a non-device or format specific manner (e.g. the UI may be presented graphically (possibly limited in size in some way), aurally, and user interaction mechanisms may vary (mouse, & keyboard, keyboard, keypad, speech recognition))

3.2 Service Model Overview

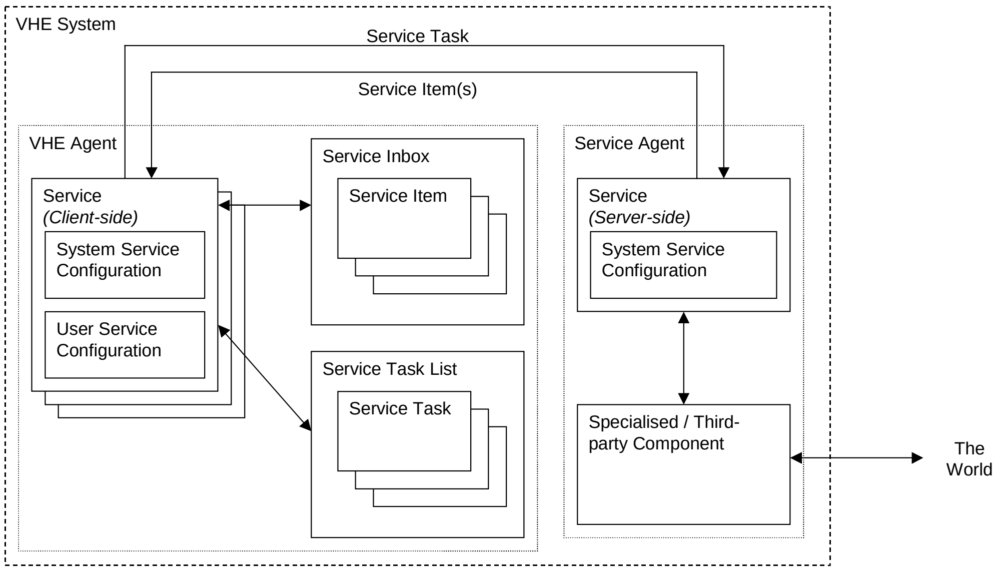


Figure 12 - System Service Component Overview

The system produced is based around the provision of generalised services to users (requirement 5), of which there may be a wide and varied selection. Thus, the design considers the following factors:

- Extendibility of the available services - The ability to add new functionality to the system without re-design of the architecture is considered paramount to the initial design of the architecture (requirement 6). This is achieved by generalising the architecture to support any type of service, as opposed to those originally conceived.

- Well defined classes to encapsulate service information and interfaces between service components - These provide the basic API's for service developers, and help enforce code reuse as well as clearly defining the boundaries between system components. This also aids testing, since common test harness can be produced for different components of the system.

- Minimisation of dynamic code distribution – As new services are brought online the interfaces they expose and new classes their "client" side code requires must be delivered to all interested clients. In a system that could scale to thousands of interested clients, the amount of code that must be distributed becomes significant. In order to minimise the amount of code that must be distributed around the system at runtime (requirement 3), the use of standard interfaces and classes which are already known to clients is emphasised.

Given these factors, Figure 12 highlights how the components of the system fit together. Each VHEA encapsulates an Inbox, containing items of information produced by subscribed service (e.g. these contain collected emails, weather reports, contacts – this remains consistent between devices as per requirement 9), and a Task-list, that contains details of tasks that the VHEA will be carrying out on behalf of the user (e.g. outgoing emails, requirement 15,16).

The VHEA contains several service components (requirement 5), which can interact with the users' Inbox and Task-list. These services can also communicate with any SA's within the system, which provide the functionality to carry out certain tasks and produce certain service-specific generalised user-interface descriptions. In order to complete certain tasks for the user, the services must locate appropriate SA's and interact with them in some fashion (requirement 19). Generally, the SA's take task details and produce some result. Internally the service agent may consist of a number of components that provide access to non-agent systems, and act as a type of gateway between the agent domain and traditional Internet services such as POP3, SMTP, WWW, etc.

3.3 Generalised User Interface Overview

The user needs to interact with their VHEA via a number of different UI's, and as described previously this is achieved through the user of generalised user-interface descriptions.

Through examination of a number of user-interface types (HTML, WML, GUI's etc…), the following components are allowed within a generalised user-interface description:

UI Element	Description
Button	Generally triggers some action to be taken. In the general case when select it causes the VHE to be queried about what UI to display next.
Choice	The UI component generally presents the user with a choice of pre-determined options from which they can select.
Text Area	Presents some text as part of the UI, which may or may not be editable
Check Box	Allows a Boolean selection of on/off true/false.
Attachment	An optional component that may be presented as part of the UI or its contents may be stored by the device displaying the UI (e.g. a graphic, sound file, text, file archive etc which can be written to disk for later viewing).

Thus, the VHEA is constrained to producing user-interface descriptions containing these elements. However, applying this constraint ensures that the maximum number of devices can be interacted from.

3.4 Overview of data managed by the system

Given the design presented above, a number of types of information are required within the system in order to achieve the goals of the project.

3.4.1 User Profiles

A user-profile contains information about a user within the system, including the following:

- Name of the user
- User-name – uniquely identifies the user within the system
- Contact email address
- Device profiles – profiles of devices that the user interacts with the system using

This information defines a user within a system, and hence every VHEA within the system contains a copy of their users profile. The device profiles it contains are used to determine how to filter and adapt content when the VHEA is about to migrate.

The SMA is the only Agent to have direct access to this information, a users VHEA only receives the appropriate user profile from the SMA when it is initially created. Any changes to a users profile therefore have to be accepted by the SMA before they are persistently stored.

Whenever a users' profile changes, the SMA propagates the updated user profile to the users VHEA.

3.4.2 Service Profiles

Each service within the system requires a service profile to define the following information:

- Service id - uniquely identifies the service within the system
- SA class name – The class containing the SA associated with the service
- Service class name – The class containing the VHEA service component for this service
- Service name – Human readable name of the service
- Service description – Brief description of the service
- Service status (activated or inactive)

Both the SMA and VHEA use the service profiles within the system. The SMA uses this information to populate the system with appropriate SA's for active services, whilst the VHEA uses this information to create the appropriate service components within itself (based upon which services the user is subscribed to) and to present details of the service to the user.

The SMA owns this information, and hence can only be changed by the SysAdmin. If a service profile is modified, the SMA propagates the modified service profile to all VHEA's to ensure they always have the most up to date version of the service profile.

3.4.3 Service Configuration

The profile of a service describes a services components, service configuration information describes how a service should go about its job. As such, configuration information can be viewed as two distinct parts, the system service configuration and user service configuration.

System Service Configuration

A system service configuration describes service-specific information that is required throughout all components of a particular service, including SA's and VHEA service components.

The VHEA and SA's require this information, and it is owned by the SMA. The VHEA has copies of this information for all available services, whereas the SA's only have access to the information pertaining to the service they are part of. This allows the SysAdmin to configure a service at run time with service-specific details.

VHEA's and SA's are provided with this information at start-up, and when changes are made to a system service configuration, it is propagated to all VHEA's and SA's that are part of that service.

User Service Configuration

Completely opposite to system service configurations, a user service configuration includes a particular users service specific configuration details for a particular service within the system.

This information is owned by the VHEA, since user service configuration information may contain private user information that no-one else (including the SysAdmin) should have access to.

3.4.4 Service Data

Every service component within a VHEA is provided with two repositories of data, the service inbox and service task-list. Generally, the inbox contains items of information as a result of carrying out some task, or other persistent information (such as emails, contact details, etc…). The task-list generally contains pending service-specific tasks that the VHEA has agreed to do on behalf of the user.

Both of these repositories are owned by a particular VHEA on behalf of their user, and are not shared with any other Agents.

3.5 System Security

Due to the nature of the information to be contained within the system and the nature of mobile Agents, some research into possible security mechanisms has been made.

Due to the sheer complexity of introducing adequate security mechanisms to the system [9], it has been decided that although security is a prime concern for a real-world application of this type, introducing any degree of security is a futile effort due to the limited time scales involved to implement the system.

Any security measures produced within the given timescales could easily be overcome, thus implementing them would be a waste of time. The system design and implementation therefore completely ignores all security considerations. Acknowledgement of the fact that there are many security implications for this project is given however.

In the future, it may be possible that an Agent architecture will exist which provides mechanisms which ensure that all potential security issues related to mobile Agents are dealt with automatically. Some architectures have made steps in this direction (such as the Ajanta architecture [10], [11] & [12]), however these are still many issues.

Chapter 4: Detailed System Design

This chapter describes in detail the design of the system based upon the high-level design overview (see Chapter 3:System Design Overview), the tools chosen to carry out the task, design decision that have been made and any problems that arose during the implementation of the system and their solution.

4.1 Design Decisions

A number of decisions were made before any detailed design for the specified system could be considered. Based upon the background information collected (see Chapter 2: Background Information), the following decisions were made:

- **Grasshopper is to be used as the underlying Agent platform**
  Given requirement 3, the system was build using MAT. Since the platform used influenced many design decisions, this was decided up front. The constraints that this decision placed upon the potential implementation were as follows:

    - Agent implementation language is Java v1.2.2
    - Agent messaging is RPC based

  However, Grasshopper provided the following functionality that simplified Agent implementation:

    - OMG MASIF compliance
    - All aspects of Agent mobility are handled automatically by the platform
    - Logical grouping of Agents in Places & Agencies
    - Agent proxies provide automated message forwarding to mobile Agents
    - API includes Agent yellow pages access for all Agents within a Region via the MASIF Finder interface
    - Lightweight RPC based communications are more efficient than ACL based communications

4.2 Implementation Tools & Techniques

This section highlights the various tools and techniques that were used in order to ensure that the implementation is of a high quality.

4.2.1 Tools

Java Development Kit v1.2.2

Sun's JDK v1.2.2 provides a feature-rich set of API's to enable application and system developers to quickly and efficiently produce systems based upon reusable component. Java also provides the ability to allow platform-independent systems to be produced.

Through my own personal study, courses at Imperial College and use on work placement at Nortel Networks, my personal skill set favours this language above others. I also had experience with the following Java components: AWT; Swing; JNI; Reflection; RMI; CORBA;

These facts, coupled with the now wide-spread adoption of this version of the JDK as the de-facto revision for application development (and hence the release of stable versions of JRE's based on this version of the JDK on several platforms - notably Windows, Linux, Solaris and PalmOS) lead to the decision to use this as the basis for developing the system.

CVS v1.10

This application provides multi-user revision control over a source-code repository, allowing changes to individual classes to be tracked and controlled. The ability to roll-back code to previous revisions is also provided. During my work placement at Nortel Networks, I became very familiar with this system and its use. Thus, I decided that (despite not requiring multi-user access control) this system would be the best to use in order to provide a revision control mechanism for the system source code.

Forte for Java – Community Edition v1.00

Forte (based upon the Netbeans Developer range of IDE's) provides a feature-rich IDE to reduce the amount of redundant tasks an application or system developer must undertake during the course of a project. Some of its features include:

- Code auto-completion (for JDK classes, classes being developed and third party classes)
    - Insertion of missing methods to concrete classes implementing/extending various interfaces/abstract classes.
    - Auto-lookup of method and field names "as you type"
- Two-key JavaDoc lookup (for JDK classes, classes being developed and third party classes)
- Fast Javac compiler (native code version of javac compiler)
- JavaDoc auto completion (based upon method signatures present in code)
- Integrated support for CVS
- Integrated debugger
- Built-in AWT/Swing component builder

Despite never using this IDE previously, the number of features incorporated in this product that could potentially reduce "redundant" development effort and hence increase the amount of time devoted to "non-trivial" development effort overweighed the advantage to be gained by using an already familiar IDE.

4.2.2 Techniques

Coding Standards

In order to reduce the effort involved in maintaining the code base of the system it is important to follow a set of well-defined coding standards in order to improve the readability of code. Having been introduced to a set of standards being used at Nortel Networks during my placement that I found quite intuitive, the standards I followed are based upon their own coding standards.

Class naming

The standard adopted for class naming follows that of Sun's implied standard. All class-name should start with an uppercase letter. Underscore's should not be used to separate words contained in the name of the class, instead the words should be truncated together, and the first letter of each word should be capitalised. All other letters of the class-name should be lowercase.

Examples:

• MyClass	OK
• myClass	WRONG
• SILLYClass	WRONG
• No_Name_Class	WRONG

Class packaging

All classes developed within the project reside within the following packages, in order to provide some logical grouping of classes:

- **vhemat.agent:** Contains the Agent implementations for the core system, along with any classes or sub-packages which are solely associated with these Agents.
- **vhemat.profile:** Contains the interfaces which describe the access methods to classes encapsulating profile information within the system.
- **vhemat.service:** Contains classes related to services, including exception types, data-type interfaces and service agent shell. Sub-packages include service implementations.
- **vhemat.terminal:** Contain classes related to the terminal Agent shell, including exception types. Sub-packages may include terminal Agent implementations and associated classes.
- **vhemat.ui:** All classes related to the description of a generalised user-interface reside within this package.
- **vhemat.util:** This package contains classes which are generally used within the system, and do not fit into one specific package (e.g. ”data-types”).

Method naming

As per Sun’s implied standard, method names should start with a lower-case letter. In the case that the method name is comprised of several words, underscores should not be used to separate words. Instead the words should be truncated together, and the first letter of each work (other than the first) be capitalised. All other letters should be lower-case.

This allows method-names to be distinguished from class-names by looking at the case of the first letter of an identifier.

Examples:

• getSomething()	OK
• GetOthers()	WRONG
• Get_EvenMore()	WRONG

Variable naming

The style used for a variable name varies based upon the type of variable the name belongs to. In all cases, the underscore (_) character should be used to separate words within a variables name (in order to distinguish between variables and class/method names). All variable types except constants should only contain lower-case letters.

Constant values (i.e. prefixed with the “final” modifier) should contain only upper-case letters (as per C/C++ constant definitions) so that they can easily be recognised as constants.

Class variables (i.e. prefixed with “static” modifier but not “final” modifier) should always have two underscore characters pre-pended to their name in order that they can easily be recognised and picked out from a section of source code (due to the “ugly” underscores).

Instance variables (i.e. not prefixed with the “static” or “final” modifiers) should always have just a single underscore character pre-pended to their name in order that they can easily be recognised and picked out from a section of source code (due to the “ugly” underscore).

Local variables should never have any underscores pre-pended to their name.

Given these examples, a variables type can be ascertained just by examining its name.

Examples:

- CONSTANT
- __class_variable
- _instance_variable
- local_variable

Commenting style

Other than the expected commenting within the body of methods, all methods and instance variables of the core implementation have associated JavaDoc style comments explaining their use. This provides a two-fold benefit:

- Provide a mechanism for ensuring that the arguments and return types of all methods are documented, as well as the operation of the methods, and describing what variables are used for.
- JavaDocs can be produced automatically from the source code, allowing third party developers to understand how to use the API’s exposed by the classes within the project, enabling third party extensions to be developed without access to the source code in order to comprehend what it does.

The use of Forte allows JavaDoc style comments to be added quickly and efficiently with its ability to automatically generate “bare-bones” JavaDoc comments, which merely need to be completed a developer.

Modularised Testing

The system to be produced can be split into several smaller components, at the Agent level and the object level. Where possible, these individual components are tested in isolation to ensure the correct semantics of the components are ensured, before integration with the whole system.

In the case of Objects, a `main()` method can be specified which carries out a number of tests on the operation of the class it is defined within which are repeatable, and highlight any problems arising as a result of modifications to the class. Special test-cases have been constructed for bugs which have been found and fixed in order to ensure that modifications to the class that re-introduce these bugs can be spotted quickly.

4.3 System Architecture Detailed Design

This section focuses on the low-level implementation of the system, providing details of how system components interact, class models, Agent interaction models and the interfaces of core components of the system.

4.3.1 Core Data-type Interface Definitions

Before describing the design of the core classes within the system, this section introduces the interfaces of core "data-types" (classes which just act as containers of information) that were described in section 3.4 (Overview of data managed by the system).

ServiceTask

An object implementing this interface describes a particular task that the user has directly or indirectly asked the VHEA to carry out. As described above, the only way to interact with SA's is by passing `ServiceTask` instances to them describing the task to be carried out. Any `ServiceTask` implementation essentially acts as a container for name-value pairs, where the name is a string and the value any Java object. The interface contains the following methods:

- `public void setTaskType( String task_type )`
  Sets the task-type to be that given (this is service specific)

- `public String getTaskType()`
  Returns the task type the instance represents.

- `public String getSender()`
  Returns the persistent identifier (PID) of the Agent making the request for this task to be carried out.

- `public void setSender( String sender )`
  Sets the PID of the Agent making the request for this task to be carried out.

- `public Object get ( String name )`
  Gets the Object associated with a given "parameter" name.

- `public void set ( String name, Object value )`
  Associates the given Object with the given name.

- `public String getInboxName()`[9]
  Returns a description of this `ServiceTask`, which should be displayed in the VHEA tasklist UI (e.g. for an email this might be the subject of the email, for a contact item it might be the name of person whose details are contained within it).

- `public void setInboxName( String name )`[9]
  Sets the description of this `ServiceTask`(see above).

ServiceItem

This interface is similar to the `ServiceTask` interface, except that the implementing class contains name-value pairs pertaining to an item of information produced by a service on behalf of a user (e.g. an email, news report or contact information for a person). The methods contained within a `ServiceItem` include:

- `public String getType()`
  This method returns the type of information contained within the `ServiceItem`

[9] Added after initial design to allow a simple mechanism for discovering a human-readable description for `ServiceTask/ServiceItem`'s.

instance.

- `public void setType( String type )`
  This method sets the type of information that the ServiceItem instance represents.

- `public void set( String name, Object value )`
  Associates the given value with the given name.

- `public Object get( String name )`
  Retrieves the value associated with the given name.

- `public String getInboxName()`[9]
  Returns a description of this `ServiceItem`, which should be displayed in the VHEA tasklist UI (e.g. for an email this might be the subject of the email, for a contact item it might be the name of person whose details are contained within it).

- `public void setInboxName( String name )`[9]
  Sets the description of this `ServiceItem` (see above).

UserServiceConfig & SystemServiceConfig

These interfaces provide access to name-value pairs for service configuration. `UserServiceConfig` & `SystemServiceConfig` both have the same interface, the only difference being `UserServiceConfig` instances are used to describe a user selected configuration for a service, and `SystemServiceConfig` instances are used to describe administrator specified configuration for a service. Generally, these two types of information are mutually exclusive. The definition of these interfaces allows requirements 7 & 21 to be met. The interface they define contains the following methods:

- `public String getServiceID()`
  Indicates the service to which this configuration information belongs via its unique identifier.

- `public void setServiceID( String id )`
  Sets the unique identifier of the service to which this configuration belongs.

- `public void set( String name, Object value )`
  Associates the given configuration option name with the given value.

- `public Object get( String name )`
  Returns the value associated with the given configuration option name.

ServiceProfile

This interface provides methods for describing a service within the system. It relates an internal service identifier with a human readable name and description, details of the `Service` and `ServiceAgent` sub-classes that comprise the service and its status within the system.

The interface contains the following methods:

- `public String getServiceID()`
  Returns the unique identifier for the service that this is describing.

- `public void setServiceID(String id )`
  Sets the unique identifier for the service that this is describing.

- `public String getDescription()`
  Gets a human readable description of the service and all of its abilities for

presentation to users deciding whether to subscribe to this service.

- `public void setDescription(String desc )`
  Sets the human readable description of the service.

- `public String getServiceClassName()`
  Gets the name of the class that implements the Service class half of the service.

- `public void setServiceClassName( String class )`
  Sets the name of the class that implements the Service class half of the service.

- `public String getServiceAgentClassName()`
  Gets the name of the class that implements the ServiceAgent class half of the service.

- `public void setServiceAgentClassName( String class )`
  Sets the name of the class that implements the ServiceAgent class half of the service.

- `public boolean getActive()`
  Indicates if the service is active within the system, or suspended. In the later case no SA's of this service are active within the system, users are not able to subscribe to the service and `Service` implementations are not activated by VHEA's.

- `public void setActive( boolean active )`
  Sets the state of the service.

DeviceProfile

This interface provides methods for describing a device that a user interacts with the system using. It relates a device identifier with a human readable name, and the characteristics of the device (e.g. processing power, network connection type etc…).

The interface defines some constants for convenience:

- `NETWORK_SLOW`
  Used to describe a network connection with low bandwidth (e.g. <128KBit/s), in which case care must be taken over how much information is sent to a device with this property so a large delay is not incurred during information exchange.

- `NETWORK_FAST`
  Used to describe a network connection with high bandwidth (e.g. ≥128Kbits/s), clarity of information is more important than the amount of bandwidth it consumes.

- `NETWORK_RELIABLE`
  Describes a network connection with "low" packet loss (e.g. a wire-line network) – large data transfers will incur little or no retransmission during transit.

- `NETWORK_UNRELIABLE`
  Used to describe a network connection with a potentially "high" packet loss (e.g. a wire-less network) – large data transfers will likely incur multiple retransmissions during transit, utilising much more network resources than for a reliable network.

- `GRAPHICS_SMALL`
  Describes a graphical display with limited dimensions – images quality/size can be reduced without loss of definition on target device.

- `GRAPHICS_LARGE`
  Describes a graphical display with relatively unlimited dimensions - image quality/size should be maintained where possible.

- `AUDIO_PLAYBACK`
  Indicates that a device can playback audio information.

- `INPUT_AUDIO`
  Indicates that a device can record audio.

- `INPUT_TEXT_KEYBOARD`
  Indicates that a device has a keyboard or keypad that can enter alphanumeric data.

- `INPUT_TEXT_KEYPAD`
  Indicates that a device has only a keypad for data entry – this limits the UI to interactions via numbers only.

- `INPUT_POINTER`
  Indicates that the device has either a mouse or stylus based pointing device.

- `PROCESSOR_SLOW`
  Indicates that a devices' processing power is limited – highly compressed data is no use due to high demands on processor to decompress this data.

- `PROCESSOR_FAST`
  Indicates that a devices' processing power is not limited – highly compressed data can be dealt with quickly.

- `MEMORY_SMALL`
  Indicates that a devices' memory space is limited – data should be presented in a form which minimises memory usage (e.g. images are minimum size, or not displayed at all).

- `MEMORY_LARGE`
  Indicates that a devices' memory space is virtually unlimited.

These constants may be logically OR'ed together in order to produce an integer value representing the capabilities of a device. The methods contained within the interface are as follows:

- `public String getDeviceID()`
  Returns the unique id assigned to this device

- `public void setDeviceID( String dev_id )`
  Sets the unique id assigned to this device

- `public int getCapabilities()`
  Returns an integer which contains the logical combination of the constants defined in this interface which describe the capabilities of this device.

- `public void setCapabilities( int capabilities )`
  Sets the capabilities of the device by passing an integer which contains the logical combination of the constants defined in this interface.

UserProfile

This interface provides methods to allow access to information about a user, including the DeviceProfile's of their devices. This provides a common entity to describe a user within the system.

The methods it contains are as follows:

- `public String getUserID()`
  Returns the unique identifier associated with this user.

- `public void setUserID( String user_id )`
  Sets the unique identifier associated with this user.

- `public String getName()`
  Gets the name of the user.

- `public void setName( String name )`
  Sets the name of the user.

- `public String getEmail()`
  Gets the email address of the user.

- `public void setEmail( String email )`
  Sets the email address of the user.

- `public List`[10] `getDeviceProfiles()`
  Gets a list of the device profiles of the devices that the user interacts with the system using (enabling requirement 2).

- `public void setDeviceProfiles( List`[10] `device_profiles )`
  Sets the list of device profiles of devices that the user interacts with the system using (enabling requirement 2).

DataContainer Class

The `DataContainer` class implements all of the interfaces of data-types used within the system (see Figure 13), in order to promote code re-use and permit polymorphism of data-types (e.g. a `ServiceTask` can easily be converted to a `ServiceItem` by simply type-casting it). Each of the methods defined in the various interfaces it implements simply invoke `get()` or `put()` on a `Map` containing all attributes associated with the particular instance of `DataContainer`.

[10] Changed from `DeviceProfile[]` to `List` for implementation convenience.

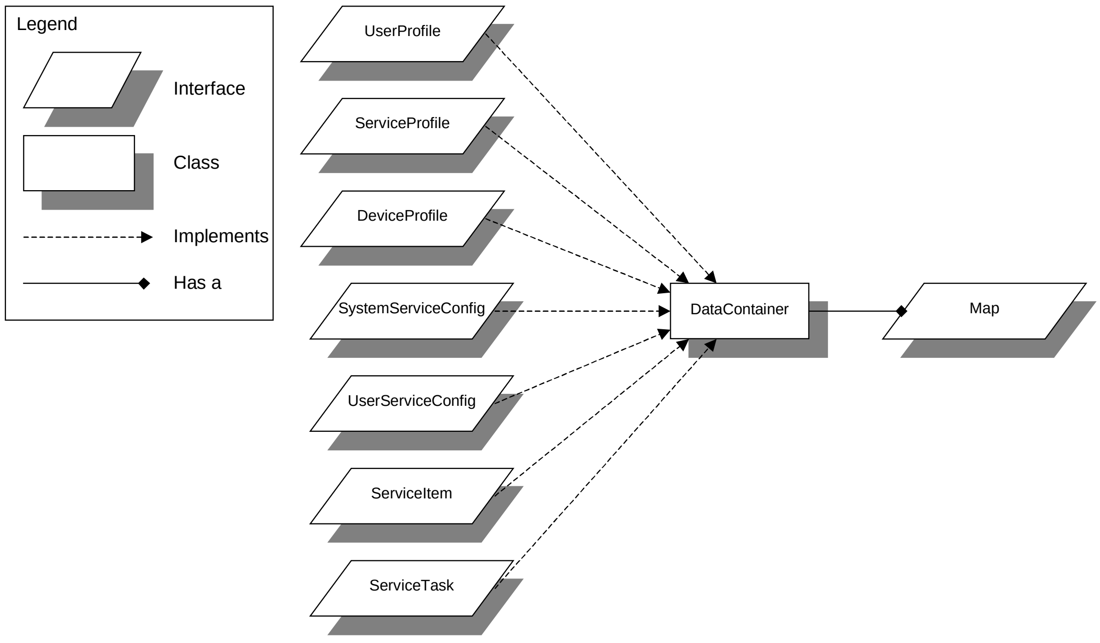


Figure 13 - DataContainer Classes

This class also defines a number of useful methods to duplicate `DataContainer`'s, track changes to `DataContainer`'s, locate a particular `DataContainer` in a `List`, and compare `List`'s of `DataContainer`'s:

- `public DataContainer()`
  Constructs an empty `DataContainer`

- `public DataContainer( DataContainer dc )`
  Creates a duplicate `DataContainer` of the given `DataContainer`.

- `public static List cloneList( List list )`
  Clones the `DataContainer`'s in the given `List`, and returns a `List` containing these new `DataContainer`'s.

-
  ```
  public static void compareLists( String attribute,
                          List current, List old, List not_in_old)
  ```
  Compares two lists containing `DataContainer` objects. The given `attribute` is used to compare the objects in both `List`'s. All objects in the `current list` but not in the `old List` are added to the `not_in_old List`.

-
  ```
  public static DataContainer find( String attribute,
                                 DataContainer dc, List entries )
  ```
  Finds a `DataContainer` in the given `List`, which has the same value for the given `attribute` as the given `DataContainer`.

- `public static DataContainer find( String attribute, Object value, List entries )`
  Finds a `DataContainer` in the given `List`, which has the same value for the `attribute` parameter as the `value` parameter.

- `public boolean isModified()`
  Returns `True` if this `DataContainer` has been modified since creation/last `setModified( false )`.

- `public void setModified( boolean mod )`
  Sets the modification flag of this `DataContainer`.

- `public static List getModified( List in )`
  Returns a `List` of modified `DataContainer`'s in the given `List`.

- `public static void setModified( List entries, boolean modified )`
  Changes the modification flag of all `DataContainer` objects in the given `List` to the value given.

- `public static boolean isModified( List entries )`
  Returns `True` if any `DataContainer` in the given `List` is modified

4.3.2 Extending Grasshopper Agent Communications Capabilities

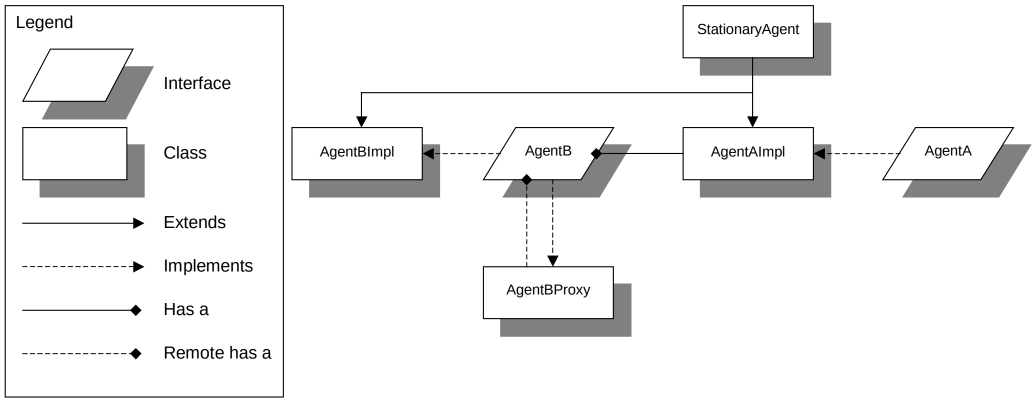


Figure 14 - Grasshopper Proxy Comms. Classes

Grasshopper provides a generalised mechanism for invoking methods on remote Agents using RPC. Figure 14 highlights an example of the classes involved in Grasshoppers' communication model. Each Agent implements a given interface (e.g. `AgentA`, `AgentB`). A proxy class for that interface is then automatically generated by a Grasshopper utility called "StubGen"[11], which implements the interface (The proxy class generated is generally called `interfaceP`, where `interface` is the name of the interface that the Agent implements). In the example, `AgentAImpl` has a reference to an object that has the `AgentB` interface. Agents do not have direct references to one another, so `AgentAImpl` actually has a reference to a `AgentBProxy` instance – other than the act of creating the proxy for communication, this can be treated as a direct reference to the target Agent. The `AgentBProxy` instance has a remote reference to the actual Agent that implements the `AgentB` interface. When the methods the proxy exposes through the `AgentB` interface are invoked, it remotely invokes the same method through the remote objects interface.

Despite RPC being more efficient that ACL communications in the domain of this project, the ability for Agents to refuse or report failure within ACL communications is considered to be one aspect of Agent communications which differentiates them from distributed object communications, and provides a far greater degree of decoupling than simple RPC.

In order to enhance the functionality provided by the Grasshopper system, all methods that Agents expose within the system indicate that they throw `AgentException`'s[12], of which

[11] During testing it was found that this utility did not produce correct code in the proxy objects within Grasshopper BETA 2. This was reported to IKV, along with a solution – see Week 8, 26/4/2000 of Appendix E:Implementation Logbook for more details.
[12] It was originally intended that `ServiceAgent`'s throw sub-classes of `ServiceException`, however once the implementation of the VHEA began, it was clear that a more general mechanism for all Agents to refuse/fail was

there are two primary types, `RefuseException` and `FailureException` (See Figure 15). Thus, the ability for Agents to indicate that they are refusing a request, or that they have failed to carry out the request can be maintained despite using Grasshoppers RPC style communications.

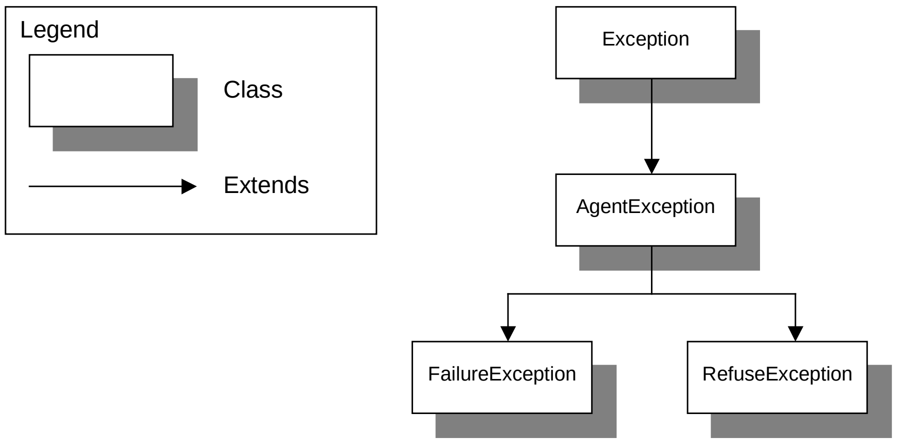


Figure 15 - AgentException Classes

In order to ensure that only `AgentException`'s are propagated back to the requesting Agent, the following code is used within all methods exposed externally by Agents:

```
public Object doTask() throws AgentException
{
    try
    {
          // Carry out the requested task
          …
          …
          …
    }
    catch( AgentException e )
    {
          // Continue throwing AgentException’s, since invoking
          // Agent will be prepared for this
          throw e;
    }
    catch( Throwable t )
    {
          // Catch all possible exceptions, including runtime
          // exceptions, and prevent them propagating back to
          // the invoking Agent.  Throw a FailureException to
          // indicate that a failure occurred.
          throw new FailureException(
                                “Unexpected failure occurred” );
    }
}
```

As can be seen, even if a runtime error occurs the invoking Agent only ever receives an `AgentException` instance, which is required to be caught by the invoking Agent within its code. Thus, even if an Agent fails it is unlikely that any other Agents will fail as a direct result of this, improving the reliability of the system.

required. Hence the `AgentException` class was created. (see Week 11, 29/5/2000 of Appendix E:Implementation Logbook for details).

Persistent Agent Naming (PID's)

Within DAE it is important to be able to uniquely identify Agents across multiple life-cycles. To this end, each Agent within the system is assigned a unique Persistent Identifier (PID) that can be used in order for other Agents to locate and communicate with that Agent across life cycles of the Agents.

Only two Agents within the system have a well-known PID, the SMA (see section 4.5.2 for details) and DBA (see section 4.5.3), since this is the only Agent guaranteed to exist within the platform at any time. The SMA guarantees that each Agent is assigned a unique PID across lifecycles, although certain types of Agents do not require a PID that exists across life-cycles. For example, a VHEA belonging to a particular user requires the same PID every time it is instantiated, but SA's do not because SA's of the same type are identical so SA type is the only discriminator between them.

4.3.3 Service Model Detailed Design

This section aims to highlight the core classes `ServiceAgentImpl` and `Service` that provide the necessary mechanisms to allow a particular service to interact with the system. Due to requirement 6, the classes attempt to minimise the amount of constraints placed on the underlying service implementation.

ServiceAgentImpl (Abstract Class)

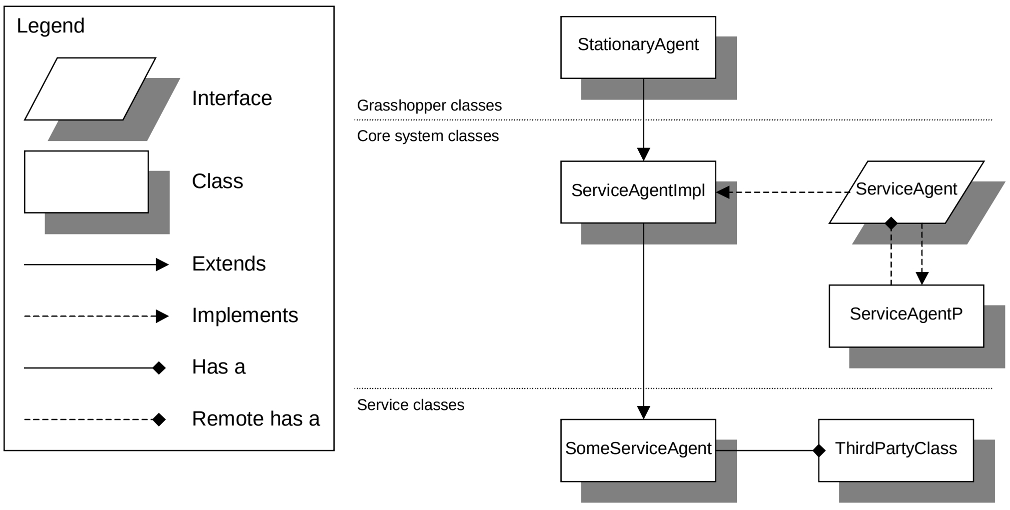


Figure 16 - Service Agent Composition

This class is extended by all Agents offering services to other Agents, and can be considered the SA shell described previously. The inclusion of this 'shell' simplifies SA development since all common code dealing with the handling of requests and Agent related operations is included here, leaving the SA developer to concentrate on the functionality of the service.

There is no direct way of interacting with the Grasshopper system from the service implementation, so it is effectively sheltered from any changes to the Grasshopper Agent system, providing the `ServiceAgentImpl` shell is kept up to date. It is also conceivable that the SA shell could be implemented using a completely different Agent architecture, and the underlying SA implementation would still function correctly. Figure 16 highlights the classes that would make-up a typical SA, with the `ServiceAgentImpl` shell wrapping the SA implementation.

The interface that the `ServiceAgent` class exposes to other Agents includes the following methods:

- `public Object executeTask(ServiceTask st)`
  Invoking this method is the equivalent of asking the Agent to carry out the task specified by the `ServiceTask` instance passed. If an error occurs whilst carrying out this task, an appropriate `AgentException` is thrown. If the task is completed successfully, an Object may be returned to the invoking Agent (the exact type of which is dependant upon the task being carried out).

- `public void setSystemServiceConfig(SystemServiceConfig ssc)`
  Changes the system configuration of this service agent (requirement 21).

The `ServiceAgentImpl` class exposes the following methods to the service implementation for convenience:

- `protected SystemServiceConfig getConfig()`
  This method gets the configuration information for the service, which is cached locally.

-
  ```
  protected Object carryOutTask( String service_type,
                                 ServiceTask st )
  ```
  This method attempts to locate a SA of the specified type to carry out the given task. Any exceptions thrown by the SA carrying out the task is thrown back to the invoking code, and if no SA could be found to carry out the task, a `NoSuitableServerException` is thrown. This allows an implementation to utilise other services within the system (requirement 19).

The service implementation is required to implement the following methods:

- `protected String getServiceType()`
  This method should return a string which describes the type of the service being implemented (e.g. "object-oriented-database"). This value is used when registering the agent with the Grasshopper platform in order to advertise the SA's services.

- `protected void updatedConfig( SystemServiceConfig ssc )`
  Invoked whenever the SA receives an updated `SystemServiceConfig` from the SMA. This is a convenience method to allow SA implementations to change their internal configuration as soon as their configuration details change.

- `protected abstract void onStartup()`
  Invoked when the SA first starts up, and is ready to interact with other Agents. This is always invoked before other Agents can send messages to the SA.

- `protected abstract void onRemoval()`
  Invoked just before the SA is removed from the system. The `ServiceAgentImpl` super-class ensures that there are no active tasks before this method is invoked, so SA implementations simply have to release any resources they have acquired.

The SA implementation is required to contain methods of signature:

```
protected Object performTask_X( ServiceTask )
```

X is the type of task to be carried out. The `ServiceAgentImpl` class utilises Java Reflection to dynamically invoke the method required of this signature to carry tasks other Agent's ask of it by replacing X with the type of task specified within the given `ServiceTask`.

If a `ServiceTask` instance is passed to the `ServiceAgent`, and a method of the required name and signature is not found a `TaskNotSupportedException` is thrown back to the invoking Agent. This exception type extends `RefuseException`, allowing a more specific reason for refusing to carry out the task to be returned to the invoking Agent. Figure 17 highlights the class hierarchy for all service exceptions used to specify more precisely specify the reason for refusal/failure.

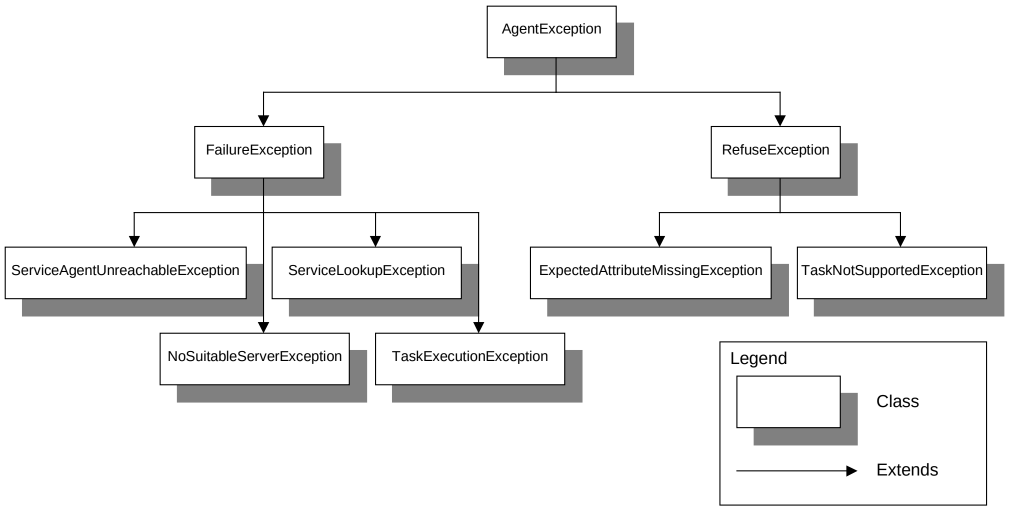

AgentException
FailureException
RefuseException
ServiceAgentUnreachableException
ServiceLookupException
ExpectedAttributeMissingException
TaskNotSupportedException
NoSuitableServerException
TaskExecutionException
Legend
Class
Extends

Figure 17 - Service Exception Classes

The various service exception types are thrown in these conditions:

- `TaskExecutionException`
  Thrown when an unexpected exception occurred during the course of executing the given task.

- `ServiceLookupException`
  Thrown when the mechanism for locating a particular type of SA cannot find any instances of the required SA, this exception is only thrown from the `ServiceAgentImpl` class to sub-classes.

- `NoSuitableServerException`
  Thrown when all possible SA's within the system have thrown `FailureException`'s in response to the request to carry out a given task. This exception is only thrown from the `ServiceAgentImpl` class to sub-classes.

- `ServiceAgentUnreachableException`
  Thrown when no SA's of the required type respond to requests to carry out the task. This exception is only thrown from the `ServiceAgentImpl` class to sub-classes.

- `TaskNotSupportedException`
  Thrown when a SA is requested to carry out a task that it does not known how to perform. The SA implementation class not containing an appropriate `performTask` method for the task type specified by the `ServiceTask` instance passed causes this.

- `ExpectedAttributeMissingException`
  Thrown when the `ServiceTask` passed to a SA is missing some attribute that is expected or required in order to carry out the task it describes.

Service (Abstract Class)

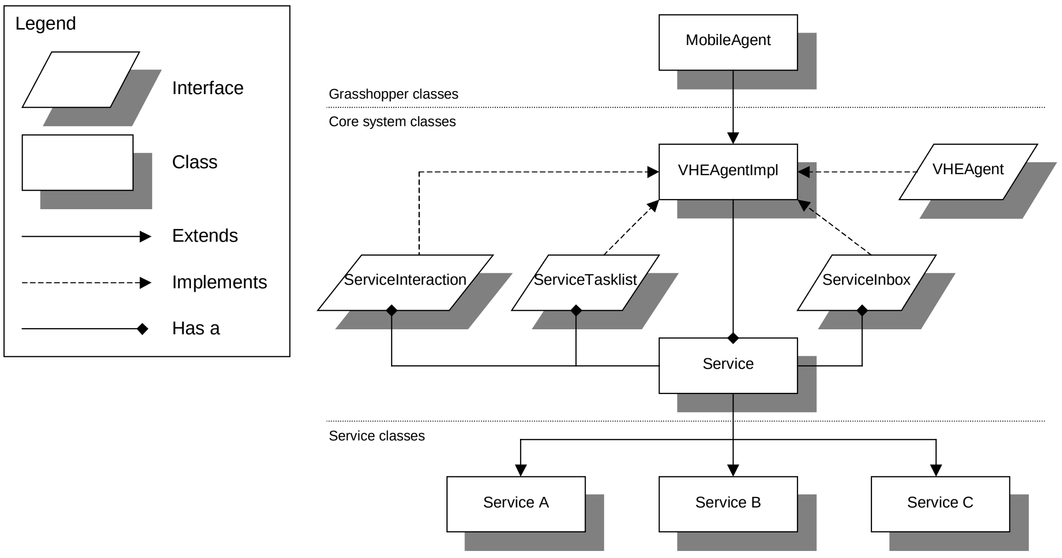


Figure 18 - Service Classes within VHEAgent

This class provides a 'shell' for the client-side / VHEA-side of a particular service (requirement 6). Figure 18 highlights where the `Service` class fits within the VHE component model. A VHEA has an instance of a sub-class of `Service` for each of the services to which the user is subscribed. This class provides access to service specific functionality for the VHE on its public side, and access to VHE functionality from its other-side. The service implementations cannot directly interact with the `VHEAgentImpl` within which it is contained. The reasoning behind this approach is:

- `Service` developers require little or no knowledge of how the system works in order to interact with it from their VHE service-specific code.

- VHE implementation and services are separate components, which can be tested and developed in isolation. Changes to the VHE do not require changes to `Service` implementations, since all direct interactions with the VHE are via the `Service` class, or one of the well-defined interfaces that the `VHEAgentImpl` implements.

When the VHEA is idle within the network, it invokes the `Service` instances it has one at a time, allowing them to carry out tasks requested by the user. These tasks are accessed via the `ServiceTasklist` interface. When tasks have been completed, and a `ServiceItem` is produced representing the result, they are added to the users inbox via the `ServiceInbox` interface. `Service`'s interact with `ServiceAgent`'s or other `Service` instances via the `ServiceInteraction` interface.

When the VHEA is interacting with its user, the `Service`'s it contains may be invoked to provide user-interface descriptions for `ServiceItem's`, `ServiceTask's` or its configuration details.

Each service within the VHEA exposes the following interface to `VHEAgentImpl` class via the `Service` class:

- `public List`[13] `getServiceItems()`
  Returns a list of `ServiceItem` objects belonging to this `Service` within the

[13] The return type was originally `ServiceItem[]`, but this was changed for implementation convenience.

users' inbox.

- `public void removeServiceItem( ServiceItem si )`
  Removes the given `ServiceItem` from the users' inbox, performing any service-specific cleanup at the same time.

- `public List`[14] `getServiceTaskTypes()`
  Returns a list of the types of task types that the user can ask this service to carry out.

- `public String getServiceTaskTypeDescription( String type )`
  Returns a textual description of a particular task type. This is presented to the user by the VHE if the user requires more information about the type of task given.

- `public List`[15] `getServiceTasks()`
  Returns a list of all tasks that are yet to be carried out for the user (requirement 15).

- `public void addServiceTask( ServiceTask st )`
  Adds a new task to the users' task-list for the VHE to carry out (requirement 15).

- `public void removeServiceTask( ServiceTask st )`
  Removes a task from the users' task list (requirement 15).

- `public String getInboxName()`[16]
  Returns the name given to the inbox of this service (e.g. "Inbox" for emails, "Contacts" for contact details), or null if the `Service` does not have an inbox.

- `public String getTasklistName()`[16]
  Returns the name given to the tasklist of this service (e.g. "Outbox" for emails, a contacts' service would not have a tasklist), or null if the `Service` does not have a tasklist.

- `public void prepareFor( DeviceProfile dp )`
  Alerts the service that it should prepare itself to present information on a device with the given profile – all items in the users' inbox and tasks in the users' task list should be filtered so that they only contain information relevant to the given device (requirements 12, 14).

- `public void activate()`
  This is invoked by the VHE to allow a service to carry out tasks it is assigned. The VHEA invokes this method periodically, allowing a chance for the `Service` to divide its tasks into smaller chunks, as well as providing the ability to re-attempt tasks (requirement 16).

- `public void setServiceConfig( SystemServiceConfig sc )`
  Invoked when the system service configuration has been changed (requirement 21).

- `public void setUserConfig( UserServiceConfig uc )`
  Invoked when the user changes their configuration for this service (requirement 7).

[14] The return type was originally `String[]`, but this was changed for implementation convenience.
[15] The return type was originally `ServiceTask[]`, but this was changed for implementation convenience.
[16] Added to allow the VHEA UI to be more specific about what is contained within an inbox/tasklist (see Week 12, 8/6/2000, Appendix E:Implementation Logbook)

- `public UIState`[17] `getServiceItemUI( ServiceItem si )`
  Creates a UI description for the given item from the users' inbox.

- `public UIState`[17] `getServiceTaskUI( ServiceTask st )`
  Creates a UI description for the given task from the users' task list.

- `public UIState`[17] `getUserConfigUI( UserServiceConfig uc )`
  Creates a UI description of the given user configuration.

- `public void init( ServiceInbox si, ServiceTasklist sl, SystemServiceConfig ssc, UserServiceConfig usc, ServiceInteraction parent )`
  Invoked when the Service is first instantiated. The default implementation stores the parameters given within the instance variables `_inbox`, `_tasklist`, `_sys_config`, `_user_config` and `_parent` respectively, and are accessible from sub-classes. These references provide the mechanisms to allow `Service` sub-classes to interact with the VHEA and SA's.

Given the states defined in section 4.5 for the VHEA, it should be noted that the `activate()` and `prepareFor()` methods are only invoked within the AT HOME states. Service implementations should take advantage of this by dynamically instantiating third-party classes rather than directly referencing them at compile-time. Due to the way that Java's class-loading mechanism works [17], any "statically" referenced classes must be loaded into a JVM at the same time as the referring class. In the event that a third-party class is used, and not available within a target JVM during migration, the third-party classes also have to be transferred to the target JVM which may be unnecessary under certain circumstances (e.g. third-party classes only referenced from the `activate()` and `prepareFor()` methods, when migrating nearer to a TA and moving into the INTERACTING WITH USER state). In order to achieve requirement 3, Service implementations should dynamically load third-party classes whenever possible in these methods. (A convenience class, `Reflect`, is provided as part of the architecture to simplify this).

ServiceInbox Interface

This interface defines methods that an object representing the users' inbox implement. An implementing object is effectively a list of `ServiceItem` objects with a method for list searching. The methods exposed are:

- `public List`[13] `getItemsOfType( String type )`
  Returns a list of all ServiceItem's contained within the inbox.

- `public void addItem( ServiceItem si )`
  Adds the given ServiceItem to the list.

- `public void removeItem( ServiceItem si )`
  Removes the given ServiceItem from the list.

ServiceTasklist Interface

This interface defines methods that an object representing the users' task list implement. It is very similar to the interface for `ServiceInbox`, since it is effectively a list of `ServiceTask` objects with a method for list searching. The methods exposed are:

[17] Return type changed to `UIState` from `UserInterface` in original design to allow a much richer set of features to be incorporated into service UI's. This also removed the need for the `getServiceTaskFromUI()`, `getServiceItemFromUI()` and `getServiceConfigFromUI()` methods originally intended.

- `public List`[15] `getTasksOfType( String type )`
  Returns a list of all ServiceTask's contained within the task list (requirement 15).

- `public void addTask( ServiceTask st )`
  Adds the given ServiceTask to the list (requirement 15).

- `public void removeTask( ServiceTask st )`
  Removes the given ServiceTask from the list (requirement 15).

ServiceInteraction Interface[18]

This interface defines a number of methods that provide mechanisms to interact with the rest of the Service's and ServiceAgent's within the system. The methods exposed are:

- `public Object carryOutTask( String service_type, ServiceTask st )`
  Similar to the `carryOutTask()` method of the `ServiceAgentImpl` shell, attempts to locate a SA to carry out the given task.

- `public Object carryOutTaskSpecificSA( String agent_pid, ServiceTask st )`
  Similar to the `carryOutTask()` method described above, attempts to locate the specified SA to carry out the given task. This is useful in situations where several requests have to be made to the same SA due to the nature of the task.

- `public List getActiveServices()`
  Returns a List containing all `Service` instances that are active within the VHEA, allowing `Service's` to interact within the VHEA (requirement 19).

- `public void updatedServiceInbox()`
  Invoked when the contents of the `ServiceInbox` have been updated and should be persistently stored immediately.

- `public void updatedServiceTasklist()`
  Invoked when the contents of the `ServiceTasklist` have been updated and should be persistently stored immediately.

- `public void updatedUserServiceConfig()`
  Invoked when the contents of the `UserServiceConfig` belonging to the `Service` have been updated and should be persistently stored immediately.

4.3.4 Generalised User Interface

This section looks at the classes used to describe a generalised user-interface within the system, maintain its state, and the `TerminalAgentImpl` shell that is a basic building block for integration of "legacy" interaction mechanisms into the system.

User Interface Object Model

One method of generalising a user interface such that its description is independent of its presentation is by producing an object-oriented description [4]. Based upon this notion, a `UserInterface` class has been devised to encapsulate an object-oriented description of a UI. The concrete representation of a `UserInterface` object would be a window, a web page, a WML card or a menu in a voice/touch-tone menu system.

[18] Replaces methods originally defined with `protected` visibility within the `Service` class. This removes the need for a direct reference to the `VHEAgentImpl` class from the `Service` class (i.e. via a well-defined interface).

Each component within the UI requires a “display” name to present adjacent to each component (in time or space), and can be changeable or non-changeable (for display/presentation only). Given this functionality and the types of UI component described in section 3.3, Figure 19 is the object-model used to describe UI’s.

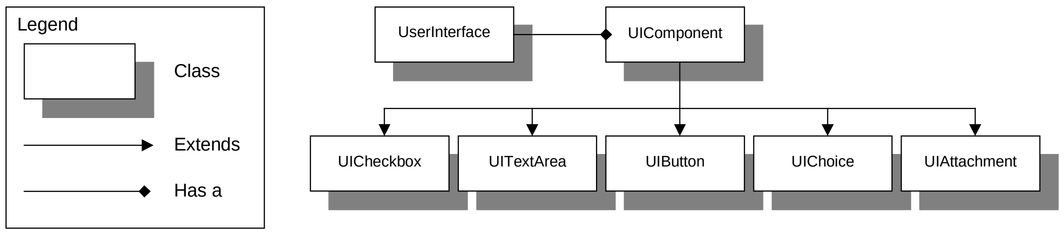


Figure 19 - Object-Oriented UI Model

The order in which `UIComponent`’s are added to the `UserInterface` object determines the order they are presented to the user, but due to the generalised nature of the UI description, no other guarantees about the way that the UI will be present can be made.

`UIAttachment` objects are not guaranteed to be displayed, but if the device the UI is presented is capable of viewing / storing this information the option will be available – it is contemplated that this will rarely be the case since the information being presented is filtered by the VHEA for the device the user is using.

UIState Class & UIStateListener Interface

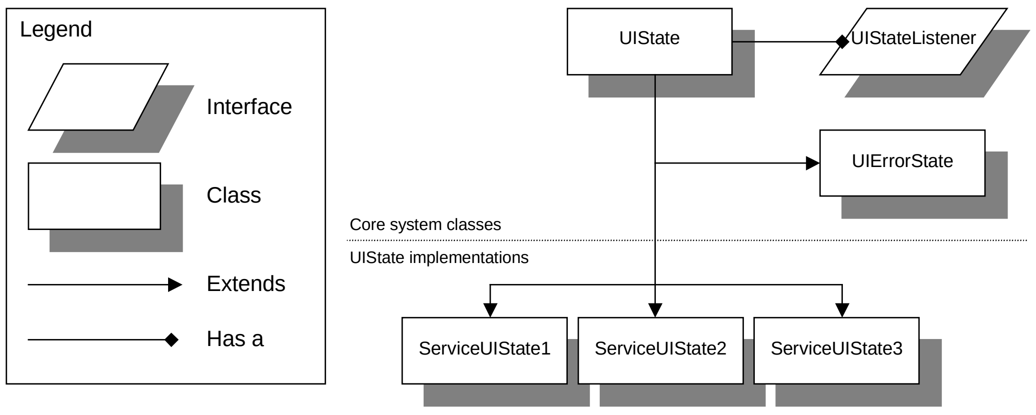


Figure 20 - UIState Classes

The aim of these classes is to provide a generalised mechanism to keep track of the current state of a UI when using the `UserInterface` and `UIComponent` classes.

Figure 20 highlights how these classes interact. The model used it based upon the “State Pattern” from OMT. The `UIState` class has a reference to a `UIStateListener` instance, which is informed whenever the current `UIState` changes. The `UIState` object defines a `getInitialUI()` method that returns the `UserInterface` for the current `UIState`, and a `getNextUI()` method which takes a `UserInterface` object that has been interacted with, and produces the next `UserInterface` to be presented.

Generally, when `getNextUI()` is invoked the current `UIState` is changed and the `getInitialUI()` of the new `UIState` is invoked, and its result returned. `UIState` provides a method `changeState()` to `UIState` sub-classes to allow a change in state to be propagated to the `UIStateListener`. When invoked, this method populates any instance variables of the new state necessary (i.e. a reference to the `UIStateListener` is added, and a reference to the previous `UIState` is added), invokes the getInitialUI() of the new UIState, and returns its results. A getPreviousUIState() method is also provided so that the previous UIState instance can be returned to. Thus knowledge of the current state of a generalised user-interface can be easily maintained within the system.

As an example, the following code segment for two UIState sub-classes highlights how the state of a simple UI where one UI is displayed after the other in turn can be maintained:

```
public class UIStateA extends UIState
{
    public UserInterface getInitialUI()
    {
          // Code to produce initial UI
          UserInterface ui = new UserInterface();
          ………
          ………
          ………
          return ui;
    }

    public UserInterface getNextUI( UserInterface ui )
    {
          // Read info back from the UI
          ………
          ………
          ………

          // Change to the next state
          return changeState( new UIStateB() );
    }
}

public class UIStateB extends UIState
{
    public UserInterface getInitialUI()
    {
          // Code to produce initial UI
          UserInterface ui = new UserInterface();
          ………
          ………
          ………
          return ui;
    }

    public UserInterface getNextUI( UserInterface ui )
    {
          // Read info back from the UI
          ………
          ………
          ………

          // Change to the previous state
          return changeState( getPreviousUIState() );
    }
}
```

Terminal Agent (TA)

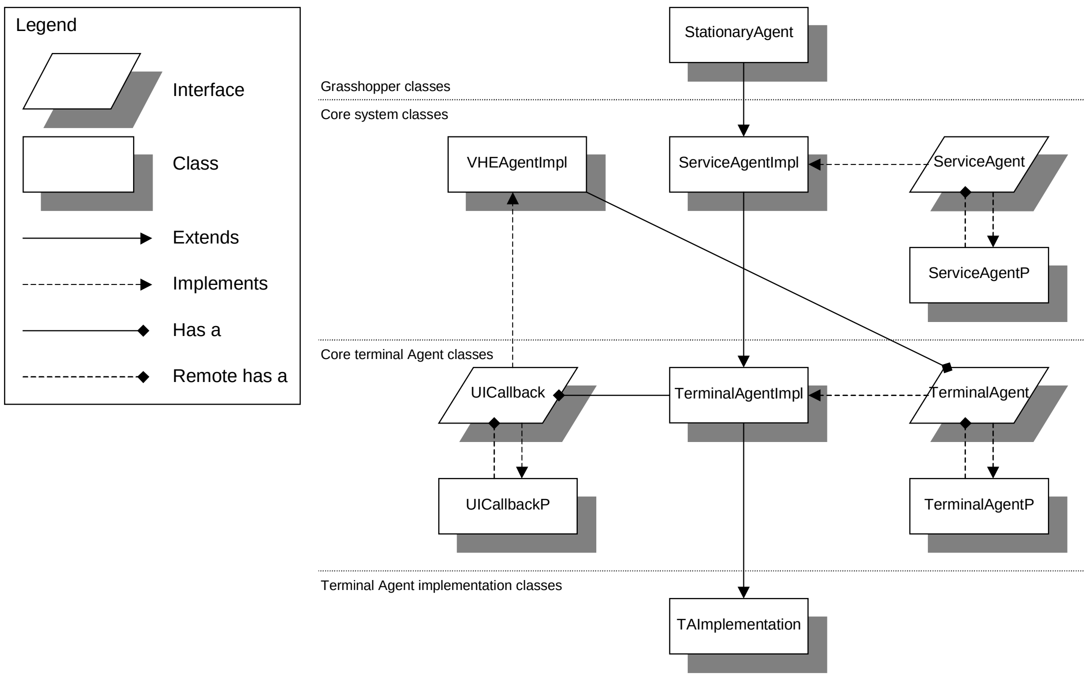


Figure 21 - Components of a Terminal Agent

As for the `ServiceAgentImpl` class, the `TerminalAgentImpl` class (see Figure 21) is a 'shell' such that:

- Developers of TA implementations need not concern themselves with the operation of the rest of the system or Grasshopper. It is envisaged that as new devices appear, new TA's could be produced quickly to extend the number of devices supported by the system.

- The TA implementation can be easily tested independently of the system.

The `TerminalAgentImpl` shell can support several users at once, keeping track of "sessions" via user-id (hence only one session can be open per user), removing some of the effort required to track sessions within sub-classes. This is dependant upon the TA implementation in use (e.g. a WAP TA may deal with several users at one, whereas a Java Swing-TA only ever deals with one user at a time). TerminalAgentImpl sub-classes should track the device a user is interacting via in a manner suitable to the interaction mechanism being used (requirement 22).

Due to the nature of UI's, an event-based model is used in interactions between the TA and VHEA. It is also necessary for the TA's to initiate VHEA migration since they are the part of the system that caters directly for interaction with a particular type of device. Thus, once a user directly or indirectly informs the appropriate TA that they wish to interact with their VHE, the TA contacts their VHEA and indicates this. Once the VHEA has finished migrating, it registers itself with the TA so it can receive event call-backs. The TA then "asks" the VHEA to provide a UI description, and it then presents it to the user.

Whenever a new UI is required, the TA calls-back the VHEA with the current UI description (containing any changes made by the user), and requests a new UI. The VHEA can refuse to return a UI, at which point the user-interaction session is terminated.

Given these interactions, the VHEA includes these methods in its `UICallback` interface:

- `public void userCalling( AgentInfo ta, String device_id )`
  Invocation of this method indicates that the user wants to interact with the VHEA via the given TA. The VHEA then takes appropriate steps to configure itself ready to migrate to the location of the TA.

- `public UserInterface getInitialUI()`
  This is invoked once the VHEA has registered itself with the TA through which it will interact with the user. The VHEA will produce the initial UI description.

- `public UserInterface getNextUI( UserInterface old_ui )`
  Once a UI has been presented to the user, and the user has finished interacting with it this method is invoked. The VHEA then looks at the changes made to the UI, and return a new UI description to be presented to the user if the user has not indicated they wish to finish. If the VHEA decides it has finished, it returns a `null` description.

- `public void userEndedSession()`
  If the user terminates the session abruptly, the TA invokes this method on the VHEA. The TA de-registers the call-back mechanism, and the VHEA returns to its home location/sends its updated state back to the original VHEA.

The TA contains the following methods in its public interface:

- `public boolean registerUI( AgentInfo vhea, String user )`
  This is invoked by a VHEA after this TA has informed it that their user wishes to interact with them, and it has migrated to the same place as the TA. Once this method has been invoked, the VHEA receives call-backs from the TA until it is deregistered.

No deregister method is required, since this occurs whenever the VHE or user decides that the session is over.

The interface of the `TerminalAgentImpl` internally contains the following methods:

- `protected boolean callVHEAHere( String user_id, String device_id, String password, boolean force_vhea )`
  Prompts the TA to contact the VHEA owned by the given user, check that the password given is correct and then ask the VHEA to migrate to a location close to the TA. Returns true if the VHEA migrated successfully.

- `protected UserInterface getNextUserInterface( String user_id, UserInterface old_ui )`
  When a user has completed interacting with one `UserInterface` and the next is required, this method is invoked to prompt the TA to send the UI information back to the VHEA and retrieve the next UI.

- `protected void terminateSession( String user_id )`
  Invoked when the user has finished interacting with the VHEA.

In addition to AgentException's, the TerminalAgentImpl class throws some other exceptions to allow more specific error handling when using these methods. Figure 22 highlights the exception hierarchy.

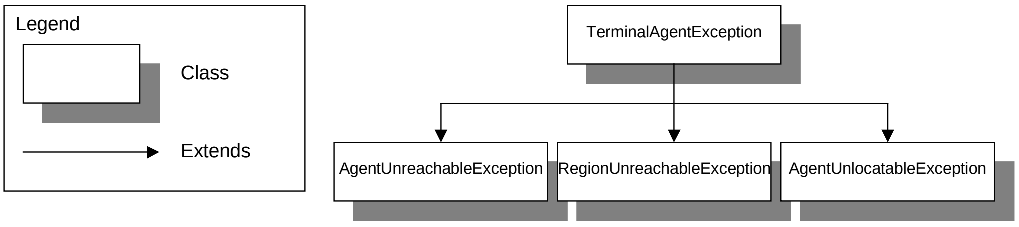

Legend
Class
Extends
TerminalAgentException
AgentUnreachableException
RegionUnreachableException
AgentUnlocatableException

Figure 22 - Terminal Agent Exceptions

In brief:

- `AgentUnreachableException`
  Thrown when the users VHEA cannot be contacted within the system Region, but was located.

- `AgentUnlocatableException`
  Thrown when the users VHEA cannot be found within the system region.

- `RegionUnreachableException`
  Thrown when the system Region cannot be contacted for some reason (the TA may exist outside of the system Region, in which case it is possible that the RegionRegistry provided by Grasshopper is unreachable for some reason).

4.4 Support API's

A number of miscellaneous utility and support classes have been designed in order to provide third-parties with an API which enables rapid development of extensions which make use of third-party classes without introducing the need to have these transported with a VHEA whenever it migrates.

4.4.1 Dynamic support for third-party classes

The Reflect class

This class provides an API that simplifies the process of using Java's Reflection mechanism to dynamically load and interact with classes. The main difference between using this class and directly using Java's Reflection API is that static/non-static methods can be invoked on a class/object with one line of code (compared to 10-20 per method invocation using the Reflection API directly). The other difference is that only one type of exception (`Reflect.ReflectException` - indicating that some failure has occurred whilst using reflection) needs to be caught (compared with the 6 or 7 that are required per method invocation when directly using the Reflection API). Any exceptions from the method being invoked are propagated back to the invoking code as normal.

Due to the need to ensure that only core system-classes and a minimal number of service related classes are referenced at compile time, this is the recommended mechanism to use in order to dynamically use classes which are only required whilst the VHEA is in the AT HOME super-state.

The API provides a number of methods, the most useful of which are:

-
  ```
  public static Object createInstance(String class_name,
                               Class[] arg_types, Object[] args)
  ```
  Attempts to create an instance of the class with the given name, using the constructor which accepts arguments of the given argument types, and the given arguments.

-
  ```
  public static Object invokeStatic(String class_name,
                String method, Class[] arg_types, Object[] args)
  ```
  Attempts to invoke a `static` method in the given class with the given name which has the argument types specified with the specified arguments.

-
  ```
  public static Object invoke(Object o, String method,
                                Class[] arg_types, Object[] args)
  ```
  Attempts to invoked the given method on the object specified which has the specified argument types with the given arguments.

4.4.2 Content adaptation

Since one of the requirements of the project is that content of services should be adapted based upon device profiles (requirement 14), a number of convenience API's have been produced in order to enable this via a common mechanism for a number of data-types.

The ImageAdaptor class

Through use of the `Reflect` class, this class utilises the Java Advanced Imaging API[19] [30] to manipulate image data of several varieties. Essentially this class provides a number of methods to enable manipulation of image data, however one general method for use by services is provided which amalgamates all of the available manipulations into one simple step.

The signature of this method is:

```
public static byte[] translateImage(byte[] image, int max_x,
                         int max_y, String format, double quality)
```

This method takes the given image data (in the form of a `byte[]`), and carries out the following steps:

- Attempts to decode the image. Supported MIME ([20], [21]) image formats are: image/jpeg; image/gif; image/png; image/bmp; image/tiff;

- Checks that the dimensions of the image are smaller than those specified by `max_x` and `max_y` parameters. If not, the image is scaled to ensure that these constraints are met (the aspect ratio of the image is maintained however).

- Encodes the image in the MIME image format specified by the `format` parameter. If the encoding algorithm supports variable quality encoding or compression techniques, the `quality` parameter specifies the quality of the image in the range 0.0 to 1.0. A value of 0.0 specifies that minimising the size of the encoded image is the priority at the sacrifice of image quality, and a value of 1.0 implies that minimising the loss of quality of the image is more important that the size of the encoded image[20].

If the image cannot be decoded or encoded for any reason, a null value is returned.

The DataAdaptor class

This class is provided in order to simplify the process of encoding and decoding binary data. It provides loss-less compression for binary data using the ZLIB compression

[19] The Early Access Java Imaging I/O [31] package is more suitable, however as of the current version (0.4) it is only capable of decoding images, not encoding in any form. Hence, the Java Advanced Imaging [30] package is used instead. See Appendix B:Java Extensions for more details of this API.
[20] Due to limitations in the Java Advanced Imaging API, the only image format supported with variable compression is JPEG. JPEG & PNG support compression, but the other formats do not. GIF images can only be decoded.

algorithm that is part of the JDK, hence this class relies solely upon JDK 1.2.2 features, and does not use the `Reflect` class.

The methods it provides are:

- `public static byte[] compress(byte[] in, double quality)`
  Takes the given `byte[]`, and applies ZLIB compression, returning the resulting `byte[]`.

- `public static byte[] decompress(byte[] in)`
  Takes the given `byte[]`, and attempts to decompress its contents using ZLIB. The inflated `byte[]` is returned if the data was inflated correctly, otherwise `null` is returned.

4.5 Core Agent Detailed Design

The previous section introduced the core classes and concepts that the Agents within the system are based upon. This section builds upon the designs in the previous section to detail the design of the core Agents within the system.

4.5.1 VHEAgent (VHEA)

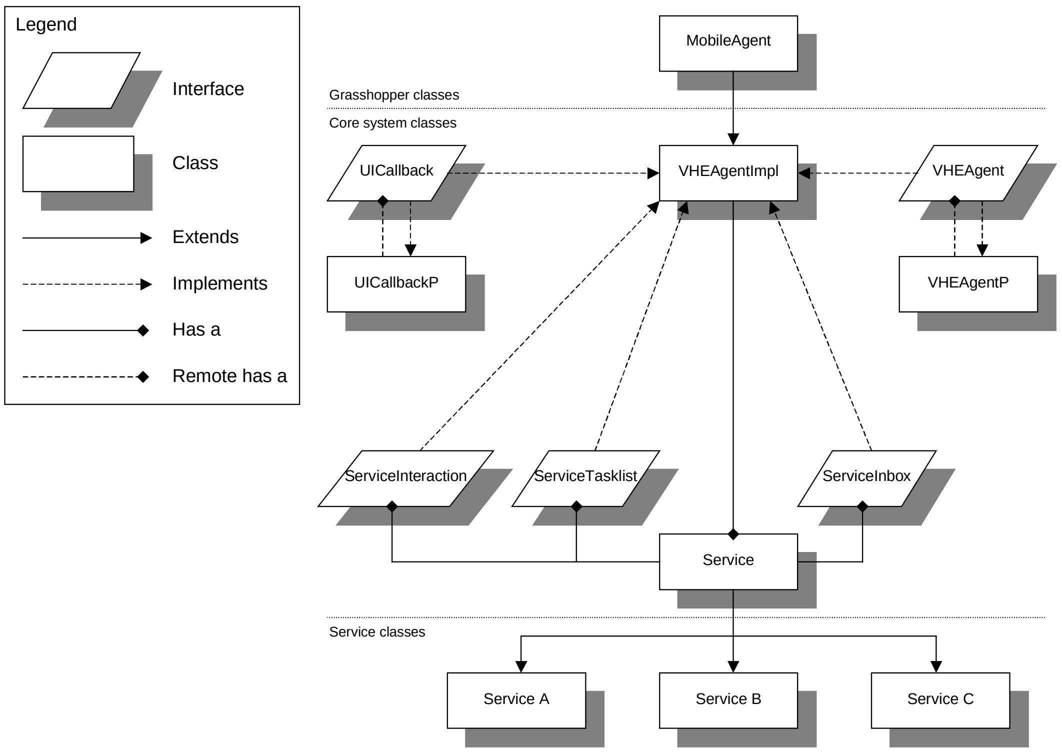


Figure 23 - Object-oriented VHEA Component Composition

As described in section 4.3.3, the VHEA consists of a number of `Service` implementations, a `ServiceInbox` and `ServiceTasklist`. Figure 23 depicts how these components fit together, with the VHEA implementation contained within the `VHEAgentImpl` class. As described previously, this class implements the `ServiceTasklist` and `ServiceInbox` interfaces, allowing changes to the `ServiceInbox` or `ServiceTasklist` to be tracked and their content to be persistently stored (requirement 10) with the DBA (see section 4.5.2). The aim is that should the VHEA fall over, or the system abruptly stopped, the contents of the VHEA's inbox and task list will be available when the VHEA / system is reactivated.

The `ServiceInteraction` interface is also implemented to provide inter-service interactions to the `Service` instances within the VHEA (requirement 19).

The VHEA operates in two main states:

- At home: In this state, the VHEA is located at its home place, as decided by the SMA. The VHEA is actively attempting to carry out tasks that the user has assigned to it.

- Interacting with a user: In this state, the VHEA is not at its home place, and is interacting with a TA in order to communicate with its user. In this state, its sole purpose is to present the information it has gathered so far on behalf of the user, and add any tasks the user wants carried out to its internal task list. `Service`'s it contains are not permitted to contact SA's in this state (requirement 17).

TA's are the primary Agents to trigger changes in these states, since the TA informs the VHEA when to migrate to the TA's place, when the VHEA's user wants to interact with it. The TA also indicates when the user dismissed the TA, and hence when it should return home/send its updated state back to the original VHEA.

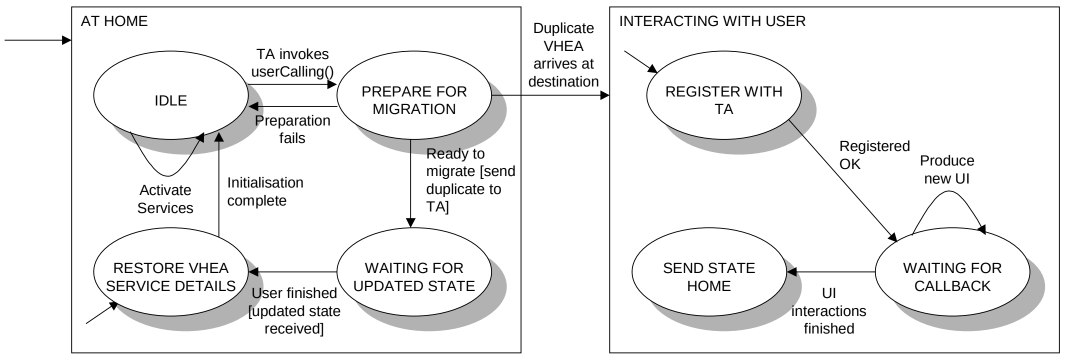


Figure 24 - VHEA State Diagram

Figure 24 is a diagram highlighting the state model for the VHEA. These states are implemented within the VHEA through the use of the "State Pattern".

RESTORE VHEA SERVICE DETAILS

This is the initial state of the VHEA when it is first created, and after the updated state of a duplicate VHEA has been sent back home from a user interaction session. In the first case, the only detail it has is its users `UserProfile`[21]. Given this information the VHEA attempts the following steps:

- Contacts the system DBA, and requests details of its' users' subscribed services' `UserServiceConfig` objects

- Retrieval of users Inbox and Task list from the DBA

- Initialise each of the users subscribed service implementations, and configure them

In the later case, the following step is carried out instead:

- Items in the users' inbox and task list are synchronised with those persistently stored with the DBA. This is to update the content of the databases' in line with `ServiceItem`'s, `ServiceTask`'s and `UserServiceConfig`'s which have been

[21] This has changed from the initial design, since the SMA owns the `UserProfile`'s within the system.

added, removed or modified during interaction with the user (requirement 19).

- The `DeviceProfile`'s of the VHEA's user are synchronised with the SMA.

If this step cannot be completed, the VHEA continues as normal and trys again shortly. The VHEA then moves into the IDLE state.

IDLE

In this state, the VHEA is not doing anything except periodically activating the `Service` implementations it contains (allowing requirement 16 to be met by services).

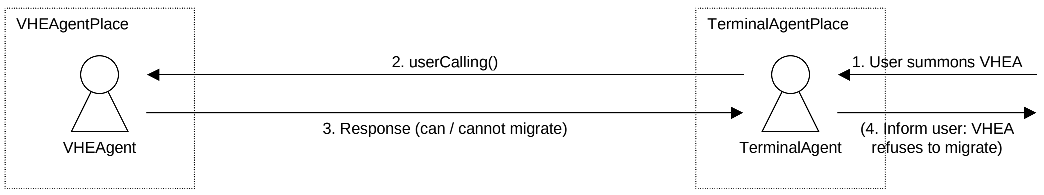


Figure 25 - VHEA / TA Interactions (User Summons VHEA)

If the VHEA receives a request from a TA indicating that its user wishes to interact with it, it moves into the PREPARING FOR MIGRATION state and responds positively to the request. Figure 25 highlights this interaction.

PREPARING FOR MIGRATION

There are a number of steps that must be carried out in this state:

- Determine which device the user wishes to interact with the VHEA via. This can be achieved by matching the device id provided by the TA with a `DeviceProfile` belonging to the user (requirement 22).

- Ask each service instance to prepare the `ServiceItem`'s and `ServiceTask`'s for the migration by filtering out any content that will be irrelevant to the device the user will be using to interacting with the VHEA (allowing requirements 12 & 14 to be met by services). The filtering done is service dependant.

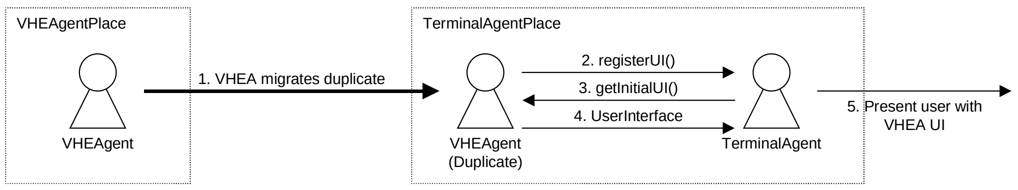


Figure 26 - VHEA / TA Interaction (VHEA Migrates & Registers with TA)

Provided these steps are carried out successfully, a duplicate[22] of the VHEA then migrates to the same Place as the TA the user is communicating with (requirement 13) – see Figure 26. The VHEA keeps a note of the TA that a duplicate has been dispatched to (requirement 22).

In the case that these steps cannot be carried out, a fault message is sent to the TA agent, and the VHEA returns to the IDLE state

[22] The original design intended that the VHEA migrated to the TA's location itself, however due to a bug in the Grasshopper Agent architecture, this was abandoned in favour of sending a duplicate Agent. See Week 11, 29/5/2000 in Appendix E:Implementation Logbook.

WAITING FOR UPDATED STATE

In this state, the VHEA waits for the updated state information from a duplicate VHEA which has been dispatched to interact with the user. Upon receiving updated state information, the VHEA moves into the RESTORE VHEA SERVICE DETAILS state to resynchronise any changes the user has made with the DBA.

REGISTER WITH TA

This state is entered straight after a duplicate VHEA has completed migrating to the TA's location. The first task is to register the VHEA's with the TA in order to allow the TA to drive "events" to the VHEA. If this is successful, and the TA requests the initial UI, the VHEA moves into the WAITING FOR CALLBACK state and sends back the initial UI (see Figure 26).

WAITING FOR CALLBACK

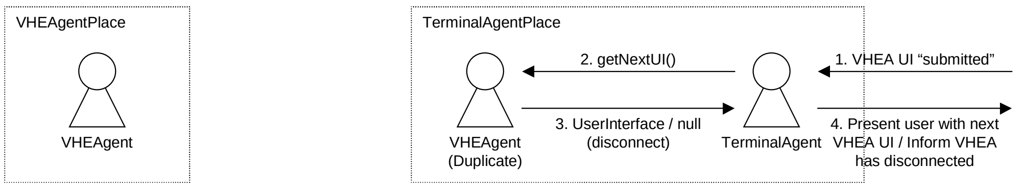


Figure 27 - VHEA / TA Interactions (VHEA & TA Exchange UI's)

In this state, the duplicate VHEA is waiting for the TA to invoke methods on its `UICallback` interface (see Figure 27). The TA returns the state of the UI after the user has interacted with it, and the VHEA produces a UI in response. The duplicate VHEA uses a set of `UIState` objects and the `UIStateListener` interface to maintain knowledge of the current state that the UI has reached. Figure 28 is a state diagram of the `UIState`'s that are accessible whilst within this state. All user interactions with the system are via the UI's these states produce (requirement 4).

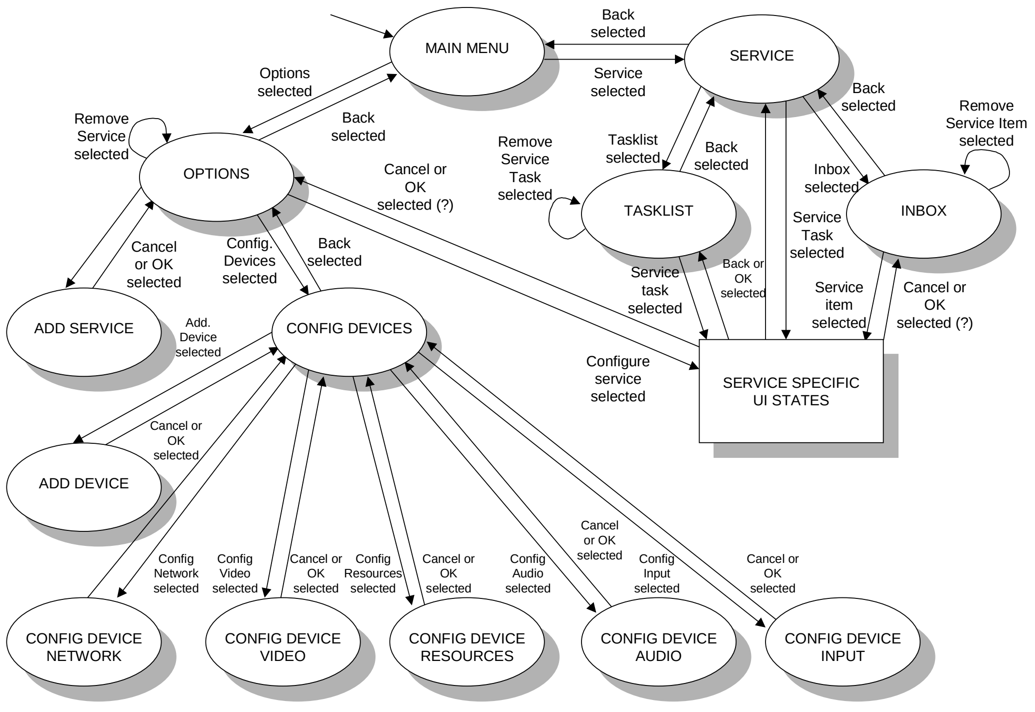


Figure 28 - VHEAgentImpl UIState's

The main `UIState`'s are:

- **MAIN MENU:** This state provides a menu from which the user can choose to enter a service-specific menu (by selecting the service they wish to enter), or enter the Options menu.

- **OPTIONS:** This state contains the main Options menu. From this state, the user can add, remove and configure subscribed services (requirements 5 & 7), or select to modify their device profiles. As the user interacts with the menu, the internal configuration of subscribed services changes according to those interactions.

- **CONFIG DEVICES:** From this state, the user can add, remove or modify device profiles within their user profile (requirement 18).

- **SERVICES:** This provides a service-specific UI that contains options to enter the inbox or tasklist of the service, or instruct the VHEA to carry out a service-specific task (requirement 15).

- **INBOX:** This state displays the contents of the users' service inbox for a particular service. The user has the option of viewing individual items presented, or removing them.

- **TASKLIST:** This state displays the contents of the users' service tasklist for a particular service. The user has the option of modifying individual tasks presented, or removing them (requirement 15).

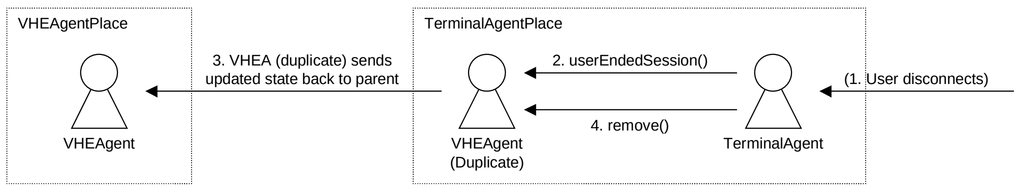


Figure 29 - VHEA / TA Interactions (Session Ended)

The TA can also indicate that the user wishes to finish interacting with the duplicate VHEA (see Figure 29), at which point the duplicate VHEA moves into the SENDING STATE HOME state.

SENDING STATE HOME

In this state, the duplicate VHEA attempts to send its updated state back to the original VHEA (see Figure 29). Only changes in state are actually sent, so this is very small in size relative to the overall size of the VHEA.

The TA then removes the duplicate VHEA from the system[23].

[23] This is due to the another bug within Grasshopper (see Week 11, 29/5/2000 in Appendix E:Implementation Logbook), which prevents `MobileAgent`'s from removing themselves from the system, otherwise the duplicate VHEA would remove itself without assistance from the TA.

4.5.2 System Manager Agent (SMA)

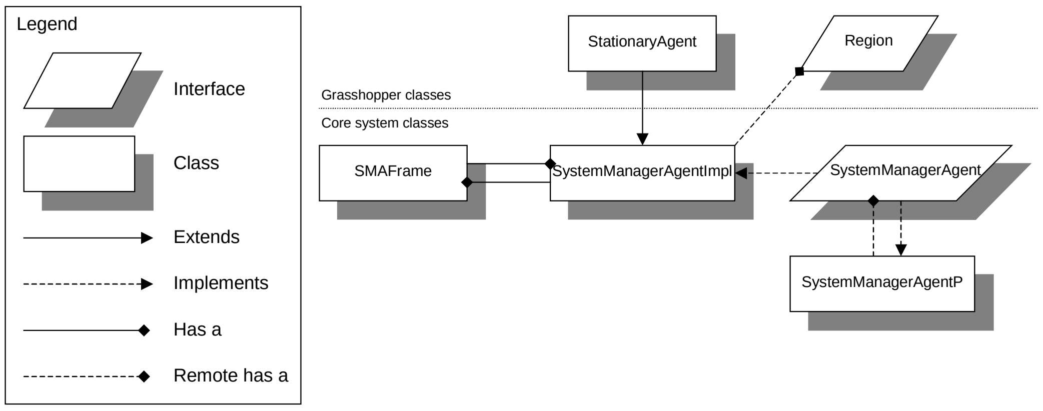


Figure 30 - SMA Classes

As described in section 3.1.2, the SMA is responsible for a number of tasks within the system. This section highlights these tasks and provides a detailed design of the SMA. Figure 30 highlights the SMA's class model.

Responsibility for User and Service Profiles and Configuration

The SMA stores all of this information within itself, and propagates this information to other Agents as required. The SMA allows multiple `UserProfile`'s and `ServiceProfile`'s to be defined for use within the system at once (allowing requirement 1 to be met**)**.

Due to the requirement that this information is persistently stored, the SMA ensures that a copy of all information is stored within the DBA of the system, and that appropriate updates are sent to the DBA as its internal version of this information changes.

User Interface

One of the primary roles of the SMA is to provide a graphical representation of the state of the system to a SysAdmin user, and allow them to modify the various profiles contained within the system. As a result, one of the major differences between the SMA and other Agents in the system is the inclusion of a GUI.

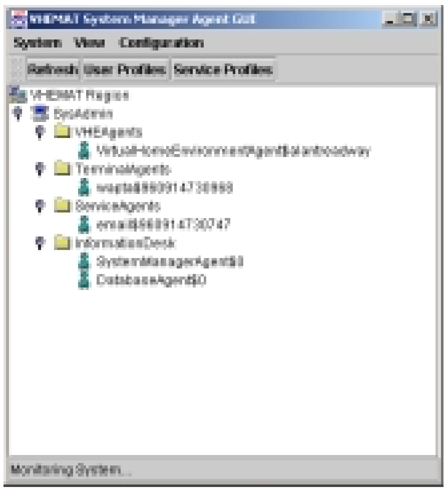
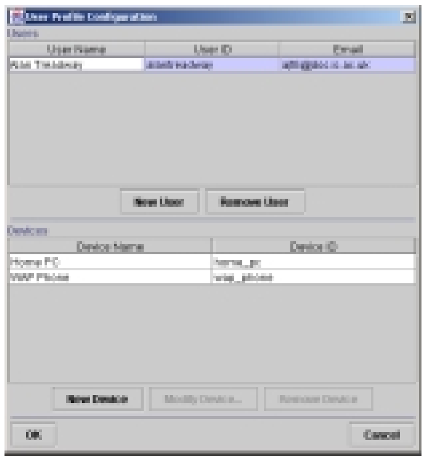


Figure 31 - SMA Explorer UI

The GUI consists of three main parts:

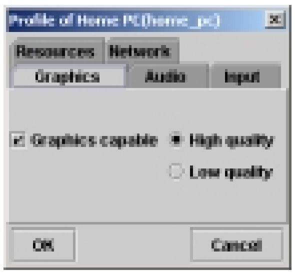
- **Explorer-like view of the system:** This part of the UI provides an at-a-glance view of the state of the system and the Agents that are currently populating it, and the current activity that the SMA is undertaking (see Figure 31). Agents can be removed from the system, Places can be created and removed, and the properties of a particular Agent displayed. The Explorer-like UI can be configured to display Agents within the system Region either by the Agency/Place they are currently located within, or by Agent type.



User Name	User ID	Email
[illegible]	[illegible]	[illegible]

Device Name	Device ID
Home PC	home_pc
WAP Phone	wap_phone

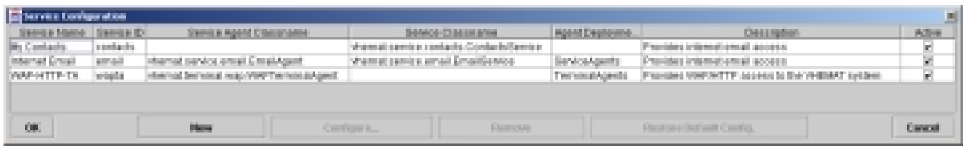
Figure 32 - SMA User Profile UI



[✓] Graphics capable (●) High quality ( ) Low quality

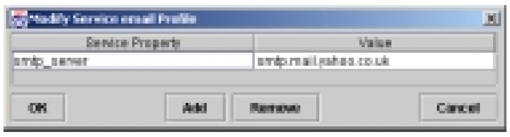
Figure 33 - SMA Device Profile UI

- **User profile configuration UI:** This section of the UI allows the profiles of the users within the system to be created, removed or modified (see Figure 32). Selecting a user from the list of current users allows the list of current device profiles to be modified, so devices can be added, removed or modified (see Figure 33). (Requirement 20)



Service Name	Service ID	Service Agent Classname	Service Classname	Agent Deployme...	Description	Active
My Contacts	contacts		vhemat.service.contacts.ContactsService		Provides internet email access	✓
Internet Email	email	vhemat.service.email.EmailAgent	vhemat.service.email.EmailService	ServiceAgents	Provides internet email access	✓
WAP-HTTP-TX	[illegible]	vhemat.terminal.wap.WAPTerminalAgent		TerminalAgents	Provides WAP/HTTP access to the VHEMAT system	✓

Figure 34 - SMA Service Profile UI



Service Property	Value
smtp_server	smtp.mail.yahoo.co.uk

Figure 35 - SMA System Service Config. UI

- **Service profile & system service configuration UI:** This part of the UI allows the details for each of the services within the system. Services can be added, removed or modified (see Figure 34), and there is the ability to modify the system-service configuration of each service (see Figure 35). (Requirement 21)

  In order to reduce the amount of data-entry within this part of the UI, the `Reflect` class is used to invoke the methods `getDefaultSystemServiceConfig()` and `getDefaultServiceProfile()` on the `Service` and `ServiceAgent` classes specified in the currently selected service by selecting the “Restore Default Config” button. If these static methods exist in either class, they are expected to return the default system-service configuration and service profile respectively.

Each of these components works independently of the SMA. The user-profile and service configuration UI’s simply modify data-structures within the SMA, and it is down to the SMA to detect these changes in order to propagate them to the appropriate Agents. Care is taken to ensure that thread synchronisation problems do not occur – concurrent modification of the data-structures could result in ambiguous profiles and configurations. Generally, duplicate copies of the data-structures are taken from the SMA, and when the user clicks the “OK” button the data-structure containing the original data is updated (only where necessary) in one go. Mechanisms are in place to ensure that during the course of this operation mutual exclusion is maintained.

The main Explorer-like UI simply uses a `TreeModel` and `TreeModelRenderer` provided by the SMA to create its display, so the SMA does not expose any internal data-structures to enable the UI contents to be produced. Events to trigger removal of Agents or creation/removal of Places simply invoke methods on the SMA which actually carry out the requested task.

Agent Population Control

The SMA is responsible for ensuring that the system is fully populated with Agents, to achieve this, it requires the information contained within the `ServiceProfile`’s and `UserProfile`’s. In the case where the SMA is just starting up, it attempts to obtain this information from the system DBA. In the event that the DBA has not been started yet, the SMA creates an instance of it, and then proceeds to retrieve the required information from it.

When a system first starts up, the SMA is the first (and only) Agent within it, with the exception of the DBA which the SMA automatically instantiates. This situation is dealt with no differently to the case when the system is populated with Agents.

The SMA periodically (every 10 seconds) updates its view of the system. This is achieved by interrogating the system Region for a list of all Agencies, Places and Agents, and then creating a tree-like model of the world. It then proceeds to carry out the following checks and subsequent actions, which form an algorithm for ensuring the population of the system is maintained:

- For every Place within the system, check the following:
    - For every `ServiceProfile` defined, check the following:
        - Should this Place contain SA’s for the service defined by the `ServiceProfile`?

          If so, check that there is at least one SA of the type defined by the `ServiceProfile` within this Place – if not, instantiate the type of SA specified by the `ServiceProfile` (requirement 6, and allowing requirement 1 to be met).

          If not, check that there are no SA’s for the service defined by the `ServiceProfile` – if so, remove them.
    - Should this Place contain VHEA’s?

      If so, add this Place to a list of “VHEA-Places”.

    If not, remove any VHEA's that are not interacting with a TA in this Place.

    - Are there any Agents that are not known SA's or VHEA's?

      If so, remove them.

- For every Place within the list of "VHEAgent-Places":
    - Strike off every VHEA's user for every VHEA in the place, from a list of all users.
- For every user without a VHEA:
    - Deploy a VHEA in a "VHEA-Place" (requirement 4, and allowing requirement 1 to be met), attempting to ensure an even balance of VHEA's across all "VHEA-Places".

This algorithm therefore ensures that system is "fully" populated with Agents based upon the `ServiceProfile`'s and `UserProfile`'s the SMA contains. As these profiles change, naturally this algorithm removes any Agents that are no longer in the right place, or belong to a deactivated service.

Distributing updated profiles

As mentioned previously, the SMA monitors its internal ServiceProfile's, SystemServiceConfig's and UserProfile's for updates caused as a result of the SysAdmin interacting with its GUI. Changes are checked for periodically (every second), by checking the status of the modification flag on each item within these internal data-structures.

When a modification is found, the following actions are taken:

- The change is propagated to the DBA, to ensure that a persistent version of the changes has been kept.
- The appropriate Agents are forwarded the updated item.

  In the case of a SystemServiceConfig changing, the relevant SA's are notified of the change, as well as all VHEA's.

  In the event that a ServiceProfile is updated, the change is propagated to all VHEA's.

  In the event that a UserProfile is updated, it is forwarded to the appropriate VHEA.

In order to prevent the SMA from blocking whilst sending these updates serially, a separate thread for each message is created so that the messages can be sent in parallel and the SMA can continue its other tasks without waiting for the updates to be sent.

4.5.3 Database Agent (DBA)

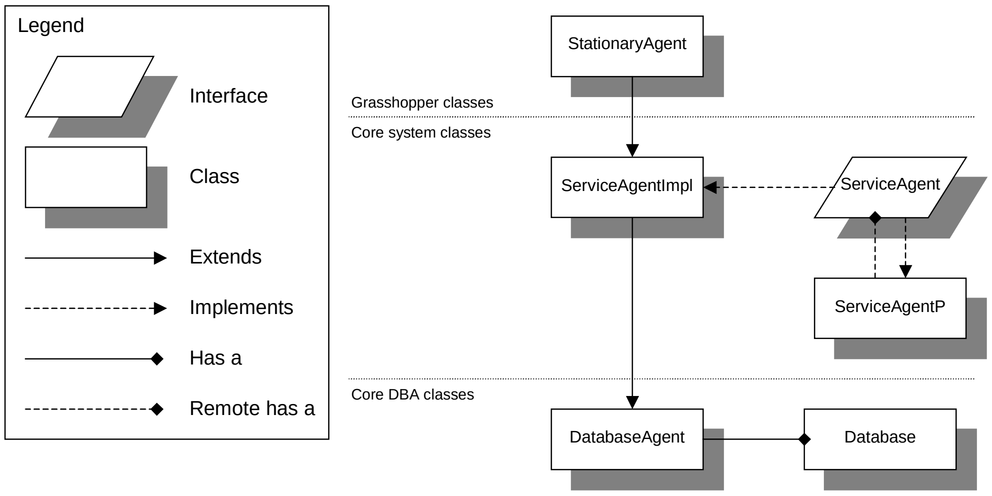


Figure 36 - Database Agent Classes

The DBA exposes a `ServiceAgent` interface, and hence other Agents interact with it by passing it `ServiceTask` objects containing tasks such as 'create a database' or 'locate an object' (assuming they have permission to do so). The DBA forms the basis for ensuring that requirement 10 is met.

ServiceTask Types

Since the DBA utilises the `ServiceAgentImpl` shell, it is required to deal with requests from other Agents in the form of `ServiceTask` objects. The following is a description of all of the types of `ServiceTask` that the DBA is able to carry out, along with descriptions of the parameters that can be specified for these tasks.

ServiceTask Type	Parameter	Description
"create-db"		Creates a new persistent database with the given access rules and name. Doesn't return anything, but throws exceptions if anything goes wrong.
	"name"	The name associated with the database
	"primary-key"	The name of the attribute of objects within the database that should be used as the primary key.
	"access-rules"	The DBA-rule governing what Agent can view what data and perform what operations on this database (see later)
"delete-db"		Deletes the entire persistent store of information for the database given. Only the creator of the DB or the SMA is allowed to do this. Does not return anything, but throws exceptions if anything goes wrong.
	"name"	Name of the database to delete.
"insert-into-db"		Inserts a serialisable object into the given database overwriting any object that previously existed with the same primary key attribute (hence this can also be viewed as an "update" operation in this case). Doesn't return anything, but throws exceptions if anything goes wrong.
	"db-name"	The name of the database to insert the object into.

ServiceTask Type	Parameter	Description
	“object”	The object to insert
“query-db”		Searches the given database for all objects that match the given criteria. Return’s a List[24] of Object’s within the database matching the query.
	“db-name”	The name of the database to query
	“query”	The DBA-query to be performed (see later)
“delete-from-db”		Deletes an object with the given primary key from the given database. Doesn’t return anything, but throws exceptions if anything goes wrong.
	“db-name”	Name of the database to remove the object from
	“primary-key”	The primary key of the object to remove

Should any of these operations fail, the DBA throws `AgentException`’s with appropriate error messages to be caught by the invoking Agent.

Object Attributes

The model of a database used assumes that each object (entry in database terms) has a number of attributes (fields in database terms) that contain ASCII/UTF8 strings, integers or boolean values. Since it is not good object-oriented programming practice to expose the internal variables of objects, the DBA uses Java Reflection to dynamically invoke methods on objects in order to obtain attribute values. Any method with the following signature is considered to allow access to an attribute value:

```
public <attribute-type> getX()
```

where X is the attribute name (not case-sensitive), and <attribute-type> is one of String, boolean or integer for simplicity of implementation. As an example, any object with the `ServiceProfile` interface is considered to have the following attributes: ServiceID; Description; ServiceClassName; ServiceAgentClassName; Active;

If an object has a method with the following signature, it assumes that it can be used to access the values of attributes:

```
public Object get( String name )
```

Thus `get(“size”)` would be a valid call to get the value of attribute “size”.

Access Rules

In order to incorporate some degree of security into the DBA with regard to access to information persistently stored within the system, creators of databases can specify simple access rules for the three basic types of operation on objects within the database. The assumption is made that no operations can be made by an Agent other than the creator and the SMA on the database unless they are specifically allowed to in the access rules.

The BNF grammar for defining these rules is as follows:

```
Rules     :=    <Rule > ( ‘;’ <Rule> )*
Rule      :=    <Op-type> ‘:’ <Rule-expr>
Op-type   :=    ‘QUERY’ | ‘INSERT’ | ‘DELETE’

Rule-expr :=    <LHS> <Op> <RHS > ( ‘&’ <Rule-expr> )
```

[24] Changed from `Object[]` in the original design, for programming convenience

```
Op        :=    '=' | '!='

LHS       :=    'invoker' | <attribute>
RHS       :=    'object-owner' | '*' | <Constant>

Constant  :=    <String> | <integer> | <Boolean>
String    :=    '"' <string> '"'
Boolean   :=    'true' | 'false'
```

The token <attribute> can be any attribute name within the database, <string> and <integer> are the base types String and int in Java.

The LHS token defines that either 'invoker' (the Agent invoking the operation) or an attribute name common to objects in the database be specified. The RHS token defines that 'object-owner' may be specified, which refers to the Agent that last inserted / updated a particular object.

If more than one rule is specified per access type, the results of each rule are logically OR'ed together to form the result for each access type.

Access Rule Examples

-
  ```
  INSERT:invoker=object-owner;
  QUERY:invoker=object-owner;
  DELETE:invoker=object-owner
  ```

  This would provide the ability for new objects to be inserted, but only the owner of the object to replace it, locate it in a query and delete it.

-
  ```
  QUERY:Active=true&ServiceID="sid2314"
  ```

  This would allow the invoker to get a particular object back during a query provided either they own the database, or the object has attributes Active and ServiceID with values true & sid2314 respectively. Only the database owner and the SMA can insert and remove objects.

-
  ```
  INSERT:invoker=object-owner;
  QUERY:invoker=*;
  DELETE:invoker=object-owner
  ```

  This would allow only the creator of objects in the database (or database owner and SMA) to add, update and delete objects. Any Agent can query the database and get back the results.

Queries

In order to allow a database to be searched for appropriate objects, a simple query language is to be used to allow the amount of data transferred between Agents when sifting through persistently stored data.

The following BNF describes the grammar that is used in order to formulate queries of a database:

```
Query     :=    <QueryPart> ( ( '&' | '|' ) <QueryPart> )*
QueryPart :=    <LHS> <Op> <RHS>

Op        :=    '=' | '!='

LHS       :=     'invoker' | <attribute>
RHS       :=    'object-owner' | '*' | <Constant>
```

```
Constant  :=    <String> | <integer> | <Boolean>
String    :=    '"' <string> '"'
Boolean   :=    'true' | 'false'
```

The 'object-owner' and 'invoker' terms are defined as per the access-rule definition, and the query is evaluated with the logical AND operator (&) taking precedence over the logical OR operator (|).

Query Examples

- `object-owner=invoker&Active=true&ServiceID!="sid5423"`

  This produces all objects that are owned by the Agent making the query, have an attribute Active that has a value of true, and doesn't have an attribute ServiceID of value sid5423.

- `UserName="Alan Treadway"|UserID="uid3534"`

  This produces all objects where the attribute UserName is "Alan Treadway" or the UserID attribute is uid3534.

Persistent Storage Mechanism – Database Object

Owing to the complexity of using the JDBC Java extension, the DBA simply utilises Java serialisation to persistently store information within the system. Serialisation effectively allows an object to be converted into a byte stream that can be stored on disk or transmitted over a socket connection, along with any objects it refers to. The aim is that each database is modelled by a `Database` object that can serialise a database to disk, and de-serialised it to recover its information. Each database is stored in a file with path `~/.vhemat/db/<db-name>.jsdb` such that the DBA can easily locate it when first asked to interact with it (where '~' is the "network-administrators" home directory).

The `Database` object provides all of the necessary methods for adding objects, removing objects and querying the database. The `Database` object has the following interface:

-
  ```
  public boolean insertElement( Object new_object,
                                                   String invoker )
  ```
  Inserts the given object into the database, or overwrites the object already present that has the same primary key. Updates the owner to be that of the given Agent name.

-
  ```
  public void deleteElement( Object removed_object,
                                                   String invoker )
  ```
  Removes the given object from the database

- `public void deleteElementByKey( Object key, String invoker )`
  Removes the object with the given primary key value from the database.

- `public List performQuery( String query, String invoker )`
  Executes the given query on the database. The invoker argument indicates the Agent that is making the request in order to evaluate the 'invoker' token within the DB-query grammar.

4.6 Extensions to core functionality: Services

This section aims to provide details of the services produced to extend the core system functionality.

4.6.1 My Contacts Service

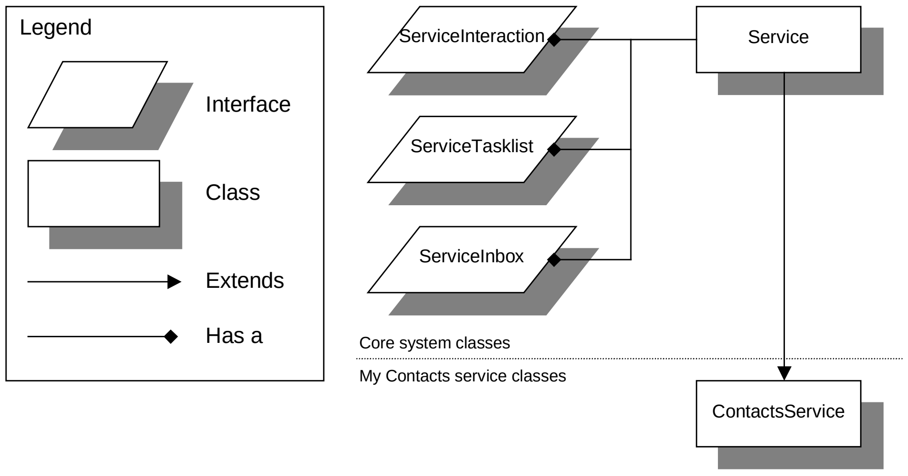


Figure 37 - ContactsService Classes

This service provides an address book type service for use within the VHEA. This service requires no SA's to provide its services, since all required functionality can be contained within the VHEA, and the service only needs to be invoked when the VHEA is interacting with the user.

The following sections define the `ServiceItem` type used by this service, and give a brief description of the `ContactsService` class that comprises the whole functionality of the service.

ServiceItem's of Type "contact-service.contact"

This type of `ServiceItem` represents a persons contact details.

ServiceItem Type	Parameter	Description
"email"	"first_name"	The first name of this contact
	"last_name"	The last name of this contact
	"mobile"	The mobile phone number of this contact
	"home"	The home phone number of this contact
	"email"	The email address of this contact
	"address"	An address associated with this contact

UserServiceConfig for this Service

The `UserServiceConfig` associated with this service contains these details:

Key	Value description
"sort-by"	Indicates the field of `ServiceItem`'s of type "contact-service.contact" in the users inbox to sort by.

ContactsService class

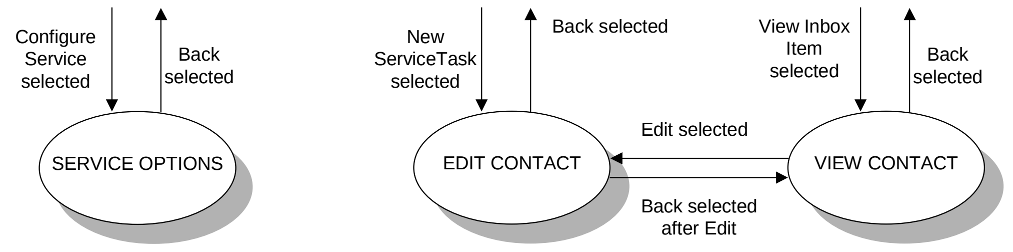


Figure 38 - ContactsService's UIState's

The `ContactsService` class is a straightforward implementation of the `Service` class, and requires no interaction with other `Service` instances or SA's. Its primary functionality is to produce `UIState`'s to allow creation, viewing and editing of `ServiceItem`'s (and `ServiceTask`'s[25]) of type "contact-service.contact". Figure 38 highlights the state model used by the `UIState`'s of this service.

4.6.2 Internet Email Service

The Internet Email Service provides the ability for a user to integrate POP3/SMTP email capabilities into their VHEA. This service allows a users' VHEA to monitor their POP3 account for incoming mail, download new mails as they arrive and present them to its user when summoned. It also allows emails to be sent by the VHEA on behalf of the user, or proactively itself. This service makes use of content filtering and adaptation to ensure that a user doesn't have to wait for inappropriate content to be sent to the device they are using, and reduces the amount of bandwidth used to deliver emails to users (due to both the content filtering/adaptation and the use of a binary encoding for the emails, compared with an ASCII-based encoding used when retrieving email by POP3).

The service is composed of two classes, `EmailAgent` and `EmailService`, which are described below along with the internal representation of emails within the system.

ServiceItem's of Type "email-service.email"

The only type of `ServiceItem` that the Internet Email Service uses is of type "email". Below is a description of the contents of a `ServiceItem` of this type.

ServiceItem Type	Parameter	Description
"email"	"from"	The email address of the sender of the email
	"to"	A `List` containing the email address of the recipients.
	"cc"	A `List` containing the email address of the recipients to be CC'ed (Carbon-Copied).
	"bcc"	A `List` containing the email address of the recipients to be BCC'ed (Blind CC'ed).
	"subject"	Subject line of the email
	"body"	Text contained in the body of the email.
	"date"	The date that the email was sent

[25] `ServiceTask`'s get automatically converted to `ServiceItem`'s straight after they have been created since this service has no tasklist.

ServiceItem Type	Parameter	Description
	“attachments”	A `Map` containing either `String`’s, or `Object[]` representing the attachments contained in this email. In the former case, the attachment is MIME type “text/plain”. In the later case, index 0 of the array contains the MIME type of the content, and index 1 contains a `byte[]` containing that content. In either case, the key’s to the `Map` are the filenames of the attachments.

ServiceItem’s of Type “email-service.saved-email”

This type of `ServiceItem` is identical to the “email-service.email” type, except it represents an email that has been “saved” (i.e. it is an out-going email in draft status, which shouldn’t be sent yet).

UserServiceConfig for this Service

The UserServiceConfig associated with this service contains these details:

Key	Value description
“email-address”	Users’ email address.
“username”	Users’ POP3 mailbox username.
“password”	Users’ POP3 mailbox password.
“mailhost”	Users’ POP3 mailboxes’ host
“mailbox-check”	Timeout between checks of mailbox.
“pending-msgs”	Message-id’s of items in POP3 mailbox that need to be downloaded.
“received-msgs”	Message-id’s of items in POP3 mailbox which have been downloaded previously. Message-id’s are added to this list as messages are downloaded, and removed as messages are removed from the POP3 mailbox (used to prevent downloading the same message multiple times until the user removes it from the POP3 mailbox).

SystemServiceConfig for this Service

The SystemServiceConfig associated with this service contains these details:

Key	Value description
“smtp_server”	The hostname of the SMTP server to use for outgoing mail.

EmailAgent Class

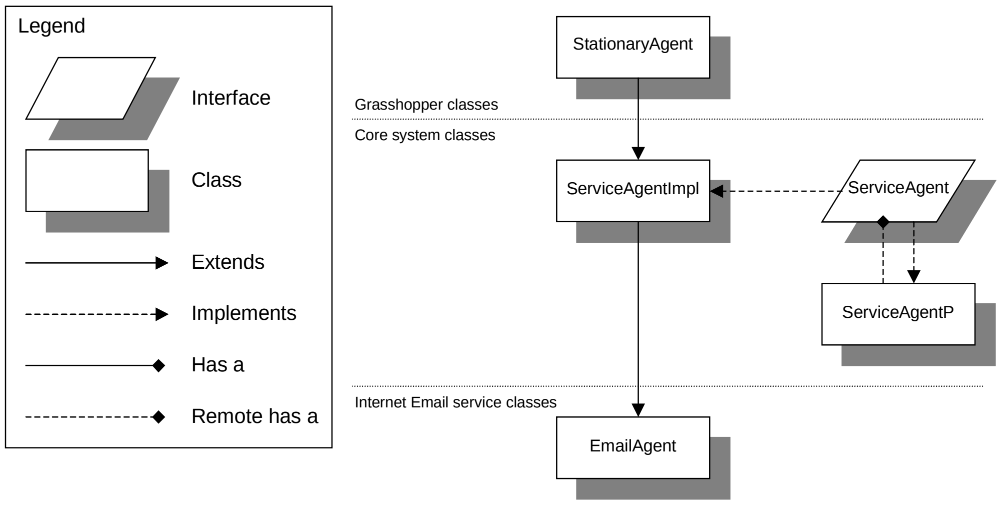


Figure 39 - EmailAgent Classes

The `EmailAgent` class is a SA that utilises the Java Mail Extension [32] (see Appendix B: Java Extensions for more details) in order to provide the ability to interact with POP3 and SMTP servers. All interactions with the `EmailAgent` are via the `executeTask()` method exposed by the `ServiceAgent` interface, Figure 40 through Figure 42 highlight the possible interactions.

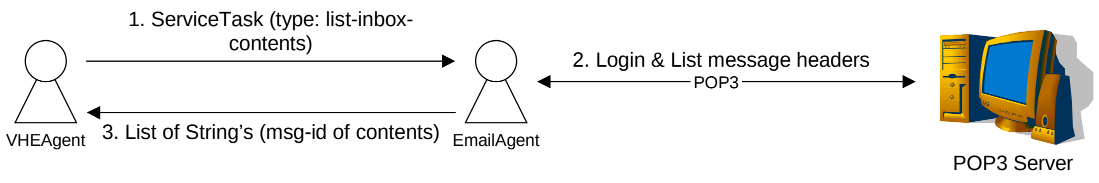

1. ServiceTask (type: list-inbox-contents)
VHEAgent
3. List of String's (msg-id of contents)
EmailAgent
2. Login & List message headers
POP3
POP3 Server

Figure 40 - VHEAgent / EmailAgent Interactions (List POP3 Inbox Contents)

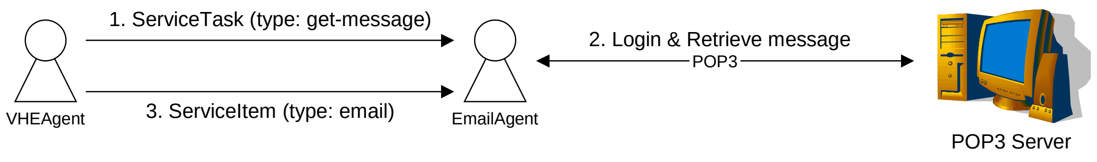

1. ServiceTask (type: get-message)
VHEAgent
3. ServiceItem (type: email)
EmailAgent
2. Login & Retrieve message
POP3
POP3 Server

Figure 41 - VHEAgent / EmailAgent Interactions (Retrieve Email From POP3 Mailbox)

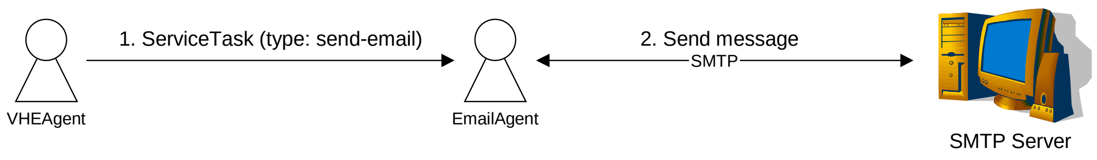


Figure 42 - VHEAgent / EmailAgent Interactions (Sending Email via SMTP)

The specifics contents of the `ServiceTask`'s used within these interactions are highlighted below.

ServiceTask Type	Parameter	Description
"list-inbox-contents"		Attempts to connect to a POP3 server & collect the headers of all messages contained within a particular mailbox. Returns a `List` of message-ids contained within the inbox.

ServiceTask Type	Parameter	Description
	"username"	The user-name associated with the POP3 mailbox to be interrogated.
	"password"	The password of the mailbox.
	"mailhost"	The host containing the POP3 server program on which the mailbox is located.
"get-message"		Attempts to connect to a POP3 server & retrieve a particular message from the given POP3 mailbox. Returns a `ServiceItem` of type "email-service.email" containing the message if successful.
	"username"	The user-name associated with the POP3 mailbox.
	"password"	The password of the mailbox.
	"mailhost"	The host containing the POP3 server program on which the mailbox is located.
	"uid"	ID of the message to retrieve.
"send-email"		Attempts to connect to a SMTP server & send a message.
	"from"	The email address of the sender of the email
	"to"	A `List` containing the email address of the recipients.
	"cc"	A `List` containing the email address of the recipients to be CC'ed (Carbon-Copied).
	"bcc"	A `List` containing the email address of the recipients to be BCC'ed (Blind CC'ed).
	"subject"	Subject line of the email
	"body"	Text to be contained in the body of the email.

EmailService Class

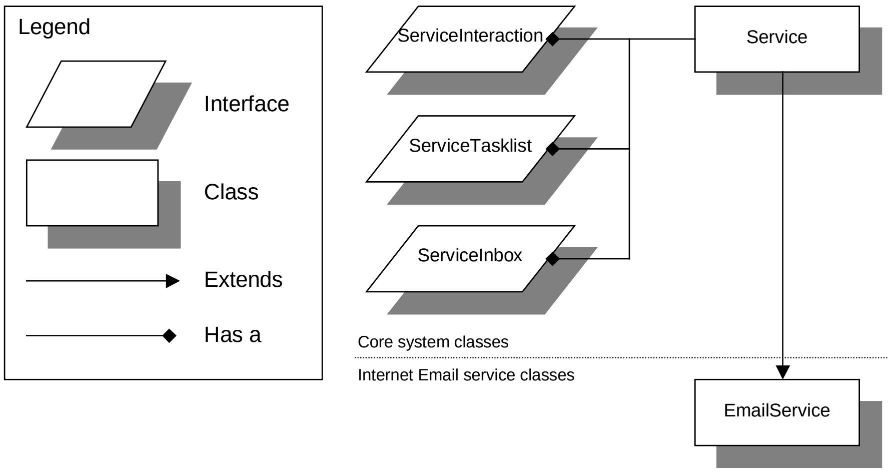


Figure 43 - EmailService Classes

The `EmailService` class provides the VHEA component of the Internet Email service. Through interactions with instances of this class, the VHEA can provide access to the Internet Email service. This is achieved through implementation of the abstract methods defined in the `Service` class.

Whilst the VHEA is in the IDLE state, the `activate()` method of the `EmailService` class is invoked (as described in section 4.3.3) providing the ability for the service to send emails, check for new emails, and download new emails via instances of the `EmailAgent` within the system.

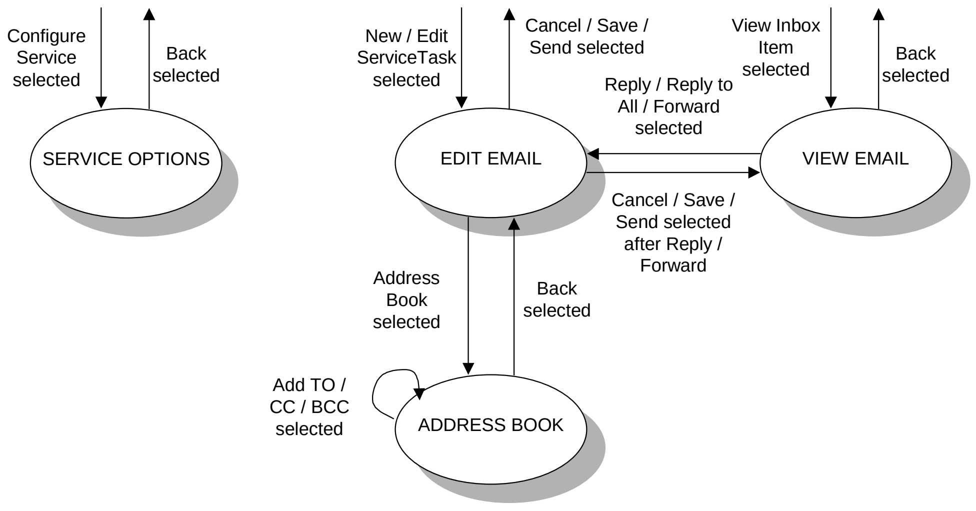

Configure Service selected
Back selected
SERVICE OPTIONS
New / Edit ServiceTask selected
Cancel / Save / Send selected
View Inbox Item selected
Back selected
Reply / Reply to All / Forward selected
EDIT EMAIL
VIEW EMAIL
Cancel / Save / Send selected after Reply / Forward
Address Book selected
Back selected
Add TO / CC / BCC selected
ADDRESS BOOK

Figure 44 - UIState's Defined by EmailService

When the VHEA invokes the `prepareFor()` method of this service, the following steps are taken to prepare the emails in the inbox for the target device:

- List of all email `ServiceItem` is obtained via the `ServiceInbox` interface.
- Iterate through list. For each email:
    - Iterate through attachments:
        - If attachment primary MIME-type (e.g. “image”, “audio” or “application”) is not supported by the target device specified, remove attachment.
- Iterate through list again. For each email:
    - Iterate through attachments:
        - Determine if attachment can be adapted to the capabilities of the target device.
        - If so, adapt the attachment using the appropriate Adaptor class
        - If adapted attachment is smaller that original attachment, replace the original attachment with the adapted attachment. Update the filename of the attachment if the MIME-type of the attachment has changed (e.g. change .bmp extension to .jpg if attachment is converted to “image/jpeg” MIME-type).

Whilst the VHEA is interacting with the user, it provides a number of `UIState` objects to provide `UserInterface` objects that describe the emails contained within the service, as well as the configuration dialog (see Figure 44).

The ADDRESS BOOK `UIState` provides the ability to add emails addresses of contacts defined within the My Contact service. If the user is not subscribed to the My Contacts service, the option to do this is not made available. This is achieved through use of the `ServiceInteraction` interface, which allows a `List` of active services to be provided to the active `EmailService` instance within a VHEA. The `EmailService` can then search through the `List` for an instance of `ContactsService`, and if found request a `List` of `ServiceItem`'s belonging to that service. The `EmailService` simply needs to know the field names for each contacts first name, last name and email address, and it can provide a list of contacts whose email addresses are known. This list can be presented to the user, and the email address of the selected contact can be added to the appropriate `List` within the `ServiceTask` being edited.

4.7 Extensions to core functionality: Terminal Agents

The following sections describe the design of some of the types of Terminal Agent intended to extend the core functionality of the system. The provision at least two TA implementations/types permits requirement 2 to be demonstrated.

4.7.1 Swing Terminal Agent

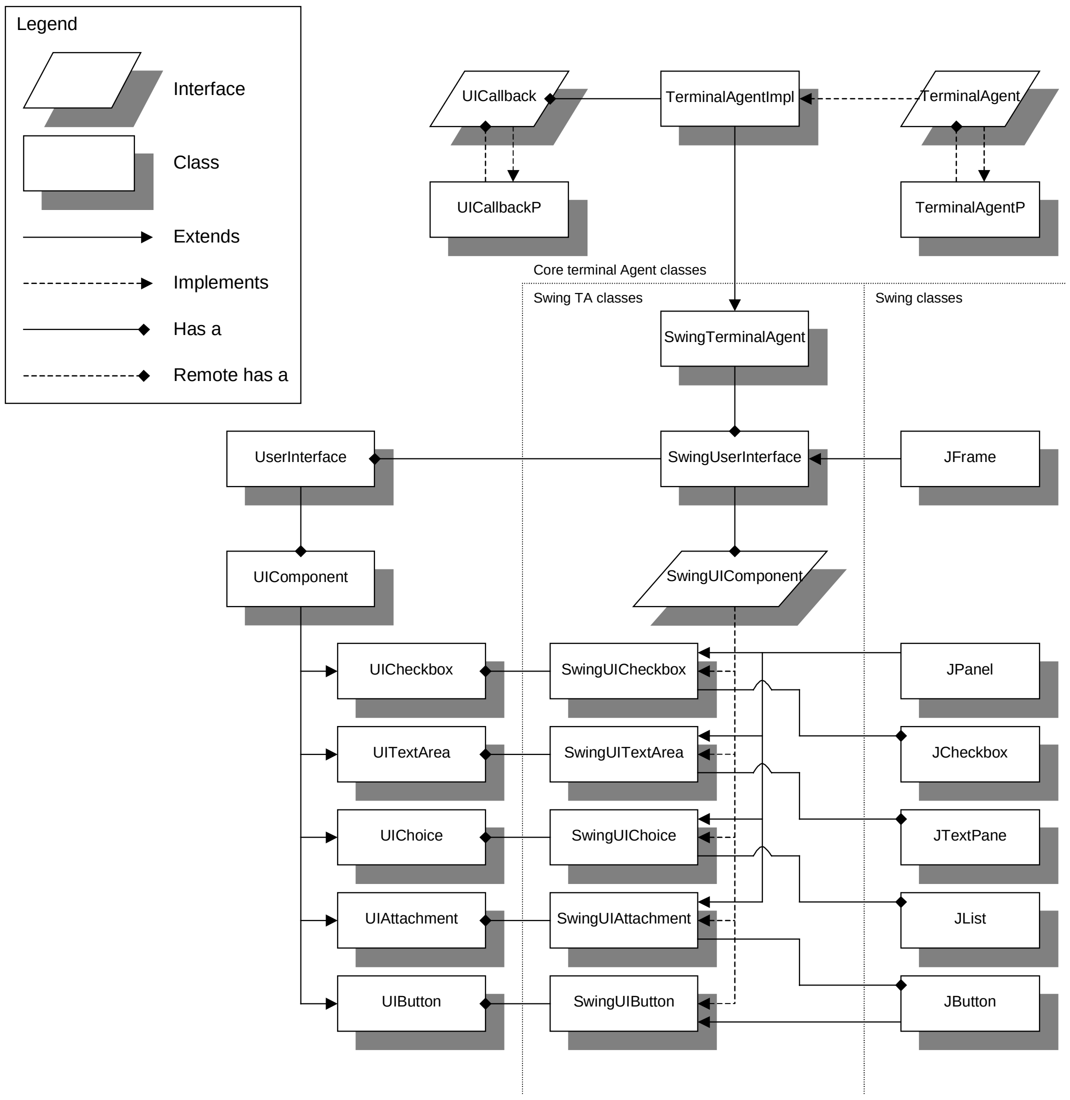


Figure 45 - SwingTA Classes

This TA provides a Java Swing based GUI to interact with a VHEA. It is generally executed directly on a users device, providing the device supports a JVM with Swing.

As Figure 45 highlights, the `SwingTerminalAgent` wraps each of the `UIComponent`'s within a `UserInterface` object with an appropriate `JComponent` sub-class, and these are then added to the `SwingUserInterface` class which is a JFrame. Hence construction of a concrete GUI based upon a `UserInterface` description becomes a trivial matter.

Each `SwingUIComponent` automatically takes care of updating the state of the `UIComponent` it wraps. When a user presses a `SwingUIButton`, the `SwingTerminalAgent` simply passes the original `UserInterface` object back to the VHEA without having to update the individual state of `UIComponent`'s contained within it.

The `SwingUIAttachment` component supports `UIAttachment`'s of the following MIME types:

- "text/plain"
- "text/html" (providing HTML3.2 or earlier is used)
- "image/jpeg"
- "image/gif"
- "audio/wav"
- "audio/aiff"
- "audio/au"
- "audio/rmf"

MIME types not explicitly dealt with can be saved to disk however, since the use of a Swing GUI implies that the device within which the SwingTA is sufficiently powerful that a persistent storage medium with file-system is available.

4.7.2 HTTP Terminal Agent Shell

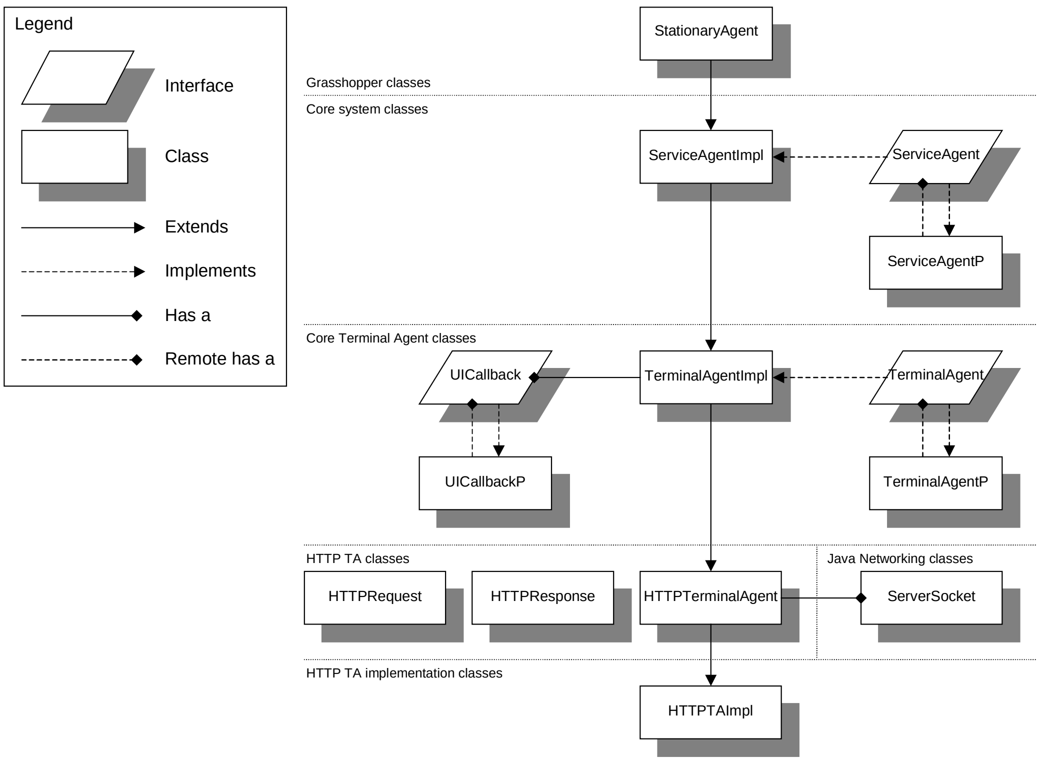


Figure 46 - HTTP-TA Classes

Due to the wide range of possible content types which can be delivered over HTTP (e.g. HTML, WML, XML with XSL etc…), this HTTP-TA shell provides the basic building block for implementing a TA which can interact with a piece of client software which communicates over the HTTP 1.1 protocol ([18], [19]). Figure 46 highlights the how the `HTTPTerminalAgent` classes fit into the `TerminalAgent` hierarchy.

Rather than take the approach of using Java Servlet's [29] to implement this "HTTP Gateway" between HTTP 1.1 clients and VHEA's, the HTTP-TA utilises its own mini-web-server. The reasoning behind this approach is:

- Grasshopper Agent's cannot exist outside a Grasshopper Agency, and the only way to instantiate an Agency is to execute "java de.ikv.grasshopper.Grasshopper" in a new JVM. Hence, a hybrid Servlet/Agent is out of the question.

- From the point of view of deployment, Servlet's are not standalone components. They generally require cumbersome[26] web-server software, which would be many times the size of an Agent.

- More control over the HTTP layer is required than Servlet's provide.

Due to these reasons, a customised mini-web-server exists within the `HTTPTerminalAgent` shell which deals with incoming requests is separate threads. The `handeRequest()` method in implementing sub-classes is invoked for each incoming request.

The `handleRequest()` method has the following signature:

```
protected abstract void handleRequest( HTTPRequest req,
                                       HTTPResponse resp)
```

The `HTTPRequest` object contains information about the incoming request, whilst the `HTTPResponse` object is provided to be populated with the response to the incoming request.

The `HTTPTerminalAgent` also exposes a number of useful method to sub-classes for use when dealing with incoming requests, and to locate other HTTP-TA's to minimise round-trip-delay between the system and a HTTP client:

- `protected static String urlDecode(String in)`
  Decodes a URL-escaped string sequence.

- `protected boolean isPrimaryHTTPTA()`
  Indicates if this is the HTTP-TA dealing with all incoming requests (i.e. has a "well-known" address) – should be used in conjunction with `getNearestTA()` to provide dynamic distribution of incoming requests to other HTTP-TA's of the same type based upon round-trip-delay to the client device.

- `protected String getNearestTA( InetAddress host )`
  Attempts to locate a HTTP-TA which has a shorted round-trip-delay to the given host. This locates all other HTTP-TA's in the Region with the same type, queries them to find their round-trip-delay to the given host, and returns the URL that the client should be redirected towards to minimise the round-trip-delay during UI interactions. If this returns null, there is no other HTTP-TA with shorter round-trip-delay.

An example of the use of `getNearestTA()` is given in Figure 47 through Figure 51.

[26] From the point of view of dynamic Agent deployment – web servers are relatively "static".

Since the `TerminalAgentImpl` shell is actually a SA, each TA can potentially have its own `SystemServiceConfig`. The HTTP-TA make use of this mechanism to determine what port to start the HTTP server on (determined by value of “default_port” in `SystemServiceConfig`), and which host has the “primary” or “central” HTTP-TA (determined by the value of “default_host” in `SystemServiceConfig`).

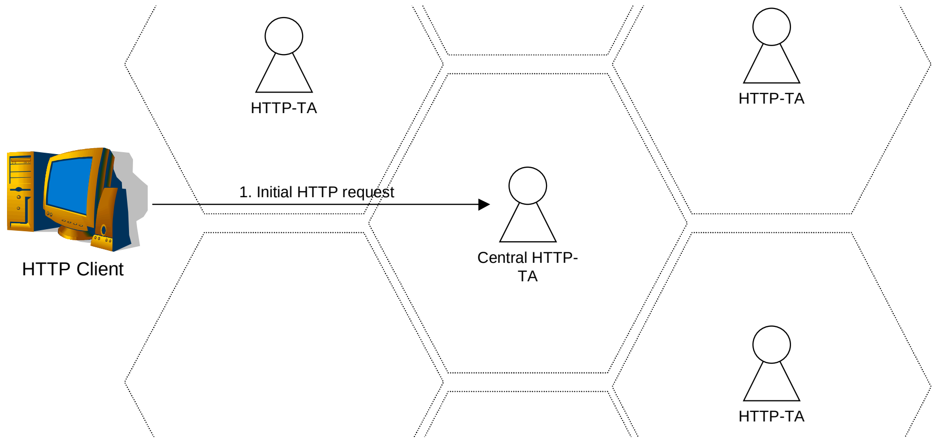


Figure 47 - HTTP-TA Interactions

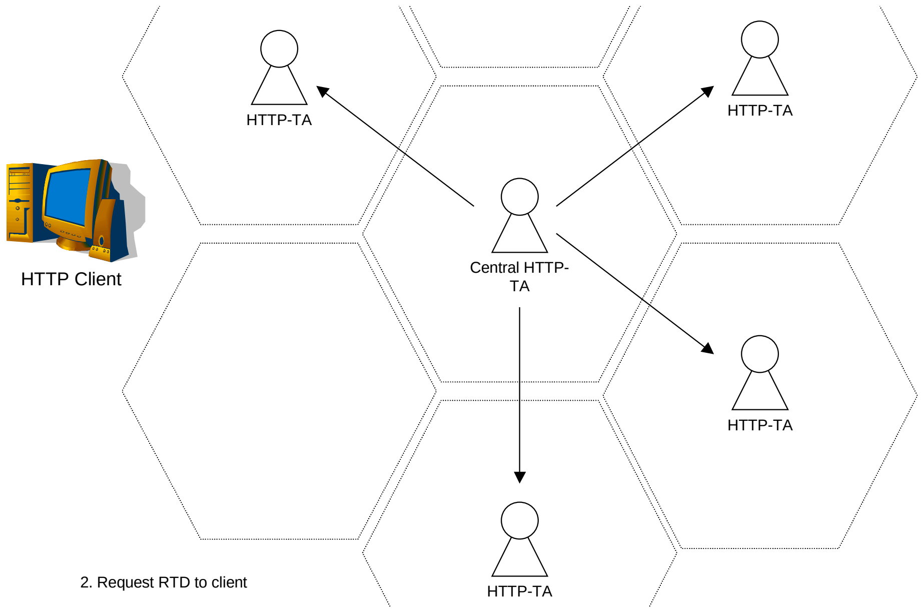


Figure 48 - HTTP-TA Interactions

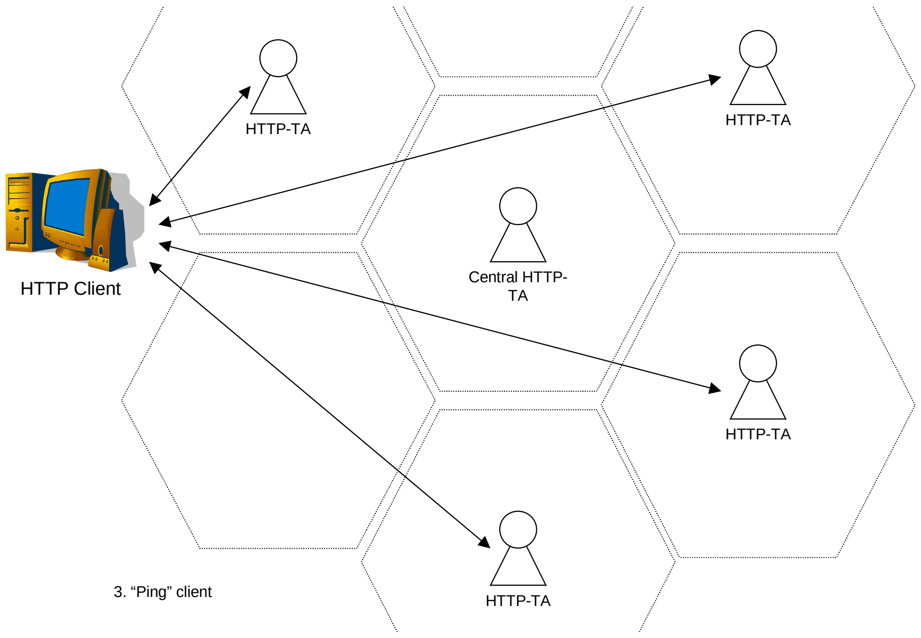


Figure 49 - HTTP-TA Interactions

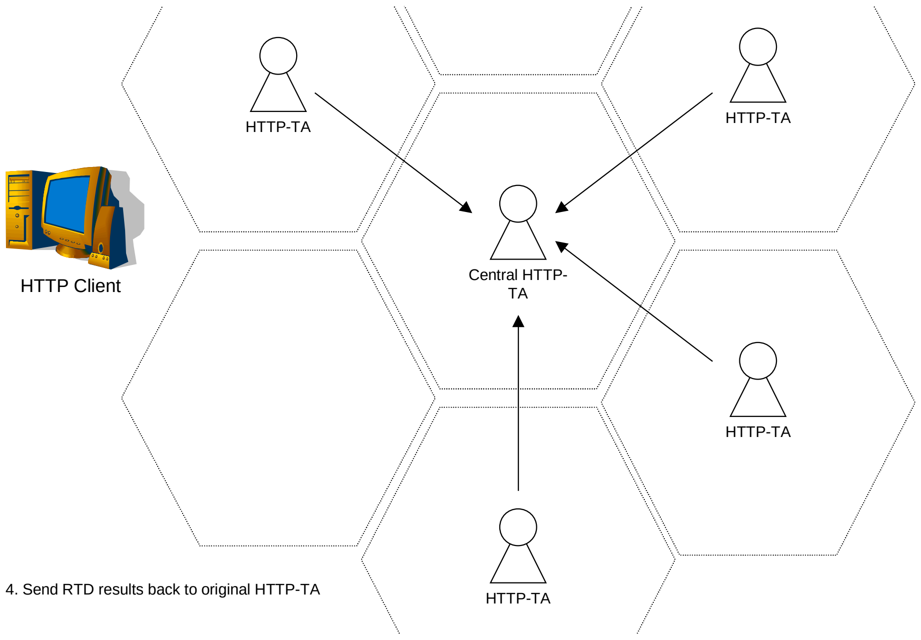


Figure 50 - HTTP-TA Interactions

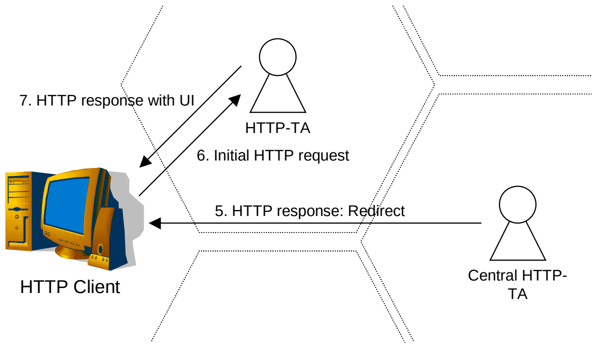


Figure 51 - HTTP-TA Interactions

4.7.3 WAP Terminal Agent

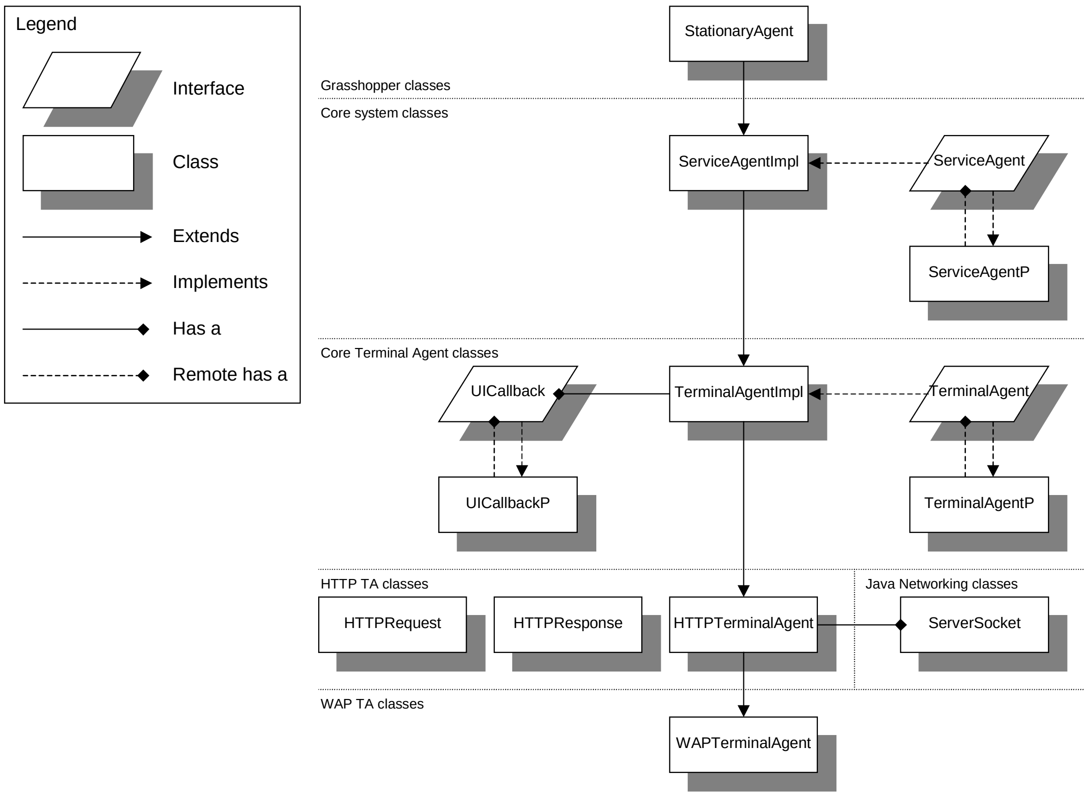


Figure 52 - WAP-TA Classes

Based upon the HTTP-TA shell, this HTTP-TA implementation provides WML [16] content over HTTP for display on a WAP enabled device such as a mobile phone. Figure 52 highlights the class hierarchy for the `WAPTerminalAgent` class.

The WAP-TA simply returns a WML version of `UserInterface` objects, using the following trivial mappings to WML elements:

UIComponent Type	WML Representation
UICheckbox	<select> elements, with <option> tags for “true” and “false”
UIButton	<anchor> element (inline-link)
UIChoice	<select> element, with an <option> element for each choice
UITextArea	For editable text, and <input> field[27]. Otherwise, text is displayed inline.
UIAttachment	N/A

Each editable `UIComponent` within a `UserInterface` is assigned a variable within the corresponding WML deck produced. All inline links representing `UIButton`’s submit the values of these variables back to the WAP-TA, so that the `UIComponent`‘s of the `UserInterface` that the WML deck is representing can be populated with the potentially modified variables.

Along with the details of the variables, the URL used to retrieve the next WML deck contains a session-id that allows an incoming HTTP request to be related to a `UserInterface` and VHEA that the user is interacting with. Since the current state of the WML UI must be tracked in order to ensure correct operation of the UI, use of “back” or

[27] Due to the limitations of WML input fields, editable text is limited to 200 characters. Testing revealed that although the Nokia WAP Toolkit 1.2 could handle much longer text, real WAP devices (such as the Mitsubishi Trium Geo@ used for testing) crash when presented with such a large amount of editable text. See Week 12, 11/6/2000 of Appendix E: Implementation Logbook.

"forward" functionality of the WML browser should not be used[28], since these functions do not generally generate HTTP requests, preventing the current UI being displayed to be tracked.

[28] As a side-effect of ensuring that editable `UIComponent`'s are assigned the correct value within the present WML (since WML browsers cache variable values, resulting in problems when re-displaying a page with variables of the same name), the WML browser history is removed upon entry to each WML deck. This is achieved by specifying that the WML browsers cache should be cleared on entry to each WML deck. Thus back and forward functions are unavailable anyway.

Chapter 5: Evaluation of VHEMAT System

This chapter aim to evaluate that the completed system matches the specification provided (see Chapter 1: Specification), and provide evidence of system "performance" (or "how well the goals of the project have been met") versus a static implementation.

5.1 Requirements Testing

Testing that the requirements of the specification have been met requires that evidence is produced to support this claim that each requirement has been met. The evidence takes either the form of documented design details that correlate with the implemented system (it is assumed that Appendix E: Implementation Logbook is evidence of correct implementation of the design) and/or evidence of the correct functionality of the system via screen-shots of the functioning system.

Requirement 1: *"The system will support multiple users, any number of which can interact with the system at a given time".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the system design is inherently multi-user capable given these assumptions:

- The SMA allows multiple user-profiles to be defined, and multiple VHEA's can exist within the system at once, each representing a user defined in the system via these user-profiles (see section 4.5.2).

- Multiple TA's can exist within the system at once (TA's are a type of SA, and multiple SA's of the same type can exist at once – see section 4.5.2), each allowing interaction between one user and their VHEA at a minimum. The TA shell provides the capability to handle multiple-users at once, although only one session per user (see section 4.3.4).

- The limiting factor depends on the TA's in use within the system. The SwingTA provides only a single user access to the system at a time, due to the limitation of only being able to display a single GUI at a time. The HTTP-TA shell allows multiple HTTP connections at once (see section 4.7.2), and the WAP-TA can handle these multiple connections and multiple user sessions at once (see section 4.7.3) due to inheritance of behaviours from the TA shell and HTTP-TA shell.

Requirement 2: *"There will be support for each user to interact with the system via multiple devices. A number of these may be emulated due to project limitations, and only one may be used by a user at one time".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the system design supports interaction with the system via a number of different devices and mechanisms:

- User-profiles can contain a number of device-profiles, allowing descriptions of all devices that a user will interact with the system via to be contained within a user-profile (see section 4.3.1).

- Multiple types of TA can exist within the system (see evidence for requirement 1). Each TA type can interact with a different sub-set of devices, or at least interact via a different mechanism (see section 4.7 for evidence of different types of TA).

Given the above assumptions, the user can interact with the system via a variety of devices, providing TA implementations are provided to enable communication via these devices.

Requirement 3: *"The system will be developed using DOT and MAT wherever appropriate in order to minimise* ***user interaction overhead*** *and* ***network resource utilisation*** *compared to systems based on client-server solutions".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the system meets this requirement, given the following assumptions in combination with evidence in section 5.4:

- The main pre-design decision made was to adopt Grasshopper as the underlying framework upon which to build the system (see section 4.1). Thus, the core system design and implementation is inherently based upon DOT and MAT.

- The VHEA composes the mobility element of the system (see section 4.5.1), since this Agent is comprised of the information within the system that benefits most from being encapsulated within a mobile Agent.

- The only non-DOT/MAT components to the system are those that integrate with legacy systems (i.e. `EmailAgent` and `HTTPTerminalAgent` shell), due to the limitations of legacy systems. These limitations can be overcome through appropriate deployment of the appropriate Agents however (i.e. deploy an `EmailAgent` onto/close to the SMTP/POP3 servers they will be interacting with; The `HTTPTerminalAgent` shell provides a mechanism for allowing a centralised contact point to be created, whilst also allowing de-centralisation of incoming HTTP requests – see section 4.7.2).

Requirement 4: *"Each user has their own VHE Agent that lives within the system, and acts on their behalf. All user interactions with the system will be carried out via their VHE".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the system design incorporates the notion of an active VHEA within the system for each user:

- The SMA ensures that there is an active VHEA in the system for each user (see section 4.5.2).

Given this assumption, and evidence from section 5.2 (since all user-interactions with the system are defined by the use-cases), the implemented system meets this requirement.

Requirement 5: *"The VHE will provide access to a number of services. These can be subscribed to, un-subscribed from and interacted with via the VHE".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the system allows each VHEA to have a number of services active within it, and that the user can subscribe to, and unsubscribe from available services, given these assumptions:

- Each VHEA can contain a number of services (see section 4.5.1).
- The VHEA presents a UI to the user allowing services to be subscribed to, and un-subscribed from (see section 4.5.1, WAITING FOR CALLBACK state).

Requirement 6: *"The system will be designed with future extension of services in mind, which could be deployed at run-time without stopping the system".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the system design provides scope for a wide variety of services to be developed, and that these services can be deployed at run-time, given these assumptions:

- The Service architecture has few constraints on what can be implemented, especially given the ability to decompose a service into VHEA-side and SA components. The VHEA-side component simply provides mechanisms for interactions with the service, whilst the SA component provides the ability to offset functionality (such as integration with legacy systems) into an environment that is less constrained (see section 4.3.3).
- New service-profiles can be added to the system via the SMA (see section 4.5.2).
- The SMA has the ability to automatically instantiate SA's based upon the service-profiles in the system, hence new service-profiles will result in new SA's being instantiated (see section 4.5.2).
- The SMA distributes details of updated service-profiles to all VHEA's within the system, enabling users to subscribe to new services when a new service-profile is added to the system (see section 4.5.2).

Requirement 7: *"The VHE will provide the ability to configure subscribed services to the users preferences".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the system meets this requirement. Section 4.5.1 highlights that the VHEA allows the user to interact with a UI provided by a subscribed service in order to configure it to the users preferences.

Requirement 8: *"The VHE will provide a high degree of **uniform service access** across devices".*

Evidence: See section 5.4.

Requirement 9: *"The VHE will provide **coherent information** within services to users between devices".*

Evidence: See section 5.4.

Requirement 10: *"The system will persistently store user information. In the event that the system crashes, user information will not be lost".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the system always persistently stores user information within the system, given these assumptions:

- The `Database` object provides a persistent storage mechanism based upon Java serialization (see section 4.5.3, Persistent Storage Mechanism – Database Object).

- The DBA provides the ability for other Agents within the system to add, remove, and perform queries on `Database` instances (see section 4.5.3).

- The SMA always synchronises its internal data-store with the DBA whenever a change occurs in the internal data-store (see section 4.5.2).

- The VHEA always synchronises its internal data-store with the DBA as soon as it receives updated state-information after interaction with its user, and whenever a service modifies the contents of the users inbox, tasklist or user-service-config when in the AT HOME state (see section 4.5.1).

Requirement 11: *"Ensure an appropriate* ***user interface*** *is provided for whatever device the user will be interacting with the system via".*

Evidence: See section 5.4.

Requirement 12: *"The VHE will base decisions on what and how to present information based upon the capabilities of the device and its network connection, which are part of a devices profile".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the VHEA filters and adapts information based upon the device-profile of the device the user is interacting via, given these assumptions:

- Assuming the device-profile is correct (i.e. capabilities described match that of the users device), it will be used as the basis for each of the users' subscribed services to filter and adapt their content (see sections 4.5.1 & 4.3.3), provided the services are implemented "correctly" (i.e. semantics of `prepareFor()` have been maintained).

Requirement 13: *"When the user connects to the system, their VHE will migrate to a host close to the users' device / onto the users device, in order to uphold requirement 3 during user interactions".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the VHEA migrates as stated by the requirement, given these assumptions:

- A VHEA will migrate to the same location as the TA which summoned it to minimise VHEA / TA interaction overhead (see section 4.5.1).

- The summoning TA will be the closest to the user, or on the users device. TA's such as the SwingTA (which provide some form of Java-based GUI) are always located on the users machine (see section 4.7.1). TA's such as the HTTP-TA have the ability to redirect the users' device to the "closest"[29] TA to the user, assuming the user contacts the "central" HTTP-TA (see section 4.7.2).

Requirement 14: *"Before migrating to a node closer to the user (requirement 13) the VHE will set aside / filter / adapt any information it is holding which cannot be presented to the user given the capabilities of the device they are interacting with the system using. This will uphold requirement 3 during migration".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the VHEA filters and adapts information based upon the device-profile of the device the user is interacting via before migration, given these assumptions:

- When a TA contacts the VHEA, it provides the device-id of the device the user is interacting with it via (see section 4.5.1).

- Assuming the device-id is valid (and if not the VHEA will not interact with the user), the device-profile for the device can be located.

- Assuming the device-profile is correct (i.e. capabilities described match that of the users device), it will be used as the basis for each of the users' subscribed services to filter and adapt their content (see sections 4.5.1 & 4.3.3), provided the services are implemented "correctly" (i.e. semantics of `prepareFor()` have been maintained).

- The VHEA will then migrate to the location of the TA.

Requirement 15: *"The user can request that their VHE carry out service-specific tasks, which can be modified or deleted before they are carried out".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the VHEA can be instructed to carry out service-specific tasks, which can be subsequently modified/deleted, given the following assumptions:

- The VHEA displays the service-specific tasks that the VHEA can perform within the appropriate services' menu. If the user selects a task, a UI will be presented allowing the user to specify the details of the task (see section 4.5.1).

- The VHEA allows the user to view the contents of a services' tasklist. From there, tasks can be modified or removed (see section 4.5.1).

[29] Closest in terms of round-trip-delay, not necessarily the physical number of hosts between the device and TA.

Requirement 16: *"The VHE will attempt to carry out tasks assigned to it by its user within the system until they are complete".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the VHEA will attempt to carry out tasks assigned to it, given the following assumptions:

- The VHEA will invoke active services whilst it is in the IDLE state, in order to allow each service to carry out tasks (see section 4.5.1).
- Services will attempt to carry out tasks within the tasklist, and retry at a later point if a task fails (see section 4.3.3).

Requirement 17: *"If the VHE is able to migrate directly to the users device, the network connection can be closed until the user is finished interacting with the VHE".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the VHEA will not attempt to interact with any other Agents other than the TA it is connected to (see section 4.5.1). Hence, if the VHEA is interacting with a TA on the users device, there is no need for a connection to a network.

Requirement 18: *"The VHE will provide the ability for users to modify the profiles of their devices to meet their needs".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the VHEA provides a device configuration UI allowing a user to add, remove and configure device profiles (see section 4.5.1).

Requirement 19: *"The system will allow services to interact with one another in order to accomplish tasks that are impossible to complete alone".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the each `Service` instance with a VHEA has a reference to an object that implements the `ServiceInteraction` interface (see section 4.3.3).

Given this fact, and that the semantics of the interface are maintained by the VHEA, `Service`'s have the ability to interact with other `Service`'s and SA's. Hence the requirement is met.

Requirement 20: *"The system will allow the system administrator to add, remove and configure users details".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the SMA provides the ability to carry out these tasks (see section 4.5.2).

Requirement 21: *"The system will allow the system administrator to add, remove and configure services".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that the SMA provides the ability to carry out these tasks (see section 4.5.2).

Requirement 22: *"The system will attempt to keep track of the device the user is currently using, such that services may attempt to contact the user at that device if necessary".*

Evidence: Through examination of Chapter 3: System Design Overview, it is possible to show that between the VHEA and TA involved in a user interaction, the system contains knowledge of the device the user is interacting with the system via, assuming:

- The primary VHEA is aware which TA a duplicate VHEA has been dispatched to interact with its user (see section 4.5.1).
- The TA is aware of the device the user is interacting with it via (see section 4.3.4).

5.2 Use-case Testing

These tests set out to establish if the desired functionality of the system matches the functionality of the implemented system. In order to accomplish this, the use cases described in section 1.2 have been used as the basis for testing the functionality of the implemented system. Evidence is provided in the form of screen-shots of the system in section A.1.

Use Case	Description	Implemented
1	User connects / logs in to system	☑
2	User disconnects / logs out of system	☑
3	User subscribes to a service	☑
4	User un-subscribes from a service	☑
5	User configures a service	☑
6	User views information from a service	☑
7	User requests that a service-specific task is performed	☑
8	User configures device capabilities profiles	☑
9	System administrator adds a user	☑
10	System administrator removes a user	☑
11	System administrator adds a service	☑
12	System administrator removes a service	☑
13	System administrator configures provider-specific details	☑

Each use-case has been successfully implemented.

5.3 User Interface Testing

This project is not concerned with the best use of HCI principles in generalised user-interfaces (although they have not been ignored), but some gauge of how appropriate the user-interfaces are for the device being used is required to measure how well the primary goal of the project has been met. Since there is no definitive way to measure a metric such as this, a set of user-trials have been arranged in order that a number of potential users could trial the system, and provide their view on the suitability of the user-interfaces for the device(s) they used to interact with the system.

Section A.2 includes the questionnaire that trial-users where asked to complete after using the system, and a summary the collected results. Generally, the opinion of the trial users was favourable concerning the suitability of the UI's for the device they were used on, but their comments highlighted a number of areas where further improvements related to the ease of navigation could be made/investigated.

5.4 Evaluation of System Performance / Goals

In order to measure the success of the project, it is necessary to measure how well the goals of the project have been met.

Given the definition of the primary goal, and of its sub-components (see section 1.1), a measure of how well the primary goal has been achieved can be produced:

- Uniform service access – Due to the relatively limited number of services and devices, this statistic (although providing a measure of success in this instances) is unrealistic. Through the provision of a generalised user-interface that all devices are capable of presenting to the user, and are capable of sending changes back to the system, all devices considered by the implementation can theoretically and practically provide the same functionality given device limitation. Thus, the measure of *uniform service access* from inspection is 100%.

  However, through considering the following assumptions, it can be shown that this rating can be maintained in general:

  - All devices that can access the system support the representation of all generalised user-interface components (except the UIAttachment type, which varies according to device type and capabilities). This can be shown to be true – whether the user-interface is presented via audio only (and a combination of voice-recognition & touch-tone detection), text only or any other currently used interface type, the basic UI components can be represented.

  - The functionality of all services' is accessible via generalised UI components. This can be shown to always be true, since a UIAttachment is a "one-way" UI component (i.e. it is non-modifiable), so feed back is never required by a service from it – hence the capabilities of a device to display a UIAttachment do not effect the functionality of a service.

  - All devices therefore allow the functionality of all services to be provided, the only difference being the representation of UIAttachment components, which are a non-essential to the ability for a user to access the functionality of a service.

  Given the above, in the general case it is likely that a measure of *uniform service access* of 100% is achievable.

- Coherent information – Essentially this measures if an item of information within the system is always available to the user, given the limitations of the device that the user is interacting with the system via. For the limited number of services and devices implemented for the project, it can be shown that this goal has been met.

  As for *uniform service access*, given a number of assumptions this can be shown to be true in general:

  - The VHEA always maintains the most up-to-date user information within the system. This can be shown to be true, given that an exceptional condition such as a duplicate VHEA not returning its updated state to the VHEA does not occur.

  - The user always interacts (indirectly) with a recently created duplicate VHEA. Thus the user always interacts (indirectly) with a VHEA that has the most up-to-date user information contained within it.

- o Filtering of information by the VHEA before duplication and migration (which is based upon the capabilities of the users device), and filtering that occurs at the TA (which is based upon the capabilities of the interaction mechanism being used) is based on the capabilities of the device.

- o Services' provide the same UI's no matter the device presenting the UI to the user. Due to the use of a generalised user-interface description, and lack of context information provided to the service (other than the profile of the device at migration time, which cannot be relied upon other than at the time the `prepareFor()` method is invoked), this can always be shown to be true.

Given these assumptions, the implementation of the system will provide *coherent information* in the general case, given that the presentation of a certain piece of information is dependant upon that devices ability to present it (i.e. described via the devices'-profile and capabilities of mechanism used to interact with the device).

- Appropriate user interface – see section 5.3.

The success of the secondary goals require collection of statistical data based upon the behaviour of the implemented system, and static implementations of similar systems:

- User interaction overhead – where possible, the amount of time taken for a user interface to react to user interactions has been measured for both the implemented system and a number of similar static systems. The collected data is shown in the tables below.

System	User-interface Type	Interaction overhead[30]
Yahoo! Mail (Static, centralised)	HTML	2364 ms
	WML	1237 ms
YourWAP.com (Static, centralised)	HTML	1992 ms
	WML	1284 ms
YAC.com (Static, centralised)	HTML	1251 ms
	WML	524 ms
VHEMAT, WAP-TA	WML	477 ms
VHEMAT, Swing-TA	GUI	<100 ms[31]

User TA	Preparation overhead[32]	Migration overhead[32]	Total
WAP-TA[33]	10 ms	530 ms	540 ms
Swing-TA – Hi-spec device[34]	250 ms	541 ms	791 ms
Swing-TA – Low-spec device[35]	530 ms	1863 ms	2393 ms
Swing-TA – Low-spec device[36] without content adaptation	10 ms	10688 ms	10698 ms

[30] For HTML content, this excludes downloading of images or other extraneous data that a HTML client is not required to download. For WML, this excludes time for conversion to byte-code format. This is an average based upon 10 requests for pages over a 56K modem connection, the pages are simply the index pages to the websites – thus the values stated should be considered the minimum time for a response.
[31] The UI is updated so quickly that the time taken is negligible.
[32] Based upon a VHEAgent containing an email with attachments: JPEG image (10.2k); GIF image (17.1k); Word Document (23.0k).
[33] Refers to a device-type that filters out everything except text data.
[34] Refers to a device that requires very little content filtering/adaptation: High quality graphics, fast/reliable network connection, fast processor/large memory.
[35] Refers to a device that requires a lot of content filtering/adaptation: Low quality graphics, slow/reliable network, slow processor/small memory.
[36] Refers to a device that requires a lot of content filtering/adaptation: Low quality graphics, slow/reliable network, slow processor/small memory.

User TA	Preparation overhead[32]	Migration overhead[32]	Total
Nominal time	10 ms	490 ms	500 ms

In summary:

- o Generally, a user can expect to wait a minimum of 1.25 seconds for a HTML web-based service to react to a user interaction, without considering the amount of time it takes for the service to do anything more meaningful than send back a pre-defined web-page. Compared to the interaction times recorded for the Swing-TA, this is a relatively long time, even for a best-case scenario.

  At times of generally high network traffic, this overhead will increase dramatically. Although this will effect the VHEMAT system, their VHEA will be in a location where the round-trip-delay is minimised or nil, so the VHEMAT system would be no worse under these conditions, and most likely provide a faster response time in comparison.

- o The WAP-TA appears slightly quicker than a WML web-based service, but half a second is still quite slow. The comparatively longer times observed for WML web-based services may be as a result of server load, which the WAP-TA would overcome through dynamically distributing new sessions to appropriate WAP-TA's through the system.

- o A VHEAgent spends a certain amount of time preparing the content it contains for the target device with which it will interact. For a typical email containing a number of attachments, this time ranges between 10-530 ms. Despite this, in situations where a slow network connection is present the total preparation time is far smaller (in the ratio 1:5) than the time it takes for a VHEAgent to migrate with the content in its original form (which is comparable to a static system). Assuming the values obtained can be shown to increase linearly with the size of the content (which can be shown), this holds for all possible types of content.

- Network resource utilisation – the services to be considered when collecting this data are:

  - o Internet Email
  - o My Contacts

  The system will be considered to be deployed either within a Network Service Providers network that deals with users directly (good & average case), or the Service Providers network will be somewhere on the internet (worst case).

  Three types of static system have been highlighted to enable comparison:

  - o Mail/Contacts Web-based Service on Internet (i.e. My Yahoo!)
  - o Mail/Contacts Web-based Service in Service Providers Web-servers (i.e. My Yahoo!, but in Network Service Providers network)
  - o Traditional POP3/SMTP servers.

  These systems will be compared in different configurations, ranging from best (least bandwidth required within all stages of the system) to worst (most bandwidth required).

Statistics have been collected, and models produced based upon the following scenarios. Collected data for these scenarios and values for **within**, **close to**, **within minimum distance of user**, **within average internet-server to user distance** and **within average internet-server to internet-server distance** are defined in section A.3. The results and graphs from the models produced can also be found there (see Figure 53 through Figure 58).

Case Type	No.	Description
Best	1.1	VHEMAT System – Within "Network Service Providers" Network: • Agencies containing EmailAgents' located **within** POP3/SMTP servers • Agencies containing VHEAgents' located **close to** POP3/SMTP server, and **within minimum distance** of user devices/TA's. • (a) Users have devices capable of supporting Swing-TA's • (b) Agencies containing WAP-TA's are located on WAP Gateway's at network extremities **close to** users mobile WAP device.
	2.1	Static System – Mail/Contacts Web-based Service on Internet: • Distance between POP3/SMTP server(s) and Web-server(s) is negligible. • Distance from user to web-server(s) is **within average internet-server to user distance**. • (a) Web-server(s) are providing HTML content • (b) Web-server(s) are providing WML content
	3.1	Static System – Mail/Contacts Web-based Service in "Network Service Providers" Web-servers: • As case 2, except web-server(s) are **within minimum distance of user**.
	4.1	Static System – No 3rd-party System • POP3/SMTP server(s) **within minimum distance of user**.
Average	1.2	VHEMAT System – Within "Network Service Providers" Network: • Agencies containing EmailAgents' located **close to** POP3/SMTP servers • Agencies containing VHEAgents' located **close to** EmailAgents', and **within minimum distance of user** devices/TA's. • (a) Users have devices capable of supporting Swing-TA's • (b) Agencies containing WAP-TA's are located **close to** WAP Gateway's at network extremities **close to** users mobile WAP device.
	2.2	Static System – Mail/Contacts Web-based Service on Internet: • POP3/SMTP server(s) **close to** Web-server(s). • Distance from user to web-server(s) is **within average internet-server to user distance**. • (a) Web-server(s) are providing HTML content • (b) Web-server(s) are providing WML content
	3.2	Static System – Mail/Contacts Web-based Service in "Network Service Providers" Web-server(s): • As case 2, except web-server(s) are **within minimum distance of user**.

Case Type	No.	Description
	4.2	Static System – No 3rd-party System • POP3/SMTP server(s) **within minimum distance of user**.
Worst	1.3	VHEMAT System – Service Based in Internet: • Agencies containing EmailAgents' located **within average internet-server to internet-server distance** of POP3/SMTP servers • Agencies containing VHEAgents' located **close to** EmailAgents', **close to** TA's, and **within average internet-server to user distance** of users device. • (a) Users have devices capable of supporting Swing-TA's • (b) Agencies containing WAP-TA's are located **within average internet-server to user distance** of users' device. WAP Gateway's at network extremities **close to** users mobile WAP device.
	2.3	Static System – Mail/Contacts Web-based Service on Internet: • POP3/SMTP server(s) **within average internet-server to internet-server distance** of Web-server(s). • Distance from user to web-server(s) is **within average internet-server to user distance**. • (a) Web-server(s) are providing HTML content • (b) Web-server(s) are providing WML content
	3.3	Static System – Mail/Contacts Web-based Service in Service Providers Web-servers: • As case 2, except web-server(s) are **within minimum distance of user**.
	4.3	Static System – No 3rd-party System • POP3/SMTP server(s) **within average internet-server to user distance**.

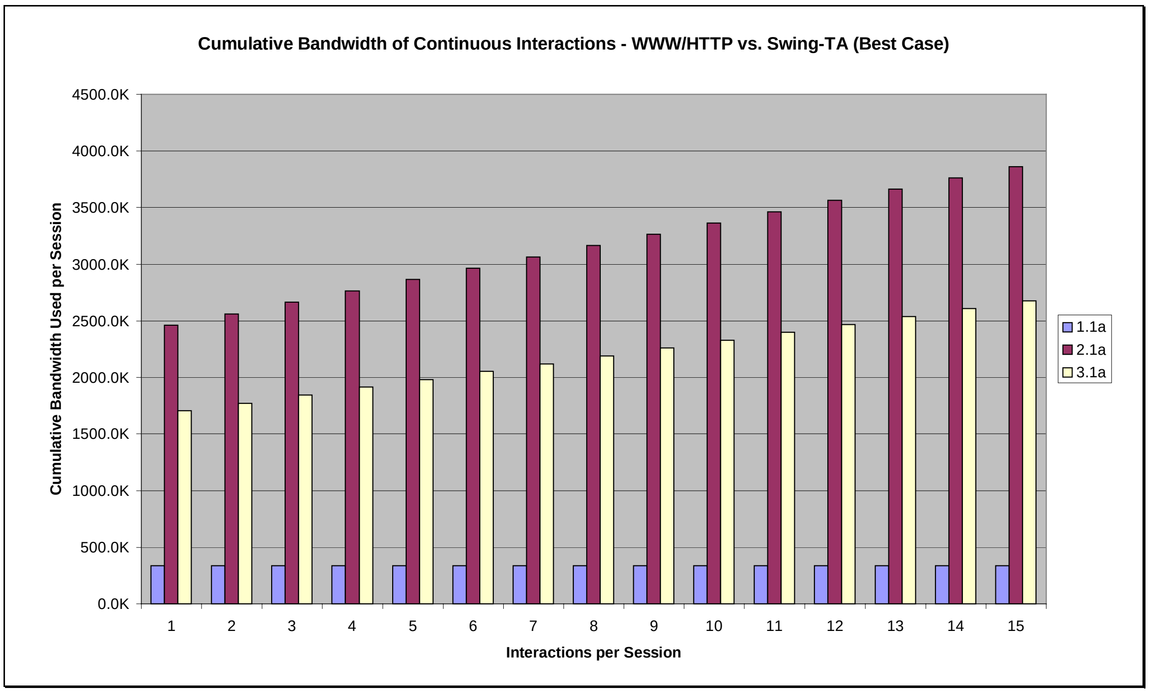


Figure 53 - Graph of Modelled Bandwidth Usage (1)

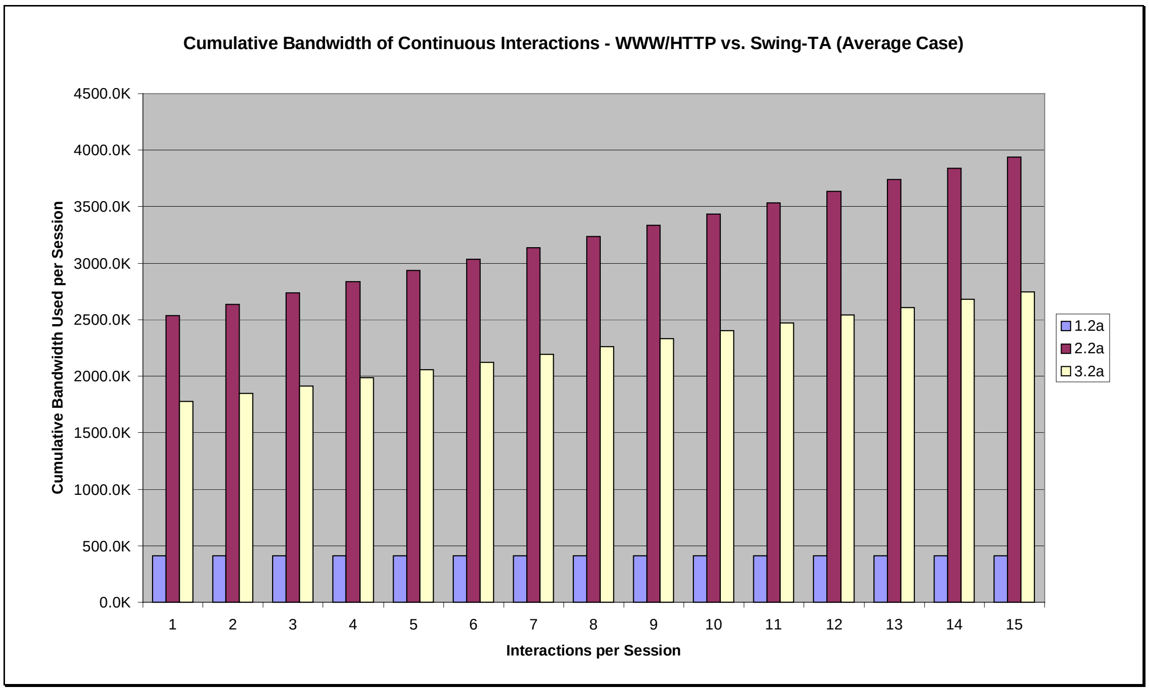


Figure 54 - Graph of Modelled Bandwidth Usage (2)

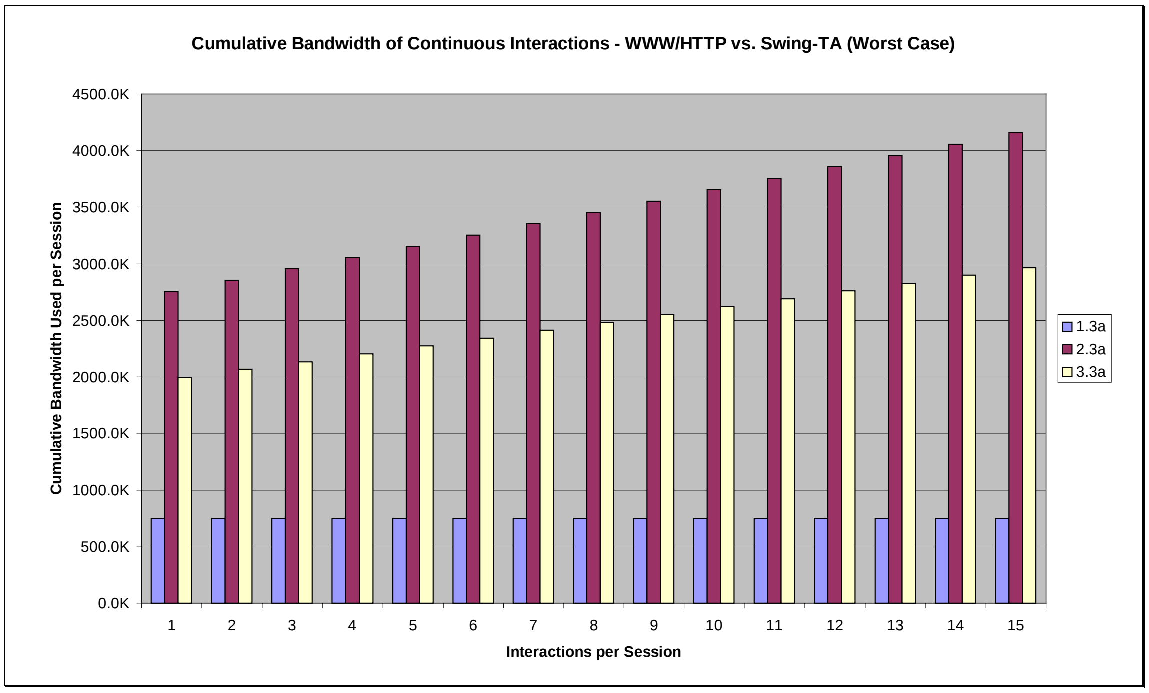


Figure 55 - Graph of Modelled Bandwidth Usage (3)

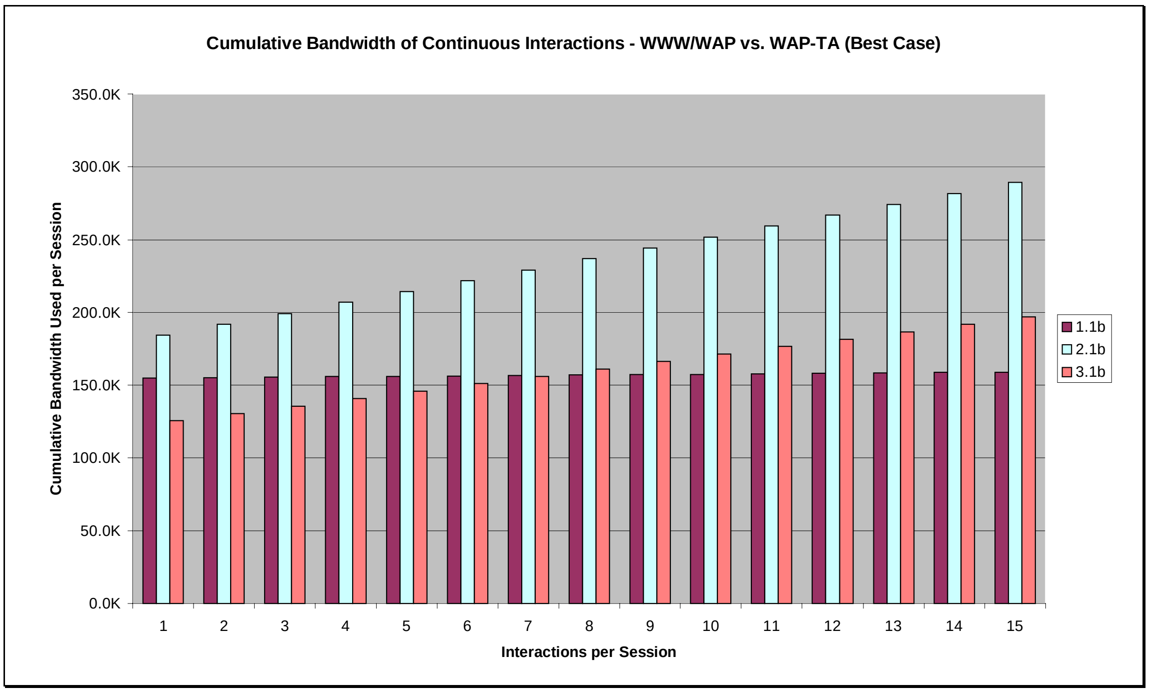


Figure 56 - Graph of Modelled Bandwidth Usage (4)

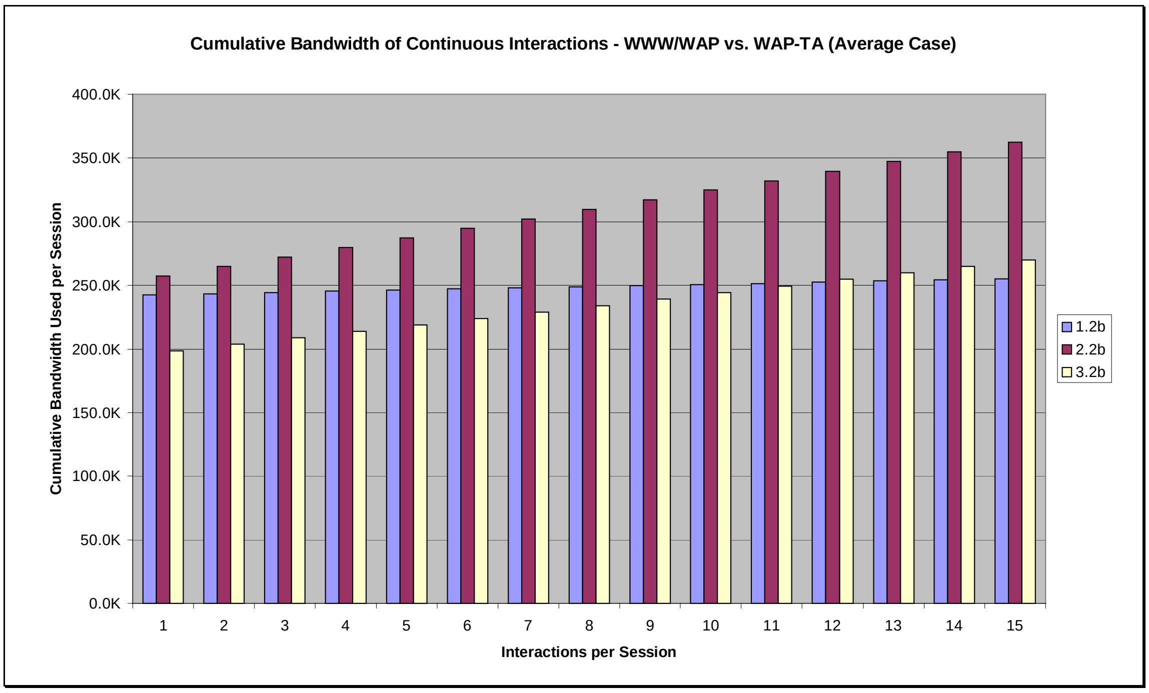


Figure 57 - Graph of Modelled Bandwidth Usage (5)

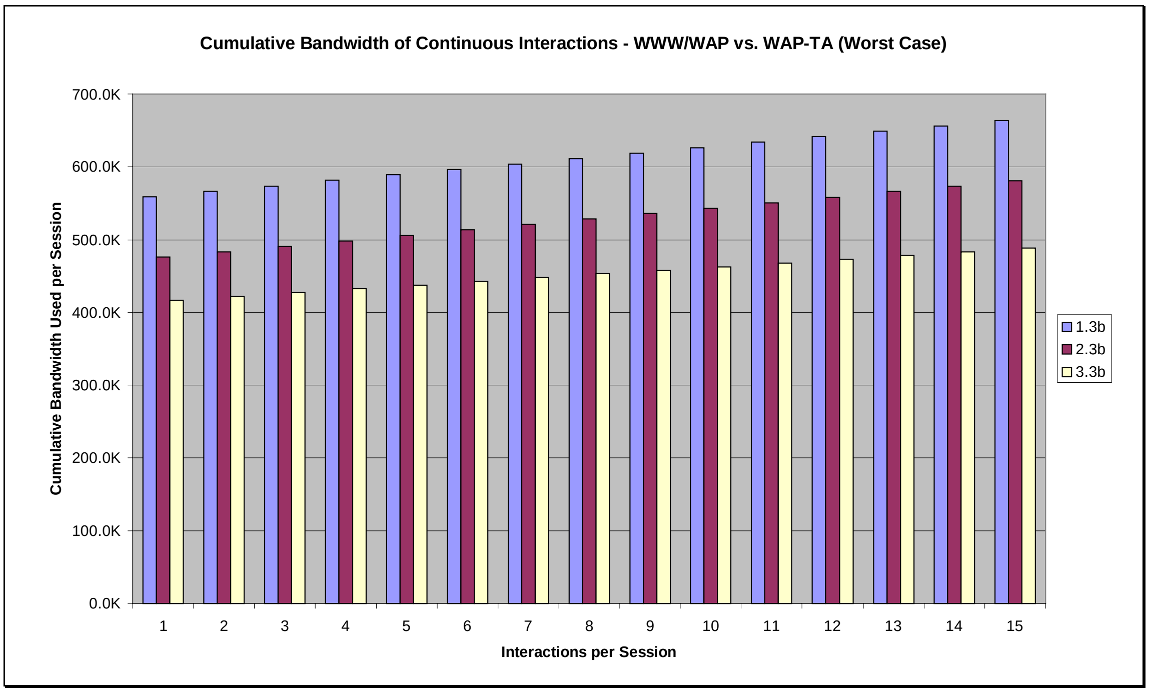


Figure 58 - Graph of Modelled Bandwidth Usage (6)

Summary of the data collected from the models:

- In all cases, the use of the Swing-TA is more efficient bandwidth-wise than centralised systems using HTML to present information to the user. This is primarily due to the efficiency gains provided by content adaptation and because no bandwidth is consumed when interacting with the VHEA locally.

- In some cases, use of the WAP-TA (HTTP-TA) is more efficient than centralised systems – this is generally dependant upon the distance between WAP-TA and WAP-gateway and number of interactions between the WAP-TA and user. In these cases, it is a result of reduced interaction bandwidth consumption due to the WAP-TA being located on the same host as a WAP-gateway, or within a hop of the gateway which results in longer-term benefits due to the comparatively large overhead of migrating the VHEA.

Chapter 6: Conclusion

This chapter attempts to draw conclusions with regard to the goals of the project, highlight the accomplishments of the project, the limitations of the implemented system and possible future work on the system, and the VHE & MAT concept in general.

6.1 Accomplishments

This project has shown that the concept of a VHE based upon MAT is a viable solution to the problem of unifying a disparate collection of services users currently use across a variety of heterogeneous devices. Using MAT and advanced concepts such as intelligent content adaptation and filtering, not only can network utilisation be improved, but the whole "user experience" as well by bringing the intelligence within the network closer to the user.

In summary:

- The implemented system attempted to amalgamate a variety of related ideas from various sources into a highly extendible and adaptable system, with a degree of success. The openness of the system promotes rapid development of new services and the introduction of suitable mechanisms for presenting information to the user, rather than the current commercial view that HTML/WML suits all.

- It has been demonstrated that the use of MAT yields benefits with regard to network resource utilisation, user-interaction overhead, and deployment of new services within the domain of VHE's, in comparison to traditional centralised systems which aim to offer similar unified services. This was achieved due to a combination of "moving" the service logic closer to the user, and through content filtering and adaptation based upon device profiles.

- Generalised user interfaces have been shown as a potential mechanism for representing the same logical user interface across a variety of different devices with differing input/out mechanisms and capabilities. Mechanisms such as this enable coherent information access and uniform service provision in the context of VHE's. In comparison to the current centralised systems available, this is a very flexible mechanism for ensuring that a multitude of devices can be used to access the same services without reliance upon a particular interaction mechanism.

- Overcame a number of seemingly random, non-trivial problems with the system, which were eventually tracked down to the Grasshopper platform itself.

6.2 Limitations

Despite the accomplishments of the project, there are a number of limitations with regard to the implemented system that have been highlighted during the course of the system design/implementation:

- Lack of security. Any "real-world" system such as that implemented would need to consider security implications such as:

    - Having personal user-information within the system, and preventing anyone except the user having access to it.

    - Preventing malicious/faulty hosts modifying the behaviour of MA's.

    - Preventing malicious/faulty MA's from modifying the behaviour of hosts/other Agents.

    - Preventing malicious/faulty MA's from causing DOS attacks through

  - Ensuring that confidentiality of messages is maintained.
  - Preventing un-authorised access to resources and Agents by Agents or other entities.

  There are many more security implications, but these should be regarded as the main threats to system security.

- The generalised user-interface is the limiting factor with regard to the services that can be introduced to the system, since this constrains the information that can be presented to and received by the user. Although suitable for services with non-specialised user-interface requirements, services that require components that are more advanced simply cannot be implemented with the given framework.

- Network utilisation within the proposed type of system is not minimal under some circumstances. This depends on how a user makes use of such a system. For example if the user uses their VHE as a permanent inbox as opposed to removing items after they have been read, the size of the VHE will increase dramatically. The proportion of information that is actually used by the user per session will decrease rapidly, thus making the majority of information redundant and make the system less efficient than centralised systems.

- Grasshopper is used as the underlying Agent platform. A number of serious issues with regards to the functionality of the Grasshopper platform have been highlighted during the course of this project, including:

  - Semantics of local and remote message parsing are completely different, due to the subtle issue of decoupling of objects when serialisation occurs in remote message passing, versus the coupling of objects when objects are passed by reference locally (see Week 12, 10/6/2000 of my logbook).
  - Migration of Agent's does not appear to function correctly – the original version of a migrated Agent is left executing after the migration has occurred, and instance variables within the migrated Agent appear to have different values from the perspective of different threads. This may be related to the issue above.
  - Search functions do not work correctly – searching for an Agent by name always returns no results, compared with searching for Agents with a specific string contained within a name does return results.

  These are quite serious issues with the Grasshopper system. Given the opportunity again I would not opt to use this system – the platform has a well designed and simple to use API, but no amount of programming simplicity can make the system more robust when the system has major bugs at its very core.

- There are only a limited number of services. I would have liked to implement more services to highlight some of the more advanced concepts that could be incorporated into the VHE. Unfortunately the time-scale for implementation of the project was to limited to enable this, although in hind-sight I should have spent less time getting the SMA GUI "right" than I did.

6.3 Possible Extensions

There are a number of areas in which the project could be extended, some of which are highlighted below:

6.3.1 New Services & Terminal Agents

Due to the generality provided by the design of the system, one possible course for future extension of the project would be to introduce more elaborate services. For example, a service that makes use of its ability to be activated whilst the VHEAgent is idle within the network to intelligently search for information that may be of interest to its user would highlight the usefulness of Agents within the network.

The other area of extension that the system was designed to handle is he addition of more Terminal Agents to handle not only other interaction mechanisms, but allow the system to be integrated with new devices in the future.

The WAP-TA could be improved to handle images or other content types (if they become adopted by the WAP consortium). Alternatively, due to the use of generalised user-interface descriptions, creating a number of new Terminal Agents' would provide an excellent means of stretching the ability of this mechanism to represent user-interfaces. For example, a Telephony Terminal Agent that interacts with the user via a touch –tone phone keypad and voice recognition.

6.3.2 Improved Content Adaptation and Translation

Due to time constraints, this project was only able to consider adaptation of images and "application" data. The image adaptation could be improved due to the limitation that only JPEG images may be encoded with any deal of improvement in image data size with the Java extensions currently available.

Audio adaptation has not been considered. The Swing-TA has the ability to deal with audio, so all that would be required to include audio adaptation is an API to achieve this, and integration with the existing services.

Another mechanism that could be introduced is that of content translation, where a content-type is converted to another content-type whilst retaining its meaning (e.g. text to audio and audio to text translation).

6.3.3 Improved Generalised User-Interface

As mentioned previously, the generalised user-interface is the limiting with regards to the services that can be incorporated within a system such as that implemented. More research needs to be carried out into suitable ways to generalise the user-interfaces of more advanced features that services may offer, such as presentation of real-time video or audio, telephony features and the mechanisms that would be required in order to enable these. Some mechanism for dynamically matching the capabilities of a device with the generalised user-interface components available to a service could be investigated.

Within the scope of the produced system, trial users' comments highlighted the need for the UI's produced in the case of the Swing-TA to be less "clunky". This could be achieved by either:

- Investigating adding intelligence to the Swing-TA in order to produce better looking GUI's through some model of what a good user-interface looks like.

- By improving the descriptive power of the generalised-user-interface with regards to spatial relationships between components.

6.3.4 Content Filtering Based Upon User Requirements

As described previously, if the system is used improperly the bandwidth utilisation benefits of using a system such as that proposed by this project are made completely redundant. This could be overcome by allowing the user to select the information to filter based upon device type and context, or by introducing some intelligence into the system to make a best-guess at the information a user will want to access.

6.3.5 Security

One of the major limiting factors of a system such as that implemented is the provision of security. This project avoided investigating this in too much detail due to the current complexities of ensuring security within such a system, and the time-scales involved in the development of the system.

If any system such as that proposed by this project were to be produced commercially, security would be of paramount concern. As mentioned earlier, the development of security conscious Agent platforms such as Ajanta will enable the Agent security issues to be handled by the underlying platform, leaving only the application-level security to be dealt with, significantly reducing the amount of effort required to incorporate the required security into such a system.

6.3.6 User Billing

If the system produced were to be taken into a commercial environment, the issue of billing would have to be considered. Some services would of course be free, but advanced services may incur charges on a per session or task basis. The problems of tracking service usage and billing the user appropriately for their usage of services must be considered and incorporated into the systems functionality in order for this to be achieved.

6.4 Conclusion

Despite the limitations of the system produced, I feel that the project has been successful in its aims to highlight the suitability of MAT within the domain of VHE's, and of VHE's within the general area of unified service provision. The approach of adopting MAT in order to provide a VHE to nomadic users appears sound enough, enabling dynamic movement of functionality about the network, seamless distributed operation of the system and non-centralised interaction mechanisms.

Within the broader scope of "real-world" applications, the adoption of VHE technologies within the UMTS specification will enable some of the benefits highlighted by this project to be brought to a wider user-base. However quickly UMTS is adopted by users, there will no doubt be a large proportion of legacy technology left in place for many years to come. The use of MAT within UMTS will enable not only service mobility within UMTS networks and connected devices, but the ability to bring these services to other "legacy" systems that will no doubt be popular for the foreseeable future.

It is also conceivable that traditional services such as My Yahoo! could adopt MAT as an alternative means of interacting with its services for the benefit of mobile users for the very reasons highlighted by this project. In an ideal world services could even be mixed and matched from various traditional service providers, but this would require development of an open standard for VHE service implementations or wide uptake of the UMTS VHE specification.

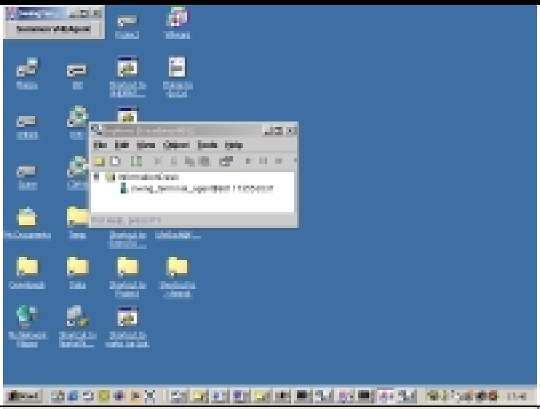
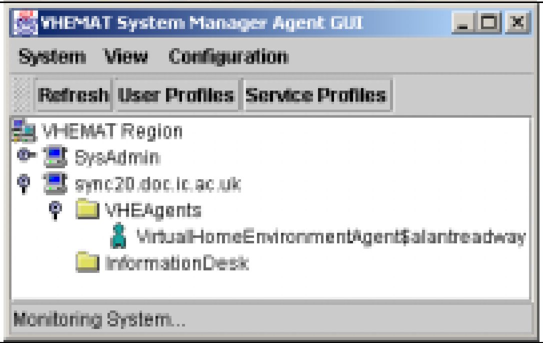
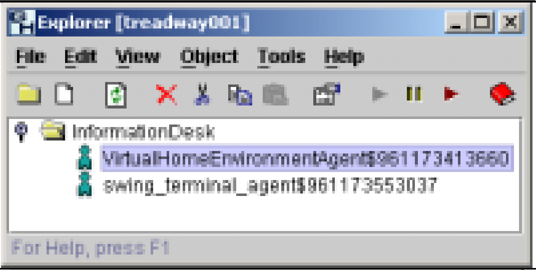
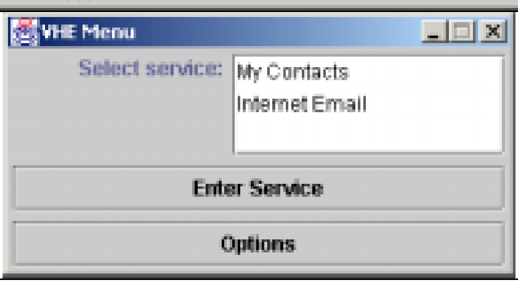
The issue of security is hindering the uptake of MAT outside of research groups however, and with good cause. In order for MAT to become used more widely and within the context of commercial applications, a robust and secure platform must be available in order to safeguard against eavesdroppers or malicious Agents introduced into the system. Moreover, an interoperable solution is required in order to remove restrictions as to where MA's can migrate within the network. Solutions such as the OMG's MASIF specification and the FIPA-ACL communication mechanisms are good steps in a direction to enable interoperability between platforms from different manufacturers, but much work is still required to enable Agent migration across heterogeneous Agent platforms.

Once these problems are overcome, I look forward to the day when commercially available systems really allow me to use any service, anywhere, anytime and through any device.

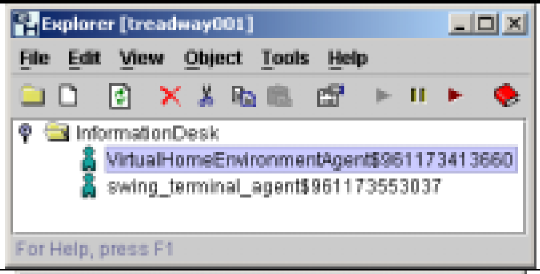
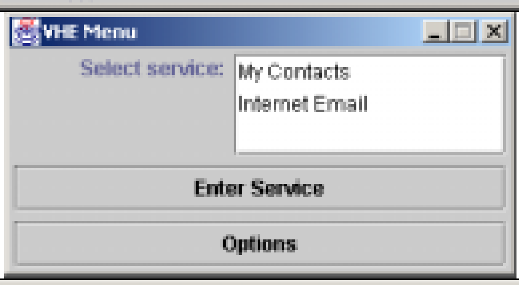
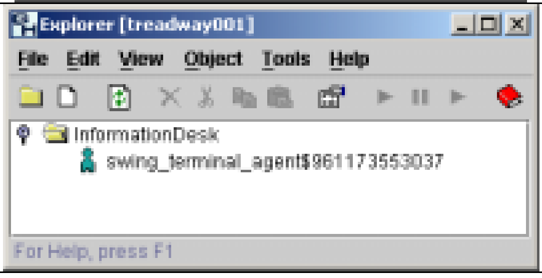
Appendix A: Evidence of System Functionality

A.1. Evidence of Implementation of Use Cases

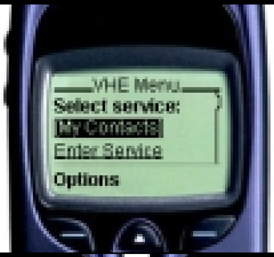
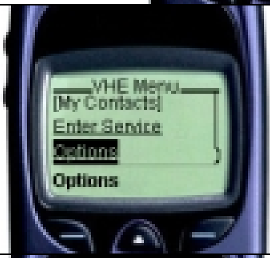
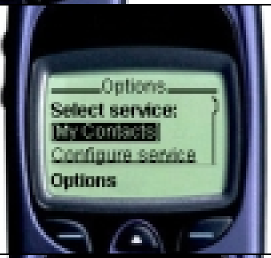
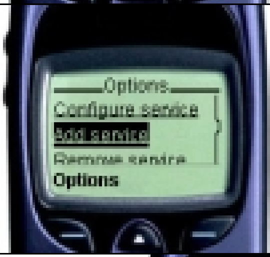
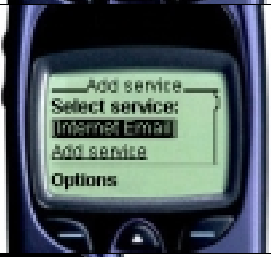
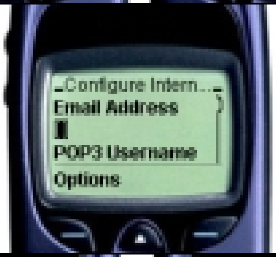
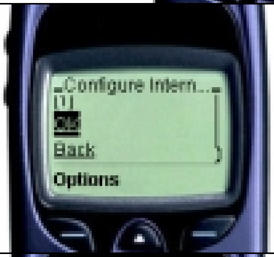
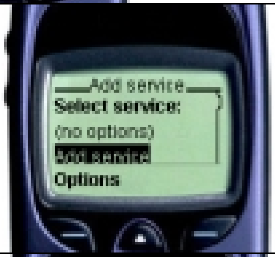
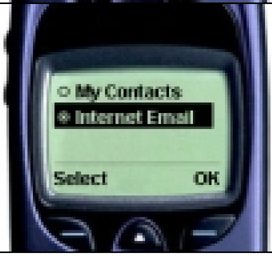
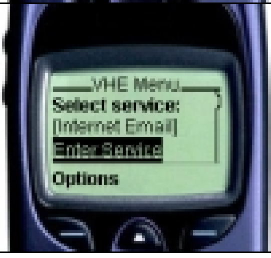
Use Case 1: User connects / logs in to system

Screen-shot	Description
	The user “alantreadway” starts a SwingTA on their PC (username, password & device id have been obtained previously).
 VHEMAT System Manager Agent GUI System View Configuration Refresh User Profiles Service Profiles VHEMAT Region SysAdmin sync20.doc.ic.ac.uk VHEAgents VirtualHomeEnvironmentAgent$alantreadway InformationDesk Monitoring System...	The user’s VHEA is located within a Place in the network
 Explorer [treadway001] File Edit View Object Tools Help InformationDesk VirtualHomeEnvironmentAgent$961173413660 swing_terminal_agent$961173553037 For Help, press F1	After the user selects the “Summon VHEAgent” button, a duplicate VHEA is dispatched to the same Place as the SwingTA
 VHE Menu Select service: My Contacts Internet Email Enter Service Options	The SwingTA displays the initial UI from the VHEA.

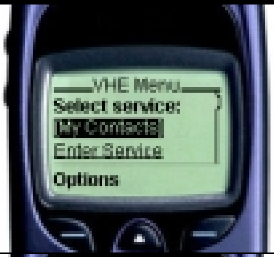
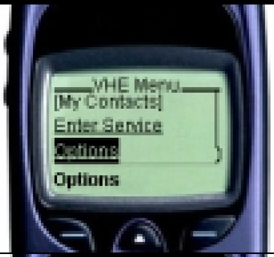
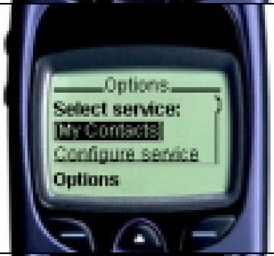
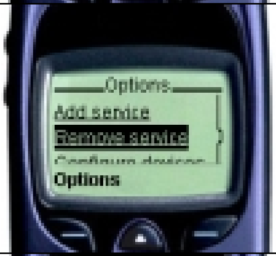
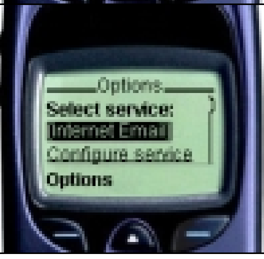
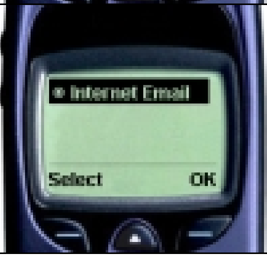
Use Case 2: User disconnects / logs out of system

Screen-shot	Description
 Explorer [treadway001] File Edit View Object Tools Help InformationDesk VirtualHomeEnvironmentAgent$9611734l3660 swing_terminal_agent$961173553037 For Help, press F1	The user “alantreadway” is interacting with a duplicate VHEA via a SwingTA.
 VHE Menu Select service: My Contacts Internet Email Enter Service Options	The user clicks the close button on the current window – this indicates that the session is over.
 Explorer [treadway001] File Edit View Object Tools Help InformationDesk swing_terminal_agent$961173553037 For Help, press F1	The duplicate VHEA sends its state back to the original VHEA, and removes itself.

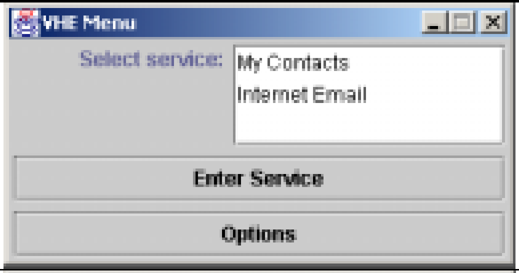
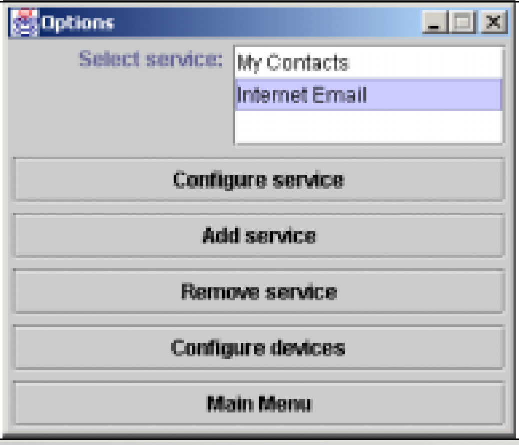
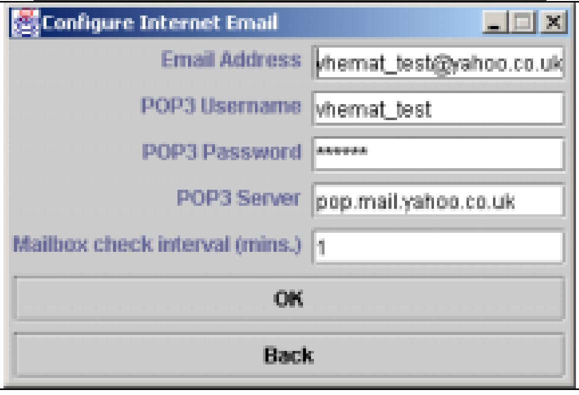
Use Case 3: User subscribes to a service

Screen-shot	Description
	User is interacting with VHEA via WAP-TA & WAP-enabled phone
  Options Select service: My Contacts Configure service Options	User selects options menu, and is presented with a number of choices.
 Options Configure service Add service Remove service Options  Add service Select service: Internet Email Add service Options	User selects “Add Service”, and is then presented with a list of un-subscribed services.
	User selects to add “Internet Email”, and is presented with the configuration options for that service.
  Add service Select service: (no options) Add service Options	User completes details, and selects OK. Service has now been removed from the “Add Service” UI.
  VHE Menu Select service: [Internet Email] Enter Service Options	“Internet Email” is now available from the main VHE menu.

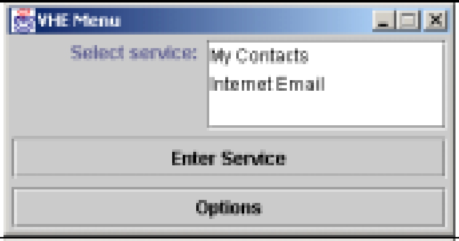
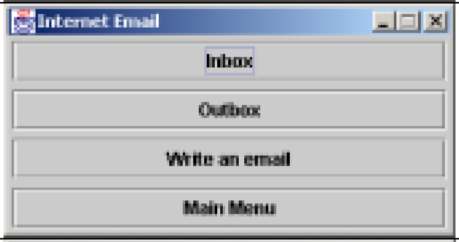
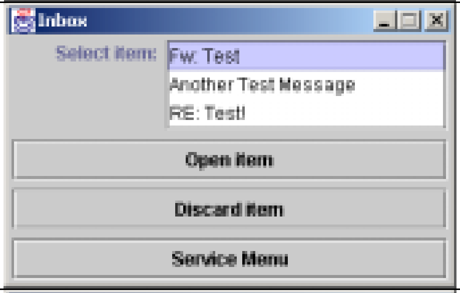
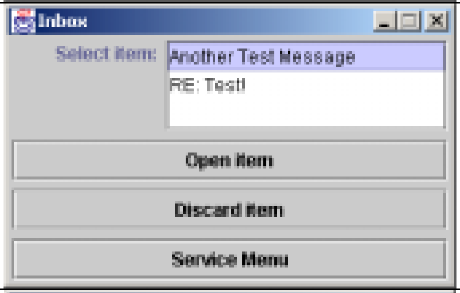
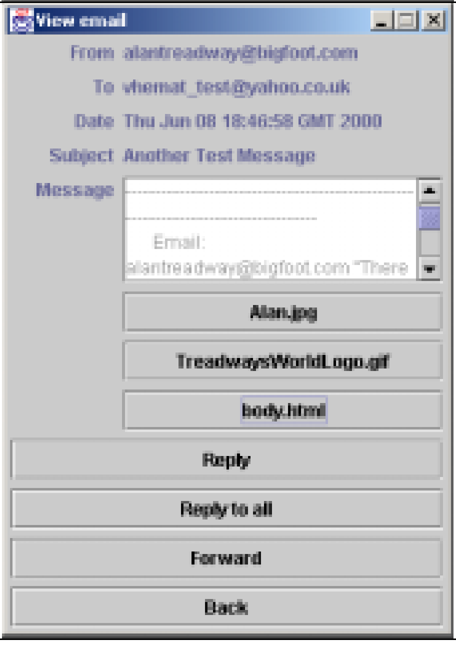
Use Case 4: User un-subscribes from a service

Screen-shot		Description
 VHE Menu Select service: [My Contacts] Enter Service Options	 VHE Menu [My Contacts] Enter Service Options Options	User is interacting with VHEA via WAP-TA & WAP-enabled phone. User enters the “Options” menu.
 Options Select service: [My Contacts] Configure service Options		User selects the “My Contacts” service, and then selects the “Remove Service” link.
 Options Select service: [Internet Email] Configure service Options		“Internet Email” is now the only subscribed service.

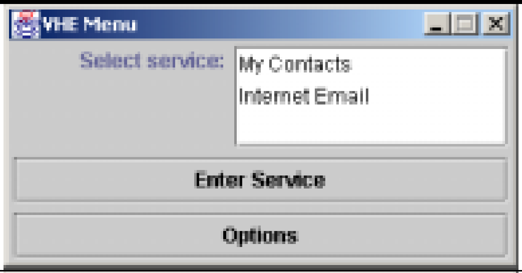
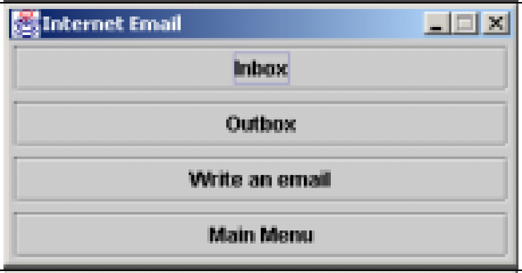
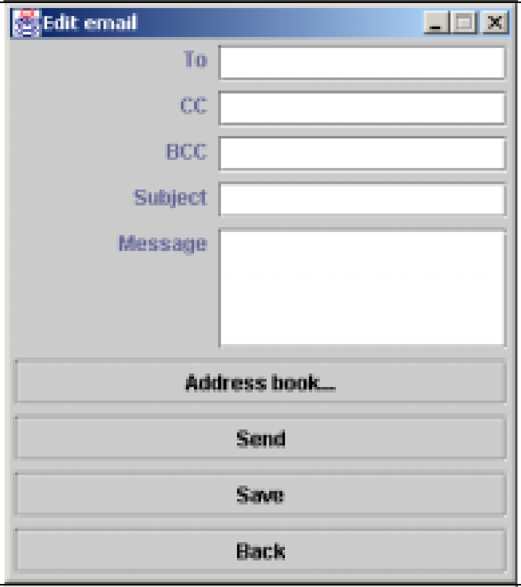
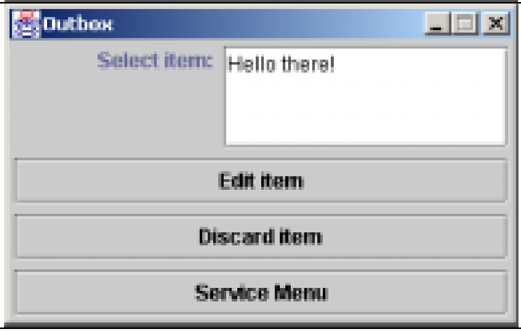
Use Case 5: User configures a service

Screen-shot	Description
 VHE Menu Select service: My Contacts Internet Email Enter Service Options	User is interacting with VHEA via Swing-TA. User enters the "Options" menu.
 Options Select service: My Contacts Internet Email Configure service Add service Remove service Configure devices Main Menu	User selects the "Internet Email" service and presses the "Configure service" button.
 Email Address: vhemat_test@yahoo.co.uk POP3 Username: vhemat_test POP3 Password: ****** POP3 Server: pop.mail.yahoo.co.uk Mailbox check interval (mins.): 1	User is presented with the service-specific options UI. Selecting OK confirms the changes, and updates the UserServiceConfig.

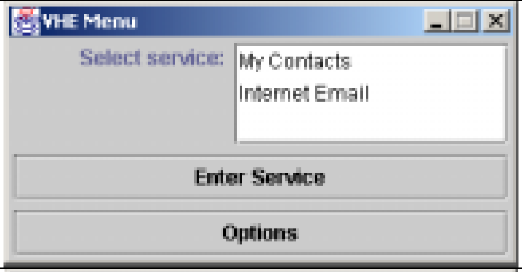
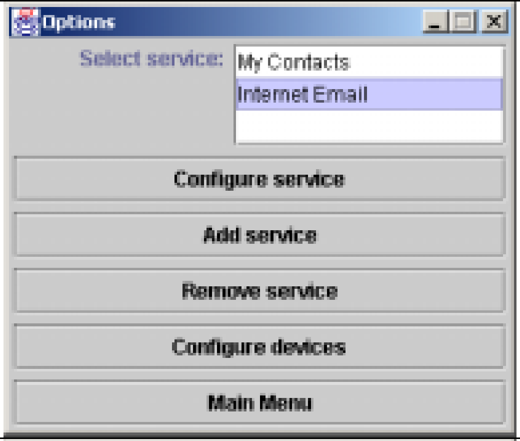
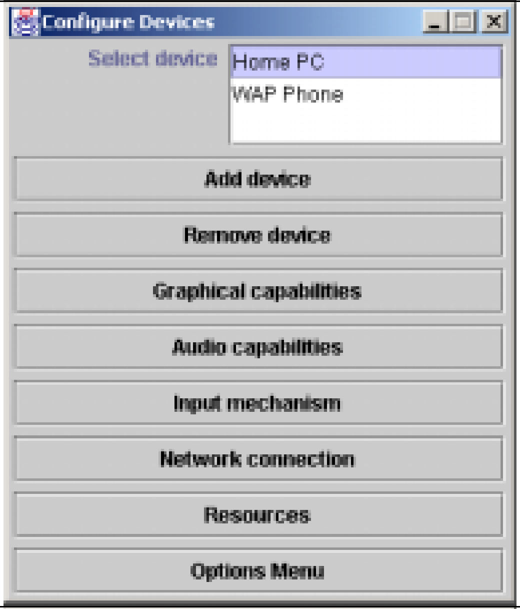
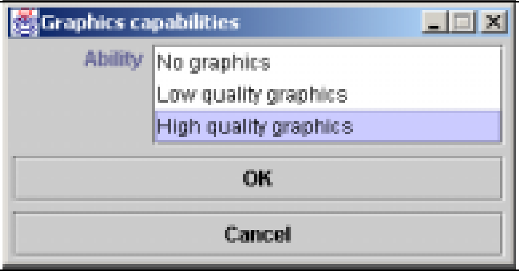
Use Case 6: User views information from a service

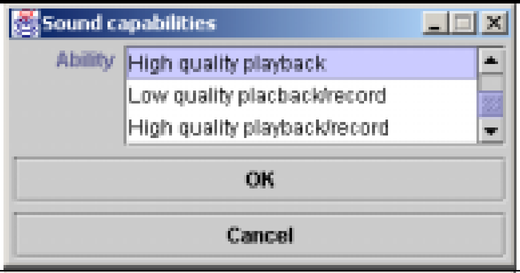
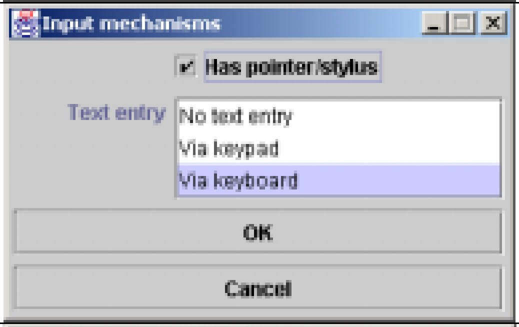
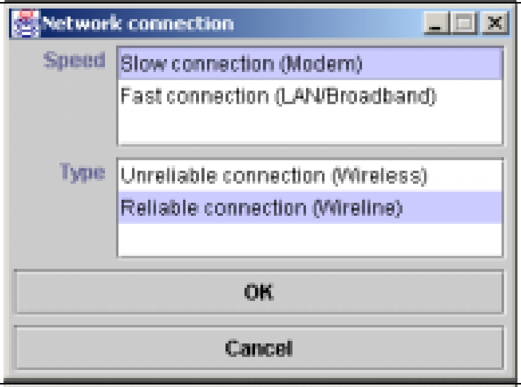
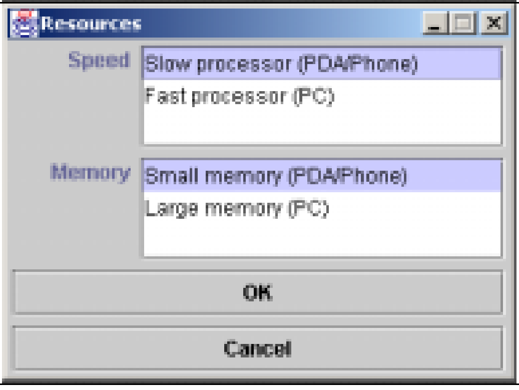
Screen-shot	Description
 VHE Menu Select service: My Contacts Internet Email Enter Service Options	The user is interacting with VHEA via Swing-TA. User selects the "Internet Email" service, and presses the "Enter Service" button.
 Internet Email Inbox Outbox Write an email Main Menu	The "Internet Email" service menu is displayed. The user selects the "Inbox" button.
 Inbox Select item: Fw: Test Another Test Message RE: Test! Open item Discard item Service Menu	The user is presented with a list of emails contained in the inbox. The user selects an email, and presses the "Discard item" button.
 Inbox Select item: Another Test Message RE: Test! Open item Discard item Service Menu	The previously selected email has been removed. The user now selects another email and presses the "Open item" button.
 From alantreadway@bigfoot.com To vhemat_test@yahoo.co.uk Date Thu Jun 08 18:46:58 GMT 2000 Subject Another Test Message Message Email: alantreadway@bigfoot.com "There Alan.jpg TreadwaysWorldLogo.gif body.html Reply Reply to all Forward Back	The email selected is displayed. Service-specific options such as "Reply", "Reply to all" and "Forward" are presented to the user.

Use Case 7: User requests that a service-specific task is performed

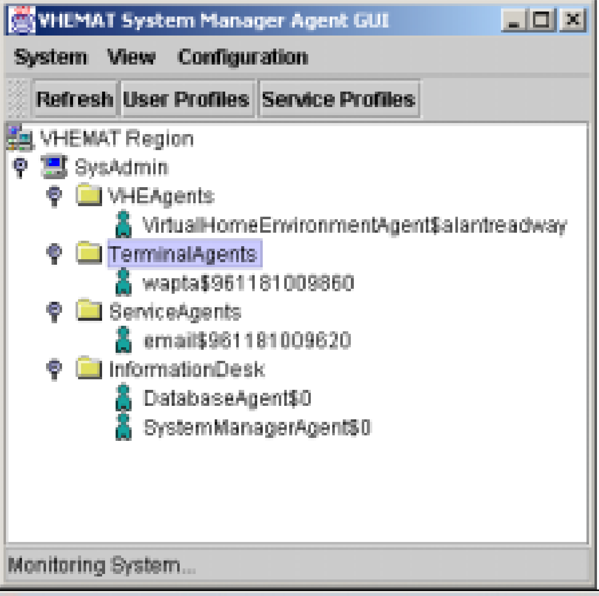
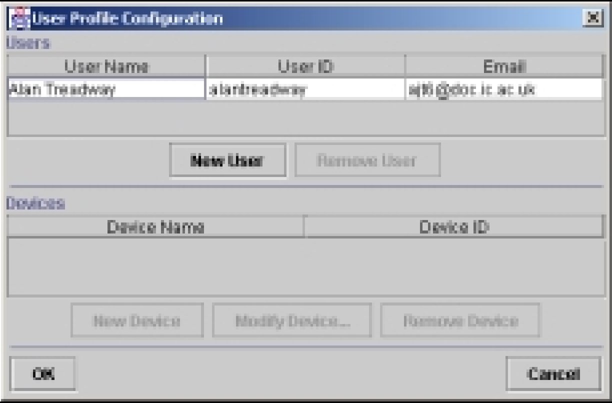
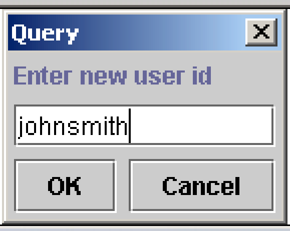
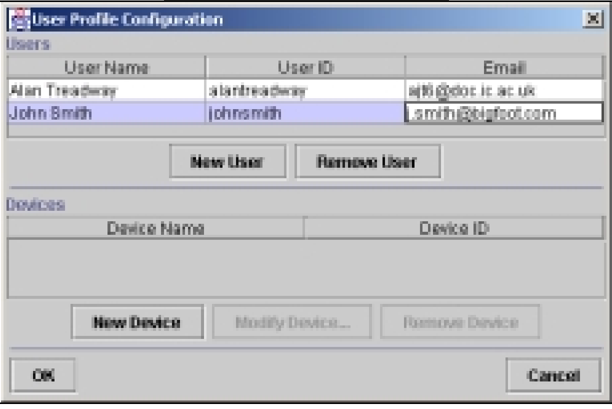
Screen-shot	Description
 VHE Menu Select service: My Contacts Internet Email Enter Service Options	The user is interacting with VHEA via Swing-TA. User selects the “Internet Email” service, and presses the “Enter Service” button.
 Internet Email Inbox Outbox Write an email Main Menu	The user selects “Write an email”, which is a service-specific task.
 Edit email To CC BCC Subject Message Address book... Send Save Back	A service-specific UI is presented, to allow the user to write an email. The user enters the required information into the fields, and presses “Send”.
 Outbox Select item: Hello there! Edit item Discard item Service Menu	The message is then visible in the outbox of the “Internet Email” service, ready to be delivered once the VHEA returns to the network.

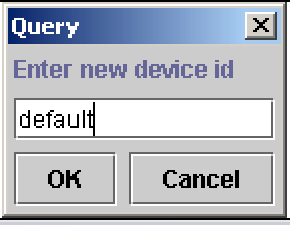
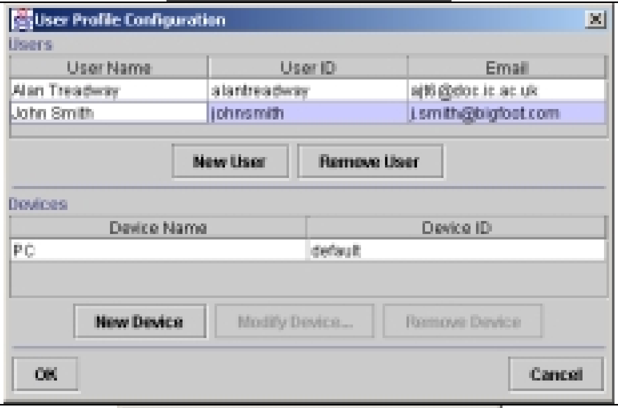
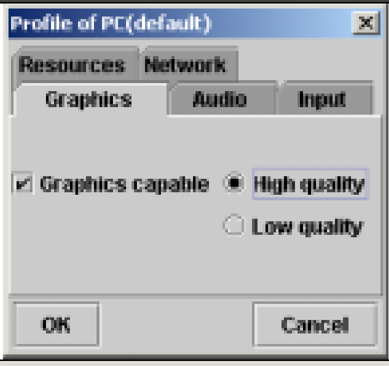
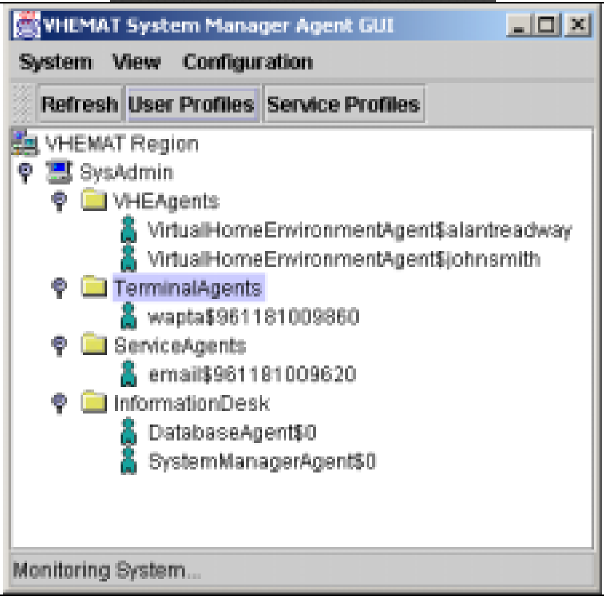
Use Case 8: User configures device capabilities profiles

Screen-shot	Description
	The user is interacting with VHEA via Swing-TA. User selects the "Options" button, to enter the options menu.
	The user then selects the "Configure devices" button.
	A list of device profiles (belonging to the user) within the system are displayed. The user selects the "Home PC" device profile, followed by the "Graphical capabilities" button.
	A list of possible configuration settings is provided. The user selects "High quality graphics", and presses "OK". The previous menu is displayed again.

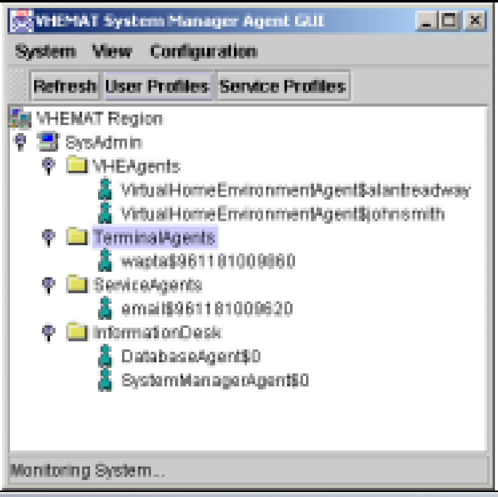
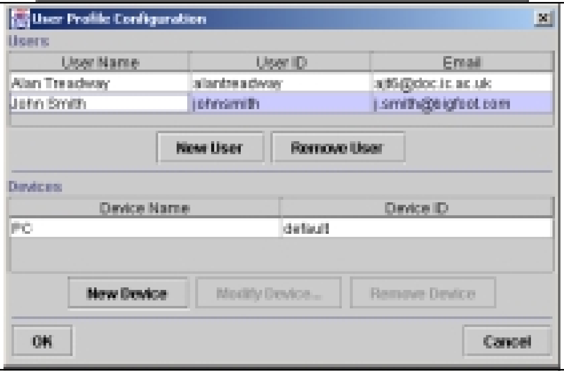
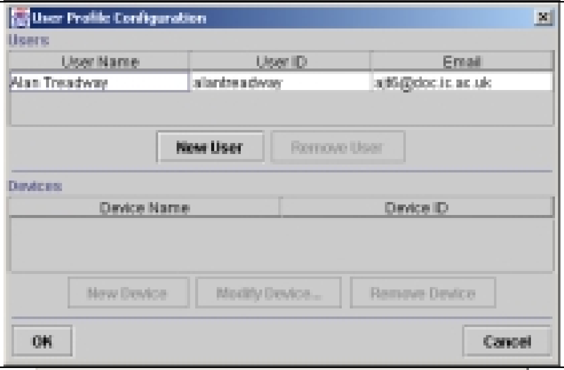
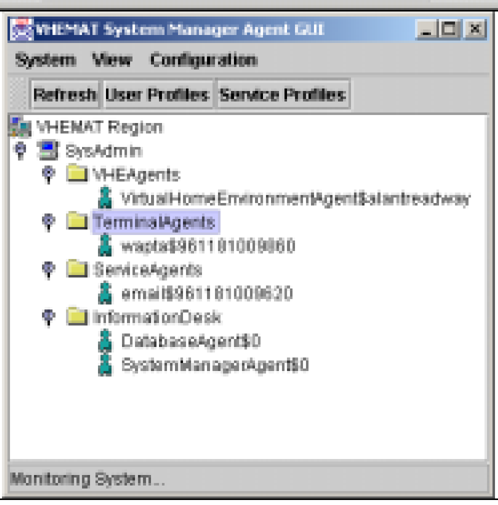
Screen-shot	Description
 Sound capabilities Ability: High quality playback / Low quality placback/record / High quality playback/record OK Cancel	The user now selects “Audio capabilities”. A list of possible options is displayed. The user selects “High quality playback”, and presses “OK”.
 Input mechanisms ✔ Has pointer/stylus Text entry: No text entry / Via keypad / Via keyboard OK Cancel	The user now selects “Input mechanism”. A list of possible options is displayed. The user selects “Via keyboard”, and presses “OK”.
 Network connection Speed: Slow connection (Modem) / Fast connection (LAN/Broadband) Type: Unreliable connection (Wireless) / Reliable connection (Wireline) OK Cancel	The user now selects “Network connection”. A list of possible options is displayed. The user presses “Cancel”.
 Resources Speed: Slow processor (PDA/Phone) / Fast processor (PC) Memory: Small memory (PDA/Phone) / Large memory (PC) OK Cancel	The user now selects “Resources”. A list of possible options is displayed. The user presses “Cancel”.

Use Case 9: System administrator adds a user

Screen-shot	Description
 VHEMAT System Manager Agent GUI System View Configuration Refresh User Profiles Service Profiles VHEMAT Region SysAdmin VHEAgents VirtualHomeEnvironmentAgent$alantreadway TerminalAgents wapta$961181009860 ServiceAgents email$961181009620 InformationDesk DatabaseAgent$0 SystemManagerAgent$0 Monitoring System...	The SysAdmin (System Administrator) has to add a new user to the system. They select the “User Profiles” toolbar button.
 User Profile Configuration Users New User Remove User Devices Device Name Device ID New Device Modify Device... Remove Device OK Cancel User Name: Alan Treadway; User ID: alantreadway; Email: ajt6@doc.ic.ac.uk	The User Profile Configuration GUI is displayed. Only one user-profile currently exists. The SysAdmin selects “New User”.
 johnsmith	The SysAdmin is prompted for the username to be associated with the new user.
 User Profile Configuration Users New User Remove User Devices Device Name Device ID New Device Modify Device... Remove Device OK Cancel User Name: Alan Treadway; User ID: alantreadway; Email: ajt6@doc.ic.ac.uk User Name: John Smith; User ID: johnsmith; Email: j.smith@bigfoot.com	A new user-profile is created. The SysAdmin can now enter other details into the user-profile just created. The SysAdmin selects “New Device”, to configure a default device for the user.

Screen-shot	Description
 Enter new device id default	The SysAdmin is prompted for the device-id of the new device.
 User Profile Configuration New User Remove User New Device Modify Device... Remove Device OK CancelUsers User Name · User ID · Email Alan Treadway · alantreadway · ajt6@doc.ic.ac.uk John Smith · johnsmith · j.smith@bigfoot.com Devices Device Name · Device ID PC · default	The device profile is then created. The SysAdmin selects the new device-profile, and clicks "Modify Device…"
 Profile of PC(default) Resources Network Graphics Audio Input OK Cancel[✓] Graphics capable ● High quality ○ Low quality	A device-profile configuration GUI is displayed, allowing the device-profile to be modified from the standard settings. The user exits the device-profile configuration, and clicks "OK" in the user-profile configuration GUI.
 VHEMAT System Manager Agent GUI System View Configuration Refresh User Profiles Service ProfilesVHEMAT Region SysAdmin VHEAgents VirtualHomeEnvironmentAgent$alantreadway VirtualHomeEnvironmentAgent$johnsmith TerminalAgents wapta$9611810098600 ServiceAgents email$961181009620 InformationDesk DatabaseAgent$0 SystemManagerAgent$0 Monitoring System...	The SMA automatically populates the system with a new VHEAgent for the new user.

Use Case 10: System administrator removes a user

Screen-shot	Description
	The SysAdmin (System Administrator) has to remove a user from the system. They select the “User Profiles” toolbar button.
	The user to be removed is selected. The “Remove User button is pressed.
	The User Profile Configuration now reflects the fact that only one user profile exists. The user presses “OK”.
	The VHEA belonging to the user that was removed is removed by the SMA.

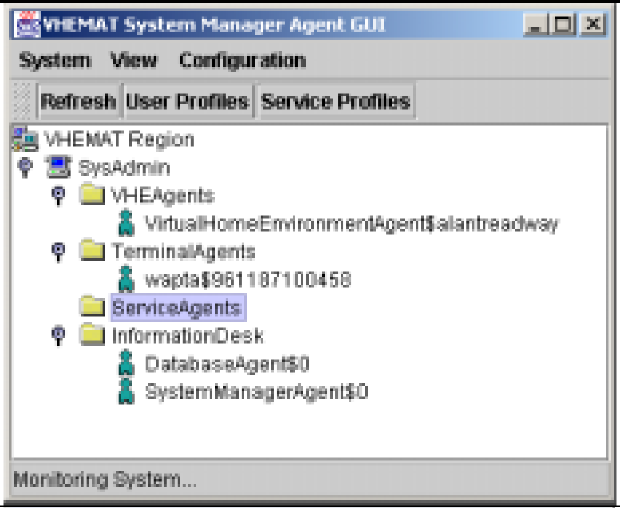
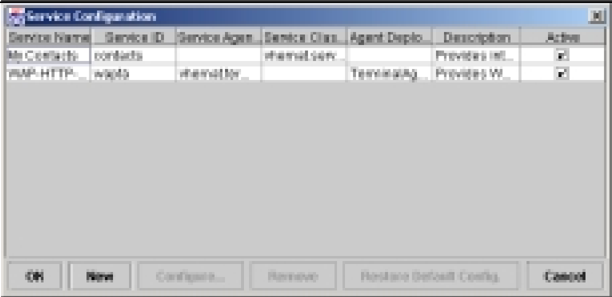
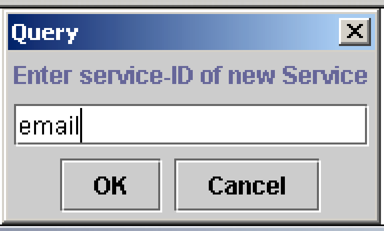
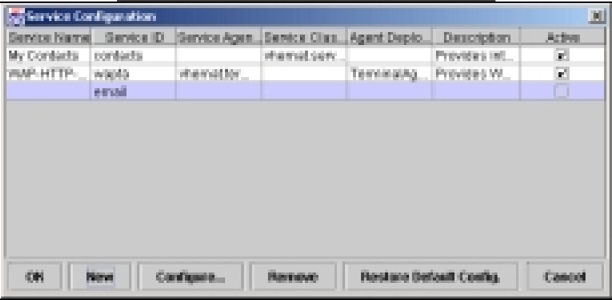
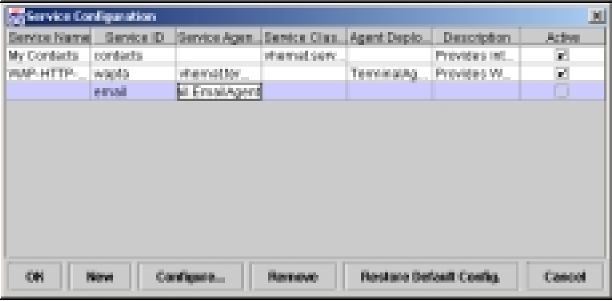
VHEMAT System Manager Agent GUI
System View Configuration
Refresh User Profiles Service Profiles
VHEMAT Region
SysAdmin
VHEAgents
VirtualHomeEnvironmentAgent$alantreadway
VirtualHomeEnvironmentAgent$johnsmith
TerminalAgents
wapta$9611810009860
ServiceAgents
email$9611810009620
InformationDesk
DatabaseAgent$0
SystemManagerAgent$0
Monitoring System...

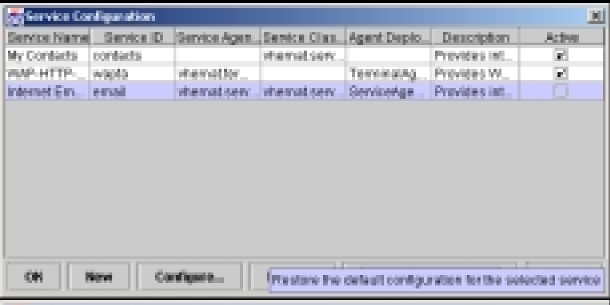
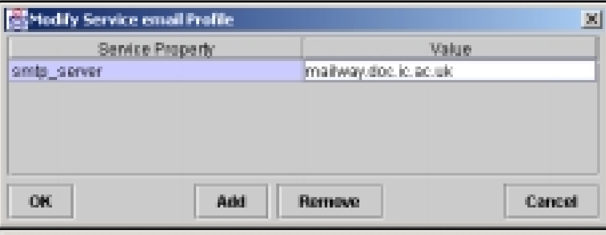
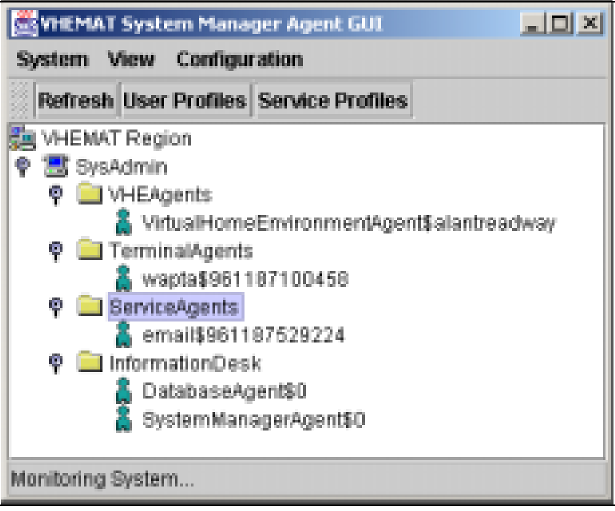
User Name	User ID	Email
Alan Treadway	alantreadway	ajt6@doc.ic.ac.uk
John Smith	johnsmith	j.smith@bigfoot.com

Device Name	Device ID
PC	default

User Name	User ID	Email
Alan Treadway	alantreadway	ajt6@doc.ic.ac.uk

Use Case 11: System administrator adds a service

Screen-shot	Description
 VHEMAT System Manager Agent GUI System View Configuration Refresh User Profiles Service Profiles VHEMAT Region SysAdmin VHEAgents VirtualHomeEnvironmentAgent$alantreadway TerminalAgents wapta$961187100458 ServiceAgents InformationDesk DatabaseAgent$0 SystemManagerAgent$0 Monitoring System...	The SysAdmin (System Administrator) has to add a new service to the system. They select the "Service Profiles" toolbar button.
 Service Configuration Service Name | Service ID | Service Agen... | Service Clas... | Agent Deplo... | Description | Active My Contacts | contacts | | vhema1.serv... | | Provides inf... | ✓ WAP-HTTP... | wapta | vhema1.tor... | | TerminalAg... | Provides W... | ✓ OK New Configure... Remove Restore Default Config Cancel	The SysAdmin is presented with a list of current service profiles. The SysAdmin presses the "New" button.
 Query Enter service-ID of new Service email OK Cancel	The SysAdmin is prompted for the service-id of the new service-profile.
 Service Configuration Service Name | Service ID | Service Agen... | Service Clas... | Agent Deplo... | Description | Active My Contacts | contacts | | vhema1.serv... | | Provides inf... | ✓ WAP-HTTP... | wapta | vhema1.tor... | | TerminalAg... | Provides W... | ✓ | email | | | | | OK New Configure... Remove Restore Default Config Cancel	A new service-profile is created.
 Service Configuration Service Name | Service ID | Service Agen... | Service Clas... | Agent Deplo... | Description | Active My Contacts | contacts | | vhema1.serv... | | Provides inf... | ✓ WAP-HTTP... | wapta | vhema1.tor... | | TerminalAg... | Provides W... | ✓ | email | al.EmailAgent | | | | OK New Configure... Remove Restore Default Config Cancel	The SysAdmin enters the class-name of the SA associated with the new service, and presses "Restore Default Config."

Screen-shot	Description
	The default service-profile and system-service-configuration are automatically loaded. The SysAdmin selects "Configure…"
	A list of system-service-configuration name-value pairs is displayed, ready to be modified. The SysAdmin enters details of the SMTP server to use, and presses "OK".
	The SysAdmin activates the service, then presses "OK" again. The SMA then populates the system with SA's belonging to the new service.

Service Configuration

Service Name	Service ID	Service Agen…	Service Clas…	Agent Deplo…	Description	Active
My Contacts	contacts		vhematserv…		Provides inf…	✓
WAP-HTTP-…	wapta	vhematter…		TerminalAg…	Provides W…	✓
Internet Em…	email	vhematserv	vhematserv	ServiceAge…	Provides inf…	

Modify Service email Profile

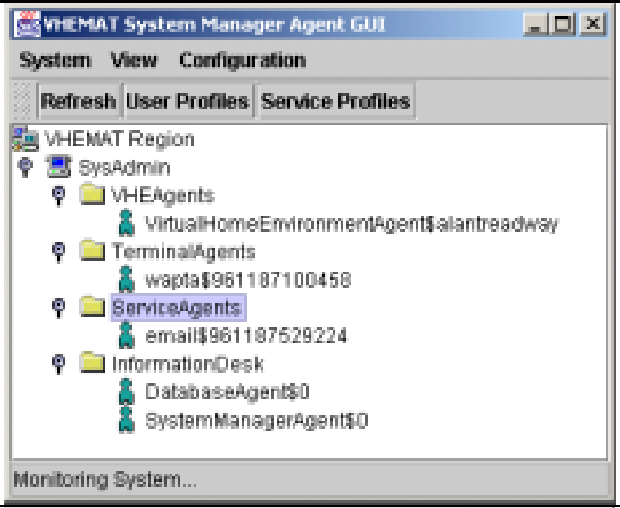
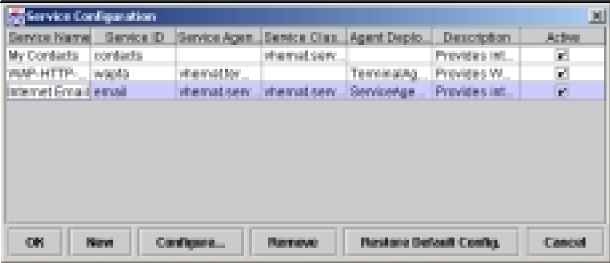
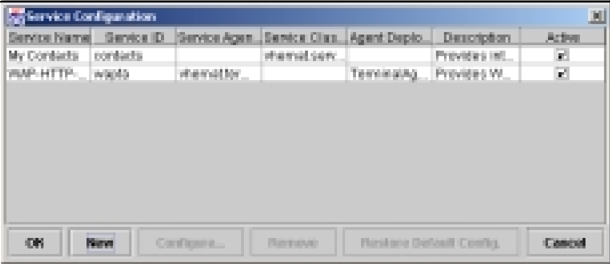
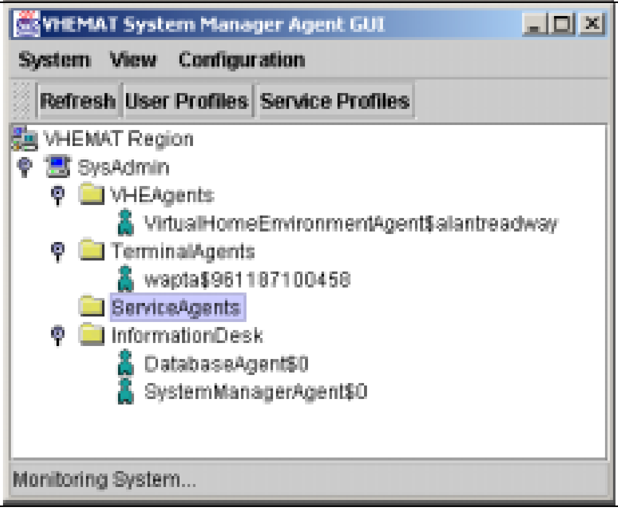
Service Property	Value
smtp_server	mailway.doc.ic.ac.uk

VHEMAT System Manager Agent GUI

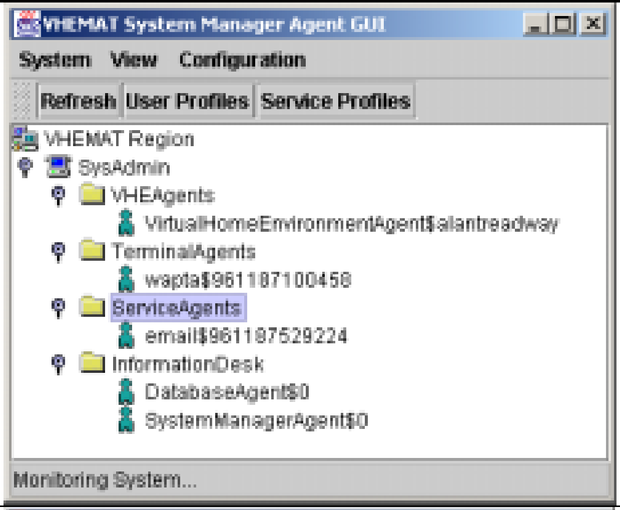
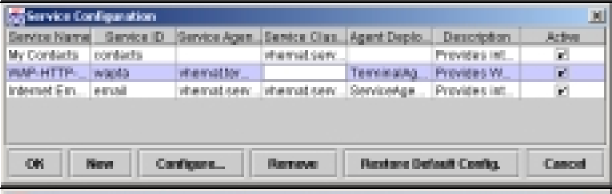
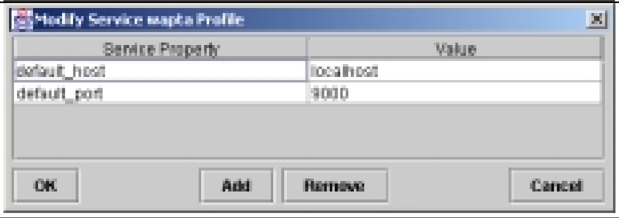
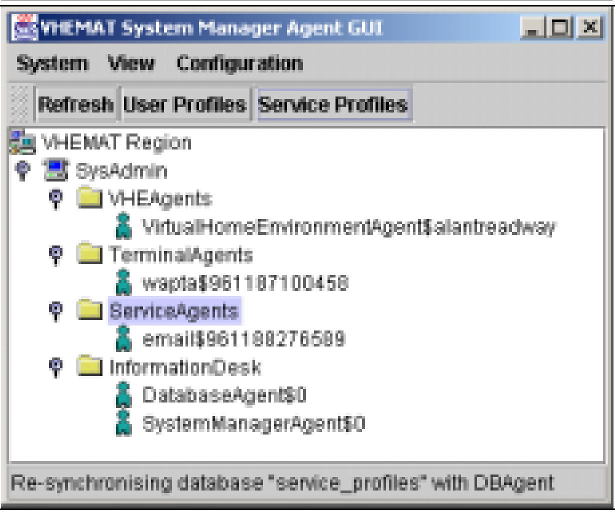
```
VHEMAT Region
  SysAdmin
    VHEAgents
      VirtualHomeEnvironmentAgent$alantreadway
    TerminalAgents
      wapta$961187100458
    ServiceAgents
      email$961187529224
    InformationDesk
      DatabaseAgent$0
      SystemManagerAgent$0
```

Monitoring System...

Use Case 12: System administrator removes a service

Screen-shot	Description
 VHEMAT System Manager Agent GUI System View Configuration Refresh User Profiles Service Profiles VHEMAT Region SysAdmin VHEAgents VirtualHomeEnvironmentAgent$alantreadway TerminalAgents wapta$9611187100458 ServiceAgents email$9611187529224 InformationDesk DatabaseAgent$0 SystemManagerAgent$0 Monitoring System...	The SysAdmin (System Administrator) has to remove the “email” service from the system. They select the “Service Profiles” toolbar button.
 Service Configuration Service Name · Service ID · Service Agen... · Service Clas... · Agent Deplo... · Description · Active My Contacts · contacts · · vhemat.serv... · · Provides inf... · ✓ WAP-HTTP... · wapta · vhemat.ter... · · TerminalAg... · Provides W... · ✓ Internet Email · email · vhemat.serv · vhemat.serv · ServiceAge... · Provides int... · ✓ OK New Configure... Remove Restore Default Config. Cancel	SysAdmin selects the “email” service from the list of service-profiles, and presses the “Remove” button.
 Service Configuration Service Name · Service ID · Service Agen... · Service Clas... · Agent Deplo... · Description · Active My Contacts · contacts · · vhemat.serv... · · Provides inf... · ✓ WAP-HTTP... · wapta · vhemat.ter... · · TerminalAg... · Provides W... · ✓ OK New Configure... Remove Restore Default Config. Cancel	SysAdmin presses the “OK” button.
 VHEMAT System Manager Agent GUI System View Configuration Refresh User Profiles Service Profiles VHEMAT Region SysAdmin VHEAgents VirtualHomeEnvironmentAgent$alantreadway TerminalAgents wapta$9611187100458 ServiceAgents InformationDesk DatabaseAgent$0 SystemManagerAgent$0 Monitoring System...	SMA removes SA’s associated with the removed service from the system.

Use Case 13: System administrator configures provider-specific details

Screen-shot	Description
 VHEMAT System Manager Agent GUI System View Configuration Refresh User Profiles Service Profiles VHEMAT Region SysAdmin VHEAgents VirtualHomeEnvironmentAgent$alantreadway TerminalAgents wapta$961187100458 ServiceAgents email$961187529224 InformationDesk DatabaseAgent$0 SystemManagerAgent$0 Monitoring System...	The SysAdmin (System Administrator) has to configure the “wapta” service. They select the “Service Profiles” toolbar button.
 Service Configuration Service Name / Service ID / Service Agen... / Service Clas... / Agent Deplo... / Description / Active My Contacts / contacts / / vhemat.serv... / / Provides int... / ✓ WAP-HTTP... / wapta / vhemat.ter... / / TerminalAg... / Provides W... / ✓ Internet Em... / email / vhemat.serv... / vhemat.serv... / ServiceAge... / Provides int... / ✓ OK New Configure... Remove Restore Default Config. Cancel	The SysAdmin selects the service to modify, and presses the “Configure…” button.
 Modify Service wapta Profile Service Property / Value default_host / localhost default_port / 9000 OK Add Remove Cancel	The service-specific configuration is displayed. The SysAdmin modifies it as required, then presses “OK”. The SysAdmin the presses “OK” again in the service-profile GUI.
 VHEMAT System Manager Agent GUI System View Configuration Refresh User Profiles Service Profiles VHEMAT Region SysAdmin VHEAgents VirtualHomeEnvironmentAgent$alantreadway TerminalAgents wapta$961187100458 ServiceAgents email$961188276589 InformationDesk DatabaseAgent$0 SystemManagerAgent$0 Re-synchronising database "service_profiles" with DBAgent	The SMA distributes the updated system-service-configuration to all SA’s belonging to the service, all VHEA’s, and the DBA.

A.2. User-Interface Questionnaire Provided to Trial Users

A number of trial users were selected to try the system, and provide an insight into any user-interface issues that are present within the system. Their computing skills ranged from novice to expert, but would be interested in a system such as that developed, if made commercially available.

Based upon the knowledge that having users "score" various attributes of a particular system is a proven methodology for collecting information from trial-users when the attributes to be tested are known, the following form was produced:

VHEMAT End-User UI Assessment Form

Please rate the following statements by ticking the appropriate box based upon you own opinions of the VHEMAT system, 1 indicates that you disagree, 5 indicates that you agree:

Statement	Score 1	2	3	4	5
1. I am familiar with an Email/Address book application.	[ ]	[ ]	[ ]	[ ]	[ ]
2. I am familiar with WAP.	[ ]	[ ]	[ ]	[ ]	[ ]
3. I found the system easy to navigate	[ ]	[ ]	[ ]	[ ]	[ ]
4. I found the system intuitive to navigate	[ ]	[ ]	[ ]	[ ]	[ ]
5. The interface was uncluttered	[ ]	[ ]	[ ]	[ ]	[ ]
6. It was easy to enter information into the system via the Swing GUI	[ ]	[ ]	[ ]	[ ]	[ ]
7. It was easy to enter information into the system via WAP	[ ]	[ ]	[ ]	[ ]	[ ]
8. Contact details were presented in a clear manner	[ ]	[ ]	[ ]	[ ]	[ ]
9. Emails were presented in a clear manner	[ ]	[ ]	[ ]	[ ]	[ ]
10. It was easy to add a new service	[ ]	[ ]	[ ]	[ ]	[ ]
11. It was easy to edit the properties of my devices	[ ]	[ ]	[ ]	[ ]	[ ]
12. The Swing user-interface was suitable for use on a PC/Palmtop	[ ]	[ ]	[ ]	[ ]	[ ]
13. The WAP user-interface was suitable for use on a WAP device	[ ]	[ ]	[ ]	[ ]	[ ]

Further comments:

Thank you for your time!

The data collected from the 12 trial users is shown below:

Statement	Average vote
1	4.67
2	3.33
3	3.16
4	4.16
5	3.33
6	4.75
7	2.50
8	4.67
9	4.33
10	3.75
11	4.25
12	3.83
13	4.42

Summary of (sensible) user-interface related comments given:

- Navigation using WAP is not simple enough due to lack of ability to use “back” within micro-browser.
- Swing user-interface is “clunky”.
- Ability to view attachments of emails using WAP should be added.
- Could use menu’s to aid navigation within Swing user-interface.

A.3. Data Collected for Secondary-Goal Test Cases

A.3.1 Definition of terms

- **Within**
  On the same host – no network bandwidth is required for interactions with servers on the same host. Distance used is 0 hops by definition.

- **Close to**
  At most one hop between hosts. Distance used is 1 hop by definition.

- **Within minimum distance of user**
  Minimum number of hops between user and “Network Service Providers’” servers (i.e. POP3, SMTP, HTTP etc…). This is considered the absolute minimum distance between a users device and a server, with the exception of servers placed at the leaf nodes of a network (e.g. WAP-gateways). Distance used is 9 hops[37].

- **Within average internet-server to user distance**
  Average distance between a users device and a server on the internet. Distance used is 13 hops[38].

- **Within average internet-server to internet-server distance**
  Average distance between two servers on the internet. Distance used is 4 hops[39].

[37] Based upon tracert distances to network service providers’ POP3/SMTP/HTTP servers (see section 0). Median value of 9 hops used.
[38] Based upon tracert distances to web-service providers servers (see section 0). Average value of 13 hops used.
[39] Based upon difference between distance from a user to an internet-based-server, and distance between a user and network-service-providers’ servers.

A.3.2 Observed Data

Below is a table of values collected from static real-world systems, and the system produced, upon which a model of the various systems can be made to compare bandwidth consumption.

Constant	Value
Size of average email (containing: JPEG image (10.2k); GIF image (17.1k); Word Document (23.0k))	72.9K
Size of contact data of 2 persons	Negligible
Size of web-page containing above email (including attachments)[40]	63.7K
Size of web-page containing a contact[40].	7.7K
Combined size of web-pages: login, selection of service, and selection of email[40].	61.7K
Combined size of web-pages: login, selection of service, and selection of contact[40].	56.4K
Size of VHEA when migrating (for display on high-spec PC) containing above email and 2 contacts	29.6K
Size of VHEA when migrating (for display on WAP-device) containing above email and 2 contacts	8.0K
Combined size of WML-decks: login, selection of service, and selection of email.	6.1K (2.8K) [41]
Combined size of WML-decks: login, selection of service, and selection of contact.	5.8K (3.0K)[41]
Size of WML-deck containing email (excluding attachments)	2.2K (0.8K)[41]
Size of WML-deck containing a contact.	0.6K (0.3K)[41]

A.3.3 Extrapolation of Observed Data

Given the definition of terms in section A.3.1, observed data from section A.3.2 and the scenarios described in section 5.4, a model of the cumulative bandwidth[42] used in each scenario has been created. These models make the following assumptions:

- Bandwidth usage outside of the systems is completely ignored (i.e. delivery of an email via SMTP into a mail-server is not considered – depending on where the mail-server is located more or less cumulative bandwidth may be required for its delivery).

- Only “simple” interactions take place (i.e. just browsing information rather than creating or editing), hence the bandwidth used requesting the next UI/HTML page/WML deck is negligible and can be ignored[43].

- Any overhead incurred as a result of using the HTTP/POP3 protocols is negligible compared to the size of the payload sent using these protocols, and can be ignored.

Given these assumptions, the model includes the bandwidth used to collect an email from a POP3 server, the bandwidth used for the user to log into the system (where appropriate) and the bandwidth to view the email and all attachments possible, as well as a contact stored with the system (where appropriate). The model then considers how multiple interactions within the same login session affects the overall bandwidth usage for that session.

[40] As created by Yahoo! Mail
[41] Size in brackets denotes WAP byte-encoded size.
[42] “Cumulative Bandwidth” refers to the total of bandwidth used on each “hop” between a source and destination host, across all interactions in a given session.
[43] For most interactions, the bandwidth usage for submitting data would be nearly identical anyway when comparing a WAP-TA vs. a traditional WWW/WML server. In the case of a Swing-TA, all interactions are carried out locally, and any changes sent back at the end of all interactions, resulting in less bandwidth usage in any case (even if only the overhead of each interaction is considered, this approach will be much more efficient).

	Case	1.1a	1.2a	1.3a
Constants	Size of email	72.9K	72.9K	72.9K
	Distance to POP3 server		1	4
	Distance to EmailAgent	1	1	1
	Size of migrating VHEAgent	29.6K	29.6K	29.6K
	Migration distance	9	9	13
	Distance from user to TA			
Cumulative bandwidth	Retreiving email from POP3 server		72.9K	291.6K
	Retreiving email from EmailAgent	72.9K	72.9K	72.9K
	Migrating Agent to TA	266.4K	266.4K	384.8K
	Total bandwidth used	339.3K	412.2K	749.3K

Table 1 - Scenario 1 Cases (Swing-TA)

	Case	1.1b	1.2b	1.3b
Constants	Size of email	72.9K	72.9K	72.9K
	Distance to POP3 server		1	4
	Distance to EmailAgent	1	1	1
	Size of migrating VHEAgent	8.0K	8.0K	8.0K
	Migration distance	9	9	1
	Distance from user to TA	1	2	13
	Distance from TA to WAP-GW		1	12
	Size of WML login/sevice/inbox	6.1K	6.1K	6.1K
	Size of encoded login/service/inbox	2.8K	2.8K	2.8K
	Size of WML login/sevice/contacts	5.8K	5.8K	5.8K
	Size of encoded login/sevice/contacts	3.0K	3.0K	3.0K
	Size of WML email UI	2.2K	2.2K	2.2K
	Size of encoded email UI	0.8K	0.8K	0.8K
	Size of WML contacts UI	0.6K	0.6K	0.6K
	Size of encoded contacts UI	0.3K	0.3K	0.3K
Cumulative bandwidth	Retreiving email from POP3 server		72.9K	291.6K
	Retreiving email from EmailAgent	72.9K	72.9K	72.9K
	Migrating Agent to TA	72.0K	72.0K	8.0K
	TA to WAP Gateway (login/service/inbox)		6.1K	73.2K
	WAP Gateway to device (login/service/inbox)	6.1K	6.1K	6.1K
	TA to WAP Gateway (login/service/contacts)		5.8K	69.6K
	WAP Gateway to device (login/service/contacts)	2.8K	2.8K	2.8K
	TA to WAP Gateway (email)		2.2K	26.4K
	WAP Gateway to device (email)	0.8K	0.8K	0.8K
	TA to WAP Gateway (contacts)		0.6K	7.2K
	WAP Gateway to device (contacts)	0.3K	0.3K	0.3K
	Total bandwidth used	154.9K	242.5K	558.9K

Table 2 - Scenario 1 Cases (WAP-TA)

	Case	2.1a	2.2a	2.3a
Constants	Size of email	72.9K	72.9K	72.9K
	Distance from Web-services to POP3 Server		1	4
	Size of HTML login/service/inbox	61.7K	61.7K	61.7K
	Size of HTML login/service/contacts	56.4K	56.4K	56.4K
	Size of HTML encoded email	63.7K	63.7K	63.7K
	Size of HTML encoded contact	7.7K	7.7K	7.7K
	Distance to User	13	13	13
Cumulative bandwidth	Retreiving email from POP3 server		72.9K	291.6K
	Loging in to system (email)	802.1K	802.1K	802.1K
	Loging in to system (contacts)	733.2K	733.2K	733.2K
	Retreiving email from Web-service	828.1K	828.1K	828.1K
	Retreiving contact from Web-service	100.1K	100.1K	100.1K
	Total bandwidth used	2463.5K	2536.4K	2755.1K

Table 3 - Scenario 2 Cases (HTML)

	Case	2.1b	2.2b	2.3b
Constants	Size of email	72.9K	72.9K	72.9K
	Distance from Web-services to POP3 Server		1	4
	Size of WML login/sevice/inbox	6.1K	6.1K	6.1K
	Size of encoded login/service/inbox	2.8K	2.8K	2.8K
	Size of WML login/sevice/contacts	5.8K	5.8K	5.8K
	Size of encoded login/sevice/contacts	3.0K	3.0K	3.0K
	Size of WML encoded email	2.2K	2.2K	2.2K
	Size of WML encoded contact	0.6K	0.6K	0.6K
	Size of WML byte-encoded email	0.8K	0.8K	0.8K
	Size of WML byte-encoded contact	0.3K	0.3K	0.3K
	Distance to WAP Gateway	12	12	12
	Distance from WAP Gateway to user	1	1	1
Cumulative bandwidth	Retreiving email from POP3 server		72.9K	291.6K
	Loging in to system (email)	77.0K	77.0K	77.0K
	Loging in to system (contacts)	72.6K	72.6K	72.6K
	Retreiving email from Web-service	27.2K	27.2K	27.2K
	Retreiving contact from Web-service	7.5K	7.5K	7.5K
	Total bandwidth used	184.3K	257.2K	475.9K

Table 4 - Scenario 2 Cases (WML)

	Case	3.1a	3.2a	3.3a
Constants	Size of email	72.9K	72.9K	72.9K
	Distance from Web-services to POP3 Server		1	4
	Size of HTML login/service/inbox	61.7K	61.7K	61.7K
	Size of HTML login/service/contacts	56.4K	56.4K	56.4K
	Size of HTML encoded email	63.7K	63.7K	63.7K
	Size of HTML encoded contact	7.7K	7.7K	7.7K
	Distance to User	9	9	9
Cumulative bandwidth	Retreiving email from POP3 server		72.9K	291.6K
	Loging in to system (email)	555.3K	555.3K	555.3K
	Loging in to system (contacts)	507.6K	507.6K	507.6K
	Retreiving email from Web-service	573.3K	573.3K	573.3K
	Retreiving contact from Web-service	69.3K	69.3K	69.3K
	Total bandwidth used	1705.5K	1778.4K	1997.1K

Table 5 - Scenario 3 Cases (HTML)

Constants	Case	3.1b	3.2b	3.3b
	Size of email	72.9K	72.9K	72.9K
	Distance from Web-services to POP3 Server		1	4
	Size of WML login/sevice/inbox	6.1K	6.1K	6.1K
	Size of encoded login/service/inbox	2.8K	2.8K	2.8K
	Size of WML login/sevice/contacts	5.8K	5.8K	5.8K
	Size of encoded login/sevice/contacts	3.0K	3.0K	3.0K
	Size of WML encoded email	2.2K	2.2K	2.2K
	Size of WML encoded contact	0.6K	0.6K	0.6K
	Size of WML byte-encoded email	0.8K	0.8K	0.8K
	Size of WML byte-encoded contact	0.3K	0.3K	0.3K
	Distance to WAP Gateway	8	8	8
	Distance from WAP Gateway to user	1	1	1
Cumulative bandwidth	Retreiving email from POP3 server		72.9K	291.6K
	Loging in to system (email)	52.6K	52.6K	52.6K
	Loging in to system (contacts)	49.4K	49.4K	49.4K
	Retreiving email from Web-service	18.4K	18.4K	18.4K
	Retreiving contact from Web-service	5.1K	5.1K	5.1K
	Total bandwidth used	125.5K	198.4K	417.1K

Table 6 - Scenario 3 Cases (WML)

	Case	4.1	4.2	4.3
Constants	Size of email	72.9K	72.9K	72.9K
	Distance to POP3 Server	9	9	14
Cumulative bandwidth	Retreiving email from POP3 server	656.1K	656.1K	1020.6K
	Total bandwidth used	656.1K	656.1K	1020.6K

Table 7 - Scenario 4

A.3.4 Miscellaneous Collected Data

Netowrk Service Provider	Hops to POP3 Server(s)	Hops to SMTP Server(s)	Hops to HTTP Server(s)
Ezesurf	9	7	9
BTInternet	22	22	22
FreeNetName	7	6	6

Network Service Provider	RTD to POP3 Server(s)	RTD to SMTP Server(s)	RTD to HTTP Server(s)
Ezesurf	180 ms	160 ms	160 ms
BTInternet	180 ms	180 ms	180 ms
FreeNetName	160 ms	160 ms	160 ms

Web-Service Provider	Hops from Ezesurf	Hops from BTInternet	Hops from FreeNetName
www.yahoo.co.uk	10	14	15
www.yahoo.com	13	14	19
www.quios.com	10	13	13
www.hotmail.com	11	13	15
www.junglemail.net	7	12	14

Web-Service Provider	RTD from Ezesurf	RTD from BTInternet	RTD from FreeNetName
www.yahoo.co.uk	171 ms	181 ms	180 ms
www.yahoo.com	230 ms	260 ms	300 ms
www.quios.com	300 ms	310 ms	301 ms
www.hotmail.com	250 ms	270 ms	260 ms
www.junglemail.net	185 ms	170 ms	151 ms

Appendix B: Java Extensions Used

B.1. Java Mail [32]

This section aims to briefly explain the functionality of the Java Mail Extension used within the system implementation.

B.1.1 Overview

The Java Mail Extension provides an abstract API for interacting with different types of mail servers. By default the Java Mail Extension can interact with IMAP4 mailboxes and SMTP servers, and with the addition of the POP3 Java Mail Extension, POP3 mailboxes can also be accessed.

The abstract classes offer the ability to handle MIME encoded messages and attachments, as well as providing JavaBean components to represent various attachment types via the use of the Java Activation Framework. The package contains a demonstration email program written on top of the Java Mail Extension to show what is capable with the API, and it is capable of the same functionality as most leading email programs (to give you some idea of the scope of the API).

B.1.2 Why use Java Mail?

Aside from the lack of any predominant 3rd party POP3/SMTP API software, the main reasons for using the Java Mail Extension to incorporate POP3/SMTP integration into the system are:

- Style of API is similar to the JDK classes, resulting in more efficient use of coding time.
- All in one-package for collecting and sending email, reduces the overhead of learning multiple API's for different products.
- Provides support for MIME encoded content within its core API – no "messing about" with encoding and decoding of MIME data.
- JavaSoft provide numerous examples of how to use the API's.
- "Re-inventing the wheel" isn't necessary – a detailed knowledge of POP3 and SMTP would be required to implement a system to enable the functionality within this extension, along with a considerable amount of effort.

B.1.3 Functionality Used

As mentioned previously, the main functionality used from this API is the ability to read email from a (POP3) mailbox, and send emails via SMTP. This involved learning how to use a couple of objects (Session, Folder and Message to be precise) by following a couple of tutorial examples and within half an hour the system was sending and receiving emails.

B.2. Java Advanced Imaging [30]

This section aims to briefly explain the functionality of the Java Advanced Imaging Extension used within the system implementation.

B.2.1 Overview

The Java Advanced Imaging Extension provides a much richer set of image manipulation primitives than are found within the standard JDK, as well as encoders and decoders for a range of image types including: BMP, JPEG, GIF (decode only), PNG and a number of other formats. This extension is aimed at software developers writing advanced imaging applications in the fields of Computer Vision or graphic manipulation packages.

B.2.2 Why use Java Advanced Imaging?

There is another Java Extension (the Java Image I/O Extension [31]) that is better suited to the needs of this project than the Java Advanced Imaging Extension, but it is still in early beta and only supports loading of images. As a result the more heavyweight Java Advanced Imaging Extension has been used for the following reasons:

- Simple API provides the ability to scale images with one method invocation.
- Ability to decode and encode various image formats.
- Numerous tutorial examples accompany the extension from JavaSoft.
- The required functionality cannot be easily implemented using the standard JDK classes.

B.2.3 Functionality Used

Despite the numerous image manipulation API's provided by this extension, the only real features utilised are the ability to decode image data into instances of `java.awt.image.RenderedImage`, encode them back into the same or another form, and the ability to scale `RenderedImage` instances, which enables shrinking of image size to match the display capabilities of devices.

Appendix C: User Guide

C.1. Installation Guide

This section describes the steps required in order to set up the user-side components of the VHEMAT system, specifically the Swing-TA components since other TA's require no user configuration.

C.1.1 System Requirements for Swing-TA

Before the VHEMAT Swing-TA can be installed, the following requirements must be met:

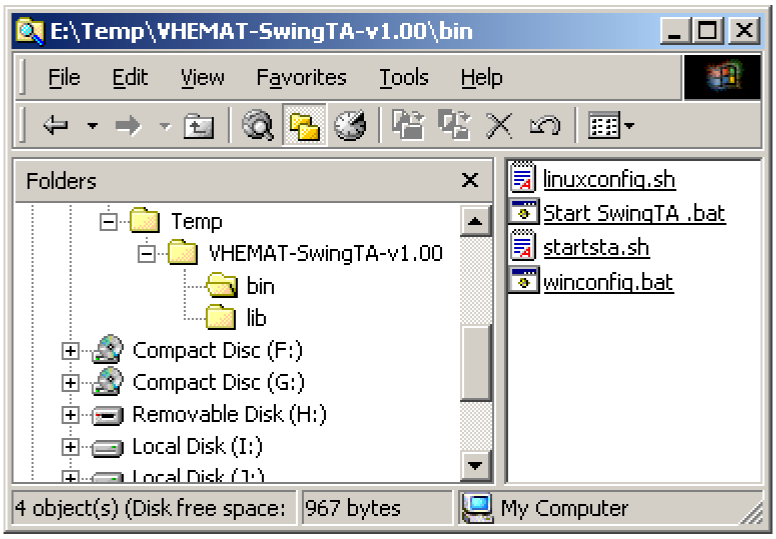
- JDK/JRE 1.2.2 installed and referenced from PATH environment variable (system can be run with JDK/JRE 1.3, but extensive testing has not been performed).

Grasshopper does not need to be installed since the required components from Grasshopper are contained within the `VHEMAT-SwingTA-v1.00.zip` file. According to the GH license, this is permitted providing the application is non-commercial.

C.1.2 Installing the VHEMAT Software

Once the requirements of the system have been met, to install the VHEMAT Swing-TA simply un-zip the `VHEMAT-SwingTA-v1.00.zip` file into the desired location. For Windows machines, the Program Files directory on your main drive is the recommended location, for Linux machines this will depend if you have root privileges. The VHEMAT scripts are be location independent, so the installation location is not important.

You should find that the extracted `VHEMAT-SwingTA-v1.00.zip` file contains two directories:


E:\Temp\VHEMAT-SwingTA-v1.00\bin

Temp
VHEMAT-SwingTA-v1.00
bin
lib

linuxconfig.sh
Start SwingTA .bat
startsta.sh
winconfig.bat

4 object(s) (Disk free space: 967 bytes My Computer

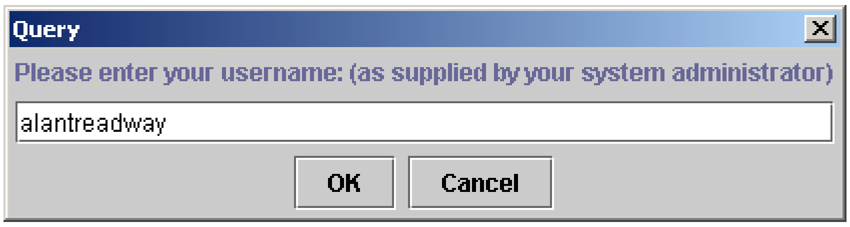
The bin directory contains Windows batch files and Linux bash scripts to launch the Swing-TA.

For convenience, Linux users may wish to add the `bin` directory to their PATH environment variable, and Windows' users may wish to add a shortcut to `Start SwingTA.bat` to their Start menu.

C.2. Configuration details

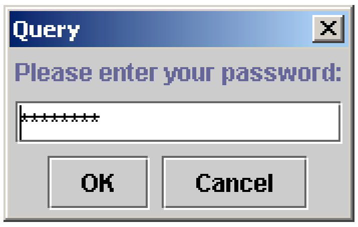
To configure the Swing-TA, you will require your username, region URL and a device-id for the device you have installed the Swing-TA on to. These details are available from your

System Administrator. Launch the `Start SwingTA.bat` file or `startsta.sh` (depending on the system you are using), and the SwingTA will start up. The first time you do this you will be prompted for these details:


Query

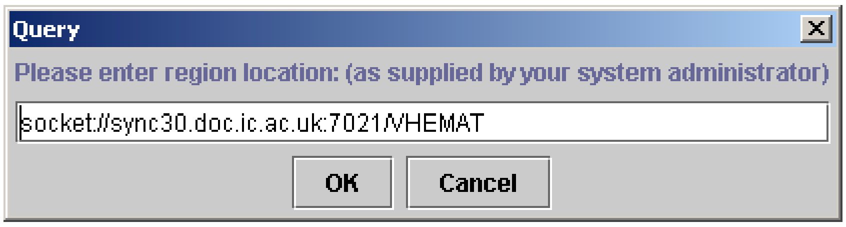
Please enter your username: (as supplied by your system administrator)

alantreadway


Query

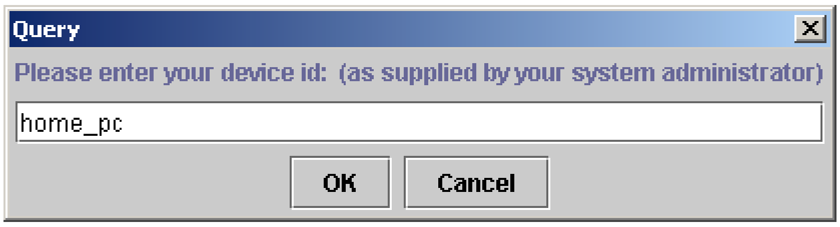
Please enter your password:

****


Query

Please enter region location: (as supplied by your system administrator)

socket://sync30.doc.ic.ac.uk:7021/VHEMAT


Query

Please enter your device id: (as supplied by your system administrator)

home_pc

If for some reason you need to reconfigure the SwingTA, simply remove the `.vhemat` directory from your user-area/profile area, and re-start the Swing-TA.

C.3. Using the VHEMAT system

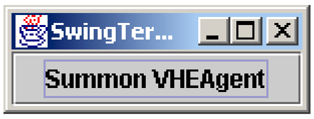
This section now concentrates on the use of Terminal Agents, and the services available in general to users.

C.3.1 Summoning your VHEAgent

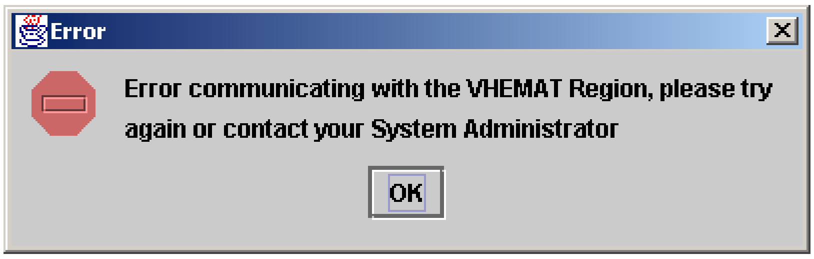
The act of summoning a VHEAgent causes the VHEAgent to migrate to the TA with which you are interacting. This process varies according to the TA you are interacting with.

Swing User Interface

After starting the Start SwingTA.bat or startsta.sh file, the following window appears:

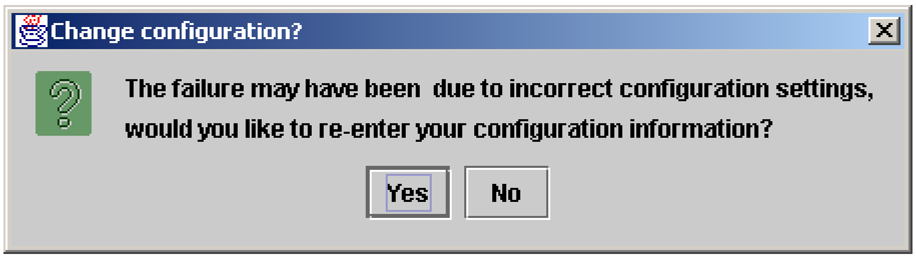


Pressing the “Summon VHEAgent” button will attempt to do just that. There are a number of common errors that may occur:

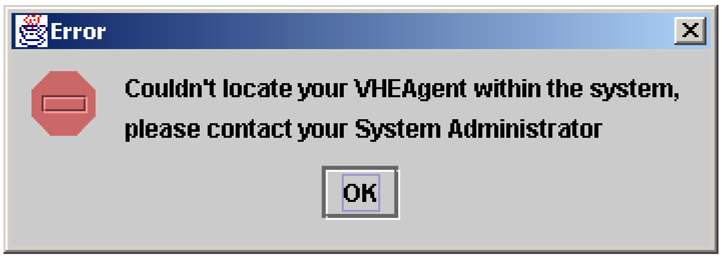

Error

Error communicating with the VHEMAT Region, please try again or contact your System Administrator

The Swing-TA could not contact the VHEMAT-system region to which you are subscribed. This is either due to an incorrect region URL specified within the Swing-TA configuration, a failure of the VHEMAT-system region, or a network problem somewhere. You will be given the option to change your configuration if this occurs:

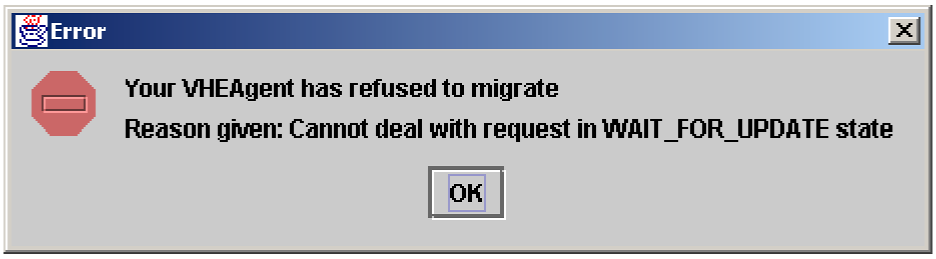

Change configuration?

The failure may have been due to incorrect configuration settings, would you like to re-enter your configuration information?


Error

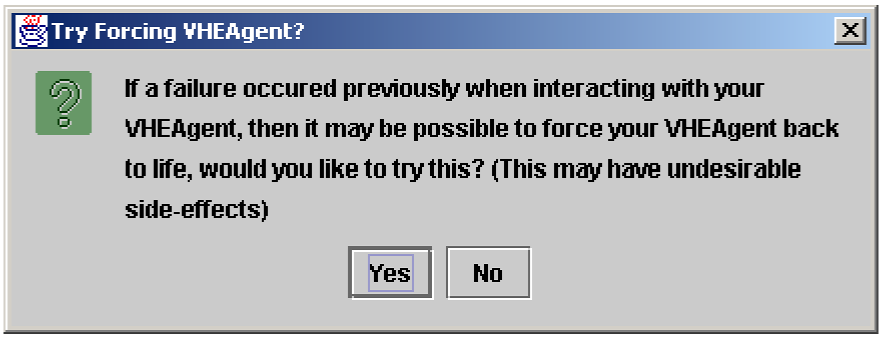
Couldn't locate your VHEAgent within the system, please contact your System Administrator

This is shown when the VHEMAT-system region has been located, but your VHEAgent cannot be located. You will be given the opportunity to modify you configuration settings, but if this problem persists, it may be that your VHEAgent has died and the system has failed to notice. Please contact your System Administrator in this case.


Error

Your VHEAgent has refused to migrate
Reason given: Cannot deal with request in WAIT_FOR_UPDATE state

If some failure occurred whilst you were previously interacting with your VHEAgent, or another session has been left open this error message may be displayed. The Swing-TA will ask you if you wish to attempt to force the VHEAgent to start a new session:

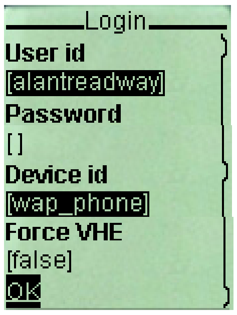


If a failure occured previously when interacting with your VHEAgent, then it may be possible to force your VHEAgent back to life, would you like to try this? (This may have undesirable side-effects)

WAP User Interface

From a WAP device, type in the URL of the central WAP-TA provided by your system administrator. It should be of the form `http://<somehost>:<wap-ta-port>/`. If you wish to bookmark this page and avoid retyping your username, password and device-id, you should append ?u=**`<username>`**&p=**`<password>`**&d=**`<device-id>`** to the above form of URL.

After a short while you should be presented with a page looking like this:



Login

User id
[alantreadway]
Password
[]
Device id
[wap_phone]
Force VHE
[false]

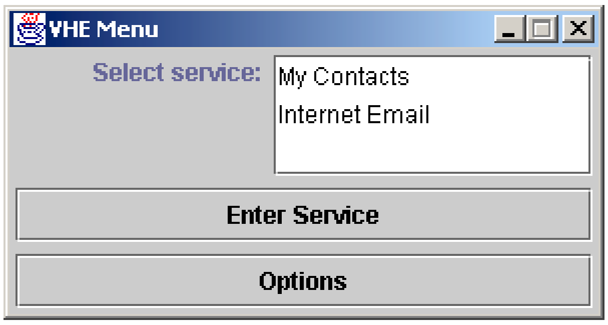
Selecting OK attempts to summon your VHEAgent. The same errors can occur as described for the Swing-TA, except that you are not given the option to re-enter configuration details, and the option to force a VHEAgent to interact with you is provided from the login screen.

C.3.2 Interacting with your VHEAgent

The options available to you are the same no matter which method of interaction you use, the only difference is in the information contained within services will vary based upon the capabilities of your device. For this reason, the rest of this guide will concentrate on use of the Swing-TA rather than attempting to cover the same ground with both it and the WAP-TA.

Main VHE Menu

Upon successfully summoning your VHEAgent, you are provided with a list of subscribed services, and two options:

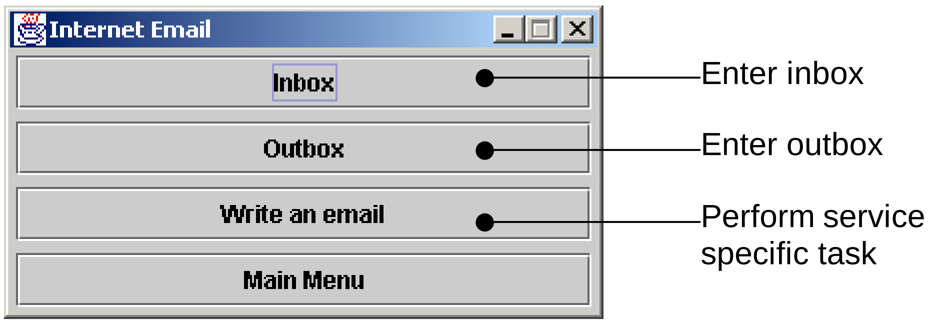


Selecting a service and clicking Enter Service take you to a service menu (see the sub-section Service Menu).

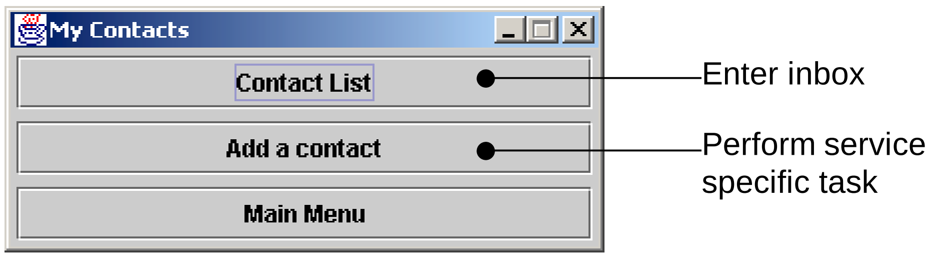
Selecting Options takes you to the VHEAgent options menu (see the section C.3.3).

Service Menus

From this type of menu a service's inbox (if it has one) can be opened, a list of pending tasks/the service's outbox (if it has one) can be opened, and service-specific tasks can be initiated. A typical service menu looks something like this:

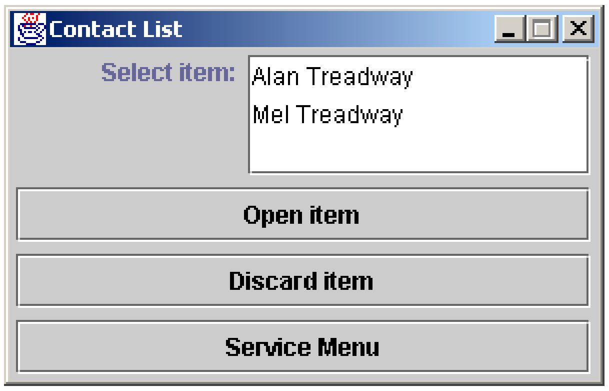


A service might call its inbox/outbox something more appropriate based upon the type of items contained within it, or hide the inbox/outbox if it isn't used. For example:



Inbox

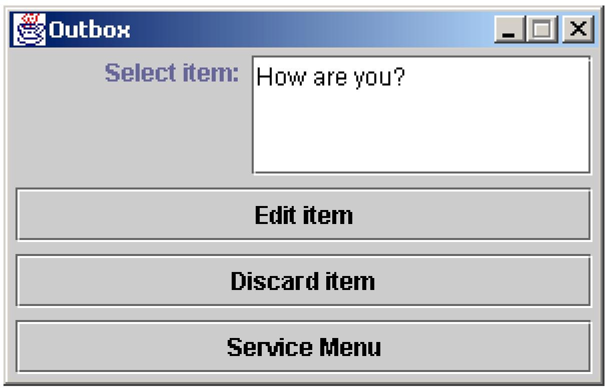
A service inbox generally contains a list of viewable items, and provides the ability to remove items no longer required:



Simply select an item, and click "Open Item" to view the item, or "Discard Item" to permanently remove it.

Outbox

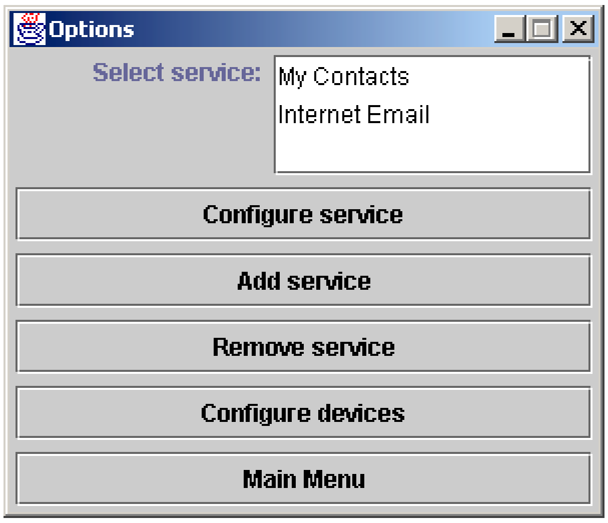
A service outbox generally contains a list of pending tasks, and provides the ability to edit tasks or remove tasks no longer required:



Simply select an item, and click “Edit Item” to edit the item, or “Discard Item” to permanently remove it.

C.3.3 Configuring your VHEAgent

The Options menu provides the ability to configure the services you are subscribed to and the devices that you will be using to interact with the VHEMAT system. The options menu looks like this:



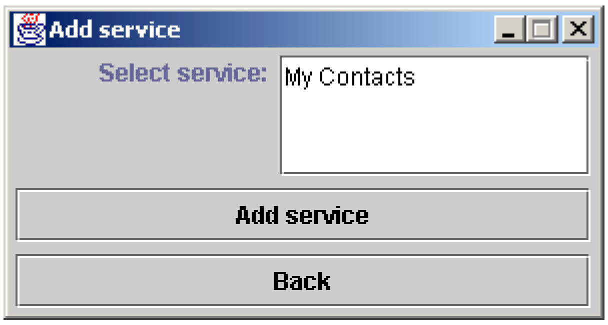
Selecting a service, and pressing “Configure Service” opens a service-specific configuration UI, or pressing “Remove Service” removes the selected service from your VHEAgent.

Clicking “Add Service” brings up the Add Service UI (see sub-section Add Service).

Clicking “Configure Devices” brings up the Configure Device UI (see sub-section Configure Devices).

Add Service

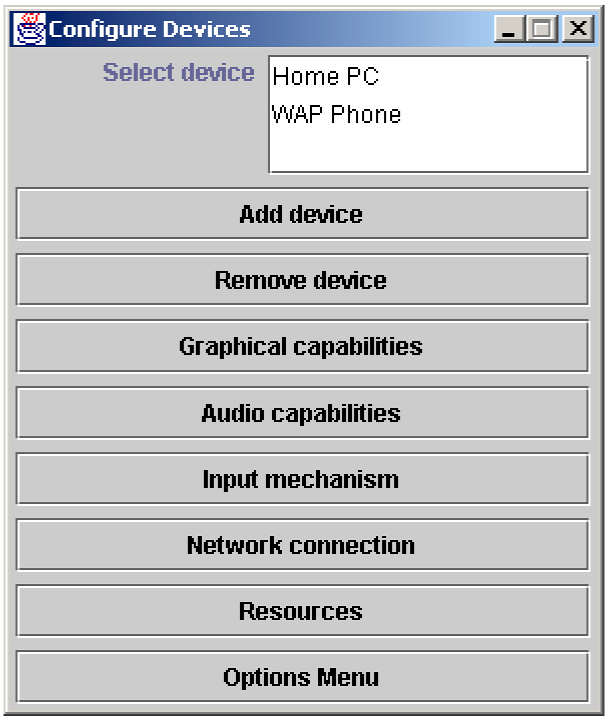
A list of currently un-subscribed services is displayed:



Selecting a service and clicking “Add Service” will attempt to start the service within the VHEAgent, and then provide you with a service-specific configuration dialog to complete.

Configure Devices

A list of currently known device profiles is displayed, along with numerous configuration options:



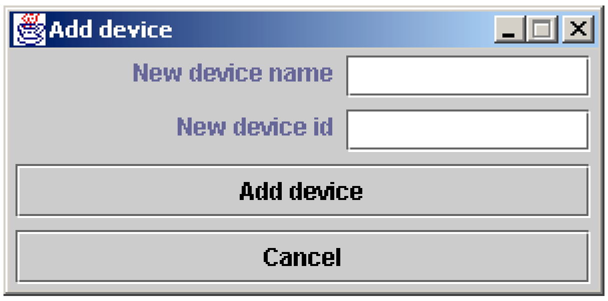
Selecting a device and clicking “Remove Device” removes the selected device profile permanently.

Clicking “Add Device” brings up the Add Device UI (see sub-section Add Device).

Selecting a device and clicking the other options allow the properties defined by the selected device profile to be modified, and are described in following sub-sections.

Add Device

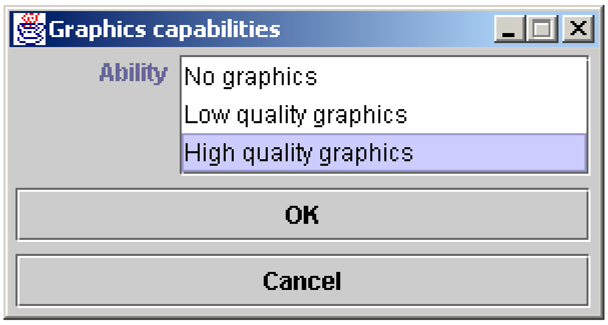
A dialog appears requesting details about the name and device-id of the new device profile to create:



Device name merely allows a meaningful name to be associated with the new device profile. The device-id uniquely identifies the device profile within your VHEAgent, and is used to indicate what device you are currently using when logging into a TA. If “Add Device” is selected, a device profile with the specified attributes is created.

Graphical Capabilities

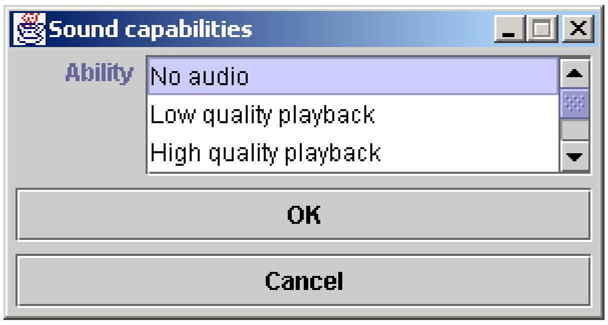
This menu allows the graphical capabilities of a device profile to be modified:



“No graphics” – Specifies device can’t display graphics
“Low quality graphics” – Specifies device can display low-resolution, low colour depth graphics
“High quality graphics” – Specifies device can display high-resolution, high colour depth graphics.

Audio Capabilities

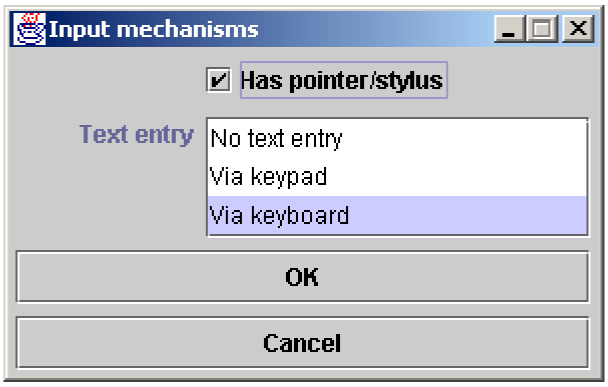
This menu allows the audio capabilities of a device profile to be modified:



“No audio” – Specifies that the device can’t play or record audio.
“Low quality playback” – As the name suggests.
“High quality playback” – As the name suggests.
“Low quality playback/record” – Low quality playback and ability to record audio.
“High quality playback/record” – High quality playback and ability to record audio.

Input Mechanism

This menu allows the input capabilities of a device profile to be modified:



Input mechanisms

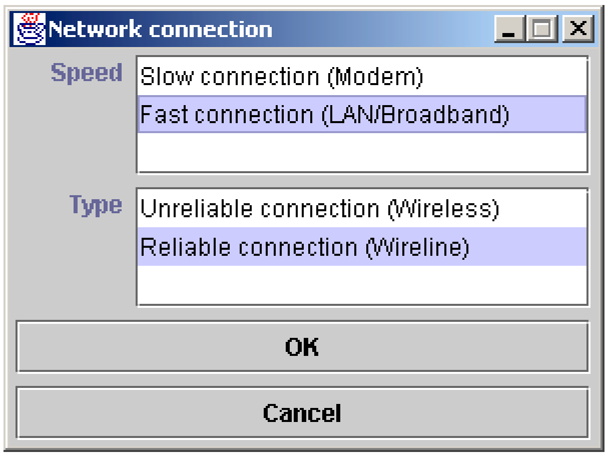
[✓] Has pointer/stylus

Text entry
- No text entry
- Via keypad
- Via keyboard (selected)

"Has pointer/stylus" – Indicates if the device has a mouse/pen or other pointer-type input device.
"No text entry" – Indicates that text entry is not possible.
"Via keypad" – Indicates that numbers only may be entered
"Via keyboard" – Indicates that alphanumeric data may be entered by the device.

Network Connection

This menu allows the network type of a device profile to be modified:



Network connection

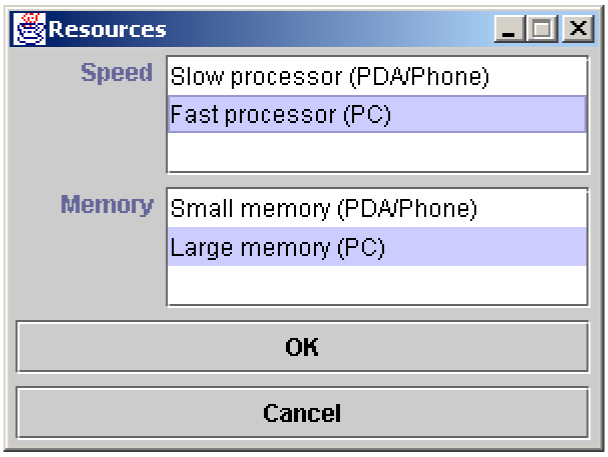
Speed
- Slow connection (Modem)
- Fast connection (LAN/Broadband) (selected)

Type
- Unreliable connection (Wireless)
- Reliable connection (Wireline) (selected)

"Slow connection" – Indicates that the device uses a slow network connection, such as that provided by a modem (i.e. bandwidth available is an issue).
"Fast connection" – Indicates that the device uses a broad-band connection of some form, or is connected to a LAN.
"Unreliable connection" – Indicates that the likelihood of dropped connections and packets is relatively high due to the nature of the network connection (i.e. wireless).
"Reliable connection" – Indicates that the likelihood of dropped connections and packets is relatively low due to the nature of the network connection (i.e. wireline).

Resources

This menu allows the resources of a device profile to be modified:


Resources

Speed: Slow processor (PDA/Phone), Fast processor (PC) [selected]

Memory: Small memory (PDA/Phone), Large memory (PC) [selected]

This allows the processing power of the device to be specified in terms of processor speed and memory available.

C.3.4 Using the services

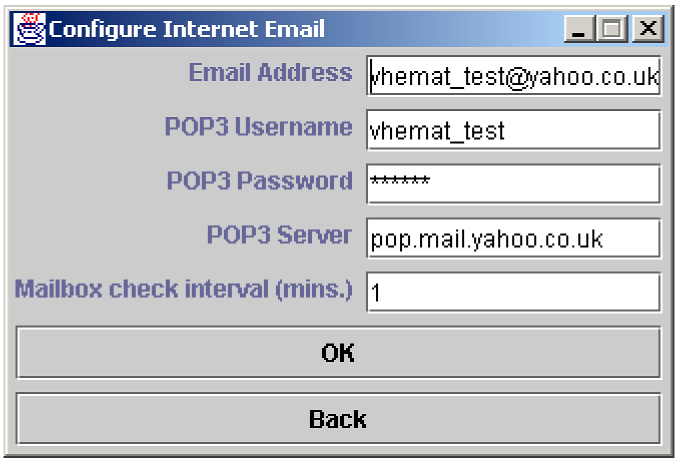
As mentioned previously, a generic service menu is provided for use by each service (see sub-section Service Menus). Thus, this section will concentrate on the UI's provided by the services themselves rather than the operation of the generic menus that allow access to the functionality of the service.

Email service

This service provides the ability to read emails from a POP3 mailbox belonging to the user, and adds the ability to write emails.

Configuration Options

A number of items are required to be configured in order for this service to function correctly:


Configure Internet Email

Email Address	vhemat_test@yahoo.co.uk
POP3 Username	vhemat_test
POP3 Password	******
POP3 Server	pop.mail.yahoo.co.uk
Mailbox check interval (mins.)	1

- Email address – This is the email address that will be placed on all outgoing mails you write. Without it recipients of emails you write within the VHE will not who the email is from.

- POP3 Username – The POP3 username that was provided to you by your service provider, and is required to access your POP3 mailbox.

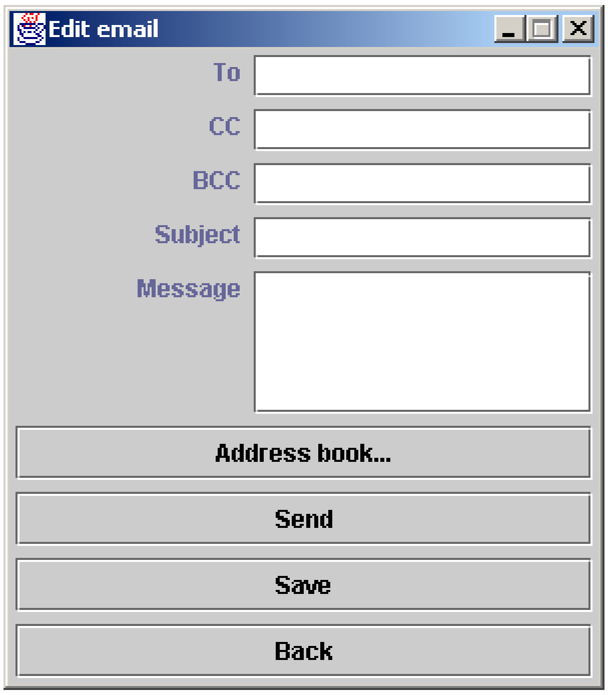
- POP3 Password – The password associated with your POP3 mailbox. This is required in order that the VHEAgent can retrieve emails.

- POP3 Server – The POP3 server on which your POP3 account exists. This is required to access your POP3 mailbox.

- Mailbox check interval – Indicates how often the POP3 inbox specified should be checked for new mail by your VHEA. The minimum interval is every minute.

Email Service Task UI

When creating a new email, a UI like this is displayed:



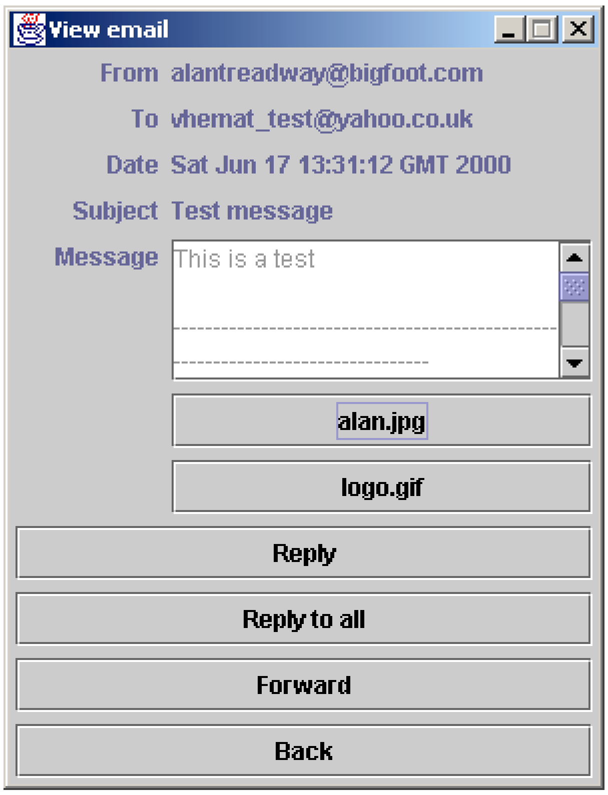
The text-entry areas are similar to those you might use in you existing mail program. The "To", "CC" and "BCC" fields should be delimited using the ';' character.

The "Address book…" button is only available if you are using the Contacts service, and provides you with a list of people you can add to the "To", "CC" and "BCC" fields.

Clicking "Send" adds the email to you outbox as an email ready to be sent. Upon returning to the network, the VHEAgent will attempt to send it for you. The "Save" button will add the email to the outbox as a draft email, allowing you to come-back and edit it further at a future point in time. When saved, an email will not be sent until you open it and click the "Send" button.

Email Service Item UI

This UI is similar to the email service-task UI, except the information it contains is readable only:


View email

From alantreadway@bigfoot.com

To vhemat_test@yahoo.co.uk

Date Sat Jun 17 13:31:12 GMT 2000

Subject Test message

Message This is a test

All of the details of the email are displayed, along with the option to view any attachments that were sent with the email (this will depend upon the device you are using).

The “Reply” button will create a new email service-task containing the original message, and with the “To” field set as the person the original email was from. You can then edit the email as described in the previous sub-section.

The “Reply to All” button is similar in nature, except that all recipients defined in the original email (including those in the “To” and “CC” fields) are set as recipients of the appropriate type.

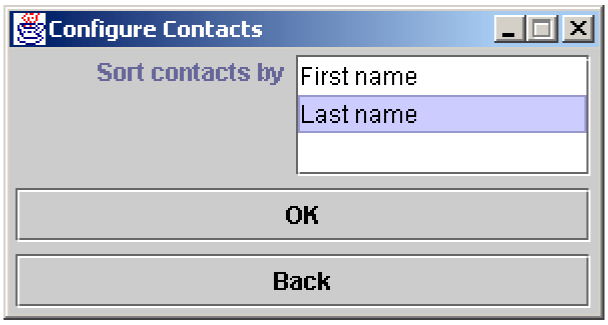
The “Forward” button will create a new email service task which contains the same content and subject, and then allow you to edit it as described in the previous section.

Contacts service

The Contacts service allows a number of peoples contact details to be stored within your VHEA.

Configuration Options

There is only one configuration option for the contacts service:

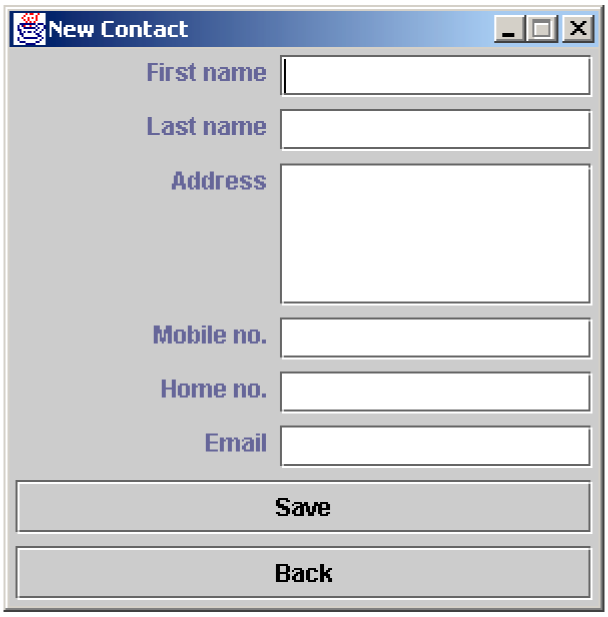

Configure Contacts

Sort contacts by: First name, [✓ Last name]

The “Sort contacts by” option allows you to configure in which order the contacts in your contacts list will be ordered.

Contact Service Task UI

When adding a contact for the first time, the following UI is presented:

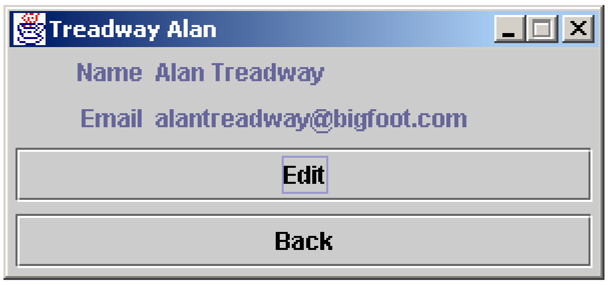


Only the “First name” and “Last name” fields are required, you can enter as much or as little information about a particular contact as you wish.

Clicking on “Save” adds the contact to your contact list (inbox).

Contact Service Item UI

When viewing a contact within your contact list, all information that has been entered for a given contact is displayed:



By selecting the “Edit” button, the contact service-item UI is displayed with the details of the contact being viewed, and you are then permitted to modify the existing contacts’ details’.

Appendix D: System Administrators Guide

D.1. Installation Guide

This section describes the steps required in order to set up the system-side components of the VHEMAT system.

D.1.1 System Requirements for the VHEMAT System

Before the VHEMAT System can be installed, the following requirements must be met:

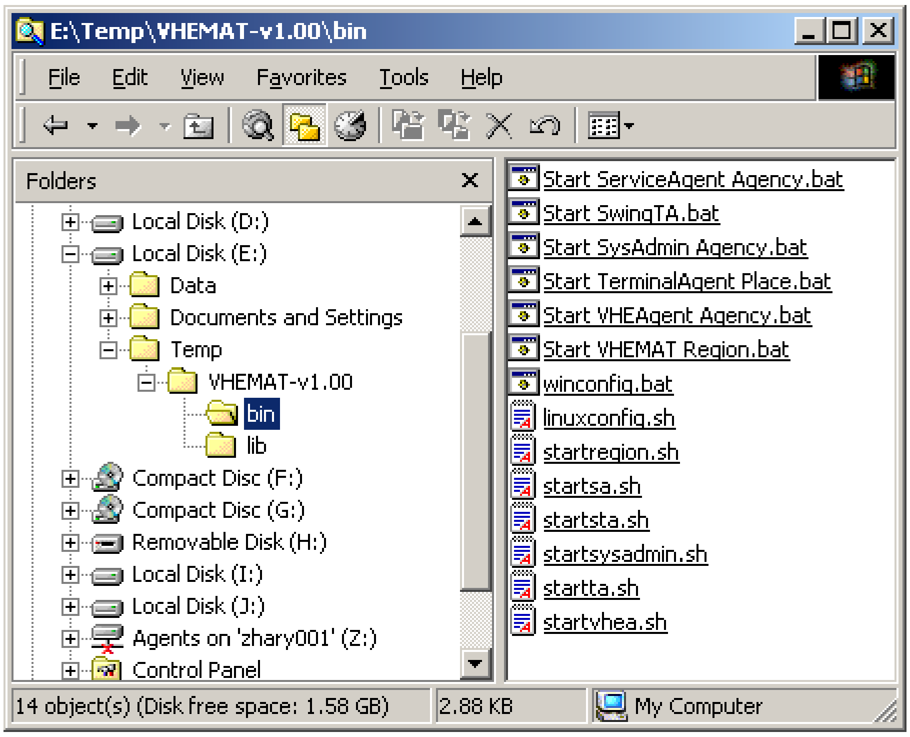
- JDK/JRE 1.2.2 installed and referenced from PATH environment variable of all machines which are to be part of the system (the VHEMAT system can be run with JDK/JRE 1.3, but extensive testing has not been performed).

Other packages do not need to be installed since the required components are contained within the `VHEMAT-v1.00.zip` file. According to the various licenses, this is permitted providing the application is non-commercial.

D.1.2 Installing the VHEMAT Software

Once the requirements of the system have been met, to install the VHEMAT System simply un-zip the `VHEMAT-v1.00.zip` file into the desired location. It is recommended that the files be places into a shared directory which is accessible from all machines which will be used to run the system. The VHEMAT scripts should be location independent, so other than that, the installation location is not important.

You should find that the extracted `VHEMAT-v1.00.zip` file contains two directories:



Start ServiceAgent Agency.bat
Start SwingTA.bat
Start SysAdmin Agency.bat
Start TerminalAgent Place.bat
Start VHEAgent Agency.bat
Start VHEMAT Region.bat
winconfig.bat
linuxconfig.sh
startregion.sh
startsa.sh
startsta.sh
startsysadmin.sh
startta.sh
startvhea.sh

14 object(s) (Disk free space: 1.58 GB) | 2.88 KB

The bin directory contains Windows batch files and Linux bash scripts to launch the various parts of the system.

For convenience, Linux users may wish to add the `bin` directory to their PATH environment variable, Windows users are encouraged to use the batch files in place.

D.1.3 Start-up Configuration of the VHEMAT Software

Due to the nature of the Grasshopper platform, a central Region Registry must be run in a well-known location, and all Agencies within the system must know the URL of this Region. Before carrying out the necessary steps to configure this, you should decide which host will run the Region Registry, and on what port (the recommended port is 7021). The Region can also be given a specific name (the default is VHEMAT), and a transport type selected (the default is “socket”, but “rmi” and “iiop” are also available – details of how to configure these settings is not documented here).

Load the `winconfig.bat` and/or the `linuxconfig.sh` into a text editor and modify the lines shown to the values you have chosen:

`winconfig.bat:`

```
…
rem Setup GH Region parameters
set TRANSPORT=socket
set HOST=146.169.49.30
set PORT=7021
set REGIONNAME=VHEMAT
…
```

`linuxconfig.sh:`

```
...
# Setup GH Region parameters
TRANSPORT=socket
HOST=146.169.49.30
PORT=7021
REGIONNAME=VHEMAT
...
```

Providing both files are consistent, both Linux and Windows machines can operate within the same region without any difficulty.

D.2. Starting the VHEMAT system

This section covers the steps required to start a working VHEMAT system.

D.2.1 Starting Grasshopper & the VHEMAT System

Providing the steps detailed in section D.1 have been followed, carry out the following steps:

- Run `Start VHEMAT Region.bat` or `startregion.sh` from the host that was decided should contain the Region Registry. This will start the Region Registry. If the command line option `–gui` is specified, a GUI for the Region Registry will appear, enabling you to monitor whether Agencies have successfully registered or not.

- Run at least one instance of `Start ServiceAgent Agency.bat/startsa.sh`, `Start TerminalAgent Agency.bat/startta.sh`, `Start VHEAgent Agency.bat/startvhea.sh` on the same/other hosts as you see fit. All of the Agencies you create will register with the Region Registry, so how they are distributed does not matter – all Agencies could be started on one host, or multiple versions started across multiple hosts.

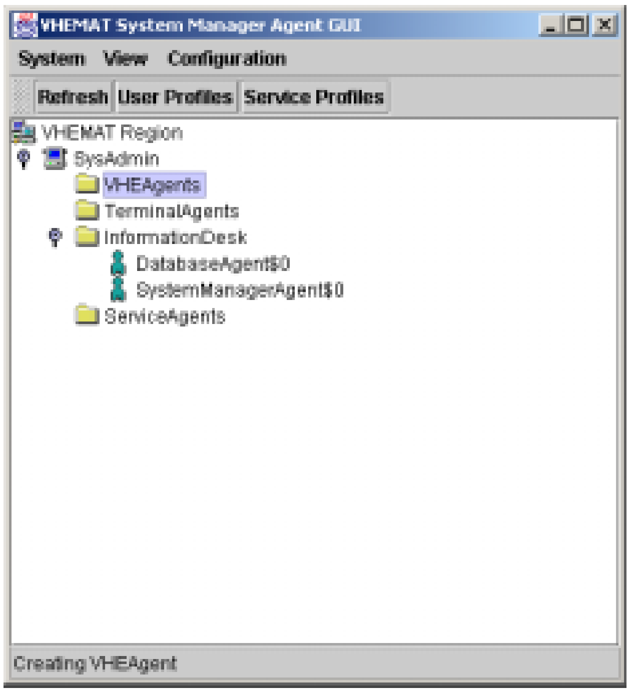
- Run `Start SysAdmin Agency.bat` or `startsysadmin.sh` from the System Administrator machine. This will start the SMA, and display the SMA GUI on the host it was started on. If the SMA has been previously configured, it will start populating all Agencies that are part of the Region it is located within with Agents.

D.3. Configuration details

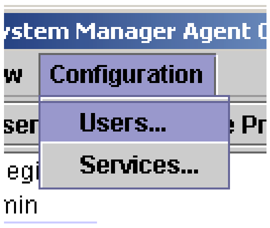
Once the system has been set-up and started (as described in section D.2), the next step is to configure the services and user profiles for the VHEMAT system.

D.3.1 Configuring user details

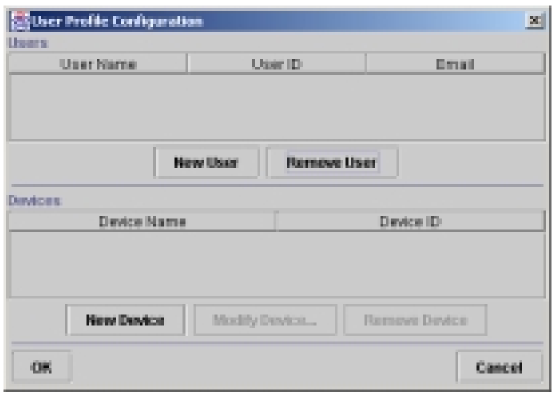
When the VHEMAT system is started for the first time, the SMA system-explorer view will be relatively empty:



To start adding/modifying user profiles for your users, click the “User Profiles” button in the toolbar, or selected Configuration→Users…. from the menu:

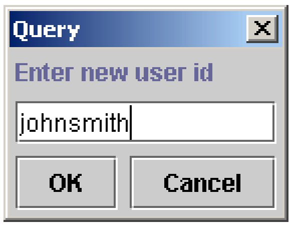


You will then be presented with the User Profile Configuration GUI:



Initially it is empty. To add a new user, follow these steps:

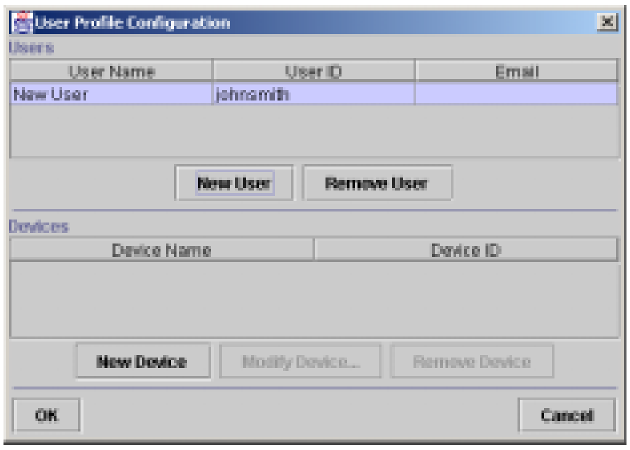
- Click “New User”. A dialog will be displayed asking for the user-id (or username) of the new user. This is the unique identifier associated with a user within the system:



Enter new user id

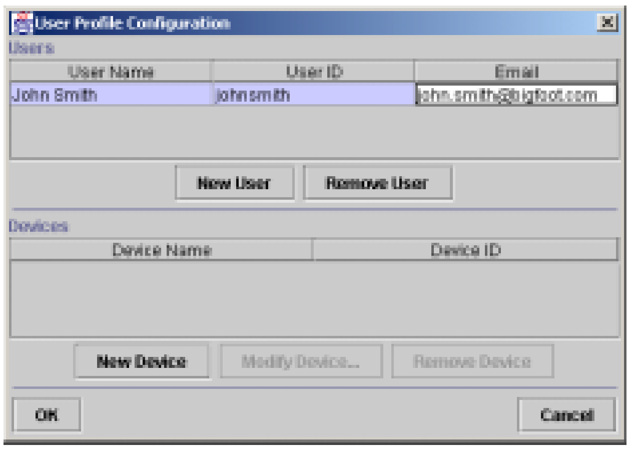
johnsmith

Once this has been entered, click “OK” and the users’ profile will be created:



User Name	User ID	Email
New User	johnsmith	

- The users real name can be entered by clicking in the area with the text “New User”, and their email address can be entered into the “Email” column by clicking in that area and typing:

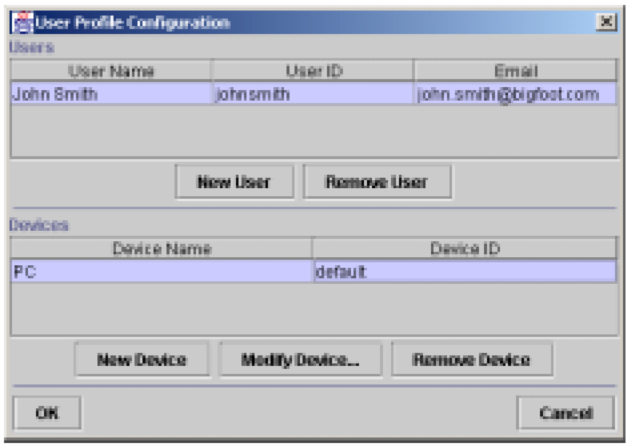


User Name	User ID	Email
John Smith	johnsmith	john.smith@bigfoot.com

Device Name	Device ID

The user will require at least one device profile setting before they can interact with the service, so setting up a default device is a good idea. This can be achieved by:

- Select a user, and clicking on “New Device”. A dialog will then be displayed asking for the device-id of the new device. To start with, just use “default”. The device profile is then added to the users’ profile:



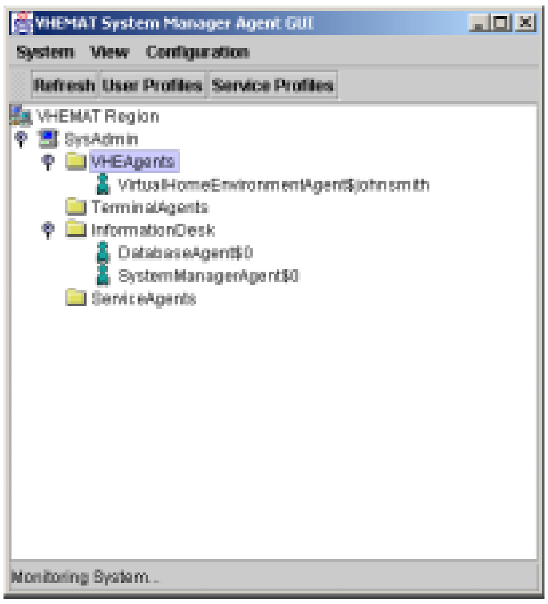
User Name	User ID	Email
John Smith	johnsmith	john.smith@bigfoot.com

Device Name	Device ID
PC	default

- The name of the device can also be entered in a similar manner to entering the users’ real name. If the device-profile needs to be modified, select it and press the “Modify Device…” button, and a dialog containing the capabilities of the device will be displayed. Modify as required, and press “OK” or “Cancel” to exit.

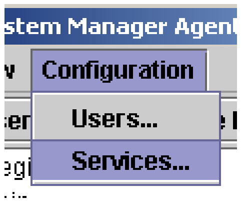
User profiles can also be removed by selecting the user profile and pressing the “Remove User” button. Device profiles can be removed in a similar manner by selecting the device profile, and pressing the “Remove Device” button. No confirmation dialog is displayed before this happens, so take care.

When all modifications to the user/device profiles, click “OK” to return to the main SMA explorer window. The SMA will then update the Agents in the system with the updated information, and start or stop the necessary VHEAgents:

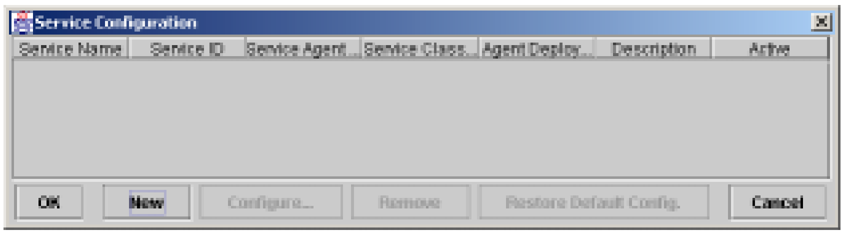

VHEMAT System Manager Agent GUI
System View Configuration
Refresh User Profiles Service Profiles
VHEMAT Region
SysAdmin
VHEAgents
VirtualHomeEnvironmentAgent$johnsmith
TerminalAgents
InformationDesk
DatabaseAgent$0
SystemManagerAgent$0
ServiceAgents
Monitoring System...

D.3.2 Configuring services

Service profiles within the system can be configured via the Service Configuration GUI. This can be reached by clicking the "Service Profiles" button in the main SMA system-explorer view, or by selecting the Configuration→Services… menu item:

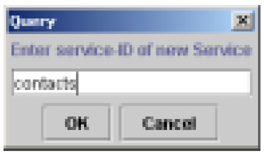


You will then be presented with the GUI:

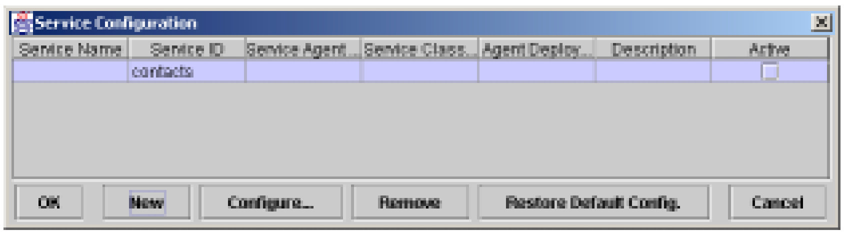


To add a service, follow these steps:

- Click the "New" button within the Service Configuration GUI. You will be prompted for the service-id of the new service:



This service-id uniquely identifies the service within the system. Press "OK", and a new service profile will be created:



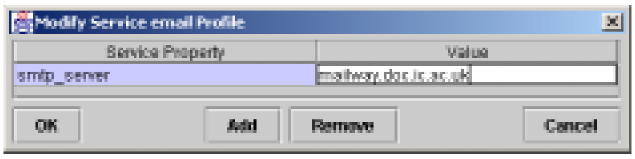
Service Name	Service ID	Service Agent	Service Class...	Agent Deploy...	Description	Active
	contacts					☐

- You can then enter the rest of the service details manually, or alternatively enter the name of either the ServiceAgent or Service class associated with this service, and hit the “Restore Default Config.” button.

  The details required are:

  - Name of the service (for display to users of the system in preference to the service-id)
  - Service Agent Classname
  - Service Classname
  - Agent Deployment Place (defines the names of the Places within the system where Service Agents for this service should be deployed, if this service uses SA’s).
  - Description of service
  - Active status (Allows a service profile that can exist for a service which is not active, but can be activated with the click of a button).

To modify any service-specific configuration details, select the service to configure and press the “Configure” button. You will be presented with a GUI to modify these configuration details:



Service Property	Value
smtp_server	mailway.doc.ic.ac.uk

To remove a service, select its profile and press the “Remove” button (caution: no confirmation prompt will be given).

When all services have been configured, click the “OK” button, and the SMA will notify all Agents of the changes, and repopulate/redeploy the system with ServiceAgents (if necessary).

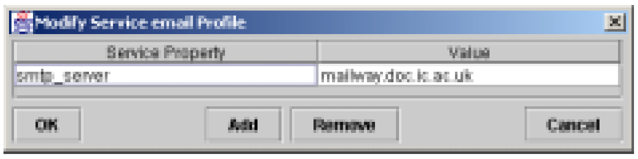
Email service

Service Agent class: `vhemat.service.email.EmailAgent`
Service class: `vhemat.service.email.EmailService`

The email service provides the ability for users to send emails via SMTP, and retrieve emails from a POP3 account.

To configure this service, create a new service profile and enter the name of the ServiceAgent class into the “Service Agent Classname” field of the profile. Now press the “Restore Default Config.” button – the service profile will be automatically completed.

Now configure the service-specific details by pressing the "Configure" button. The email-service configuration details will appear:


Modify Service email Profile

Service Property	Value
smtp_server	mailway.doc.ic.ac.uk

The "smtp_server" property defines the SMTP server through which all outgoing emails should be sent. Click "OK" to save the properties.

Click "OK" to save the service profile you have just created, and the SMA will populate the system with EmailAgents, which will be started within places called "ServiceAgents" (the default place for ServiceAgents).

Contacts service

Service class: vhemat.service.contacts.ContactsService

This service provides the ability for a user to keep a contacts list within their VHEA.

To configure the service, create a new service profile and enter the name of the Service class in to the "Service classname" field of the profile. Now press the "Restore Default Config." button – the service profile will be automatically updated.

This service has no service-specific configuration, so now click the "OK" button and the details of the service will be propagated to all VHEAgents within the system.

D.3.3 Configuring Terminal Agents

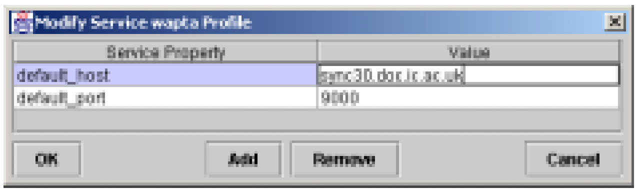
Terminal Agents are just Service Agents with specialised behaviour, so this section assumes that you are familiar with configuring service profiles.

WAP Terminal Agent

Service Agent classname: `vhemat.terminal.wap.WAPTerminalAgent`

To configure this service, create a new service profile and enter the name of the ServiceAgent class into the "Service Agent Classname" field of the profile. Now press the "Restore Default Config." button – the service profile will be automatically completed.

Now configure the service-specific details by pressing the "Configure" button. The WAP/HTTP-TA-service configuration details will appear:


Modify Service wapta Profile

Service Property	Value
default_host	sync30.doc.ic.ac.uk
default_port	9000

The "default_port" property defines the port that the WAP/HTTP-TA should attempt to start on.

The "default_host" property defines the name of the host on which the "central" WAP/HTTP-TA will exist – this TA will redirect incoming login requests to other WAP/HTTP-TA's of the same type. This will only work if there is a suitable Place on this host and the SMA populates that Place with a WAP/HTTP-TA.

Click “OK” to save the service profile you have just created, and the SMA will populate the system with WAPTerminalAgents, which will be started within places called “TerminalAgents” (the default place for TerminalAgents).

D.3.4 Details to Provide to Users

Every user will need to know the following details about the configuration of the system:

- For all Terminal Agents:
  - Their user-id (username) within the system
  - The device-id of their initial device (i.e. “default”), users can create new device profiles themselves once they have access to their VHEAgent.
- For Swing-TA’s:
  - The URL of the Region Registry. This is of the form:
    <transport>://<region-host>:<region-port>/<region-name>

    The required values are the same as those defined in `winconfig.bat` or `linuxconfig.sh` (see section D.1.3).
- For HTTP-TA’s:
  - The URL of the central HTTP-TA of each type within the system. This is of the form:
    http://<default-host>:<default-port>/

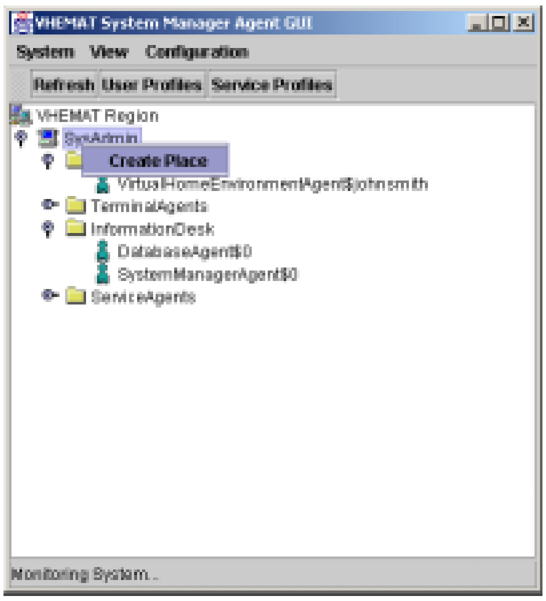
    The required values are the same as those defined in the service configuration of each HTTP-TA type.

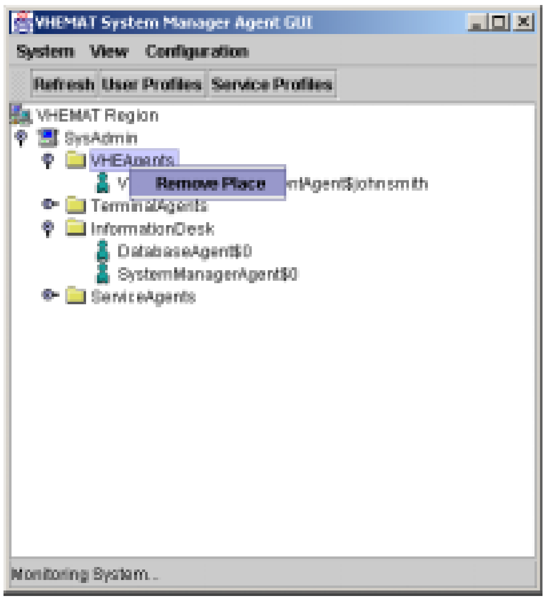
D.4. Miscellaneous features

The SMA also provides a number of other features that may assist in maintaining a VHEMAT system.

D.4.1 Ability to Add and Remove Places

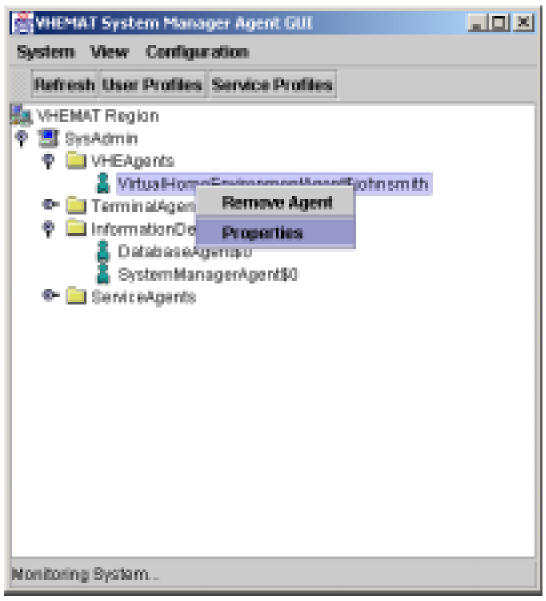
The SMA has the ability to add and remove places at runtime through the use of a context-sensitive popup-menu:

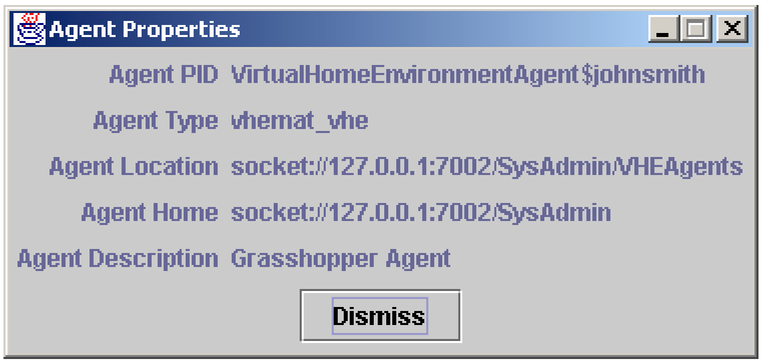

VHEMAT System Manager Agent GUI
System View Configuration
Refresh User Profiles Service Profiles
VHEMAT Region
SysAdmin
Create Place
VirtualHomeEnvironmentAgent$johnsmith
TerminalAgents
InformationDesk
DatabaseAgent$0
SystemManagerAgent$0
ServiceAgents
Monitoring System...



D.4.2 Ability to View Agent Properties and Remove Agents

The properties of Agents can be viewed via another context-sensitive popup menu:

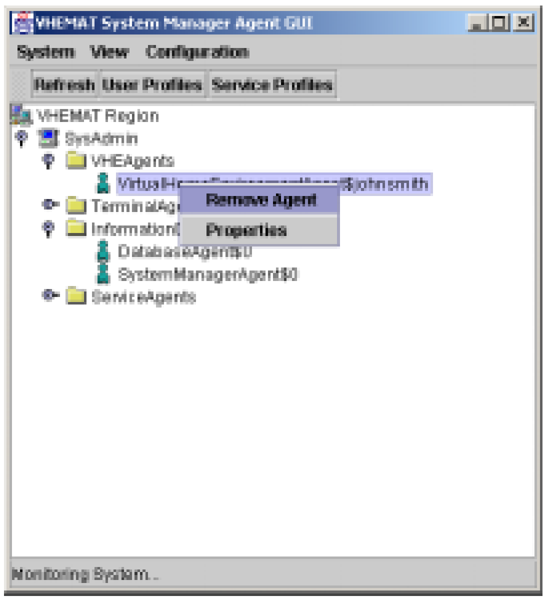




Agent Properties

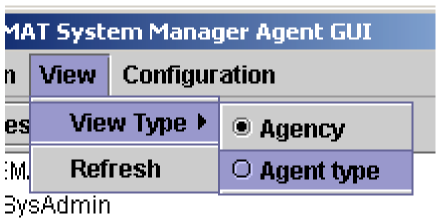
Agent PID	VirtualHomeEnvironmentAgent$johnsmith
Agent Type	vhemat_vhe
Agent Location	socket://127.0.0.1:7002/SysAdmin/VHEAgents
Agent Home	socket://127.0.0.1:7002/SysAdmin
Agent Description	Grasshopper Agent

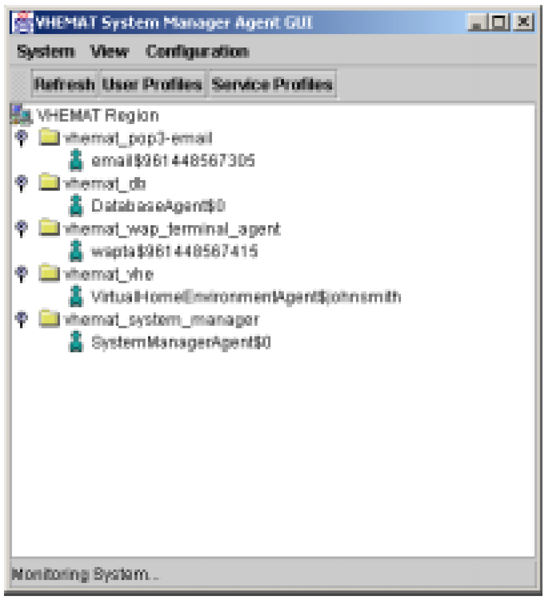
In the event of an Agent becoming unstable, it can be removed through the same context-sensitive popup-menu:



D.4.3 Changeable View Type

The view of the VHEMAT system is not confined to grouping by Agency, the Agents can also be grouped by type to make it easier to see how many of each Agent type are active within the system:





```
VHEMAT Region
  vhemat_pop3-email
    email$961448567305
  vhemat_db
    DatabaseAgent$0
  vhemat_wap_terminal_agent
    wapta$961448567415
  vhemat_vhe
    VirtualHomeEnvironmentAgent$johnsmith
  vhemat_system_manager
    SystemManagerAgent$0
```

Monitoring System..

Appendix E: Implementation Logbook

This appendix contains the logbook that was written whilst the VHEMAT system was being implemented for reference purposes.

E.1. Work break-down

Week	Hours logged	Summary
1	1.5	Stubbing of core classes
2	5.0	Stubbing of core classes & configuration of support software (JDK1.2.2 on Linux primarily)
3	11.5	Implementation of support classes (i.e. vhemat.util.Database)
4	10.0	
5	4.0	Implementation of DatabaseAgent
6	6.0	Start of implementation of SystemManagerAgent and associated GUI's
7	3.0	
8	30.0	
9	6.0	SystemManagerAgent and DatabaseAgent "integration"
REVISION / EXAM BREAK		
10	41.5	Integration of SMA with GUI's; Implementation of SMA core functionality; Implemented TerminalAgent shell & SwingTerminalAgent; Started implementing VHEAgent; Integrated VHEAgent and SwingTerminalAgent;
11	59.5	Continued implementation of VHEAgent; Integration of VHEAgent with DBAgent and SMA; ContactsService implemented; Interoperability testing (Win2000 & Linux); Core system testing; Implementation of HTTPTerminalAgent shell & WAPTerminalAgent
12	57.0	Further testing, bug fixes & efficiency enhancements; EmailService and EmailAgent implementation and integration with VHEAgent; Support classes for content adaptation implemented;
Total	**235.0**	

E.2. Contents of Logbook Entries

E.3. Logbook entries

Week 1 (W/C 31/1/200) – 1.5 hours

6/2/2000 – 1.5 hours

Created the following (empty) files, decided a package structure that would be appropriate, and placed them into a CVS repository. (CVS provides revision control for source code and related files):

```
vhemat\agent\VHEAgent.java
vhemat\agent\TerminalAgent.java
vhemat\agent\DatabaseAgent.java
vhemat\agent\SystemManagerAgent.java
vhemat\profile\DeviceProfile.java
vhemat\service\ServiceInbox.java
vhemat\service\ServiceTasklist.java
vhemat\service\Service.java
vhemat\service\ServiceAgent.java
vhemat\service\ServiceTask.java
vhemat\service\ServiceItem.java
vhemat\service\ServiceAgentInterface.java
vhemat\service\ServiceException.java
vhemat\service\SystemServiceConfig.java
vhemat\service\UserServiceConfig.java
vhemat\ui\UserInterface.java
vhemat\ui\UIComponent.java
vhemat\ui\UIButton.java
vhemat\ui\UIChoice.java
vhemat\ui\UITextArea.java
vhemat\ui\UIAttachment.java
```

I then proceeded to stub out the following files and add JavaDoc comments based upon the details given in the design chapter of my final report:

```
vhemat\service\Service.java
vhemat\service\ServiceAgent.java
vhemat\service\ServiceException.java
```

Week 2 (W/C 7/2/200) – 5.0 hours

7/2/2000 – 1.5 hours

- Continued stubbing out the following classes, and adding JavaDoc comments:

  ```
  vhemat\service\ServiceTask.java
  vhemat\service\ServiceItem.java
  ```

- Set-up my DOC file area so that I can transfer the CVS repository containing all of the source code from home to college and vice versa. NT version of CVS appears to work OK, but the Linux version seems completely incompatible (even on the same versions), so any development in the labs will have to be in NT. Having two copies of the repository will prevent the loss of all of the project work, and will prove to be life-saving in the event that either my home system fails or the lab systems fail.

8/2/2000 – 2.0 hours

- Stubbed out and added JavaDocs to the following classes:

  ```
  vhemat\service\ServiceTasklist.java
  vhemat\service\ServiceInbox.java
  vhemat\service\UserServiceConfig.java
  vhemat\service\SystemServiceConfig.java
  vhemat\profile\DeviceProfile.java
  ```

  Created, stubbed and JavaDoc'ed the following classes:

  ```
  vhemat\profile\ServiceProfile.java
  vhemat\profile\UserProfile.java
  ```

  Given all of the classes/interface currently stubbed out, work should now be able to start on implementing the ServiceAgent shell, and the first Agent to make use of it, the DBAgent.

- Discovered that the RedHat 6.1 machines in the lab support Sun's JDK1.2.2 Release Candidate 2 -- these include the Dynamic machines, which are dual-bootable (not very useful since someone could come along and reboot the machine at a critical moment) and sync26, sync32 and a few others that are NOT dual-bootable.

  The aim is that several Agencies can be run across a number of machines in the labs remotely for the purpose of testing and demonstrations.

  This version of JDK1.2.2 will be used in preference to the older version I was previously expecting to use (JDK1.2.2 pre-release version 1) which worked on all machines in the lab.

- Installed the EPOC C++ SDK, which contains an EPOC emulator for testing C++ code (although it doesn't have to be used soley for this purpose). Unfortunately it appears as if the only way to interact with another machine via a TCP/IP stack from within the emulator is to dial-out to an ISP since the emulators' COM ports are mapped directly onto the host machines COM ports. No mention of a "virtual-hub" (e.g. TCP/IP implementation in emulated EPOC device is "virtually" connected to the host systems TCP/IP stack, so EPOC device can communicate over whatever network the host system is connected to) is made in the documentation.

11/2/2000 – 1.5 hours

- Installed Sun's Forte in to my user area – given that it provides in built support for CVS (& much more) it will provide a better alternative to notepad when editing code in the labs (in fact I may decide to use it at home as well providing its doesn't contain too many bugs – its only a release candidate at the moment).

- Started filling in the methods in ServiceAgent, and created the TaskInvocationException class and sub classes TaskNotSupported and TaskExecutionException.

Week 3   (W/C 14/2/2000) – 11.5 hours

14/2/2000 – 3.0 hours

- Created the vhemat.util.Debug class that provides appropriate methods for logging debugging messages. In particular, each debug message has a "fatalness" value associated with it such that unimportant messages are filtered out from the standard output, but all messages are written into a log file on disk. A generated line of debug output is of the form:

  ```
  Mon Feb 14 12:05:34 GMT 2000: No ServiceAgent's available
  ( ServiceAgent.java:123 / level 5 )
  ```

  Hence each debug message will contain the time and date it occurred, a reference to the code generating the message and the "fatalness" level of the message, which will hopefully improve the amount of time spent bug hunting.

- Started adding code into ServiceAgent to deal with Grasshopper specific implementation details (registering services with RegionRegistry, locating and asking other Agents to perform ServiceTask's etc..)

- Discovered that the Stubgen.bat file provided with Grasshopper2 is broken, and hence generating the stubs required for inter-agent communications using specific interfaces (e.g. the SeviceAgent interface) is impossible. I have posted a message on the Grasshopper2 discussion board in order to find out if I'm doing something wrong or if there is a bug in the stub generator. Hopefully I should receive a prompt reply – all other questions posted on the board by other people have been answered fairly quickly by members of the IKV development team. [Solved: See 15/2/2000]

15/2/2000 – 3.5 hours

- Started stubbing out the DatabaseAgent class in preparation for its implementation.

- Updated the setSender() method in the ServiceTask interface to be package visible only in order to prevent service implementation from attempting to "spoof" their identity in order to gain access to services (e.g. information within the DatabaseAgent which they don't normally have access to).

- Successfully generated stubs for ServiceAgentInterface implementations after prompting from the guys at IKV++. Below is a transcript of emails posted in the Grasshopper discussion forum at http://www.ikv.de:

  Original message

  ```
  Hi

  I've been  izarre ng to generate stubs for Agents' I'm
  creating using Grasshopper2 beta using JDK1.2.2 under
  Win98. No matter what I do I get the following response:

  H:\Project>stubgen vhemat.service.ServiceAgentInterface
  ..\lib\gh.jar;..\lib\jndi.jar;..\lib\ldap.jar;..\lib\prov
  iderutil.jar;H:\Project

  ! Class vhemat.service.ServiceAgentInterface or one of
  the referred classes not found:
  vhemat.service.ServiceAgentInterface
  Please check your CLASSPATH or the –classpath option.
  ```

Done.

This is definitely NOT a classpath problem since H:\Project\vhemat\service\ServiceAgentInterface.class exists. I've also tried specifying the filename of the class, changing the current directory to that of the class file and executing "stubgen ServiceAgentInterface.class" and everything else I can think of. All of the classes it refers to are also in the classpath.

If someone could point out what I'm doing wrong, or a fix for the stub generator it would be much appreciated

Cheers,
Alan

1st reply

Hi,

I assume you have following classpath

```
set
classpath=..\lib\gh.jar;..\lib\jndi.jar;..\lib\ldap.jar;.
.\lib\providerutil.jar;H:\Project
```

and then you call

```
stubgen vhemat.service.ServiceAgentInterface
```

and this does not work?

Looking at the text it looks like you specify the classpath behind the interface class which is not correct.

Hmm, it looks like the '-classpath' option does not work. Please set the full classpath before calling the stubgen.

I hope this solves your problem

…
executing "stubgen ServiceAgentInterface.class" and everything else
…

Please use the normal java package/class syntax for the stubgen and do not append '.class'

Best regards

Christoph

My reply

Hi

Thanks for the suggestions – they prompted the solution

```
to the real problem (which turns out to be quite
 izarre). The problem was a kind of classpath problem.

In order to reduce the length of my classpath I
previously placed the Grasshopper JARs into
JDK1.2.2\JRE\LIB\EXT which has the effect of including
them in the classpath.

From the Grasshopper bin directory I ran the stubgen.bat
with the following arguments:

stubgen –d H:\Project\vhemat\service
vhemat.service.ServiceAgentInterface

and with the classpath set to:

classpath=.;H:\Project

The following two cases highlight the output from the
stubgen.bat file:

Case 1: Grasshopper JAR’s in JDK1.2.2\JRE\LIB\EXT
==================================================
..\lib\gh.jar;..\lib\jndi.jar;..\lib\ldap.jar;..\lib\prov
iderutil.jar;.;H:\Project
! Class vhemat.service.ServiceAgentInterface or one of
the referred classes not found:
vhemat.service.ServiceAgentInterface
Please check your CLASSPATH or the –classpath option.

Done.

Case 2: Grasshopper JAR’s NOT in JDK1.2.2\JRE\LIB\EXT
======================================================
Generating vhemat.service.ServiceAgentInterfaceP…
Wrote file
H:\Project\vhemat\service\ServiceAgentInterfaceP.java

Done.


So, the motto of the story is not to put things in the
JRE\LIB\EXT directory which don’t belong there! The error
message was a little mis-leading though.

Thank you for your help, I hope I haven’t wasted too much
of your time.

Cheers,
Alan
```

18/2/2000 – 2.0 hours

- Downloaded and installed into my DoC file area JDK1.2.2 Final Release for Linux which was released earlier this week. According to the documentation it fixes several bugs related to socket connections, which could have potentially affected the system I’m constructing.

- Started work on the vhemat.util.Database class that will provide all of the necessary functionality to deal with a persistent serialised database. Wrote the loading routine to de-serialise the database and a method which will attempt to interrogate an object for its primary key using reflection.

19/2/2000 – 3.0 hours

Continued work on the vhemat.util.Database class – added mechanisms for serialising the database which will operate concurrently to transactions being carried out on the database providing a speed up when a database becomes large (the Database object will write the entire database back to disk for simplicity rather than attempting to update individual entries – Java serialisation would not allow this easily anyway). Care has been taken to ensure that the data within the database being serialised is decoupled from the "live" database in order to allow this without inconsistencies appearing in the serialised database. Also added methods for inserting/updating and deleting entries from the database (although not the code to check that these are valid actions based upon the Agent requesting these transactions).

Week 4 (W/C 21/2/2000) – 10.0 hours

26/2/2000 – 6.5 hours

Continued working on the vhemat.util.Database class. Implemented the following functionality:

- Parsing of access rules
- Insertion and deletion of objects from the database
- Correct interpretation of access rules for inserting, and deleting objects within a database.
- Test harness for repeatable testing of the above functionality

Output from test harness:

```
Database test harness
=====================


Test objects have methods:
        String getName()
        int getAge()
        boolean getMale()
        Object get( String )


Access rules for database = INSERT:invoker=object-
owner;QUERY:invoker=*;DELETE:invoker=object-owner
DB owner = me, Sys.Admin = SysAdmin

Creating DB with key "name" and above access rules


Populating DB with these entries as the DB owner:
[Name="Bob";Age=35;Male=true;Other=null]
Result = true
[Name="John";Age=21;Male=true;Other=money]
Result = true
[Name="Jane";Age=29;Male=false;Other=honey]
Result = true


Attempting to insert into the database as someone else:
[Name="Bill";Age=29;Male=true;Other=null]
```

```
Result = true


Attempting to UPDATE the database as DB owner:
[Name="Bob";Age=29;Male=true;Other=null]
Result = true


Attempting to UPDATE the database as someone else:
[Name="Bob";Age=29;Male=true;Other=null]
Result = false
[Name="Bill";Age=31;Male=true;Other=null]
Result = true


Attempting to DELETE fron the database as DB owner:
[Name="John";Age=21;Male=true;Other=money]
Result = true


Attempting to DELETE the database as someone else:
[Name="Bob";Age=29;Male=true;Other=null]
Result = false
[Name="Bill";Age=31;Male=true;Other=null]
Result = true
```

27/2/2000 – 3.5 hours

Completed the implementation of the Database class. Added the following functionality:

- Parsing of queries
- Evaluation of queries
- Extended test harness to test new functionality

Test harness output:

```
Database test harness
=====================


Test objects have methods:
        String getName()
        int getAge()
        boolean getMale()
        Object get( String )


Access rules for database = INSERT:invoker=object-
owner;QUERY:invoker=*;DELETE:invoker=object-owner
DB owner = me, Sys.Admin = SysAdmin

Creating DB with key "name" and above access rules


Populating DB with these entries as the DB owner:
[Name="Bob";Age=35;Male=true;Other=null]
Result = true
[Name="John";Age=21;Male=true;Other=money]
Result = true
[Name="Jane";Age=29;Male=false;Other=honey]
```

```
Result = true


Attempting to insert into the database as someone else:
[Name="Bill";Age=29;Male=true;Other=null]
Result = true


Attempting to UPDATE the database as DB owner:
[Name="Bob";Age=29;Male=true;Other=null]
Result = true


Attempting to UPDATE the database as someone else:
[Name="Bob";Age=29;Male=true;Other=null]
Result = false
[Name="Bill";Age=31;Male=true;Other=null]
Result = true


Attempting to DELETE fron the database as DB owner:
[Name="John";Age=21;Male=true;Other=money]
Result = true


Attempting to DELETE the database as someone else:
[Name="Bob";Age=29;Male=true;Other=null]
Result = false
[Name="Bill";Age=31;Male=true;Other=null]
Result = true


Test query = male=true


Performing query as DB-owner:
Result = [[Name="Bob";Age=29;Male=true;Other=null]]


Performing query as someone else:
Result = []


Closing database!!!!!!!
```

As can be seen there is a bug in the Database object – this will be investigated at a future point (once I can get the debugger working in Forte4J).

Also completed (but not tested) the implementation of the DatabaseAgent based upon the use of the Database class. (A suitable ServiceAgent test harness has yet to be produced)

Week 5 (W/C 28/2/2000) – 4.0 hours

4/3/2000 – 4.0 hours

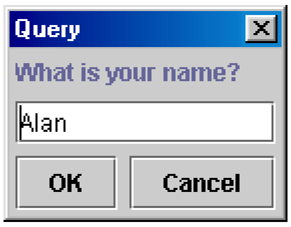
Completed debugging the Database class and successfully compiled the DatabaseAgent. Since the SystemManagerAgent will be the next Agent to be developed, I will use this to test the functionality of the DBAgent.

Week 6 (W/C 6/3/2000) – 6.0 hours

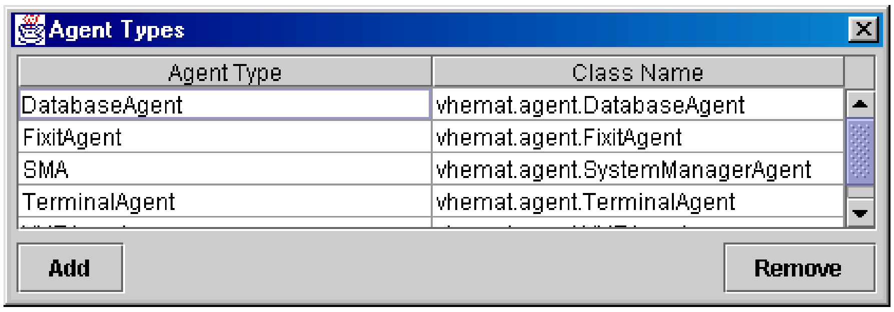
11/3/2000 – 6.0 hours

Started working on the SystemManagerAgent. Given the amount of information that is required by the SMA, I have decided to implement the management GUI initially to enable configuration of the system in a user-friendly manner. Using Forte, I have so far started 4 of the GUI's components (windows) and completed 2:

- vhemat.agent.gui.StringDialog: Displays a general purpose query and a text-field for the user to give a response. The query and default response can be set as required, and the query blocks the calling thread until either ok, cancel or the close button is pressed.



- vhemat.agent.gui.SMAAgentTypeDialog: This is a more complex GUI component – given a Map between Agent-type and associated class name, this dialog will display a list of these associations and allow them to be modified. This will allow agents to be referred to by their type rather than class name, and for new agent types to be added and removed within the SMA (and hence the system).

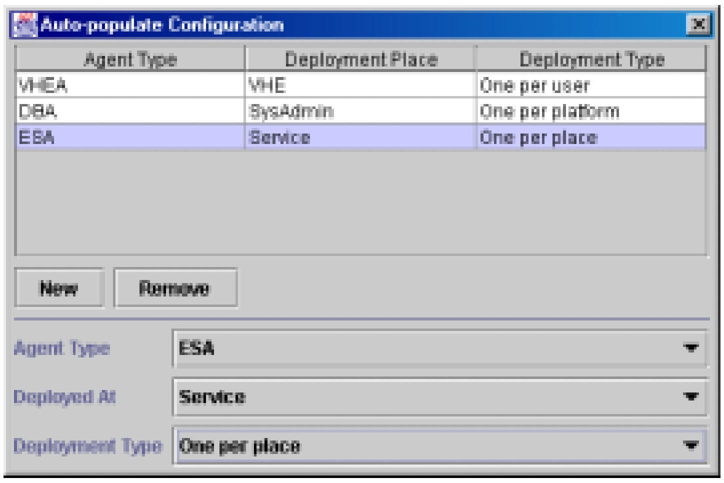


Agent Type	Class Name
DatabaseAgent	vhemat.agent.DatabaseAgent
FixitAgent	vhemat.agent.FixitAgent
SMA	vhemat.agent.SystemManagerAgent
TerminalAgent	vhemat.agent.TerminalAgent

Week 7 (W/C 13/3/2000) – 3.0 hours

18/3/2000 – 3.0 hours

Completed work on the vhemat.agent.gui.SMAAutoPopulateDialog class: It provides the ability to modify the list of Agents that are automatically populated within the system at start-up, as well as where and how many agents of each type should exist.



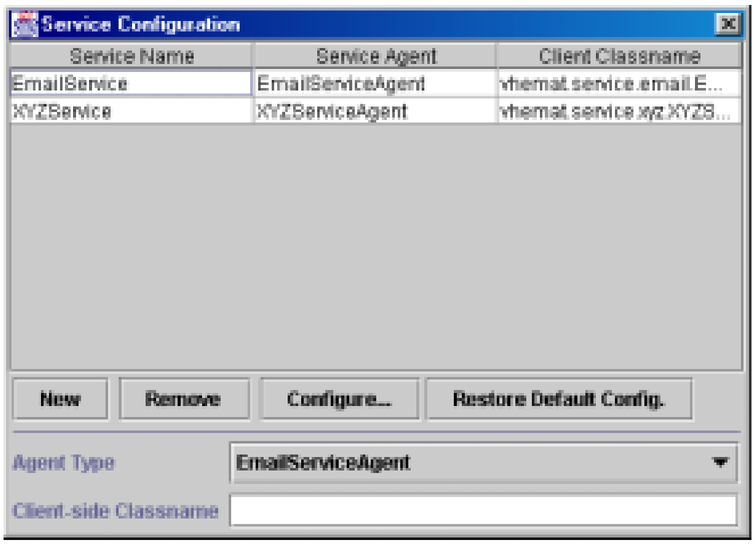
Agent Type	Deployment Place	Deployment Type
VHEA	VHE	One per user
DBA	SysAdmin	One per platform
ESA	Service	One per place

Agent Type: ESA
Deployed At: Service
Deployment Type: One per place

Week 8   (W/C 20/3/2000) – 30.0 hours

20/3/2000 – 7.0 hours

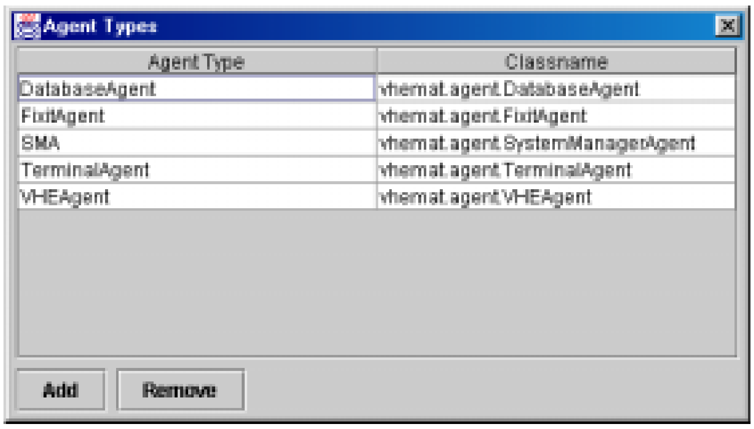
- Started work on a service configuration GUI component (vhemat.agent.gui.SMAServiceConfigDialog) that allows the services within the systems to be configured from the SMA.



Service Name	Service Agent	Client Classname
EmailService	EmailServiceAgent	vhemat.service.email.E...
XYZService	XYZServiceAgent	vhemat.service.xyz.XYZS...

Agent Type: EmailServiceAgent
Client-side Classname:

- Generalised the SMAAgentTypeDialog (scrapped) and SMAPlaceTypeDialog (created today & scrapped) into the vhemat.agent.gui.PropertyDialog that allows a Map of information to be modified in a GUI. Due to the fact that there are many GUI's that will probably be doing similar tasks, I decided that one generalised class for this purpose would lead to a much neater solution than multiple version of the same thing!


Agent Types

Agent Type	Classname
DatabaseAgent	vhemat.agent.DatabaseAgent
FixitAgent	vhemat.agent.FixitAgent
SMA	vhemat.agent.SystemManagerAgent
TerminalAgent	vhemat.agent.TerminalAgent
VHEAgent	vhemat.agent.VHEAgent

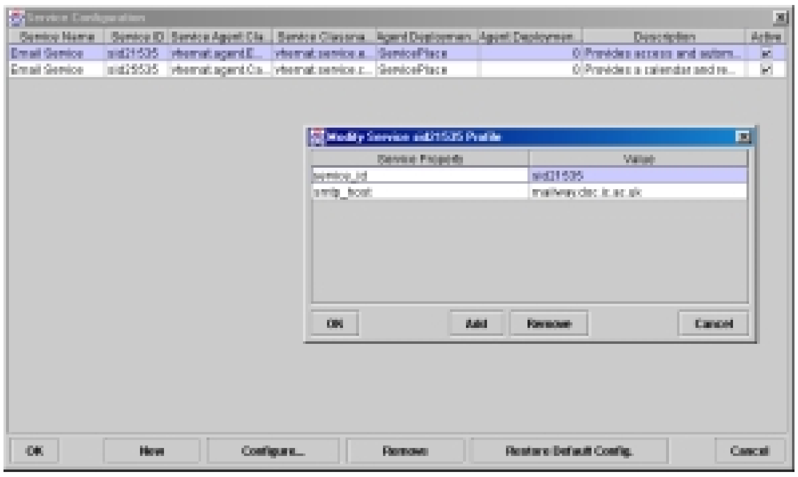
- Produced a convenience class to remove code duplication where TableModel implementations were being produced for each of the GUI's, called whemat.agent.gui.AbstractTableModel.

- Created vhemat.util.DataContainer which implements all of the interfaces within the vhemat.* sub-packages which provide methods to store/retrieve name value pairs (or can be abstracted to this model). This includes: ServiceItem, ServiceTask, SystemServiceConfig, UserServiceConfig, DeviceProfile, ServiceProfile and UserProfile interfaces. The object has been designed to be Serializable (so it can be passed between Agents) and has method which are appropriate so that it can be interrogated by the Database object – this involved renaming the methods setValue() and getValue() in some of the interface mentioned above to set() and get() respectively.

21/3/2000 – 8.0 hours

- Generalised the vhemat.agent.gui.PropertyDialog class further by changing the data structure it allows to be modified to a DataContainer.

- Completed vhemat.agent.gui.SMAServiceConfigDialog – All service profile information is now displayed within the GUI, and selecting the "Configure" button provides a PropertyDialog to modify the SystemServiceConfig associated with the service.


Service Configuration

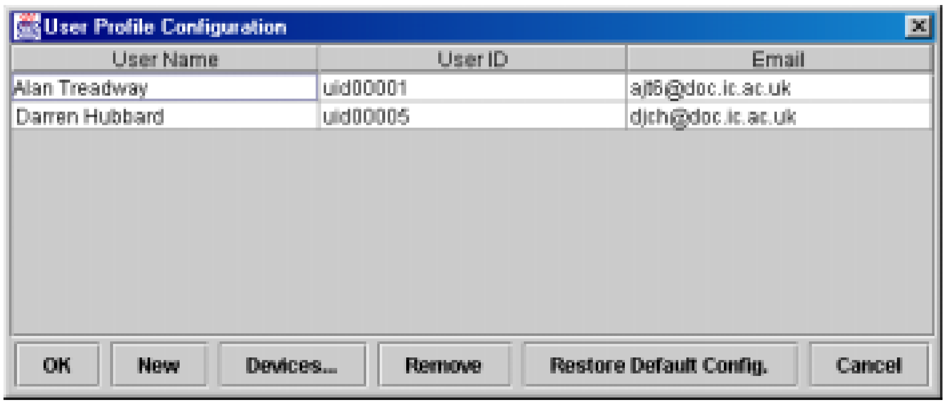
Service Name	Service ID	Service Agent Cla...	Service Classna...	Agent Deploymen...	Agent Deploymen...	Description	Active
Email Service	sid21535	vhemat.agent.E...	vhemat.service.a...	ServicePlace	0	Provides access and autom...	✓
Email Service	sid25535	vhemat.agent.Ca...	vhemat.service.c...	ServicePlace	0	Provides a calendar and re...	✓

Modify Service sid21535 Profile

Service Property	Value
service_id	sid21535
smtp_host	mailway.dcs.kcl.ac.uk

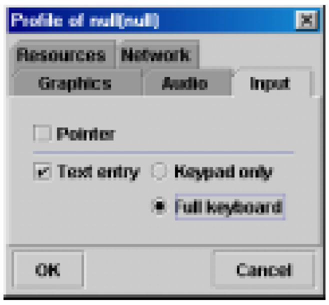
A mechanism to automatically populate the SystemServiceConfig has yet to be devised.

- Started vhemat.agent.gui.SMAUserConfigDialog – This provides a mechanism for modifying user profiles.



User Name	User ID	Email
Alan Treadway	uid00001	ajt6@doc.ic.ac.uk
Darren Hubbard	uid00005	djch@doc.ic.ac.uk

- Completed work on vhemat.agent.gui.SMADeviceProfileDialog – Provides the ability to modify DeviceProfile instances.

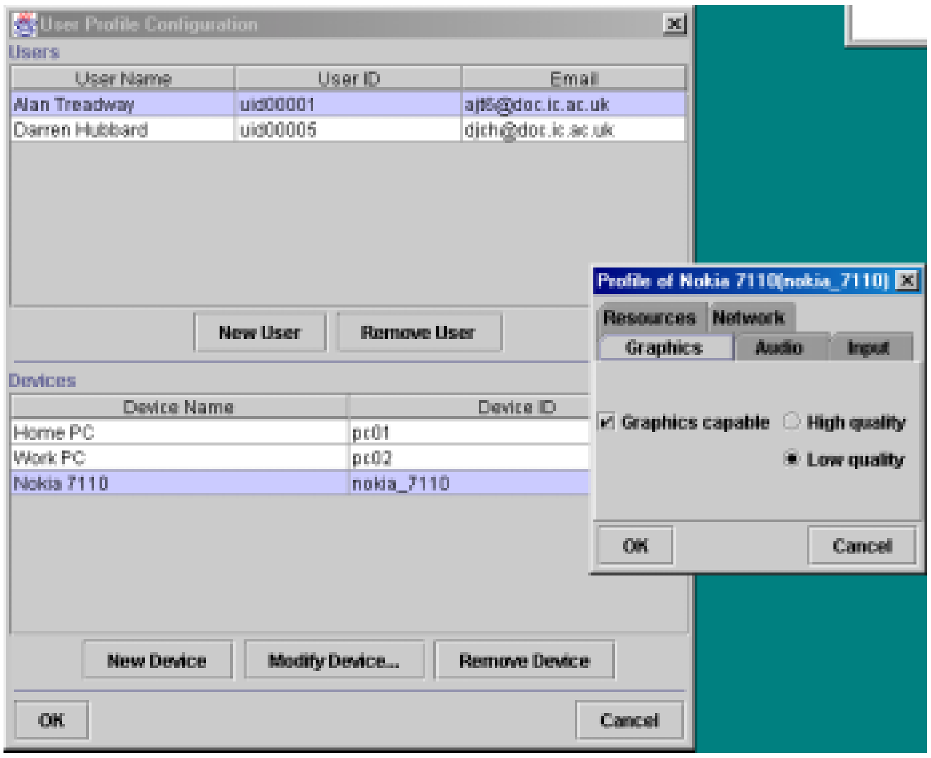


☐ Pointer

☑ Text entry ○ Keypad only ● Full keyboard

24/3/2000 – 8.0 hours

- Completed vhemat.agent.gui.SMAUserConfigDialog – Intergrated use of SMADeviceProfileDialog to enable modification of User's DeviceProfile's.



User Name	User ID	Email
Alan Treadway	uid00001	ajt6@doc.ic.ac.uk
Darren Hubbard	uid00005	djch@doc.ic.ac.uk

Device Name	Device ID
Home PC	pc01
Work PC	pc02
Nokia 7110	nokia_7110

☑ Graphics capable ○ High quality ● Low quality

During the process of creating this GUI, I discovered that due to the fact that dialogs were being opened within GUI event handler methods and that dialogs block the invoking thread until they have been completed, significant slow down was caused as a result of blocking the event-handler threads. In order to counter act this, all event handling code that may block is executed in a new thread to prevent this from happening.

- Started integrating the SMA with the SMAFrame. The SMAFrame can now interact with the SMA to provide a basic overview of the state of the system using an explorer-like view.

- SMA auto-detects presence of DBA on platform, and creates a new one if none are found.

- Renamed ServiceAgent to ServiceAgentImpl, ServiceAgentInterface to ServiceAgent, and reproduced the proxy class (now SerivceAgentP). This better matches the naming conventions adopted for the rest of the classes.

25/3/2000 – 5.0 hours

- Attempted to start SMA/DBA synchronisation methods. Code to check for user profile and service profiles within the DBA, and take appropriate action have been written but not tested. For some reason the following exception occurs whenever the SMA attempts to communicate with the DBA:

```
de.ikv.grasshopper.communication.RemoteRuntimeException: No reason
provided. Nested exception:
  java.lang.NoSuchMethodException
        at
de.ikv.grasshopper.communication.MessageForwarder.callInstanceMethod(Me
ssageForwarder.java:187)
        at
de.ikv.grasshopper.communication.MessageForwarder.invoke(MessageForward
er.java:123)
        at
de.ikv.grasshopper.communication.CommunicationClient.invokeLocal(Commun
icationClient.java:441)
        at
de.ikv.grasshopper.communication.CommunicationClient.invokeInstanceMeth
od(CommunicationClient.java:154)
        at
de.ikv.grasshopper.communication.CommunicationService.invokeInstanceMet
hod(CommunicationService.java:369)
        at
de.ikv.grasshopper.communication.ObjectP.invoke(ObjectP.java:104)
        at
vhemat.service.ServiceAgentP.executeTask(ServiceAgentP.java:65)
        at
vhemat.agent.SystemManagerAgentImpl.synchroniseWithDB(SystemManagerAgen
tImpl.java:257)
        at
vhemat.agent.SystemManagerAgentImpl.live(SystemManagerAgentImpl.java:76
)
        at de.ikv.grasshopper.agency.Agent.run(Agent.java:343)
        at java.lang.Thread.run(Thread.java:479)
```

The exception implies that the DBAgent doesn't have a method executeTask(), however it extends ServiceAgentImpl, so it must through inheritance.

- Added vhemat.util.DebugExceptionDialog to provide a GUI to display fatal(ish) exceptions, rather than just printing them to the stdout.

26/3/2000 – 2.0 hours

Discovered the reason why I was getting the NoSuchMethodException yesterday, the following email I posted to the Grasshopper discussion forum today explains the problem:

```
Hi Guys,
I think I've discovered a subtle bug in GH2.0 beta 3 which is
caused (I believe) as a result of the proxy code generated by
stubgen to be incorrect.

I've successfully reproduced the bug, and have a number of test
agents which highlight the problem and a short-term solution -
if you would like to see them just email me.

Given an interface X:

public interface X {
      public void M( Y );
}

(Y is either a class of interface) and a class Z which
extends/implements Y, the following code causes a
NoSuchMethodException (embeded in a RemoteRuntimeException) to
be thrown:

Y object = new Z();
....
X agent = (X) ProxyGenerator.newInstance( X.class, .... );
....
agent.M( object );

However, if the argument passed to M is an instance of Y, the
method is invoked without any problems.  The reason for this
appears to be: The arguments passed within the generated proxy
object (i.e. XP) to the ObjectP.invoke() method include the
classes of the parameters passed to method M as the parameter
types of the method of the remote object, rather than the
expected types. So in the example above, a method M accepting
an argument of type Z was expected to be found remotely, but
was not (hence the exception). In actual fact the signature of
the method being invoked remotely for a particular proxy method
remains constant, so there should be no reliance on the type of
arguments passed at all.


The short-term solution is to modify the generated proxy
classes to do this. E.g.:

public void M( Y inParameter0 )
{
      ....
      result = invoke ("M", new Class[]
      {inParameter0.getClass()}, new Object[] { inParameter0},
      java.lang.Void.TYPE);
      ....
}

becomes:
```

```
public void M( Y inParameter0 )
{
      ....
      result = invoke ("M", new Class[] {Y.class}, new Object[]
      { inParameter0}, java.lang.Void.TYPE);
      ....
}
```

```
(Notice the subtle change from inParameter0.getClass() to
Y.class )

This then allows the code sample given above to invoke the
method M with parameter of type Z on a remote Agent without any
problems.

Cheers,
Alan
```

The short-term solution is simply to modify the proxy objects generated for the ServiceAgent interface (and others) to accommodate this problem (as described above).

Week 9 (W/C 27/3/2000) – 6.0 hours

27/3/2000

Received a reply to my posting on the Grasshopper Discussion board confirming my discovery:

```
From: JanM (jan)
To: IKV++ Products / Grasshopper Beta20 Discussion / Bug in
stubgen generated proxy objects
Subject: RE:Bug in stubgen generated proxy objects
Message:
Hi Alan!

Thanks for your effort! It is indeed a very nasty bug and very
hard to find.
We have no "Duke-Dollars" program or something similar but we
hope that it does not prevent you from reporting further bugs
:)

Greetings, Jan
----------------------------------------
Thank you for posted message in our IKV++ Community forum,
WebMaster, IKV++ Community
kindlein@ikv.de
http://www.ikv.de
```

1/4/2000 – 6.0 hours

Completed writing the SMA's DB synchronisation methods to allow the service and user profiles local held by the SMA to be synchronised with those stored persistently with the Database Agent. The SMA will now startup, create a DBAgent, and query it for the service and user profiles it has successfully.

BREAK FOR REVISION AND EXAMINATIONS

Week 10 (W/C 22/5/2000) – 41.5 hours

22/5/2000 – 8.5 hours

Recommenced work on project after break for revision and examinations.

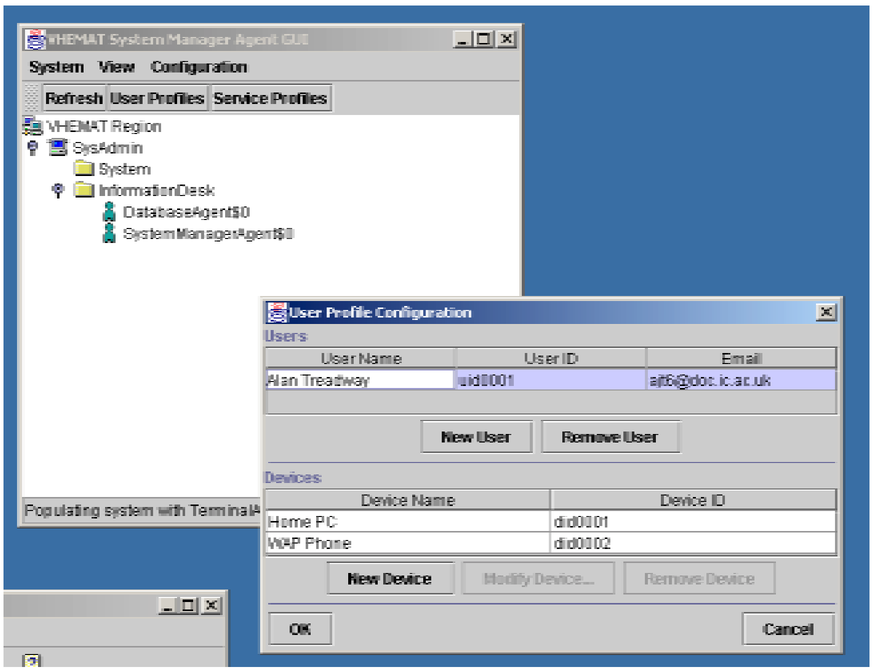
Implemented new functionality within the SystemManagerAgent:

- Added ability to modify the user profiles and service profiles from the SMA's GUI – this involved use of the previously creates SMAUserConfigDIalog and SMAServiceConfigDialog as per project requirements.
- Generalised DatabaseAgent synchronisation code to deal with any number of Database's by simply adding a list containing the name of the database, the List object to hold the local version of the database, and the primary key of the database.
- Refresh button / menu item now refreshes system view, and the type of view (i.e. agents grouped by Agency or Agent-type) can be selected from the GUI.
- Added new icons to the TreeModel used as part of the SMA's GUI.
- Added confirmation request to "shutdown" menu item / close window button.

Found and fixed the following bugs whilst doing this:

- Discovered that the list of recently accessed Database objects within the DatabaseAgent was (i) not being purged correctly, causing a fatal exception in the DatabaseAgent, and (ii) not correctly comparing database names (causing multiple versions of the same Database to be referenced in the list at once).
- Database objects would not allow PID's to appear on the RHS of access expressions, fixed such that PID's are treated as String constants for the purposes of evaluating queries.
- ServiceAgent's were not using their PID to uniquely identify themselves, they now correctly report their name as being their PID.

The SMA GUI now looks like this:



User Name	User ID	Email
Alan Treadway	uid0001	ajt6@doc.ic.ac.uk

Device Name	Device ID
Home PC	did0001
WAP Phone	did0002

23/5/2000 – 11 hours

Added the following functionality to the system:

- Added an onRemove() method to the ServiceAgentImpl class – this is invoked just before an Agent is removed from the Grasshopper system. The method ensures that all currently active tasks within the ServiceAgent are completed before the Agent terminates, and that no other new tasks are started. Once all outstanding tasks have been completed, a new abstract method onRemoval() is invoked – any concrete ServiceAgent sub-classes must implement this method in order to carry-out resource deallocation in a "system-friendly" manner.

- Updated the SMA so that it scans the system populating places within it based upon the current ServiceProfiles' it has. During this process, Agents that are "in the wrong place" or belong to a service that has been deactivated are also removed.

- Right clicking an Agent in the SMA GUI now brings up a pop-up menu that allows the properties of that Agent to be displayed to the user in a new window.

- Started working on the system-shutdown routine that removes all Agents from the system. Haven't had much luck so far due to an inability to successfully interact with remote Agencies, and ask them to remove specific Agents.

Found and fixed the following bugs:

- Debug.log() was still reporting the wrong calling method in some circumstances (stack-trace seems to change length based upon the type of runtime compiler being used!). Fixed this by replacing a search for a particular line number, with a search for a particular method invocation (i.e. the last one containing "vhemat.util.Debug.log") and assuming the next line of the stack trace has the information required.

24/5/2000 – 8.0 hours

Added the following functionality to the system:

- Created vhemat.terminal.TerminalAgent interface (and grasshopper proxy object) based upon design documentation.

- Created vhemat.terminal.TerminalAgentImpl abstract class to provide "wrapper" around TA implementations, as per design documentation. Wrote all necessary code to track users sessions and associated callback proxies (which allow interaction with VHEAgents').

- Created vhemat.terminal.UICallback interface (and grasshopper proxy object) based upon design documentation.

- Improved the system monitor to be more efficient with regards to updating the internal view of the system, and as a result the update of the GUI has also been improved.

- Added extra pop-up menus to the SMA GUI to allow adding/removing of places within Agencies connected to the system.

Found and fixed the following bugs:

- Fixed the shutdown routine within the SMA – all Agents are now removed correctly. The problem arose due to the inability to communicate with the Agencies the Agents were located within, because the Agency address could not be obtained. This was resolved by creating a class (vhemat.util.Lookup) with a number of static methods to generate the Grasshopper address of an Agency given an Agents' or Places' GrasshopperAddress within it.

- Added code to ignore the ThreadDeath exception thrown when the SMA invokes its remove() method, since it is just caused by the Agent being garbage collected.

- Added ScrollPane to the SMA GUI to allow the system view to be scrolled around.

27/5/2000 – 6.0 hours

In preperation for the creation of the first TerminalAgent implementation (i.e. the SwingTerminalAgent), added the following functionality to the system:

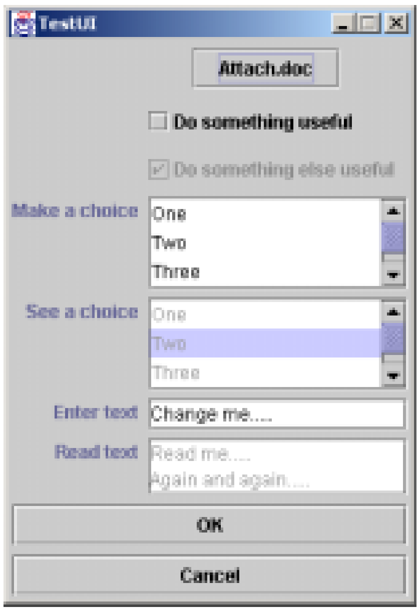
- Added all necessary code to the abstract user-interface classes in the vhemat.ui package based upon the design document.

- Created classes to support the production of the SwingTerminalAgent within the vhemat.terminal.swing package, which effectively wrap instances of the "abstract" user-interface classes in the vhemat.ui package. No major problems were faced during the creation of these classes.

The test-harness for the code produced today highlights how the generalised UI classes can be used to directly build a GUI:

```
UserInterface ui = new UserInterface( "TestUI" );

String choice[] = new String[] { "One", "Two", "Three", "Four" };

ui.add( "1", new UIAttachment( "Attach.doc", new byte[1024] ) );
ui.add( "2", new UICheckbox( "Do something useful", true, false ) );
ui.add( "3", new UICheckbox( "Do something else useful", false, true ) );
ui.add( "4", new UIChoice( "Make a choice", true, choice, -1 ) );
ui.add( "5", new UIChoice( "See a choice", false, choice, 1 ) );
ui.add( "6", new UITextArea( "Enter text", true, "Change me....." ) );
ui.add( "7", new UITextArea( "Read text", false, "Read me.....\....." ) );
ui.add( "8", new UIButton( "OK" ) );
ui.add( "9", new UIButton( "Cancel" ) );

System.out.println( "ui == " + ui );

new SwingUserInterface( ui );

System.out.println( "ui == " + ui );
```

Swing GUI produced from test harness:



28/5/2000 – 8.0 hours

Started constructing the VHEAgent, which will use the state-pattern to simplify its operation due to the finite-state-machine design in the design documentation. Created the bare minimum states (i.e. IDLE, PREPARING_TO_MIGRATE, REGISTERING_WITH_TA and WAITING_FOR_CALLBACK) in order to start testing the TerminalAgent shell.

Added all necessary code to the SwingTerminalAgent to provide the necessary functionality for VHEAgent/user interactions. Since a SwingTerminalAgent will execute on a users device rather than living in the network, the STA will locally store all required configuration details (e.g. users' name and password, system region location and device ID) and retrieve them each time it starts.

Problems encountered:

- Remote searching of RegionRegistration doesn't always filter results correctly (e.g. SwingTerminalAgent is searching for the VHEAgent "VirtualHomeEnvironmentAgent$alantreadway", but a search for an Agent with this name (which exists) produces no results, whereas a search for "$alantreadway" produces a list of one Agent (as expected)).

- VHEAgent correctly receives request to migrate to a SwingTerminalAgent when a user clicks the "Summon VHEAgent" in the SwingTerminalAgent window, but Agent doesn't migrate after its state changes to PREPARING_TO_MIGRATE.

- SMA GUI appears to lock-up sometimes, due to Agencies falling over/being removed from the system and the list of Agencies returned from the Region indicating they still exist. This will require further investigation.

Week 11 (W/C 29/5/2000) – 59.5 hours

29/5/2000 – 8.0 hours

Updated all Agent interface so that all methods now throw AgentException's, of which there are two main flavours RefuseException and FailureException. ServiceException has been removed, but its sub-classes have been re-grouped appropriately under these new classes. The aim is that this change will allow all Agents the ability to express refusal to carry out some action, or that they have failed.

Spent most of the day investigating and attempting to fix the problem highlighted yesterday of the VHEAgent migrating, in one form or another. After several hours, the following problems with (I believe) the Grasshopper system:

- Mobile Agents cannot remove themselves from an Agency, and they cannot remove other Agents, although the SMA (and other Stationary Agents) can remove them.

- When a mobile Agent migrates, the original Agent is left running after the migration has been completed, in complete contrast to details explained in the Grasshopper developers guide (although this may be related to the problem above).

- Agent state becomes inconsistent between threads after a migration. Two different threads report that an instance variable of the same VHEAgent are completely different values (I'm not completely sure how this is possible).

However, after further investigation most of these "bugs" can be sidestep by changing the design of the system slightly.

Rather than having one VHEAgent per user, duplicate VHEAgents will be deployed when a user requests that they wish to interact with theirVHEAgent – a "parent" VHEAgent will remain located in the "network", waiting for the results of the interactions from the duplicate Agent, which will then be removed from the system. This provides a number of advantages:

- VHEAgent is always contactable within the network

- Results of interactions will most likely be a sub-set of the entire Agent state (reducing bandwidth usage compared with migrating the entire Agent).

30/5/2000 – 10.0 hours

Pushed forwards with system implementation today. Added the following functionality:

- SystemManagerAgent will now attempt to deploy VHEAgents' if it believes some are missing from the system.

- VHEAgent can now synchronise its UserServiceConfig, ServiceTasklist and ServiceInbox with the DBAgent – this was achieved by further generalising similar methods in the SMA, and moving them to the DatabaseAgent class as static methods for use from both Agents (and other in the future if necessary).

- The VHEAgent can now start Services, provided all required information is present (i.e. UserServiceConfig, SystemServiceConfig and ServiceProfile)

- Started adding parts of the generalised user interface into the VHEAgent. Devised a UIState class to allow simple tracking of where in the UI the user currently is, which uses the state-pattern. Adapted the Service interface to make use of this class, allowing greater freedom within the Service implementations to add functionality to the VHEAgent user interface – rather than limiting the Service implementations to dictating how to represent ServiceItem's (e.g. how to present an email), this will allow functionality to be built into the representation of ServiceItems' (e.g. an email might have a "Reply to" or "Forward" button guilt into its UI).

- Created options UI for VHEAgent, with options to add, remove and configure services. Click add brings up a list of all non-subscribed services, but the configuration UI (produced by the Service implementation in question) cannot be brought up because the Service can not be started. It appears this is because the SystemServiceConfig details are not being passed to the VHEAgent from the SMA correctly, this will require further investigation.

- Started creating a simple "Contacts" service for testing purposes – the aim is that once this is completed and working, 90% of the system architecture will be in place, with the exception of content adaptation (since a contacts/address book service doesn't adapt its content).

31/5/2000 – 12.5 hours

Completed the majority of the VHEAgent implementation today, including the necessary support functions in the SMA to allow system configuration details to be forwarded to the SMA.

Overview of functionality added:

- Added functionality to SMA to populate the system with VHEAgents, one per user within the system

- Updates to ServiceProfiles' and SystemServiceConfigs' are now sent to all primary VHEAgents' (i.e. not duplicates interacting with a TerminalAgent) in parallel, due to the use of a separate thread for each messaging interaction that allows the SMA to continue its normal processing without blocking whilst it confirms that Agents have received the update messages.

- Updates to UserProfiles' are now sent to the relevant primary VHEAgent, via a similar mechanism to that for ServiceProfiles' and SystemServiceConfigs'

- VHEAgents can now correctly send their state information back to a primary VHEAgent, which can then integrate the changes in state into itself. The state information sent back only includes: New/modified ServiceInbox and ServiceItem objects, and the current UserServiceConfigs'.

- Extended IdleState of VHEAgent to process Services' one at a time (through activation of the active() method).

- Added code to PrepareForMigrationState to invoked the prepareFor() methods of all active services.

- Generalised the Agent proxy generation code, and inserted it into the Lookup class for convenience.

- Added functionality to the TerminalAgentImpl class to automatically remove duplicated VHEAgents' from the local Agency once they have sent their updated state information back to their primary VHEAgent

- Added a UIErrorState class to the vhemat.ui package to allow easy production of error messages using the UIState mechanism.

- Added UI's for the ContactsService class, so contacts can now be added, viewed, edited and removed. This has highlighted some glitches in the SwingTA GUI, which will need to be investigated.

- Added a Metrics class to vhemat.util, which can currently calculate the serialised size of an Agent/Object – added Debug.log() calls to record the size of migrating VHEAgents and returned state information. Nominal sizes appear to be approximately 4Kb for a VHEAgent, and 3Kb for returned state with one active service. It is envisaged that this will be useful in collecting metrics for evaluation of the system.

Bugs found and fixed:

- DatabaseAgent.synchroniseWith() method added yesterday was found to be throwing NullPointerExceptions' when removing entries from a database at the client-agent side. This was due to an oversight in the generalisation of the routine that meant that a null primary-key was being passed to the function that selected records to be removed from a database.

- Original implementation of DataContainer had its modified flag set as transient, so that modification state was not saved by Database instances. When testing the VHEAgents' ability to resynchronise its internal data-store with the information from a remote instance of itself, this became a problem since during serialisation of the agent state the modifiers were being reset (and hence it was impossible to tell which UserServiceConfigs' had been updated). This was overcome by ensuring that the modifier flag is not transient, and that the Database.synchroniseWith() method sets the modification flag of items inserted into a database are reset to non-modified.

- Removed the use Thread.yield() from the SMA - whilst allowing other applications a chance to execute using this method, the CPU was still in use 100% of the time (as viewed from the "Task Manager"). Changed over to the use of Thread.sleep(), which provided a better alternative – even with the thread sleep for 0.5s, the CPU utilisation dropped to 3%.

- Updated the PropertyDialog GUI so that it no longer displays certain values contained within a DataContainer (i.e. service-id and datacontainer UID).

- Updated the SMAUserConfigDialog so that modification flags are set correctly (previously ALL UserProfiles' were being reported as being modified). Also removed a small glitch from the GUI which meant that after a user is removed, their DeviceProfiles' can sometimes be left visible.

The basic VHE system is now up and running, so my focus will now switch to extending the functionality of the system by introducing new TerminalAgents and Services, and tidying up the user-interfaces. The only missing functionality appears to be:

- Ability for user to change their DeviceProfile (this requires a little thought as to how to overcome contention issues between Sys. Admin. changes and user changes).

- Ability for SMA to deploy TerminalAgents within the system (this can be introduced once a TerminalAgent that is designed to be deployed within the platform is produced, and is a trivial matter given the existence of code for deploying ServiceAgents and VHEAgents)

- Utility code to convert between different content formats (although it really comes down to the Service implementation to take care of this, it would be nice to have some prebuilt libraries for Service developers to make use of)

This missing functionality will be introduced as required by the extensions as they are developed.

1/6/2000 – 8.0 hours

Successfully tested the VHEMAT system across a Windows 2000 box and Linux (RedHat 6.2) box, with no problems. In the process set-up several start-up scripts, and produced a “distribution” directory structure for the system so that it can easily be deployed.

General bug hunting and error handling improvements made:

- Found glitches in several GUI’s:

  - SwingUITextArea components were not correctly populating the UITextArea objects they were wrapping correctly (last character from input was always missing). This was fixed by altering the was they are updated from being based upon events from its Swing components, to just collecting the value contained within them when the GUI is closed

  - Updates to DeviceProfiles within the SMAUserConfigDialog were not causing the UserProfile database and VHEAgents to be updated with this new information. This was caused by the fact that although the modification flag on the DeviceProfiles was being set correctly, the UserProfile encapsulating them didn’t.

  - Updated the SwingUserInterface class to ensure that it is always a minimum size of 300 pixels wide, and hence reduce the likeliness of a “cramped” GUI being produced.

- Updated the TerminalAgent shell to better handle error conditions, and more importantly differentiate between them – this was achieved by introducing a TerminalAgentException class, and several sub-classes relating to possible errors that could occur during TerminalAgent/VHEAgent interactions.

- Updated the SwingTerminalAgent to deal with these new exceptions, and provide a more verbose set of error dialogs including a limited ability to suggest ways of reminding errors (e.g. “Would you like to check the configuration settings?”).

3/6/2000 – 11.0 hours

Started producing the HTTPTerminalAgent shell today. It extends the TerminalAgent shell to provide the necessary functionality to interact with an Agent using the HTTP protocol. This class merely provides a mechanism for dealing with incoming HTTP requests, and sending back appropriate HTTP responses. The aim is that concrete sub-classes can deal with the HTTP requests by interacting with the methods provided by the TerminalAgent shell, for example by returning a WML/HTML page with the current UI presented by a VHEAgent, or with an error message.

Having tested the HTTPTerminalAgent with IE5.01, I proceeded to start work on a WAPTerminalAgent, which serves WML pages through which a user can interact with the VHEMAT system and their VHEAgent.

Problems encountered:

- Spent hours attempting to fix the HTTPTerminalAgent shell to work with a number of WAP emulators, since IE5.01 use of HTTP did not completely stretch the abilities of the HTTP protocol implemented within it, and due to other subtle differences (i.e. IE5.01 sends a CRLF sequence after data is posted in a HTTP request, whereas other clients do not).

- Spent hours attempting to get the WAPTerminalAgent to produce output that could be displayed by a WAP emulator/phone (Mitsubishi Trium Geo), however due to the rather unhelpful error messages produced (such as "Unknown URL" and "Page contains errors") or complete lock-up of some software, this proved very difficult. Fortunately, I remembered that the NokiaWAP Toolkit v1.2 had an integrated editor, and would let you see exactly what the problem was with the WML if it was unable to parse it. Within half-hour the few "minor" bugs in the WML had been fixed, and the WAP phone was quite happy to interact with a VHEAgent via the WAPTerminalAgent.

I also attempted to set up the system to run remotely on the sync machines in DoC, with no success. Aside from the fact that the scripts produced Thursday for use on Linux machines were intended for use with Bash, and the DoC machines are configured to use Csh, a NoClassDefFoundException is thrown whenever the SMA is started up. The problem would not appear to be the result of an incorrect class path, since other Agents can be started correctly. I suspect that some exception is being thrown during the loading of the class, resulting in this exception (e.g. if an exception is thrown during a static() initialiser, this exception is thrown despite the fact that the class file has been located).

4/6/2000 – 10.0 hours

- Ironed out numerous bugs in the WAPTerminalAgent, and discovered some limitations which have had an impact on the design of the generalised UI classes:

    - Discovered that WAP devices are not guaranteed to display text input fields which contain return codes – all text to be edited must have '\n' and '\r' characters stripped. Decided the only solution to this problem was to assume that input from a UI would not guarantee return codes would be retained from the original value. Also decided to add a "multi-line" hint to the UITextArea class to indicate if the input area should be multi-line or single-line to aid UI building.

    - Decided that password text entry fields should mask input, so added a "hidden" hint to the UITextArea class to indicate if its contents should be readable.

    - Had problems ensuring that the correct variable values within WML pages were being used, since cached variables were over-riding the values that should have appeared in some pages – this was solved by forcing a new browser "context" be created for each page displayed, which also has the effect of clearing the browser history and cache (this prevents users from moving backwards through pages, which is actually a good thing with regards to tracking what the user is currently looking at).

- Worked towards producing a mechanism to allow HTTP redirection within the HTTPTerminalAgent, without much success.

- Decided that the TerminalAgent shell can actually be considered a type of ServiceAgent (since other Agents within the system may wish to interact with them – e.g. by requesting the URL that a HTTPTerminalAgent is accessible from), so TerminalAgentImpl now extends ServiceAgentImpl. This allows TA deployment to be controlled by its ServiceProfile within the SMA, and allows a SystemServiceConfig to be specified for each type of TA.

- Improved "force VHEA" feature of the TerminalAgent shell to force a VHEAgent from an interrupted session to disconnect.

- Improved the structure of the VHEAgent UI, grouped ServiceInbox, ServiceTasklist and possible tasks for each service into one menu per service.

- Extended the ServiceAgent shell to include a number of extra methods that concrete sub-classes are required to implement (i.e. onStartup() – invoked when the ServiceAgent is first started up; and updatedConfig() – invoked whenever a new SystemServiceConfig is received).

- Updated the SMA to use separate threads when sending Agent shutdown messages, so that multiple requests are sent in parallel rather than series (hence improving the efficiency of the system shutdown routine).

- Added a roundTripDelay() method to the Metrics class to enable calculation of round trip delays to a host. Unfortunately, as ICMP packets are required to implement a proper ping packet, and Java doesn't support this type of packet. Having browsed a number of web-sites, one solution appeared to be that timing the creating of a Socket object (which creates a Socket to a remote host) achieves a similar measure – in either case of a successful socket being opened, or a failure, the response time will give an indication of round trip delay.

Week 12 (W/C 5/6/2000) – 57.0 hours

5/6/2000 – 9.0 hours

- Discovered why the SMA was refusing to start on the DoC lab machines remotely. The NoClassDefFoundException being thrown was indeed as a result of an exception being thrown at class-load time. A number of static variables related to the GUI were being initialised, and the initialisation was throwing an exception within non-Xwindows based sessions. Moved the initialisations to the classes static() method, and caught the exceptions – the SMA can now be launched remotely without a GUI.

- Got the HTTP-redirect to work correctly. Unfortunately, I had again been using IE5.01 for testing purposes – redirect messages are not handled as per the HTTP specification by this browser. A number of WAP gateways, the Nokia WAP toolkit and Netscape Navigator all responded to the redirect as expected.

- Updated the HTTPRequest object to include information about the host that an incoming request is coming from. In conjunction with the new getNearestTA() (which will locate a HTTP-TA of the same type which has a shorter round-trip-delay to the given host), the primary HTTP-TA for a given content type (i.e. WAP) can redirect a user to a closer TA in order to decrease network traffic (between User & TA/VHEA) and user-interface response time (between TA & User-browser).

- Spent a length of time testing the platform remotely and locally. Unfortunately GH doesn't appear to link working across a modem link (the RegionRegistry has trouble keeping track of remote Agencies due to the bandwidth limitation!).

- Updated the SMA so that SystemServiceConfigs are now only distributed to ServiceAgents if they are modified.

- Added asynchronous shutdown routine to the DatabaseAgent so that Database instances are closed in parallel rather that series, reducing the amount of time taken to shutdown the DatabaseAgent.

6/6/2000 – 7.0 hours

- Started working on the EmailAgent to provide access to Internet email services. Decided that the simplest was to implement this was to use the JavaMail extension with the POP3 provider extension to allow interactions with POP3 and SMTP servers. So far the message ID's of messages in a users inbox can be fetched, and messages can be sent using the methods that the ServiceAgent implementation exposes via the executeTask() mechanism. In the process of producing a method to retrieve emails, which will convert attachments to specific Java types to simplify the processing required by the VHEAgent-side Service implementation.

- Updated the ServiceInteraction interface (implemented by the VHEAgent to provide inter-service interaction abilities to Service instances) to include methods to allow Services to indicate to the VHEAgent that the UserServiceConfig/ServiceInbox/ServiceTasklist have been updated, and should be persistently "backed-up" (i.e. with the DatabaseAgent).

- Introduced the vhemat.util.Reflect class to provide a simple mechanism for dynamically invoking methods on objects. At the moment it supports invoking static methods and non-static methods only (its use will be expanded to instantiating objects at a future point). In the event of a reflection related exception occurring, Reflect.ReflectException is thrown by the methods it provides (in order to reduce the number of exceptions needed to be caught whilst using reflection – to invoke a method generally up to 7 exception types can be thrown), any exceptions thrown by the invoked method are allowed to propagate back freely. The aim is that he use of this class provides a mechanism for i) de-coupling classes at class-load time (to prevent unnecessary class loading) and ii) allowing dynamic access to methods of concrete classes which aren't known at compile time.

- Updated the SMAFrame to use the Reflect class to dynamically invoke the methods getDefaultServiceProfile() and getDefaultSystemServiceConfig() on the selected services' service agent class-name and service class-name when the "Restore Default Config." button is pressed. Added these methods to all current ServiceAgent and Service implementations.

7/6/2000 – 10.0 hours

- Tested the EmailAgent, and corrected a number of bugs with regard to the ServiceItems' it was producing.

- Produced the EmailService class, which periodically scans for new emails in the users POP3 account, and sends pending emails to the default SMTP host for the service. Also provides UI for the service, and can interact with the Contacts service to retrieve peoples email addresses. Can filter image attachments based upon capabilities of device that the VHEAgent is migrating to.

- Updated the SwingUIAttachment class to deal with images to be presented to the user, and added a hint to the UIAttachment class as to whether an attachment should be "inline" or not.

- Updated the VHEAgent with regard to invocation of update methods through the ServiceInteraction interface. These are now more efficient since they now only update the specific DB required, rather than all user information DB's.

8/6/2000 – 6.0 hours

- Added a couple of methods to the Service interface to allow specific names to be assigned to a Services' Inbox/Tasklist, and even indicate that they are not used by the Service, after comments that the generic names given to the inbox/tasklist were misleading, and sometime unnecessary (due to lack of tasklist for the contacts service for example).

- Fixed intermittent bug in SwingTA (which wasn't so intermittent during the demonstration to my supervisor ☹), which meant that on some occasions UIButtons were not being set to the activated state when their GUI wrappers (i.e. SwingUIButton's) had been clicked. This caused the VHEAgent to return a null value, indicating to the SwingTA that the VHEAgent had disconnected.

- Fixed a bug in the WAPTA that appeared when replying to an email, which caused the WAPTA to send invalid WML. This was caused because the reply included a message which had a quote mark contained within it – as a result the WML was unintelligible. This was resolved by introducing a mechanism to encode all characters within text in WML sent to a WAP device outside of the ranges 'a' –'z', 'A' – 'Z' and '0' – '1' into the form &#xx; (where xx is the Unicode number of the character) – the text then parsed correctly.

- Updated the email service to use MIME types internally – all "image/*", "audio/*" and "application/*" mime-types are processed by the EmailAgent and EmailService. The SwingTA can now deal with some of these content-types within UIAttachments', content which it cannot deal with it just gives the option of saving to disk.

9/6/2000 – 3.0 hours

- Started investigating mechanisms to allow content adaptation for Services. Started implementing a ImageAdaptor class to allow dynamic invocation of the Java Imaging IO extensions to allow this.

10/6/2000 – 14.0 hours

- Continued working on the ImageAdaptor class. After writing most of the implementation to use the Java Imaging IO extension, I discovered that the API is only half implemented, none of the encoders for the image formats have been written, rendering the API virtually useless for the purpose it is required for. Decided that the Java Advanced Imaging API would be a suitable substitute, and was able to reproduce the same functionality as for the Java IIO extension. Completed and tested satisfactorily, although the only viable output format for translations is JPEG, since the other formats supported (BMP, PNG, TIFF and others) don't have encoders' that support compression.

- Created DataAdaptor to allow content compression for generic data (i.e. by using ZLIB compression).

- Updated the EmailService class to make use of the ImageAdaptor and DataAdaptor classes to adapt content to target devices capabilities.

- Found a rather nasty bug in GH. It appears that objects passed to Agents during inter-Agent communications are not serialised when the target Agent of a message is local to the sender. This results in any objects being passed during RPC to be referenced by both Agents, rather than each having their own separate copy. Obviously this leads to major problems when either Agent modifies the contents of these objects, since the state of the objects (and hence both the Agents) change. This leads to completely different semantics for remote Agent interaction compared with local Agent interactions! This was discovered after several hours attempting to discover why the contents of a VHEAgent's inbox kept changing from what they should have been whenever the system is started up. It appears that after passing objects to the DatabaseAgent to persistently store, the DatabaseAgents' local cache of Database objects remain in memory (by design), and hence if any objects have been placed into a Database by a local Agent, changes to these objects by the Agent inadvertently modify the contents of the Database without any interactions directly with the DatabaseAgent. Thus, when the content of the inbox is modified prior to migration, these temporary changes were actually being stored in the VHEAgents' inbox database and causing this bug.

  The solution is to ensure that any method invocations on Agents have had their arguments serialised. The simplest solution to this was to introduce a mechanism to automatically serialise then de-serialise arguments and results for the executeTask() method exposed by ServiceAgents, which effectively decouples the Agents. In other Agent interactions (of which there aren't actually that many), this will have to be done manually.

11/6/2000 – 8.0 hours

- In order to ensure that no Agent interactions are affected by the bug found in GH yesterday, updated all methods that Agents expose externally (i.e. to other Agents) to ensure that non-trivial parameters and returned values to use the new Metrics.decouple() method (which decouples Objects passed to/returned from Agents during Agent interactions by serialising and then deserialising Objects).

- Updated the VHEAgent and SMA to allow users to change the DeviceProfiles' of their devices from their VHEAgents' UI. Discovered that the SMADeviceProfileDialog was setting the processor speed the wrong way round, due to incorrectly named UI components (i.e. they were named the wrong way around).

- Updated the SMAServiceConfigDialog and SMAUserConfigDialog classes so that device, user and service ID's cannot be modified. This prevents any possible ambiguity arising when id's are changed.

- Updated the WAP-TA to place a maximum size limit on editable UITextAreas' of 200 characters, since during testing large editable areas caused real WAP devices to crash.

Appendix F: Bibliography

F.1. Papers

[1] Hagen, Breugst and Magedanz: Impacts of Mobile Agent Technology on Mobile Communication System Evolution, *IEEE Personal Communications Magazine*, August 1998 (p.56 – 69)

[2] C. Schmandt: Multimedia Nomadic Services on Today's Hardware, *IEEE Networks Magazine*, September/October 1994 (p.12 – 21)

[3] Alanko, Kojo, Liljeberg, Raatikainen: Mowgli: Improvements for Internet Applications Using Slow Wireless Links, *8th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 1997 (p. 1038 – 1042)

[4] Roman, Singhai, Carvalho, Hess and Campbell: Integrating PDA's into Distributed Systems: 2K and PalmORB, *HUC'99 Proceedings*, September 1999.

[5] Fox, Gribble, Chawathe, Brewer: Adapting to Network and Client Variation Using Infrastructure Proxies: Lessons and Perspectives, *IEEE Personal Communications Magazine*, August 1998 (p.10 – 19)

[6] Noble, Satyanarayanan, Narayanan, Tilton, Flinn and Walker: Agile Application-Aware Adaptation for Mobility, *Proceedings of the 16th ACM Symposium on Operating System Principles*, 1997 (p.276-287) (SIGOP)

[7] Girish Welling and B.R. Badrinath: An Architecture for Exporting Environment Awareness to Mobile Computing Applications, *IEEE Transactions on Software Engineering Vol.24, No.5*, May 1998 (p.391 – 400)

[8] Roy M. Turner: Context-Mediated Behaviour for Intelligent Agents, *International Journal of Human-Computer Studies Vol. 48*, 1998 (p.307 – 330)

[9] Jansen, Karygiannis: Mobile Agent Security, *NIST Special Publication 800-19*

[10] Karnik, Tripathi: A Security Architecture for Mobile Agents in Ajanta, *International Conference on Distributed Computing Systems 2000*

[11] Karnik, Tripathi, Vora, Ahmed, Singh: Mobile Agent Programming in Ajanta, *International Conference on Distributed Computing Systems 1999*

[12] Karnik, Tripathi: Delegation of Privileges to Mobile Agents in Ajanta, *1st International Conference on Internet Computing*

[13] CAMELEON Concept Description
(Deliverable D06. CEC Deliverable ref. AC341/VOD/WP1/DS/P/9904/b1)
http://www.comnets.rwth-aachen.de/~cameleon/cameleon/d06_final1.zip

[14] CAMELEON Performance Assessment Results
(Deliverable D09. CEC Deliverable ref. AC341/CN/WP1/DS/P/9909/b1)
http://www.comnets.rwth-aachen.de/~cameleon/cameleon/D09_final.zip

[15] WAP Forum: Wireless Application Protocol Architecture Specification, version 30/4/1998
http://www.wapforum.org/what/technical.htm

[16] WAP Forum: Wireless Application Protocol Wireless Markup Language Specification Version 1.2 (4/11/1999)
http://www.wapforum.org/what/technical.htm

[17] Java Virtual Machine Specification
http://web2.java.sun.com/docs/books/vmspec/html/VMSpecTOC.doc.html

[18] RFC 1945: Hypertext Transfer Protocol - HTTP/1.0
http://www.faqs.org/rfcs/rfc1945.html

[19] RFC 2068: Hypertext Transfer Protocol - HTTP/1.1
http://www.faqs.org/rfcs/rfc2068.html

[20] RFC 2045: MIME Part One: Format of Internet Message Bodies
http://www.faqs.org/rfcs/rfc2045.html

[21] RFC 2046: MIME Part Two: Media Types
http://www.faqs.org/rfcs/rfc2046.html

F.2. Websites

[22] CAMELEON Project
http://www.comnets.rwth-aachen.de/project/cameleon/cameleon.html

[23] Yahoo! Web-based Services
http://my.yahoo.com/

[24] FIPA-OS
http://www.nortelnetworks.com/fipa-os

[25] Grasshopper
http://www.ikv.de/products/grasshopper.html

[26] Voyager
http://www.objectspace.com/products/voyager/

[27] Aglets
http://www.trl.ibm.co.jp/aglets/

[28] iPulse
http://oz.com/ipulse/index.asp

[29] Java Servlet Extension Homepage
http://www.javasoft.com/products/servlet/index.html

[30] Java Advanced Imaging Extension
http://www.javasoft.com/products/java-media/jai/index.html

[31] Java Imaging I/O Extension (Early Access)
http://developer.java.sun.com/developer/earlyAccess/imageio/

[32] Java Mail Extension Homepage
http://www.javasoft.com/products/javamail/index.html