ESPRIT Project No. 25 338

Architecture internal 1.1 deliverable

Work package A

Architecture

Architecture Report DA1.1

ID:	Architecture Report DA1.1 V. 1.0	Date:	27.11.97
Author(s):	Mike Bursell Douglas Donaldson input from all partners	Status:	Internal Release
Reviewer(s):		Distribution:	Project confidential

Change History

Document Code	Change Description	Author	Date
Architecture Report DA1.1	Draft version of document for discussion.	Mike Bursell (APM)	23.Oct.97
	New format, completely new input	Mike Bursell, Douglas Donaldson (APM)	27.Nov.97

Introduction

Background

This document is a development of a draft document made available to the project partners as a basis for discussion towards the end of October. One of the issues that came up was how better to provide architectural description for the FollowMe project - the original document was not structured enough to form a useful deliverable. It is hoped that this version will be more useful, but it should be stressed that it is only a starting point, and is expected and designed to evolve.

Quite a large number of issues were raised in the discussions following the internal publication of the first iteration of this document, and these were resolved - or are being resolved - in a number of ways. These include:

- discussion on FAST' s threaded discussion board
- email between partners
- meetings between partners
- a general meeting at UWE, 17-19 November, 1997.

It is not expected that all the issues raised were solved over the month between the original publication of this document and this iteration, but issues have been separated into three main sets, to allow them to be better addressed:

1. solved or discounted
2. the stratum in which they exist was identified, and they are now the responsibility of the partner in charge of the associated work package
3. unresolved, no stratum yet identified, but can wait.

These last are listed within *Section Architectural backbone*. To the best of my knowledge, there are no outstanding, major issues which were noted at the UWE meeting but have not been identified with particular stratum or strata.

Description

As mentioned above, in the Section Background, this document has been substantially revised, partly in response to the initial submissions made by partners, and partly in response to discussion of the document itself. This section briefly describes the status of this document, and its structure. Although the structure of the document is not expected to change a great deal over the short time, it is to be hoped that as more work is undertaken within particular strata that the

partners responsible will provide more information, allowing strata (and in some cases, the architectural background) to be updated.

Current status

This document has input from all the strata, and an architectural backbone.

Structure

This document an *Introduction* section, an *Architectural backbone* section, and Strata sections. The Introduction section provides background and general information about the document. The Architectural backbone section provides certain information which is either central to the project - such as a brief description of the project aims - or shared - such as concepts which are the same across strata. The strata sections incorporate architectural input from specific to strata. These should conform to a template provided separately, though not all sections of the template are expected to be filled in at any one point.

It should be stressed that partners should not place a complete design for associated work packages within this document, which is intended to provide a useful summary of work packages for other partners and those outside the project. It is the job of the architectural background to try to maintain conceptual consistency across these strata.

In summary, the document has a structure thus:

- Introduction
    - Inception
    - Description
    - Evolution
- Architectural backbone
- Strata
    - *for each stratum -*
        - *Status*
        - *Scope*
        - *Issues*
        - *OMT Models*
        - *Components*
        - *Function*
        - *API*
            - *Comments on API*
        - *Requirements on other strata*
        - *Other comments*
    - MOW - Mobile Object Workbench
    - IS - (Personal) Information Space
    - AA - Autonomous Agents
    - PP - Personal Profiles
    - SI - Service Interaction
    - SD - Service Deployment
    - UA - User Access
    - PA1 - Pilot Application 1
    - PA2 - Pilot Application 2

Evolution

The first release of this document is as an internal project deliverable for the end of November (month 2 of the project). Further releases (full project deliverables) are due in month 6 (March 1998) and month 12 (September 1998). However, it has become clear that it is not good enough to freeze this document between releases, and it is therefore intended that it should evolve over time, and as more work is done both within strata and on wider architectural issues.

This first release is intended to provide a base set of concepts for the architecture backbone, and to allow partners to define the scope and function of work packages as represented by strata. In some cases, it has been possible to provide more data within the strata section.

Whereas the strata sections are expected to be updated on a fairly frequent basis, it is likely that the architectural background will be updated less frequently. This is for a variety of reasons, including:

- there is less effort available for architectural updates than available within work packages themselves
- work on shared concepts should not be rushed, but rely on discussion between strata
- it is hoped that following discussion of the original draft of this document, strata are more ‘ self-contained’ than before, and that a large number of dependencies have now been identified
- major changes can only be expected to arise as differences, dependencies and requirements become visible between strata, so more than one strata is likely to change significantly before major changes are reflected within the architectural background.

Architectural backbone

Strata Overview and Rationale

This document is split into a number of sections, described hereafter as "strata", as they represent different levels of the architecture of the project. These are ordered roughly according to their level (lowest first) - for instance, the Mobile Object Workbench stratum addresses low-level engineering issues on top of which other strata are to be built. The strata have been identified with the technical / engineering work packages of the FollowMe project - work packages B, C, D, E, F, G, H, I and J, and bear the same names as these work packages. Note that work packages I and J (the two pilot applications) do not represent work which is integral to the architecture of the system - they are, in fact, applications which sit on top of the system, leveraging the possibilities and functionalities it provides. However, they are extremely important within the architecture document, providing as they do a set of "real-world" needs and requirements that might otherwise go unrecognised within the architectural vision.

Some thought was given to separating the work packages from the strata, but it was decided not to do so for two main reasons:

- the work packages represent distinct logical blocks of work and serve to identify likely possible dependencies (see the FollowMe Technical Annex, Part II, Section 3.3 "Deliverables and Dependencies")
- the work packages are split between different partners, and the communications overhead in trying to split strata differently would be counter-productive.

The list below gives the names of the strata, the abbreviations used throughout the document, and a brief description of the scope of each.

MOW ***Mobile Object Workbench***
This work package will deliver a workbench upon which the other strata will be implemented. It is intended to provide facilities for object movement to allow the creation of a variety of high level services, but does not include the creation of these services.

PIS ***Personal Information Space***
The information space provides a facility for maintaining, accessing and collating information. It provides mobile users with a single and consistent (i.e. logical) view of an information space irrespective of where they are located. Both users and agents may potentially access an information space - though it may be that only Personal Assistant agents have access to a user's personal information space.

AA ***Autonomous Agents***
This work package will provide an agent-based view of the underlying workbench. *Editor's Comment: should it not provide an agent-oriented model of computation to agent programmers, which happens to be engineered with the workbench?* It will provide a framework to allow the selection, dispatch and subsequent return of agents acting in order to complete tasks.

PP ***Personal Profiles***
The aim of work-package E is to determine a representation for the storage of personal information in support of other FollowMe components. This profile is a human-readable document from which one or more profile objects

may be constructed. The profile object will present an interface by which the content of the profile may be obtained.

SI ***Service Interaction***

The aim of this work-package is to define a pattern for making services available to FollowMe agents. This will develop a service description language and tools which use these descriptions for service location. A major aim is extensibility; the ability to add new services without rebuilding the code. Consequently, these descriptions are characterised as service profiles using similar techniques to those in the Personal Profiles work-package.

SD ***Service Deployment***

This work package will deliver two features:

- The first one is a set of low-level tools which provide applications with performance measurements and predictions.
- The second one is an example of a service deployer dedicated to Etel++. The main goal of this deployer is to provide Etel++ with load-balancing features based on performance measurements and the exploitation of profile.

UA ***User Access***

The User Access will provide the following functionality:

- Access to Java enabled devices for output and input
- Access to fax and telephones (not Java enabled) for output and input
- Error feedback to the agent who required the delivery, e.g. "device not responding"
- Feedback to the agent responding to delivered information, e.g. providing a way of communication between the agent who required the output of information and the user.

PA1 ***Pilot Application 1***

The scope of the pilots is to implement applications with the following core features:

- The applications' components can be classified as: information providers (offering access to data-objects), service providers (providing services operating on data available from the information providers), information consumers (engaging agents to make use of available services) and brokers (mediating between the other components). By separating these components the system will easily scale and adapt to new requirements (add new information sources, services or users).
- The applications enable automated task execution while the user is offline. Information retrieved and processed (i.e. filtered) by agents is stored in a location close to the user (i.e. in the user's LAN or at the ISP of the user).
- The applications allow the user to be mobile. This means that the user will be able to access the applications from different devices (i.e. Java-enabled devices, fax-machines, mobile phones) and from any geographical location.
- Information related to the user will be stored in a user trusted environment (in personal profiles).

PA2 ***Pilot Application 2***

The purpose of ETEL++ is to show the benefits of having mobility to users and data. Its goal is to unveil low-level features (mainly MOW and Service Deployment) via the manipulations of an electronic newspaper across a network.

Definitions and Object Diagrams of Concepts and Architectural Components

The foundational object modelling concepts of the FollowMe Architecture conform to those of RM-ODP. The ODP object concepts are introduced and extended with FollowMe abstractions such as agent and information space. OMT notation is prescribed for modelling diagrams, for consistency.

Object, Service and Interface

The ODP object model has a general concept of an object which can be of arbitrary granularity, can exhibit arbitrary (encapsulated) behaviours, and interactions between objects are not constrained.

The [OMT] notation diagram (Figure 1) shows some fundamental concepts and relations.

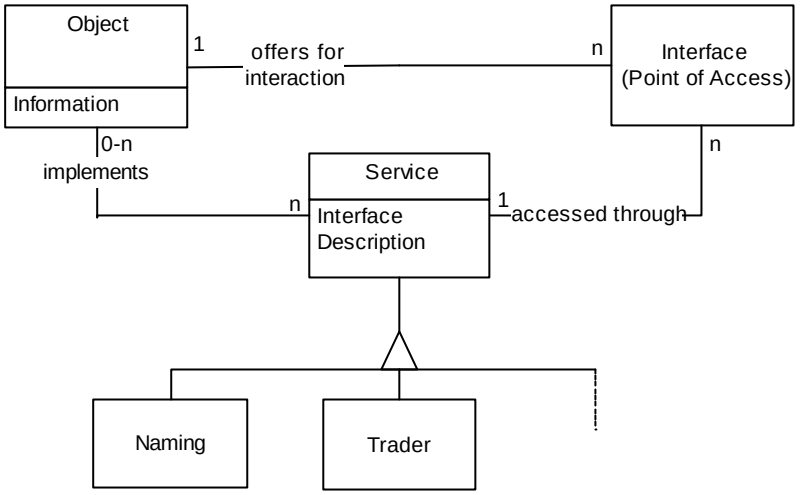


Figure 1 Object, Service and Interface concepts

Services have a description (e.g. IDL) which abstracts from the implementation mechanism. Objects are entities which implement one or more services. Objects contain information and have defined behaviours. Objects have one or more interfaces which allow access to their services. Objects can only interact at interfaces.

Examples of services which might be implemented as part of the FollowMe project include Naming, Trading, Scheduling, Event Notification, Personal Information and Diary Services.

Object Composition and Decomposition

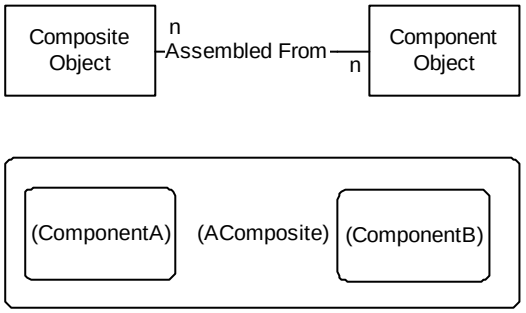


Figure 2 Object Composition and Decomposition

In order to aid description of a system at different levels of abstraction, composition and decomposition concepts are useful to describe a composition hierarchy. A higher level composite object is assembled from a configuration of lower level component objects. Composition not only allows modelling of whole-part (aggregation) relationships, but also allows the implementation of high level service (c.f. a subsystem) to be considered as a high level object. The object diagram of Figure 2 shows instances of classes ComponentA and ComponentB composed within an instance of Acomposite.

The idea of an object group is often a useful modelling concept. Grouping is an example of composition/decomposition. Examples of object groups include replication groups for availability and groups of objects in a domain. (Examples of domains include security domains, management domains and naming domains).

Located Objects

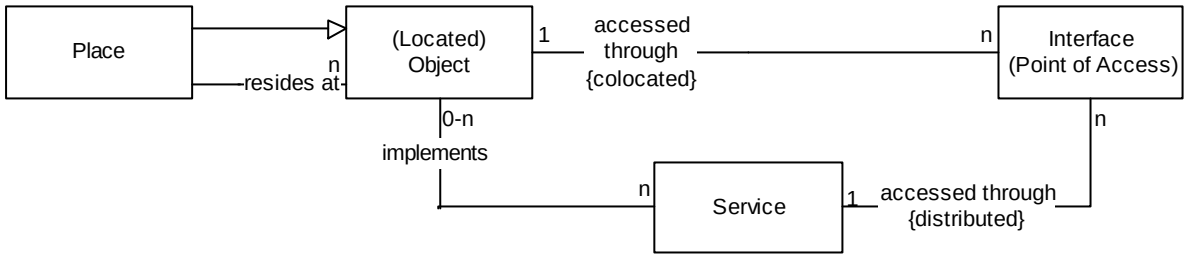


Figure 3 Located Objects

As described earlier, the object model is general, and the discussion of objects and services does not address where objects or interfaces exist. There is no assumption that objects exists at a particular machine. This is adequate at a high level or system description. At an engineering level of description, the object concept is still used but with assumptions about engineering mechanisms.

A located object is understood as the sort of entity described by conventional object programming languages. It resides (is instantiated) at a place (an execution environment), which in turn runs on a (single) machine (Figure 3). Its interfaces are addressable points of access local to the object (cf InterfaceRefs, ServiceBindings, Skeletons).

A located object may be one of a number of points of access to a distributed service; in which case the client should remain unaware of the location of the service access point, being only interested in the widely available service it is using.

Example of Decomposition into Located Objects

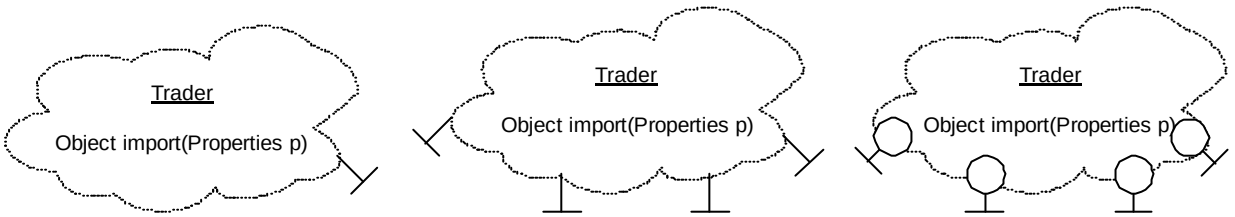


Figure 4 A High level Trader object decomposed into a set of located objects offering the Trader service

At a high level of system description, a service may be given an implementation which is a description of a (high level) object, e.g the Trader object implementing a trading service. The trading service is considered highly available, not available only at a specific place. At a lower, engineering level, the implementation of the same service may be described using a number of located objects, possibly distributed over many machines. The service may then be described as distributed, or as having a number of distributed interfaces. The objects behind a service's distributed interfaces presumably collaborate to share information. This is an example of a decomposition of the high level composite object into lower level engineering components (Figure 4).

Mobile Objects and Agents

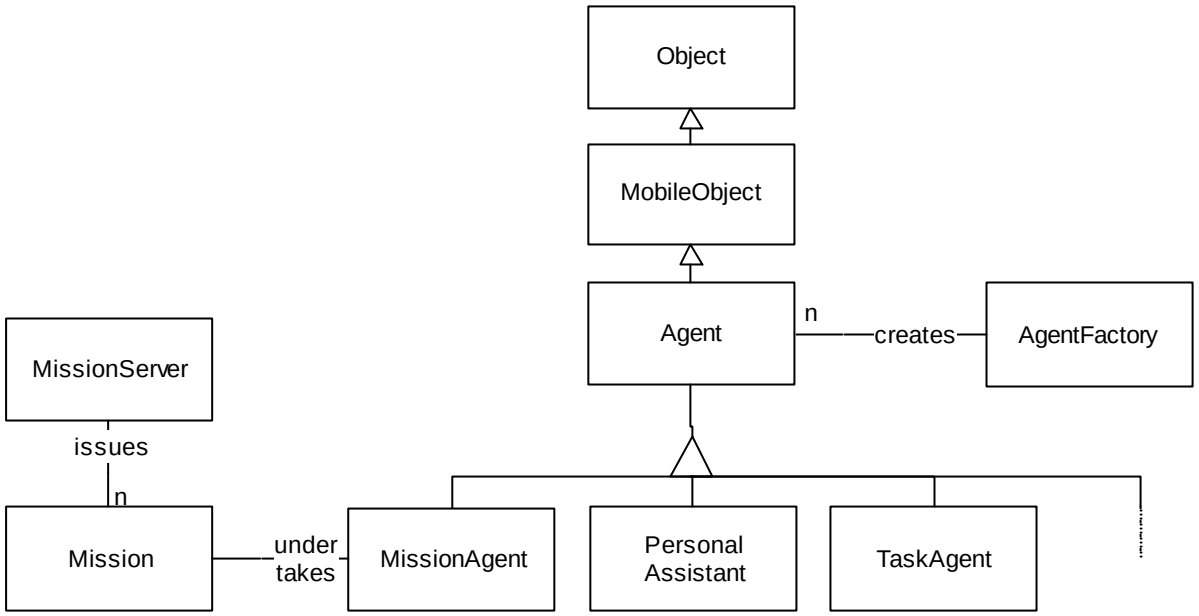


Figure 5 Mobile Objects and Agents

Mobile Object

A mobile object is a located object which can move between places subject to the security policies employed by the place and by the object (Figure 5).

Agent

An agent is a mobile object with goals (ie objectives to accomplish). Agents may be created by agent factories, which may maintain a library of agents suitable for specific jobs, or may be specialised to create specific types of agent.

Agents are not fundamentally classified as active or passive. Activity is regarded as a temporal property of all objects according to whether they have a thread busy or blocked within them at a moment of time. If some distinguished method is used to resurrect an agent after a move, and that thread continues with activity or blocks, that may allow the agent to be classified as autonomous.

Specialised Agents

Examples of specialised agents include a mission agent, a generic agent whose goals are described using a mission (e.g. a script issued by a mission server); a personal assistant, which represents a user directly within a FollowMe system; and a task agent which is designed to execute tasks in a particular domain.

Composite Mobile Objects and Agents

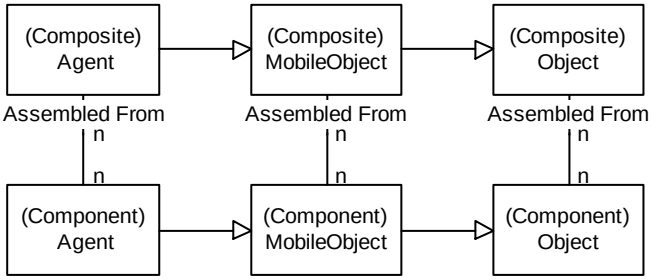


Figure 6 Composite Mobile Objects and Agents

The concept of composition/decomposition also applies to Mobile Objects and Agents (Figure 6). Particular care is needed, however, to ensure that mixing of levels of abstraction in descriptions does not cause confusion. At a high level of abstraction, it may not make sense to describe a mobile object as existing at a Place. For example when abstracting over time, it may not make sense to describe a mobile object as existing at a single place, although at any instant it clearly will be at some place or other.

Places, Hosts and Mobility

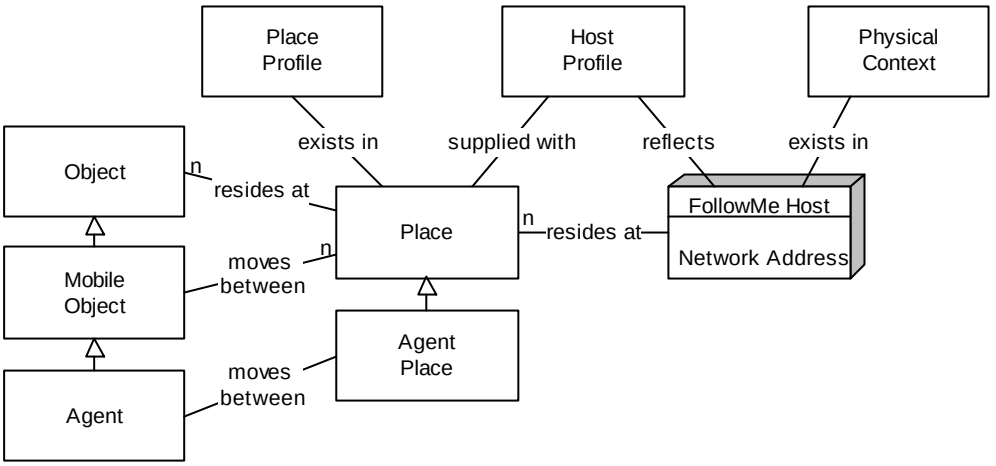


Figure 7 Places and Hosts

Place

A Place is a logical execution environment for objects. Located objects reside at one place. A mobile object can change its residence from one place to another (Figure 7).

A place resides at a host for its lifetime. It is useful to consider a place to be a distinguished object created within a host process (e.g. a virtual machine process which is part of a FollowMe system). The lifetime of the place would then be the lifetime of the process. Places do not move between hosts; however hosts may be mobile, and a place on a host which moves may have support for dynamic change of the host' s address; such a place may be referred to as a mobile place.

Place Profile

Places exist in an administrative context, which is the totality of the contextual information available to the place, for example the naming context made available through a name service, and the service context made available through a trading service. The place profile may be made available to objects at a place, allowing the object to discover information about its context.

Agent Place

An Agent Place is a Place which offers extended services to agents, for example information about other agents at that place, or other information required for an agent to achieve its goal which the most simple place doesn' t provide.

Host Profile

A Host Profile is made available to a Place, and reflects the host capabilities such as bandwidth, CPU resources and reliability. A Host Profile may be used to make decisions concerning movement, communication, processing etc. The terms Dock, Minihost, User End Device or User Terminal may be used to loosely refer to hosts or devices whose profiles reflect that the machines have greater or less reliability or capability. They are not architectural abstractions, however.

Host

A Host is a hardware machine, a physical execution environment. FollowMe Hosts are capable of running Places.

Hosts have a network address, and exist in a physical context (geographically, and on a network). A host may be dis-connected from and reconnected to the network. A mobile host may be disconnected from the network and reconnected

at a different network address (Figure 8). A host is not FollowMe capable if it does not have the (hardware or software) ability to ‘ run’ a Place.

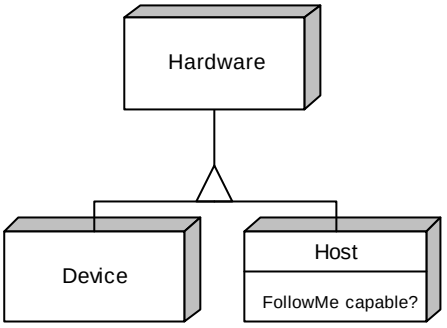


Figure 8 Varieties of Hardware

Device

The term Device is used for hardware which doesn’ t have the hardware capability for ‘ running’ a Place but may be used for user access (I/O).

Profiles and Profile Objects

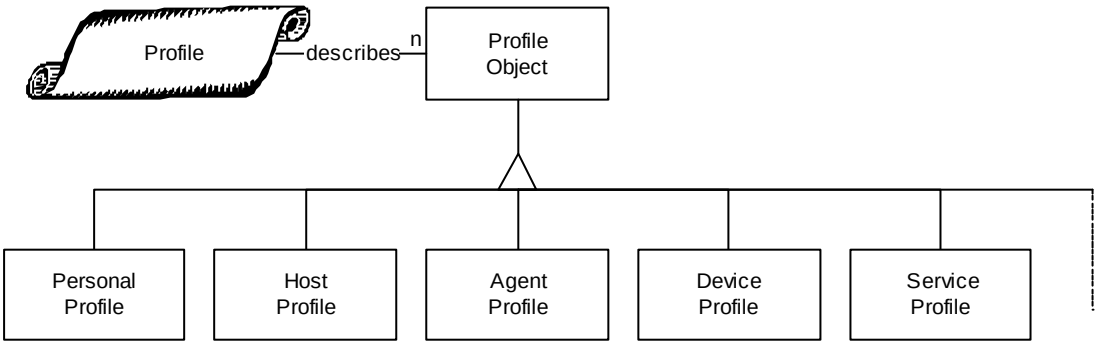


Figure 9 Profile and Profile Objects

The term ‘ profile’ is frequently used within FollowMe workpackages. A profile is a description of meta-content, a generic, structured description (Figure 9). The generic description might be captured within a Profile Object. Architectural concepts which may be described with profiles include Personal Profile (describing a user’ s preferences, personal data and including a Diary), Host Profile (describing a host’ s capabilities), Agent Profile (describing an agent’ s goals, capabilities, functions and delegations), Device Profile (describing a device’ s characteristics such as bandwidth and display capabilities) and Service Profile (describing a service’ s properties, perhaps for use in trading).

(Personal) Information Space

The phrase ‘ Personal Information Space’ has been renamed ‘ Information Space’ to reflect the requirement for provision of distributed information services, offering a containment structure for information. The term Personal Information Space may be used to refer to a user’ s personal information, but this is just one example of an information space. Other Information Spaces may hold shared information, or application information.

Example Architectural Configuration

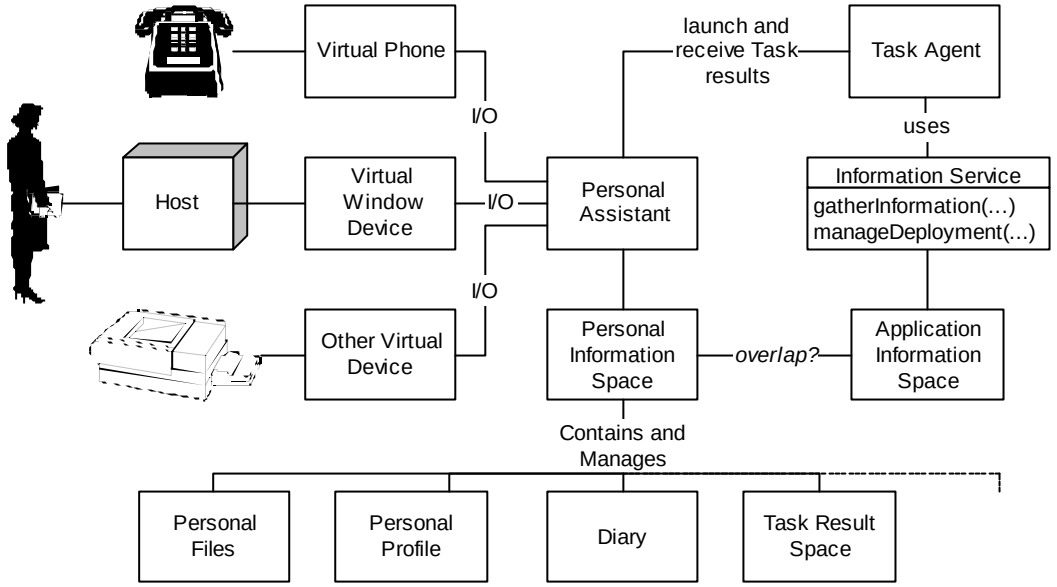


Figure 10 Example Architectural Configuration

The FollowMe architecture does not prescribe one ‘system’ configuration, rather the architectural components are intended to provide useful general abstractions which may be configured in a flexible way to provide users with services supporting mobility, or to implement domain specific applications.

Figure 10 shows an example configuration mixing objects from the user’s domain with an domain specific Information Service application.

The Personal Assistant (PA) is shown as being aware of virtual I/O devices which it can use to communicate with its boss (user). The user is ‘logged in’, providing the PA with an extra screen device. The PA is shown as the only object which can change the user’s Personal Information Space, presumably the PA provides interfaces for the user to change her mobile files, profile and diary, and for task agents to place information returned from information services.

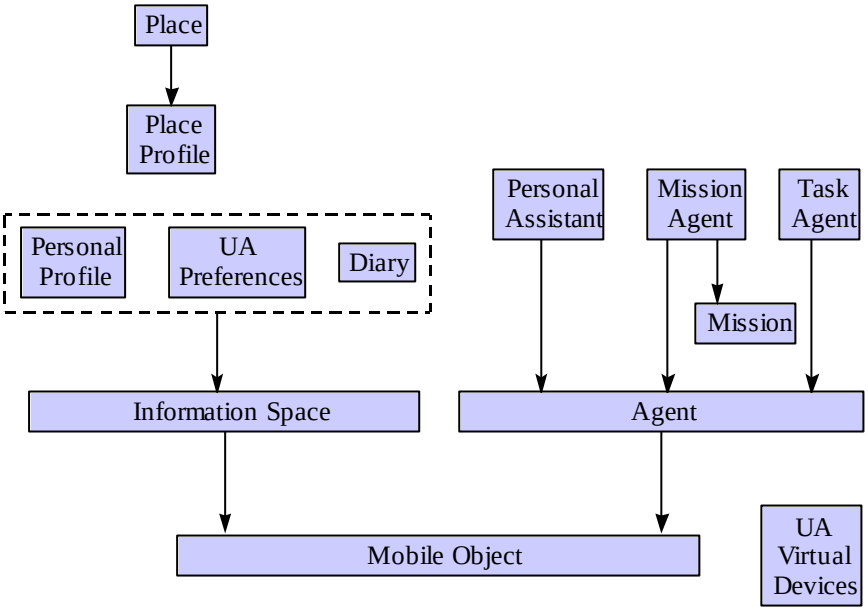
The information service is shown as offering interfaces both for information gathering and service deployment. The task agent may carry selected personal information such as preferences for specific information, which the information service uses to improve service response and availability.

The PIS and Application Information Space may overlap, if the administrative domains of personal objects and application objects overlap.

Component dependencies

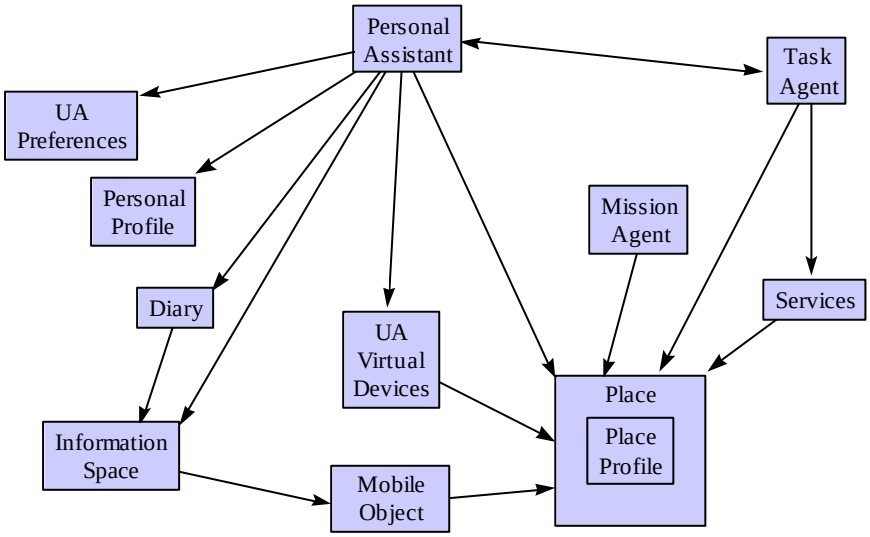
Implementation Dependencies

This diagram shows implementation dependencies between components.



Examples of implementation dependencies include inheritance (e.g. specialised agent to agent to mobile object) and containment (e.g mission agent to mission). The information objects in the dashed box are shown with a possible implementation dependency on some information space. The nature of the component-container relationship is to be defined (as for every component relationship shown), but it is noted that there is more than one possibility, for example components which notify the container of updates, or of self-redeployment, and components which are ignorant of their container.

Use Dependencies



The diagram above shows usage (talks to) dependencies between components. Mobile objects and subclasses interact with the place where they reside, and presumably with place profile via the place. Task agents use services. Services may interact with places so as to deploy themselves. Information Spaces may use mobile object facilities for information deployment. The Personal Assistant keeps its information objects in an information space, but, as described above, there is more than one possibility for component-container interaction (ie. information content to Information container interaction). To illustrate this, the Diary is shown as being aware of its information space, whereas the personal profile and user preferences are not.

Strata Relationships which need Clarified

The strata relationships listed here are taken from the dependency sections of the strata contributions, and are grouped by the stratum upon which the dependency is considered to be made. If there are other dependencies, these should be cited as soon as possible to allow them to be resolved..

This section should be seen as a 'holding tank' for issues raised by strata, and not as a long-term section. There are three courses of action for points raised in this section -

1) The dependencies are acknowledged component dependencies, which will be clarified as each workpackage refines the meaning and interfaces of its components. This is likely in quite a lot of cases, as the originators of these issues had not seen the descriptions in the strata. If this is the case, then work package leaders should expressly acknowledge that these issues are taken in by their work package. For instance, "facilities and API: need to be clearly defined" in the MOW is acknowledged by APM.

2) The dependencies are not clearly expressed and need better definition before they can be acknowledged (or discounted). In this case, the originator should discuss it with the work package leader.

3) The dependencies are remain misunderstood or discounted by the work package leader. In this case, they should move into the ' Open Issues' section for further discussion among the partners.

MOW

- **facilities and API**: need to be clearly defined. From AA (and others, presumably).
- **replica management**: possibly some low-level mechanism required? From (P)IS.
- **local directory/trader for each place**: allow mobile objects to find others in the same place. Agents should be able to find this From SI. *APM comment: The MOW will provide hooks to identify when a mobile object arrives/departs or is created/destroyed. I envisage that AA will subclass the MOW definition of place, and add code to make use of this information in order to keep a directory of colocated agents. This makes more sense than simply recording all mobile objects, as knowledge of a mobile object existence is useless unless there are some standard mechanisms that can be used to interrogate it. This sounds like an AA issue (if an issue at all). Does this agree with AA feelings?*
- **directory/trader needs MOW hooks to identify new places**: From SI. *APM comment: The MOW hooks will provide this as well.*
- **Places:** Places should be aware of firewalls. From UA. *APM comment: Question not understood.*
- **object owning**: who owns an object? Who decides backup/restore? From PA1. *APM comment: Owning: what does owning mean? Ownership is at a higher level of abstraction than the MOW object model. Backup/Restore: The MOW will provide hooks to allow a 3rd party backup service to integrate in a seamless way. I envisage that it is the objects responsibility to backup itself (using a 3rd party service) and the backup service together with MOW cooperation will restore the object's state and tell it that it has been restarted. It is up to the object to manage the side effects of restart (eg managing rollback, transactions etc).*
- **is normal Java observer/observable pattern available to trigger actions?**: from PA1. *APM comment: Yes, although I would recommend the Java 1.1 event model (which subsumes the functionality). If there are any implementation issues that arise at a later date, I would rather solve them once*
- **control of system resources**: who controls it? Hooks probably needed into JVM. From PA1. *APM comment: Should this be in this stratum? It is agreed that the MOW can exercise little control without hooks to the JVM.*
- **object authentication**: restriction of access to public methods, etc.. From PA1. *APM comment: Agreed. Security models to come. Note that between clusters, only methods on exported interfaces may be invoked. An object may therefore restrict which of its methods may be called.*
- **system events**: what standard mechanisms (e.g. Java exception handling). Event broadcast? From PA1. *APM comment: Java exceptions: all local/remote method invocations may raise Java exceptions. Event broadcast: see MOW section.*
- **PUSH communication required**: in addition to traditional PULL. From PA2. *APM Comment. I believe the MOW facilities described in this document will provide the required functionality. If this is not the case, then we need to discuss PUSH requirements in more detail..*
- **bulk-transfer mechanism required**: RT issues not important. From PA2. *APM comment: MOW will provide access to existing techniques (eg sockets) in a controlled way.*

(P)IS

- **Location transparency policies for profile objects**: specifically, template for viewing/moving diary object to be determined. From PP. *Editor's comment: What is meant by template?*
- **Use of non-personal Information Spaces needs outlining**: From SD.
- **IS Factory**: how are Information Spaces created? How are old objects garbage collected? From PA1.
- **is access to the Information Space only via the Personal Assistant?**: from PA1.
- **control over actual location of data required**: from PA2.
- **availability, consistency and response times**: presumably some exist from AA, PA1, PA2 - not defined. Raised by (P)IS, which is expecting requirements to be placed upon it by other workpackages.

AA

- **Agents:** Agents who want some information to be delivered by the user access, need to know to which place to go or to send the info in order to access a host in which the wanted output devices are available. From UA.
- **Agent owning:** who owns an agent itself (within the system)? From PA1.
- **User interfaces**: do agents carry around their own user interfaces or are there libraries available? From PA1.

PP

- **Develop techniques for description of metadata**: from SI.
- **Develop tools for construction of profile objects**: From SI.
- **Easy access to profile factory**: from SD & PA2.

SI

- **interaction by an Agent**: needs definition (in SI anyway). From AA & PA1.
- **definition of what is domain specific and what not**: From PA1.

SD

- **interaction with AA**: needs better definition. From AA.
- **how should services use Profiles?**: really from this to PP, but raised in PP.

UA

- **interaction with AA**: needs better definition. From AA.
- **libraries**: are they available? By implication from PA1.
- **rule triggering (events?) on layout adaptation**: when tweaking takes place in UA, ETEL++ wants rules triggered. From PA2.

Open Issues

A number of issues exist which have yet to be discussed, clarified, assigned to work packages or deferred.

I expect that the dependencies raised in the strata, and listed above, will also throw up one or two more unresolved issues.

Events

There still seems to be an amount of confusion as to the requirements on or necessity of an Event service, and whether Java 1.1' s default service, combined with what the MOW will provide, will fit the bill. A discussion of this issue has been posted to the Message Board and to APM area on the HyperWave server.

Security

What levels of security are needed, and are likely to be available, is yet to be fully defined. Important here is to define what is needed at what stage in the project. For instance, it may well be acceptable to provide hooks to start with, and then to go on to a more fully developed model and then implementation. Discussion and use case scenarios are needed.

Object Naming and Trading

What requirements exists for Naming and Trading, and at what level do they sit? MOW will provide basic services, but it is expected that applications will have extra requirements for directories of named objects. At the UWE meeting, 17-19 November, it was agreed to return to this issue when application requirements were clearer. It was also agreed to postpone discussion about Trading, as there was no immediate requirement. Future issues may include Trading services at the ‘ knowledge domain’ and service interaction levels.

Profile, Host and Place Profiles

Is there a general purpose Profile meta content; what in particular is a Host or Place profile?

User Access Module/Subsystem

Is there a User Access Subsystem object, or is it package/collection of classes allowing agents the illusion of direct I/O to/from (virtual) devices (e.g. AWT components, telecoms devices, etc.)

Stratum: Mobile Object Workbench (WP B)

Status

Most of the design work of the MOW has been completed. Coding has not started, although we intend to make extensive use of the FlexiNet ORB.

Survey done: Oct 1997
Requirements done: Nov 1997
Design due: Dec 1997
Interfaces due: Dec 1997
Implementations due: Jan, Feb, Apr, Jun, Sep 1997

Scope

This work package will deliver a workbench upon which the other strata will be implemented. It is intended to provide facilities for object movement to allow the creation of a variety of high level services, but does not include the creation of these services.

In particular it includes:

- a facility to allow location transparent reference to interfaces on MOW objects
- a facility to allow certain objects to move between machines, and for this movement to be transparent in that references to (or from) the object continue to work.
- facilities to allow extension of the MOW to form other higher level workbenches, for example hooks to allow a higher level workbench to be aware when an object moves.
- a model for security and a tool-kit to provide security facilities.
- facilities to enable a mobile object to determine information pertaining to the place at which it is located. The contents of this information are not determined by this stratum.

This work package does not include:

- General naming or trading services, other than those required to provide location transparent interface references. However, as a short term solution, a simple naming service will be provided. Other strata are at liberty to extend or replace this as appropriate.
- Event classification and notification. However, the stratum does include a tool-kit of facilities to allow event notification to be implemented in a straightforward manner. This follows the Java beans event model, and extends it with asynchronous notification.

Issues

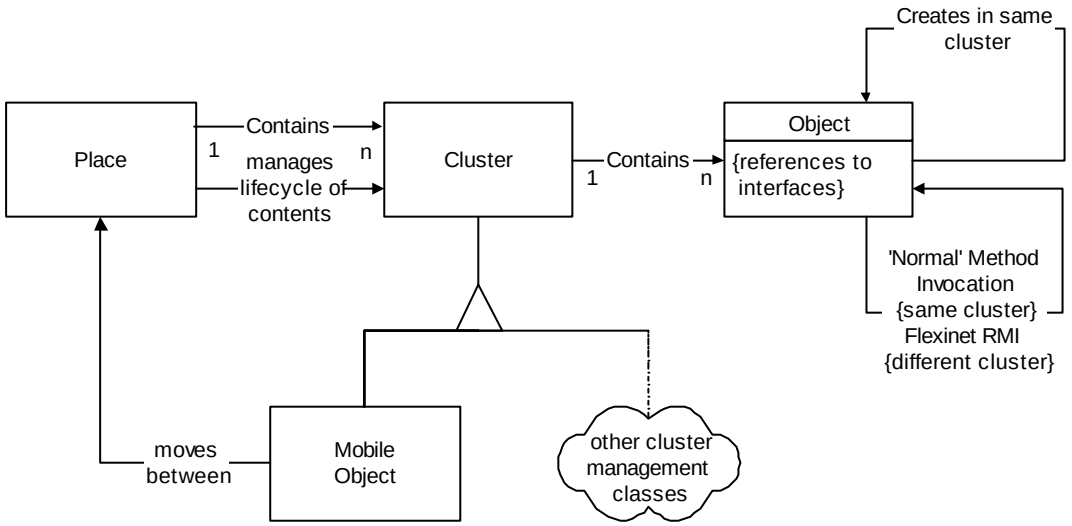
The following issues are still to be decided with respect to this stratum:

- **encapsulation**: The MOW will attempt to encapsulate mobile objects so that they may not maliciously affect each other, or consume resources which were not allocated to them. In practice the extent to which this is possible in the current JDK/JVM implementation is limited. This situation may change in the near future (with JDK 1.2)
- **security**: The extent to which agents and communication may be secure is still to be determined.

OMT Models

Computational Model

This describes the components of the MOW at a computational level. Engineering issues, such as how transparencies are realised, and ‘ grubby details’ are omitted.



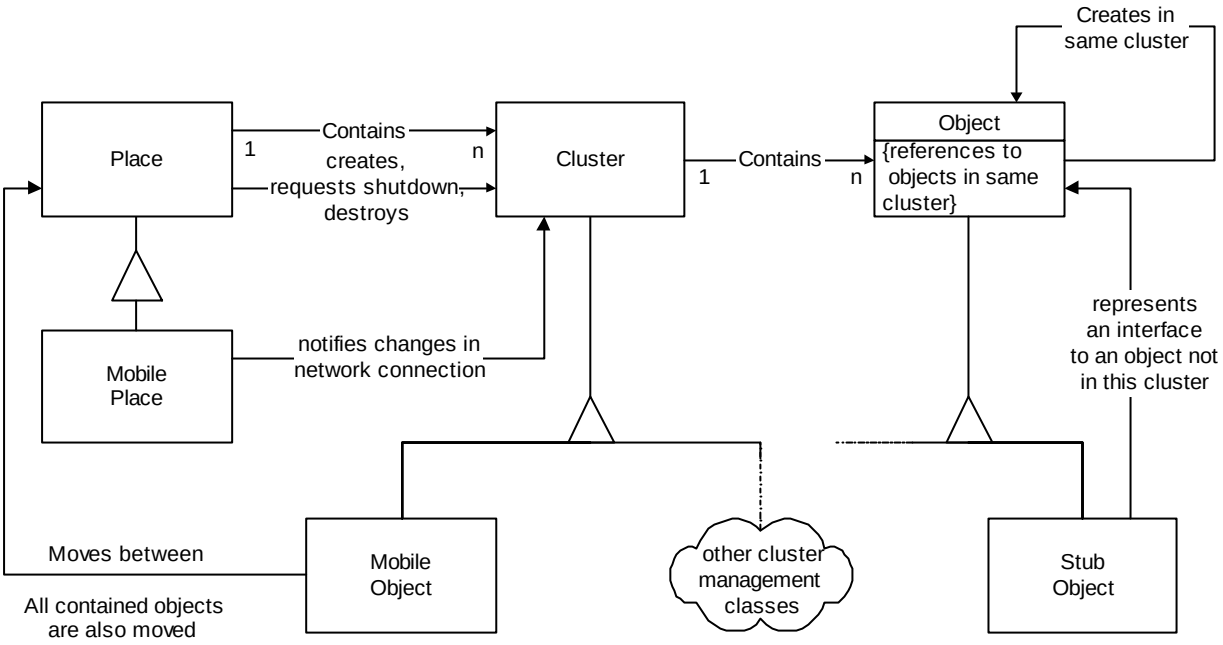
There are three primitive concepts:

- A place is an abstraction of an execution environment. It contains a number of clusters.
- A cluster is an object containing a grouping of objects that are managed together. A mobile object is a specialisation of this which is able to move between places. There may be other forms of cluster that have different group management facilities. Each cluster contains a number of objects. The cluster and objects within it together form a group of related code and data objects. Protection, movement, destruction, charging and other management functions are considered in terms of the lifecycle of clusters and the interaction between them. It is sometimes useful to consider a cluster and its contents as a virtual process.
- An object is the basic building block out of which applications or agents may be built. Objects may contain references to interfaces on objects anywhere in a FollowMe system. Objects may directly create other objects, but only within the same cluster. They may be able to arrange the creation of objects in other clusters via communication with a place. Within a cluster, access to methods/data on objects is determined by standard language protection means and takes place using standard method invocation. Between clusters, encapsulation is enforced so that object in one cluster may only access methods on objects in other clusters, if these methods form part of the interface passed between the clusters. The method invocation semantics are those of FlexiNet RMI.

(Not shown) In most respects Place, Cluster and Mobile Object are also Objects. The distinction is that objects exist within Clusters whereas Places, Clusters and Mobile Objects do not. A cluster may create objects within itself. Creation (and destruction) of clusters is under the control of the place at which they reside.

Engineering Model

This describes the externally visible components of the MOW at an engineering level. The diagram is not Java specific, although the intended implementation is. Note that internal engineering is not shown.



The notions of place, cluster, object and mobile object map directly onto the equivalent computational level entities. Engineering objects are distinguished from computational objects, in that engineering objects can only contain references to objects within the same cluster. References to (interfaces on) objects in other clusters are represented by distinguished objects called stub objects.

In order to move, a method is invoked on the Mobile Object. This method is available to all objects within the cluster, but is not normally exported outside of the cluster. When a mobile object moves, all contained objects are also moved. However, transient objects are not moved, but are instead discarded. *(not shown)* Transient objects are defined as those that are only reachable via transient references from the mobile object itself, or by transient references from objects implementing exported interfaces. An exported interface is one passed outside of the cluster.

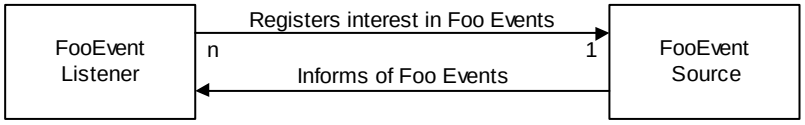
When a cluster is destroyed, objects within it are also destroyed. In a Java implementation, this cannot be enforced, and instead communications between objects internal and external to the cluster is prevented.

A distinguished class of place, called a 'Mobile Place' is defined. This is a place which may change engineering attributes (for example connectivity to the network, or network address). The cluster will be notified of such changes. By default, these changes will be ignored, however sub-classes of Cluster or Mobile Object may override this default behaviour.

Events Services

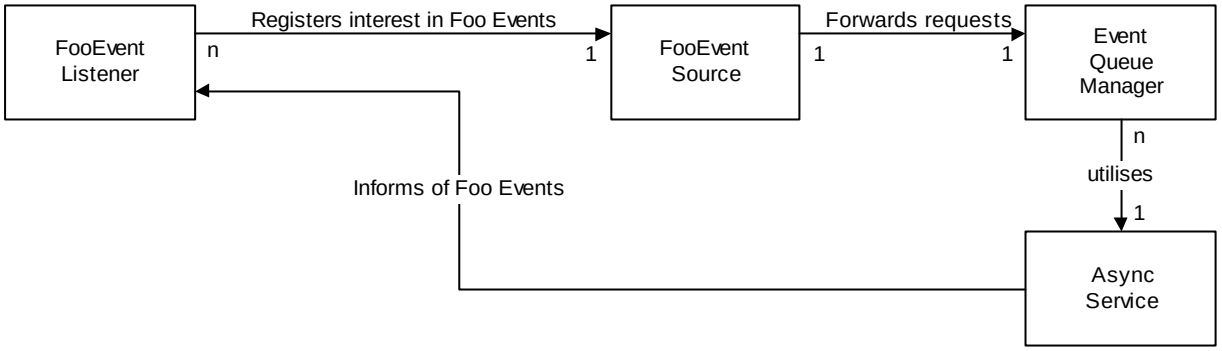
Two services will be provided on top of the workbench to aid with the notification of asynchronous events. The first of these, an *asynchronous invocation service* will allow one object to invoke a method on another object without having to wait whilst that method is serviced. Results from the method invocation (if any) are discarded. The second services *an event queue manager* is a lightweight service that an object may use to implement the management of potentially large numbers of ‘ listeners’ . This service maintains information about which other objects have registered an interest in a particular class of event, and notifies them of the event on behalf of the object generating the event. The event queue manager makes use of the asynchronous invocation service.

Computational View of Event Notification



An event source is an object conforming to a particular event interface (for example FooEvents). Similarly an event client is any object that conforms to a suitable interface for receiving events. See the Java 1.1 specification for more details.

Engineering View of Event Notification



The Event Queue Manager and Async Service together provide all the required functionality for straightforward event notification. Of course, a particular event source is at liberty to bypass the Event Queue Manager to implement more complex event notification (for example distributing events dependent on client specific criteria.)

Components

As examined in the OMT models, the components that are ‘ owned’ by this stratum are:

- **place:** the location at which mobile objects reside
- **cluster**: the mechanism for collecting objects into a unit
- **mobile object**: the unit of mobility
- **event services**: basic services to aid in event delivery.

Function

The MOW is involved with making engineering transparent. It will allow distinguished objects (mobile objects) to move from place to place by invoking a method on themselves. Communications between mobile objects (and between non-mobile objects) is performed by simple method invocation. It will be (selectively) transparent to the two parties involved in the call whether either of them has moved since the last invocation, or is in the process of moving.

In addition, the MOW will provide hooks, as required, to allow other strata to gain introspective information about the functioning of the MOW. For example this might include information about the arrival or departure of objects from a place, or hooks to allow a place to refuse an object’ s request.

API (draft)

```
public class Cluster
{
  // Only the cluster and the place ever have references
  // to this interface. They are therefore the only parties that
  // may make use of these calls.

  //invoked by the Cluster or its contained objects

  /* lock(), unlock()
   *  Prevent calls on this cluster's methods by other clusters. These calls
   *  will block until unlock() is called. lock() may be called any number of
```

```
    *  times by a particular thread, and unlock() must be called the same number
    *  of times to unlock the cluster.
    */

   public final void lock();
   public final void unlock() throws UnMatchedUnlock;

   //invoked by the cluster's current place
   //the cluster, or its contained objects

   /* requestStop(), stop(), destroy()
    *  Three severities of stop.
    *  requestStop() asks the cluster to cease processing (e.g. by moving)
    *  stop()        tells it to stop, but the cluster has a chance to clean up first.
    *  destroy()     stops the cluster dead. (cannot be ignored)
    *  when the call from requestStop() or stop() returns, the cluster is
    *  destroyed (if it has not moved).
    *  Typically, if a cluster does not 'stop' if the place requests it,
    *  after a time the place may call 'destroy'.
    */

   public void requestStop();// move or stop
   public void stop();       // graceful exit
   public final void destroy();exit now

   // invoked by the place on the cluster

   /* Object init(ifacecls)
    *  This is called when the cluster is first created
    *   this should return the object implementing
    *   an interface of class 'ifacecls'
    *   or null if no interface return is required.
    *   No calls may be made on the cluster until this function returns.
    *   The cluster may create new threads in this (or any other) call
    */

   public abstract Object init(Class ifacecls)

   /* restart(int reason)
    *  This is called in the following circumstances
    *    After a successful move, at the new location (resason=InitMoved)
    *    After a pendMove() that failed (reason=PendMoveFailed)
    *    After restart after failure (reason=RestartAfterFailure)
    *    For a different engineering reason (reason=?)
    *  No calls may be made on the cluster until this function returns.
    *  The cluster may create new threads in this (or any other) call.
    */
   public abstract void restart(int reason);
}

public interface Place
{

   //invoked by an object in a local cluster

   /* getContext()
    *get a handle on information pertanining to this Place.
    * exactly what information this should contain is TBA.
    */

   public Context getContext();

   /* newCluster(objclass,ifclass)
    *create a new cluster. objclass must be a subclass of
    * cluster. ifclass is an interface class that will be
    * returned by init(). This call returns the interface
    * returned by init().
    */

   public Object newCluster(Class objclass, Class ifclass);

   //Callbacks to inform the place when clusters move/are created

   /* They may be used by an implementation of Place to
    * monitor the current cluster population.
    */

   public void callbackNewCluster(Cluster c)
   public void callbackClusterArrived(Cluster c,Place from)
```

```
  public void callbackClusterLeft(Cluster c,Place to)

}

public class MobileObject extends Cluster
{

  /* Only the mobile object. Objects within it, and the place ever have references
   * to this interface. They are therefore the only parties that
   * may make use of these calls.
  */

  //invoked by the Mobile Object or objects within it

  /* moveTo(Place p) - does not return
   *  The thread calling this blocks until the object is able to attempt
   *  movement. If this move fails an exception is thrown. If this move
   *  succeeds, the call never returns. (If several threads make this
   *  call, only one may return an exception)
   */

  public void moveTo(Place p) throws AccessDeniedException, MoveFailedException
                                     IncompatiblePlaceException;

  /* pendMove(Place p) - returns immediately
   *  Move to the place p as soon as possible.
   *  There may be a synchronous check to see if the move is possible,
   *  if it will definitely fail, an exception is thrown.
   *  This call prevents any new invocations of this cluster's methods by
   *  objects within other clusters. When all currently executing invocations have co
pleted
   *  and all local threads have exited, the object will attempt to move to p.
   *  If this move fails, this.restart() is called with an error status.
   */

  public void pendMove(Place p) throws AccessDeniedException,
                                       MoveFailedException,
                                       IncompatiblePlaceException;

  /* lockLocation(), unlockLocation()
   * Prevent the object from moving, or allow it.
   * Useful if the object is making use of resources in a
   * way that would cause inconsistencies if it moved. For example
   * an object that is invoked in a transaction may or may not be
   * able to move without violating the transaction semantics.
   * lockLocation() may be called many times, and unlockLocation
   * must be called as many times before the location is actually unlocked
   * lockLocation() and unlockLocation() may be called in different threads.
   */

  public final void lockLocation();
  public final void unlockLocation() throws UnMatchedLock;
}

public interface MobilePlace extends Place
{

  // Typically, these calls will be made by the
  // infrastructure to the place.

  /* lockForMove()
   *  Inform the place that it should isolate itself, as
   *  network disconnection is about to occur. When this
   *  call returns, the place may be 'cleanly' disconnected.
   */

  public void lockForMove();

  /* resumeAfterMove()
   *  Inform the place that it has restarted, possibly
   *  at a new network address. The place will reread IP
   *  or other network addresses and update any naming
   *  services it is using of its new location.
   *  resumeAfterMove() may be called even if the place
   *  was not cleanly shutdown using lockForMove()
   */

  public void resumeAfterMove();
}
```

Comments on API

The API given is for the MOW itself, not the additional services provided in the MOW work package. The API is relatively stable, but security functions will be added at a later date, and these may impact slightly on the methods listed above. In addition, any object may implement any public interface and this interface may be made available to objects in other clusters (mobile or otherwise) in a location transparent way. When calls are made across these interfaces, the semantics are those of FlexiNet RMI. In particular

- The call may fail due to security or engineering reasons
- Data (primitive types and objects) are passed by copying their value
- References to other interfaces are passed as references.

For more details, see the FlexiNet Open Orb Framework document (APM 2047.01).

Requirements on other strata

As the MOW is the ‘ bottom’ strata there are no requirements on other strata, other than to ensure that we are aware, and have agreed to any additional functionality, or hooks, that must be supplied by the MOW. In order to make use of the MOW, a naming and/or trading service may be required. This has yet to be allocated to a strata.

Stratum: (Personal) Information Spaces (WP C)

Status

The work on this work package (WP C - Personal Information Space) has only just begun.

Survey due:	n/a
Requirements due:	end Dec 97
Design due:	end Jan 98
Interfaces due:	end Feb 98
Implementations due:	end Mar, Jun, Oct 98

Scope

An information space (see Figure 11) provides a facility for maintaining, accessing and collating information. It provides mobile users with a single and consistent (i.e. logical) view of an information space irrespective of where they are located. Users and agents may access an information space - though it may be that only Personal Assistant agents have access to a user' s personal information space.

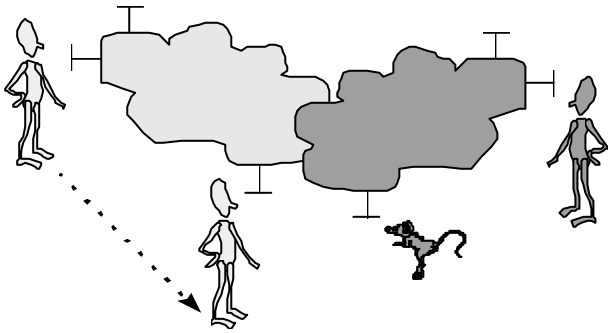


Figure 11 Users, agents and information spaces.

Issues

- **security**
- **persistence**
- **garbage collection**

- **consistency requirements**
- **what kinds of information are stored in the information space - results from foraging agents, personal profile, diary … ?**
- **which entities have access to an information space**
- **shared or private information spaces**

OMT Models

A host may support a number of places. A place provides access to a number of interfaces. These interfaces may be to agents, services, devices, and information spaces. A mobile user requires access to the same information spaces, irrespective of which place they are in. In addition to supporting places, hosts may also provide storage facilities for objects with the attendant support for replica management. A storage facility may hold parts of a number of information spaces; each information space may be distributed across several storage facilities. This is depicted in Figure 12.

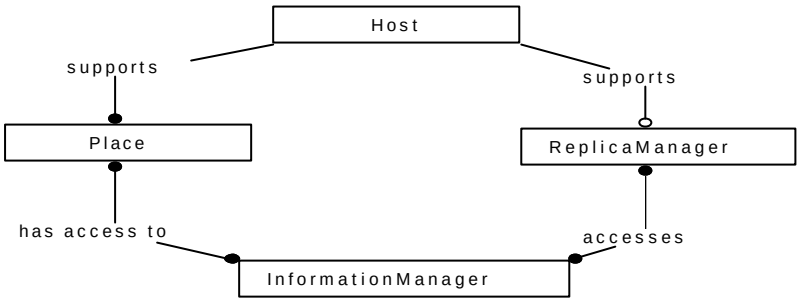


Figure 12 Hosts, Places, and Replicated Information.

Devices enable users to interact with places. They are also the means by which places interact with users. Typical devices could include WWW browsers, telephony interface, fax interface. A user views an information space via some application, and issues commands to their personal assistant. The personal assistant in turn issues commands to task agents. Task agents do not however have direct access to a user's information space - all such access is mediated by a personal assistant which may place the information in an information space and/or notify a user. To provide the required access characteristics, it may be necessary for a personal assistant to be able to deploy information to a storage facility that is (in network terms) close to a given user.

Components

Function

The physical objects which constitute an information space may be distributed across several nodes:

As a consequence of mobility - some nodes, and hence the objects resident on them, may physically move when a user moves.

- As a requirement for mobility - some nodes may become inaccessible when a user moves (as a consequence of the user becoming disconnected from the network or due to network failures). To preserve a user's logical view of an information space, it may therefore be necessary to redeploy objects to other nodes.
- There is thus a need to replicate data across nodes to provide the required levels of performance, availability and fault tolerance (see Figure 13).

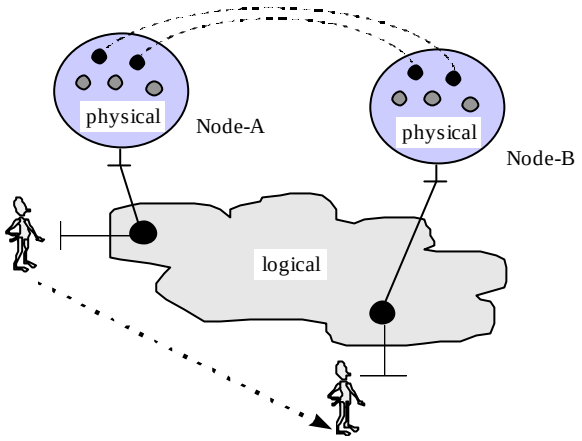


Figure 13 Logical and Physical Information Spaces.

Replication

Replication is the maintenance of on-line copies of data and other resources. It can provide enhanced performance, high availability, and fault tolerance.

However, replication brings with it the following issues:

- Replication transparency - clients should not be aware that multiple physical copies of data exist.
- Consistency - it may not be acceptable for different clients to obtain different results when they make the same requests to the same logical data items.

There are many different approaches to replica management, ranging from a simple asynchronous model in which all client requests are processed by the local replica manager and local replica manager communicate updates to all other replica managers of the same data items, to a totally synchronous model where all updates are totally ordered. In general, most schemes fall somewhere between these two extremes - the asynchronous model may not provide adequate consistency and the synchronous model may have unacceptable response times. A range of techniques have been devised to make tradeoffs between consistency, availability and response time.

API

A distributed container has two distinct kinds of interfaces: an operational interface, representing the logical view of an information space:

Access

```
put(name, Object)
Object get(name)
remove(name)
iterate or enumerate
```

and a physical view, allowing control over how the underlying data is physically distributed:

Deploy

```
make Object availableAt Place
make Object unavailableAt Place
```

Figure 14 shows how an information space presents a logical interface (IS_L) which is accessed from a number of physical locations (IS_P):

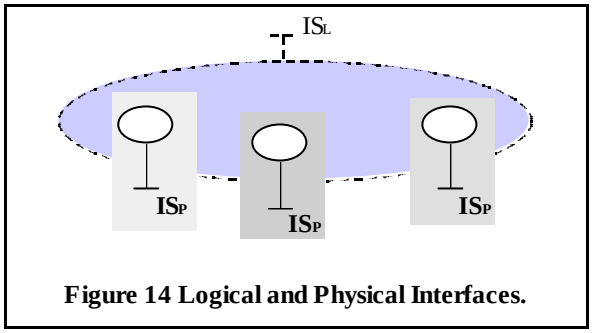


Figure 14 Logical and Physical Interfaces.

Figure 15 shows how each of these physical interfaces relies on a bag of objects that are physically deployed at some node. In this scenario all physical interfaces rely on the same set of objects - two of the physical interfaces are remote from the data, one of the physical interfaces is co-located with it.

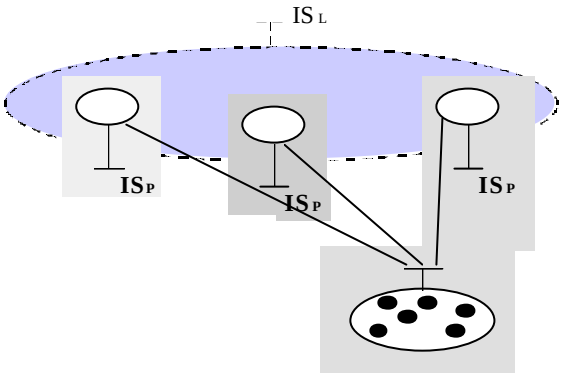


Figure 15 Physical Interfaces and Physical Data - co-located and remote.

The deploy interface may be used to physically deploy these objects at some other physical location. Figure 16 shows the arrangement when the objects have been replicated at another node.

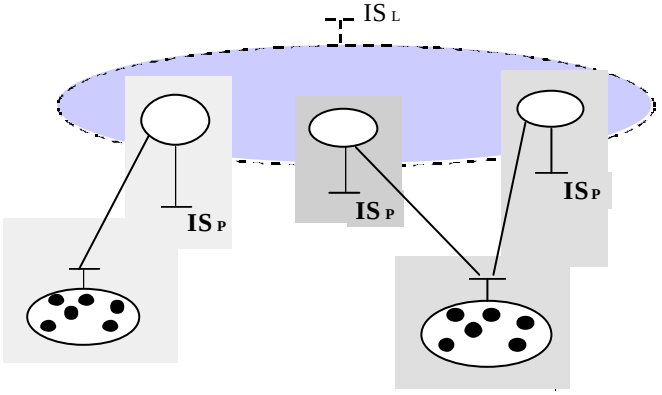


Figure 16 Replicated Data.

With replication of objects may come a requirement to maintain consistency between these distinct physical clusters - perhaps via the containers or perhaps via the physical information space interfaces (this equates to the role of replica manager described above). This is shown in Figure 17.

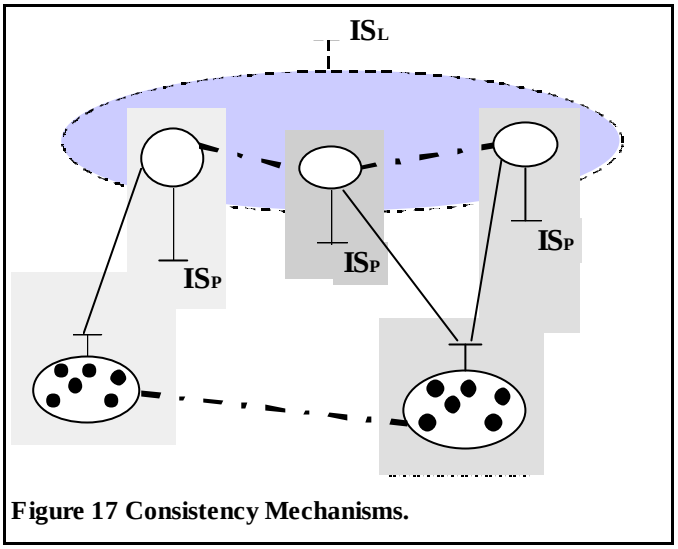


Figure 17 Consistency Mechanisms.

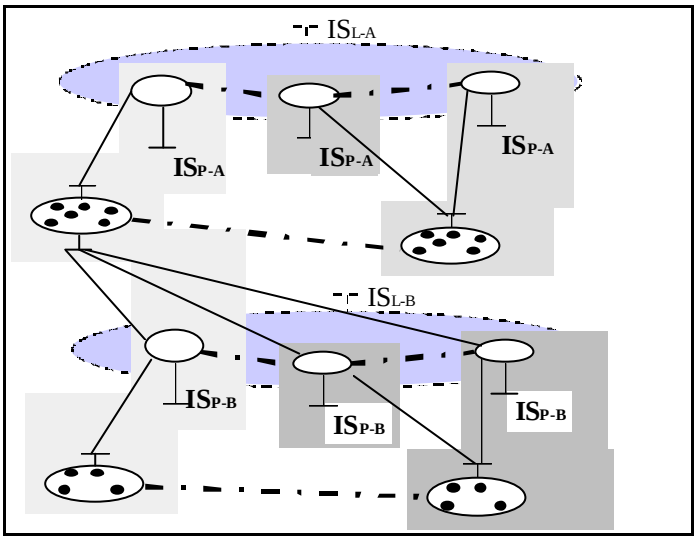


Figure 18 Multiple, Overlapping InformationSpaces.

Finally we can view sets of information spaces, with overlapping objects. Figure 18 depicts two information spaces, A and B and two logical clusters of objects. Information space A and B have access to one of these clusters and the cluster is deployed at 2 nodes. Information space B also has access to another cluster, similarly deployed, but inaccessible to Information space A.

Comments on API

Requirements On Other Strata

MOW: basic mechanism to support deployment by cloning and moving clones, possibly also support for replica management.
What are the requirements for availability, consistency and response times from PA1, PA2, AA?

Other comments

- Mutable c.f. non-mutable data - much of the information handled by the newspaper pilot will be read-only, hence the problem of consistency maintenance disappears.
- What are the implications for replica management when some cluster of objects goes off-line? For example, the distributed diary may be physically spread across a number of nodes, one of which is a disconnected PDA. What needs to happen when that PDA is reconnected?
- Semantics - putting an object into the information space deploys it at some location that is close to the client that first performed the operation. Repeated calls to this operation have no effect- the object is logically in the information space. To affect an object' s physical distribution, the deploy interface must be used.
- Explicit control over deployment of data - e.g. to achieve off-line reading when disconnected. Metaphors such as packing a suitcase before travelling may be helpful. Push objects before a mobile user moves or pull objects when a mobile user arrives at some destination
- COTS tools modify files. To support the mobile user it may be necessary to move files via objects. A wrapper object can pack and unpack the data and be responsible for detecting inconsistencies between a source object and its copies.
- Some technologies that may be of relevance include: JavaSpace, JNDI - Java Naming and Directory Interface, JavaBeans. The "Glasgow" JavaBeans specification makes some very relevant extensions to the Java 1.1 Beans specification - see below).
- There exist places that objects can be sent to and pulled from.
- Different classes of objects may have different responsibilities when they move and different capabilities for movement.
- Some objects might need to be discarded (left behind and garbage collected) when the mobile user moves.

JavaBeans - "Glasgow" Specification

JavaBeans is Java' s component model. Glasgow Beans is the code-name for the next release of the JavaBeans component model specification. It introduces the following extensions:

- Runtime Containment and Services - when a Bean is introduced to its environment (e.g. a WWW browser, word processor, etc.), it knows that it's running inside a Java VM and that it has access to the core Java APIs. However, today, it has no standard means to discover other attributes or services that may be available in its surrounding environment. The Runtime Containment and Services protocol allows developers to nest Beans within other Beans more seamlessly, where the nested Beans can procure additional services at runtime from its environment in a standard fashion. Similarly, it provides a standard mechanism for the environment or containing Bean to extend its capabilities directly to Beans within it.
- Activation Framework - enables developers to take advantage of standard services to determine the type of an arbitrary piece of data, encapsulate access to it, discover the operations available on it, and to instantiate the appropriate JavaBeans component to perform said operation(s). For example, if a browser obtained a JPEG image, this framework would enable the browser to identify that stream of data as an JPEG image, and from that type, the browser could locate and instantiate an object that could manipulate, or view that image.
- Drag and Drop - provides platform-independent drag and drop facility for Java GUI clients implemented though AWT and JFC classes. It leverages the existing `java.awt.datatransfer.*`package to enable the transfer of data, described by an extensible type system based on the MIME standard.

Stratum: Autonomous Agents (WP D)

The aim of this work-package is to develop an agent-based view of the FollowMe architecture.

Status

Survey due:	end of December 1997
Requirements due:	end of January 1998
Design due:	end of February 1998
Interfaces due:	end of March 1998
Implementations due:	April/May/July/October/November 1998

Scope

This work package will provide an agent-based view of the underlying workbench. Editor' s Comment: should it not provide an agent-oriented model of computation to agent programmers, which happens to be engineered with the workbench? It will provide a framework to allow the selection, dispatch and subsequent return of agents acting in order to complete tasks.

It includes:

- providing a general interaction framework
- off-line completion and mediation of tasks
- an agent description (scripting) language
- monitoring correctness of agents

It does not include:

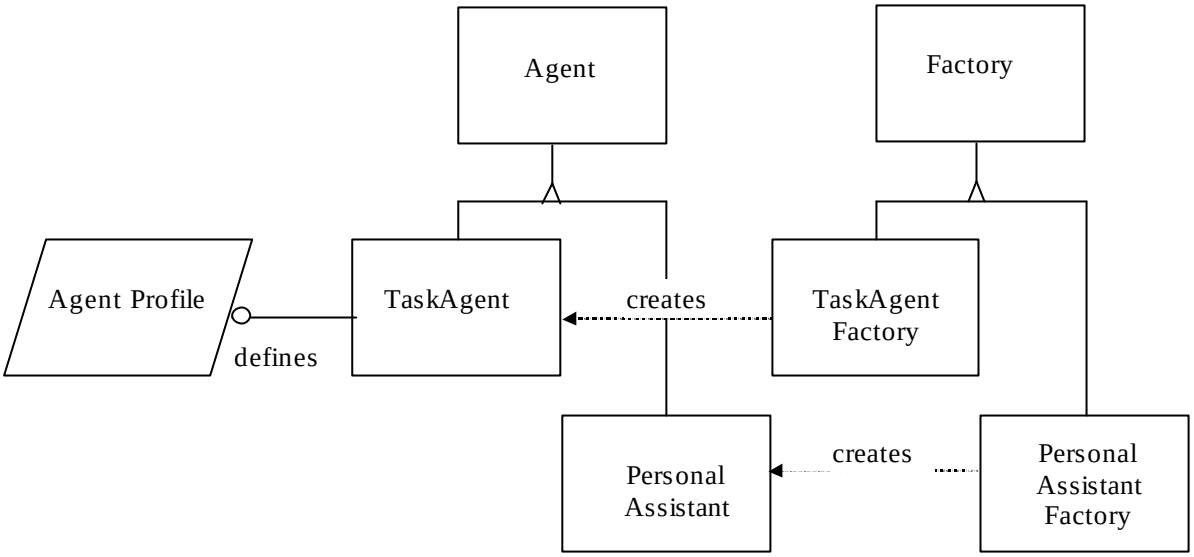
- agent "intelligence"

Issues

The following issues are still to be decided with respect to this stratum:

- **security:** it is not clear how security issues will impact this stratum
- **user access:** the way in which the user will access the Personal Assistant needs to be clarified and an interface agreed upon.
- **workbench**: what will be provided by the workbench is emerging, issues such as events need to be resolved.
- **interaction model**: a prescriptive design method is beyond the scope of this stratum however an interaction model which the stratum supports will be needed.

OMT Models



Components and Function

As examined in the OMT models, the components that are ' owned' by this stratum are:

- **Personal Assistant:** co-ordination agent responsible for receiving user' s delegated requests, interpreting the request in terms of what needs to be accomplished, creating an appropriate task agent and dispatching it. The PA is also concerned with correctness of operation. *Editor's comment: what does this mean?*
- **Task Agent:** Task agents include both pre-programmed agents and extensible agents specified by an agent profile.
- **Agent Profile**: Extensibility is achieved with the idea of an agent profile defining the components which make up the agent, and an agent script to `glue' these components together.
- **Personal Assistant Factory:** object capable of creating, managing and destroying PAs
- **Task Agent Factory:** object capable of creating task agents. *Editor's comment: managing and destroying?*

API

Various APIs will be specified, some will be visible to other work packages such as User Access and some will be for internal use within the AA work package. The following will be provided:

- **Personal Assistant**
- **Personal Assistant Factory**
- **Task Agent Factory**
- **Task Agent**
- **Agent Profile**

Comments on API

It is not possible to specify the APIs in any detail at present.

Requirements on other strata

- **MOW:** facilities and API to be provided

- **UA:** interaction between AA and UI unclear
- **SD:** interaction with SD unclear
- **SI:** how an agent interacts with services is in SI

Other comments

The role of AA is unclear due to a lack of envisaged application scenarios however we hope that it will take shape once some clear scenarios are outlined.

Stratum: Personal Profiles (WP E)

The aim of this work-package is to determine a representation for the storage of personal information.

Status

Survey due:	end of November 1997 (draft copy available)
Requirements due:	end of December 1997
Design due:	end of January 1998
Interfaces due:	end of January 1998
Implementations due:	February/May/August 1998

Scope

The aim of work-package E is to determine a representation for the storage of personal information in support of other FollowMe components. This profile is a human-readable document from which one or more profile objects may be constructed. The profile object will present an interface by which the content of the profile may be obtained.

An important aim of this work is to provide extensibility of the profile, so that new kinds of data may be added without rebuilding the software. *Editor's comment: what software?*

It includes:

- User identification and addressing
- Diary information (and a specific diary object)
- Resource information (e.g. followMe placeMarks and service handles)

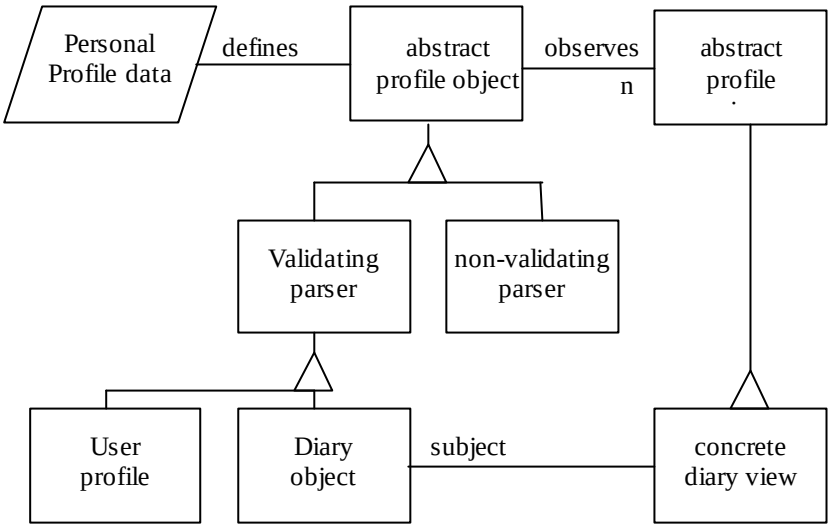
Issues

The following issues are still to be decided with respect to this stratum:

events: There is a need for the diary object to be a source of scheduled events.
security: Does security information (keys) belong in the profile?
host profiles: Is the host profile (e.g. memory space, cpu usage) a profile in the sense of meta-data described here?
device profiles: In describing the input/output capabilities of a given device, will the user access work package utilise this work-package?

geographic information: Is there a requirement for this kind of information as distinct from logical addressing? How would it be used?

OMT Model



Editor's comment: these inheritance relationships are unexpected. Doesn't a parser generate a diary object from profile data?

Components

The components of the Personal Profile workpackage include:

profile data: A MetaData description of a profile object
profile object: The abstract object described by the profile data
non-validating parser: A kind of profile object where the profile data need only be well-formed. This class supports extensibility by allowing properties to be accessed generically by name.
service profile: service profiles unknown at compilation time may be added to the profile data and accessed via a generic (non-validating) profile object.
validating parser: classes designed specifically to parse core profile data. Profile object properties will correspond directly to profile data properties. The user object includes core personal information.
diary object: The diary object retains core calendrical properties, and associated methods implementing a calendar model (see below).
profile view: A profile view implements the policies required to implement location transparency in the informationSpace.
diary view: The diary view implements an abstract template defined by the profile view. It allows the diary to be rendered and browsed on some device.

Function

Profile objects are generally passive, returning profile data items on request (though see issues re events). Information may also be written back into the profile data file where appropriate. The diary view supports interactive browsing/updating of the diary profile.

API

The API specification is due at the end of January 1998. This document will comprise:

- An interface to a generic (non-validating) profile object.
- Interfaces to specific (validating) profile objects (User and Diary).

Comments

See survey (DE1) for more details.

Requirements on other strata

InformationSpace requirements

Where profile objects exist in the Information Space, suitable policies supporting location transparency must be devised. Specifically, the template for viewing/moving a diary object in the Information Space is yet to be determined. *Editor's comment: What is meant by template?*

Service deployment requirements

The personal profile is the place in which particular services may wish to place application specific information (cookies). For example, a data structure describing a personalised newspaper.

Other comments

None.

Stratum: Service Interaction (WP F)

The aim of this work-package is to develop a framework for integrating services into FollowMe.

Status

Survey due:	end of November 1997
Requirements due:	end of January 1998
Design due:	end of February 1998
Interfaces due:	end of March 1998
Implementations due:	April/July/October 1998

Scope

The aim of this work-package is to define a pattern for making services available to FollowMe agents. This will develop a service description language and tools which use these descriptions for service location. A major aim is extensibility; the ability to add new services without rebuilding the code. *Editor's comment: what software?* Consequently, these descriptions are characterised as service profiles using similar techniques to those in the Personal Profiles work-package. The service profile will build on current methods of interface description, comprising three elements:

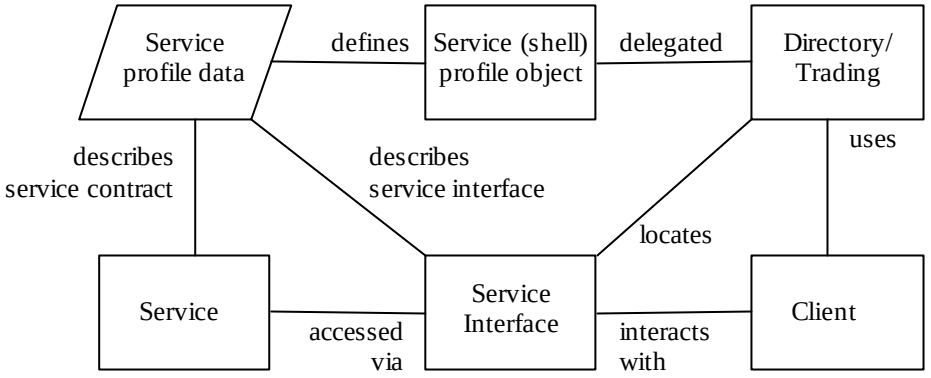
- **Interface syntax:** Definition of service interface data types and methods.
- **Interface semantics:** Definition of a *service model*; the entities and relationships within the interface domain.
- **Interface pragmatics:** Definition of a *service contract*; a description of the admissible behaviours at the interface and their effects in terms of the service model.

Directories and Trading are seen as two distinct levels of the service location problem. Using a directory, a service is located syntactically on the basis of its interface type definitions. Using trading, a service is located semantically on the basis of the service model.

Issues

One unresolved question is who is responsible for the service profile data, and for exporting the corresponding profile object to the directory/trading service. This appears to be a function outside the existing wrapped service.

OMT Model



Editor's comment: should the shell-directory relationship be 'export' and the client-directory relationship be 'import'?

Components

All client/service interactions work via the service interface. The service profile is meta-data about the service and its interface. The service shell is a profile object corresponding to this meta-data and is exported to the Directory/Trading service. This allows clients to locate service interfaces either by specifying a given interface type, or by specifying additional conditions in terms of the service contract.

Function

The service interaction patterns developed within this work-package will enable us to integrate new or existing services into FollowMe, and will be equally applicable to CGI, CORBA or RMI based services.

It will not enable agents to discover how to use new services without an existing service profile.

API

The API specification is due at the end of March 1998. This document will include the following.

Service Description Language: A definition of the content tags expressible within the service profile.
Service Shell: The interface to the service profile object.
Service Directory: A definition of the interface with the service directory/trading service.

Comments

See survey (DF1) for more details.

Requirements on other strata

Mobile Object Workbench requirements:

- A local Directory/Trader should be constructed when a new place is created.

- Agents must be able to find the Directory/Trader in a given place.
- To support distribution, the Directory/Trader needs MOW hooks to identify new places.

Personal Profiles requirements:

- Develop techniques for the description of meta-data.
- Develop tools for the construction of profile objects.

Other comments

None.

Stratum: Service deployment (WP G)

Status

What composes this stratum is well identified. A survey is under construction and requirements are already defined.

Survey due:	December 1997
Requirements due:	January 1998
Design due:	March 1998
Interfaces due:	April 1998
Implementations due:	May 1998

Scope

This work package will deliver two features:

- The first one is a set of low-level tools which provide applications with performance measurements and predictions.
- The second one is an example of a service deployer dedicated to Etel++. The main goal of this deployer is to provide Etel++ with load-balancing features based on performance measurements and the exploitation of profile.

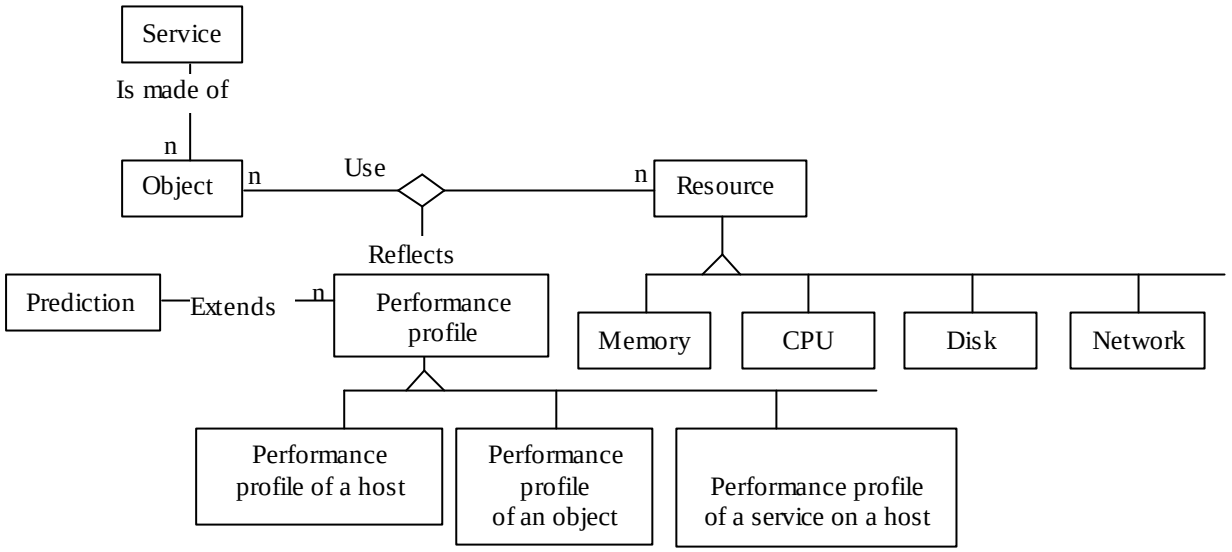
Issues

The following issues are still to be decided with respect to this stratum:

- **Scope of performance profile:** The scope of performance profile is still unclear. Are performance profiles for objects meaningful when objects move or not? Are these profiles only available on the related hosts?
- **Shrink-expand:** this policy, mentioned during the kick-off meeting, needs work. *Editor's comment: what was mentioned during the kick-off meeting? Please explain as much as possible.* The interactions between hosts, places and services have to be clarified.
- **Requirements of PA1:** we expect to receive requirements from PA1 regarding load-balancing issues.

OMT Models

The first picture describes the first feature of Service deployment work package, that is, basic tools monitoring performance.



Components

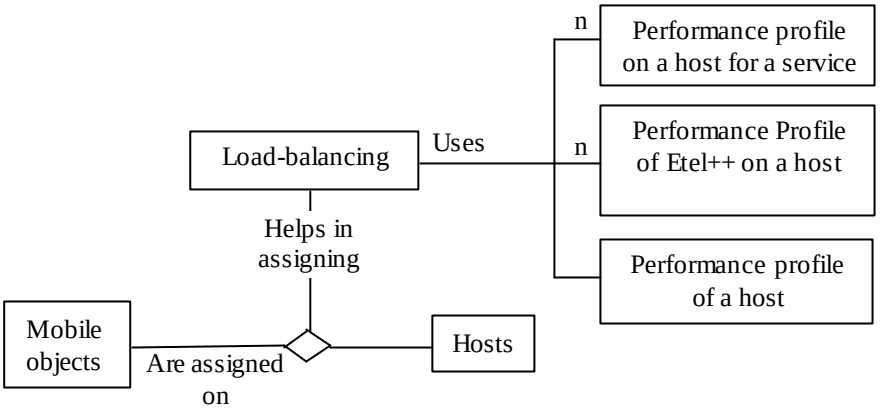
This workpackage monitors resources such as CPU, network, … The values obtained by the monitoring loop are somehow compiled into a profile that reflects the usage of each resource, i.e., a kind of history. In addition, WP-G extends, when this is possible, the basic knowledge provided by these compiled numbers by predictions that try to guess what could be the usage of a given resource in a close future (couple of hours, max 24, say). In addition, WP-G offers a built-in profile for each HOST. As such, 4 basic elements exists in the WP-G:

- **monitoring:** it gets numbers from what is monitored, i.e., instantaneous values reflecting actual performance.
- **Performance profile:** this profile reflects performance *history* of a resource. Numbers obtained by the monitoring elements are somehow compiled into a history of performance. Note the performance is impacted by the objects using that resource.
- **Prediction:** it computes predictions on usage of hardware components. These predictions are also stored in usage profiles.
- **Host profile:** WP-G offers a built-in profile that describes the performance characteristics of each host. This profile is likely to be public, that is, any participant of FollowMe can issue a query returning the profile of a host. This is useful to make smart placement decisions, for instance.

The picture above also gives other elements that are not in the WP-G but that are built on top of it. The performance profile of a service and the performance profile of an object are typical examples. (The latest is analogous to what *getrusage* returns for a Unix process.)

A possible load-balancing policy dedicated to Etel++ is described in the second picture given below. The main components that have to be highlighted in this picture are:

- **Performance profile of Etel++ on a host:** This performance profile stores the resource consumption on a host for the set of objets related to Etel++. (more on this in the WP-J – PA2 draft).
- Other services are described with a **performance profile** as well.
- Given the performance of the service, as well as the performance of hosts that sit around, ETEL can determine the best placement of the objects that implement the electronic press service around the network.



Function

The service deployment package will provide the following functionalities:
- Performance monitoring and predictions
- A service deployer for Etel++

The monitoring tools allow to get performance measurements associated to hosts, objects or a group of objects. They include measurements related to hardware components such as:
- CPU usage
- Disks usage and capacity
- Memory usage and capacity
- Network bandwidth between hosts

Predictions on CPU, disks, memory and network performance are also available. History of hardware components usage and predictions are stored in performance profile.

The service deployer of Etel++ is based on the following techniques:
- computation of the profile of a service by means of data mining and statistics based on user profile
- Decision over data movements (where to compute, what to pull or to push,…).

API

Not yet available.

Requirements on other strata

The following requirements are made by this stratum on other strata:
- **IS:** We would like to outline the new dependency between SD and IS.
- **PP:** easy access to profile factory.

Stratum: User Access (WP H)

Status

The work on this stratum has only just begun

Survey due:	11/97
Requirements due:	12/97
Design due:	01/98
Interfaces due:	01/97
Implementations due:	first : 3/98

Scope

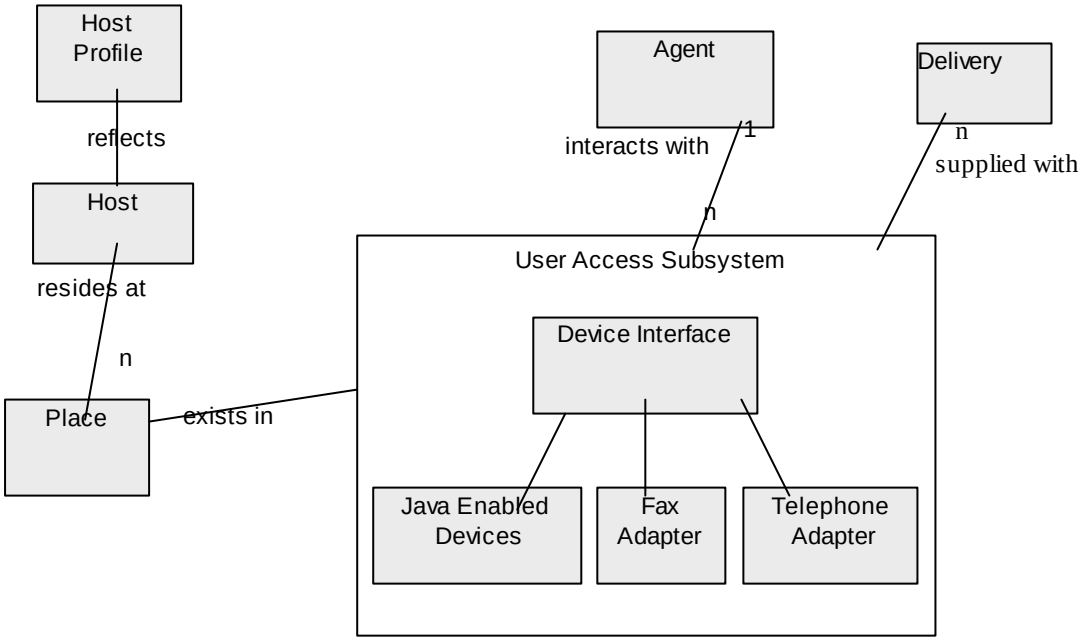
The scope is defined by the Function, later on this paper.

Issues

The following issues are still obscure with respect to this stratum:

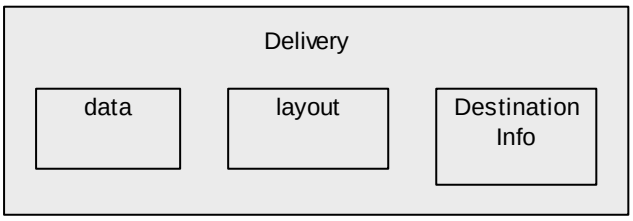
- **Using third parts telephone and fax software:** it is a decided issue, but we need to search more about the software available in the market.
- **Java enabled devices:** not many such devices actually placed in the market.
- **Exact way of communication between agent and User Access:** an "open connection" between an agent who wants to deliver info and get feedback and the user need to be provided. We have an idea concerning this issue, but we need to search more about this.

OMT Models



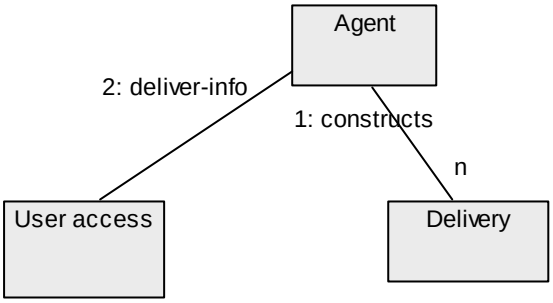
Each Place is provided with an User Access subsystem, through which the agents can access different devices types to deliver the user information.

Java enabled devices, fax adapter and telephone adapter are engineering objects, all contained in the User access subsystem.



The User Access is supplied with a delivery. The delivery consist of data, layout of the data and information about where to deliver the info, e.g. telephone number. The User Access assumes that the info concerning about where to deliver the information are probed data, i.e. the agent is not trying to deliver info to a non connected device to the host where the User Access resides. If the device has problems on receiving data or it is not connected to the host where the operating User Access lives, the User Access will only inform the agent of that fact, but only the agent is capable of looking for a new host, where the desired devices are connected.

Graphics of the process:



Components

As examined in the OMT models, the components that are ‘ owned’ by this stratum are:

- **User Access subsystem**: provides interfaces to the agents in order to deliver information to the user through different devices types.
- **Devices interfaces:** specific implementation to handle with the different devices to be supported by Follow Me.
- **Delivery:** a delivery unit, it is composed by the data to be send to the user, the layout of this data and information about where to deliver the information, e.g. telephone number, IP address, etc.

Function

The User Access will provide the following functionality:

- Access to Java enabled devices for output and input
- Access to fax and telephones (not Java enabled) for output and input
- Error feedback to the agent who required the delivery, e.g. "device not responding"
- Feedback to the agent responding to delivered information, e.g. providing a way of communication between the agent who required the output of information and the user.

Requirements on other strata

- **Places:** Places should be aware of firewalls
- **Agents:** Agents who want some information to be deliver by the user access, need to know to which place to go or to send the info in order to access a host in which the wanted output devices are available.

Stratum: Pilot application 1 (WP I)

Status

The work on this stratum has only just begun.

Survey:	due 12/97	Status: draft version available
Requirements:	due 03/98	Status: in progress; draft version by end of 97
Design:	due 05/97	
Implementation:	due 12/98	
Deployment (1)	due 07/98	
Deployment (2)	due 12/98	
Trials	due 03/99	
Evaluation	due 03/98	

Scope

The scope of the pilots is to implement applications with the following core features:

1) The applications' components can be classified as: information providers (offering access to data-objects), service providers (providing services operating on data available from the information providers), information consumers (engaging agents to make use of available services) and brokers (mediating between the other components). By separating these components the system will easily scale and adapt to new requirements (add new information sources, services or users).
2) The applications enable automated task execution while the user is offline. Information retrieved and processed (i.e. filtered) by agents is stored in a location close to the user (i.e. in the user's LAN or at the ISP of the user).
3) The applications allow the user to be mobile. This means that the user will be able to access the applications from different devices (i.e. Java-enabled devices, fax-machines, mobile phones) and from any geographical location.
4) Information related to the user will be stored in a user trusted environment (in personal profiles).

A generic scenario within such an application will look like the following:

1) All communications between the user and the FollowMe system is handled by the user's personal assistant (PA). The PA connects the user to a service broker that allows the user to select a specific service the user wants to use. This de-couples the user specific components (such as the PA) from the rest of the system (available information sources and services).
2) The broker links the user to the selected service. The user specifies input parameters required by this service, a time schedule that specifies when to execute the service and a location where the user wants the results of the service to be stored.

3) After specifying these input parameters, the user can disconnect from the system. The specified services will be executed automatically according to a user defined schedule.
4) When a service is executed (triggered by the user defined schedule for this service), the agent that is in charge of executing the respective task will contact another broker that links it to information provider sites that hold data objects related to the task. This de-couples service providers from information providers. The service provider does not need to know about available information sources – this information is provided by the broker. New information providers could join the system by registering at the broker service without the need for changes to existing services. The same holds for the integration of new services operating on data offered by information providers.
5) The agent contacts these data sources, collects data objects that match the parameters specified by the user, may performs certain operation on this data and delivers the results to the PA, which, in turn, stores the information in a location close to the user (trusted space).
6) Via his/her PA, the user can specify when and how he/she wants the results of a specific service to be delivered to him/her. Output devices could be any Java-enabled internet device, standard fax devices or phones.
7) The user can keep a diary that holds information on when he/she is reachable through which devices. This enables event triggered information delivery (in addition to scheduled delivery).

FAST is currently investigating in three knowledge domains in two of which pilot applications will be implemented:

- regional event announcing service: various internet access provider within the Bavaria Online Network provide information on regional events (schools, concerts, cinema, exhibitions,...). The agent system queries these sources with respect to a user' s preferences.
- stock information systems: host a portfolio management service for their users. Stock values are queried from existing information sources like Yahoo. The users can define the content of their individual portfolio and set price goals and stop loss marks. The system continuously updates their stock values. Alarms can be triggered by value limits.
- real estate information systems: various internet access provider within the Bavaria Online Network host a real estate information system. The user can specify parameters about real estate objects he/she might be interested in. Information is queried from real estate brokers, savings banks or news papers

Components, Functions and strata they relate to

Component Name	Component functions	Related stratum / WP
User Access	Gateway to end user devices; presents GUI provided by agents to the user; receives user input and sends it back to the respective agent; implements gateways to non-Java devices	WP H: User Access
FollowMe Entry Point	A web-server where the user can log on to the system; initiated the creation of a ' place' at the host of the user; allows the user to initiate the process of getting linked to his/her personal assistant (PA) or to create a new PA	WP B: MOW (concept of places and how to get in contact with PA) WP I: Pilot 1 (Application specific part)
Personal Assistant (PA)	Special type of autonomous agent; owner of all IS-objects, profiles and task agents acting on behalf of the respective user; manages access to IS; manages personal profile (PP); can contact user by referring to the PP (diary section); can (de-)register at the PA Directory Service (PADS); responsible for user authentication; can lookup which task agents the user is currently using; provides basic user interfaces (maintain profile/diary, lookup available services in Task Agent directory service (PADS), select new service); can request instantiation of new task agents;	WP D: Autonomous Agents WP E: Personal Profiles WP I: Pilot 1 (Application specific part)
PA Factory (PAF)	Holds class definitions for PA; can create PA from template (maybe also kill and trace by audit trails – see issue ' object owning'); create initial IS and PP for new user/PA; maybe responsible for backup and recovery of PA	WP B: MOW (concepts) WP D: Autonomous Agents WP I: Pilot 1 (Application specific part)
PA Directory Service (PADS)	Register/de-register PA; lookup location of PA by request of other components so that they can contact the PA	WP F: Service Interaction (concepts of naming and traders) WP I: Pilot 1 (Application

		specific part)
Profile Objects	All kinds of profiles including personal profile, task profile, host resource profile, etc.; store all kind of structured data including diary (where the user is when), task parameters (including diary for reporting and task execution – see Timer component), user preferences, event history, etc.; probably a XML-object stored in IS	WP E: Personal Profiles WP I: Pilot 1 (Application specific part)
Information Space	Storage facility with integrated management functionality; place to store objects; might be mobile; might consist of several parts or sub-IS; objects can be stored/deleted/referenced by authorised agents;	WP C: Information Space
Hosts and Places	Hosts / Places provide a living environment for agents (Java virtual machine); policy files restrict actions of agents on a host (i.e. amount of available resources)	WP B: MOW (concept of places)
Task Agents (TA)	Provide task specific user interfaces; make use of a Timer by defining interrupts that can be called by the Timer; method to quit from a service (involves cleaning up objects in IS, PP and Timer); generate reports (XML-pages) out of data stored in IS and profiles; scheduled reporting (diary functionality; Timer); data retrieval planing (lookup addresses of IPs in IPDS; query IPs for their updating schedule; use Timer); scheduled data retrieval from IP (uses Timer; updating schedule of IP) and store results in IS for later report generating;	WP D: Autonomous Agents WP I: Pilot 1 (Application specific part)
TA Factory (TAF)	Holds class definitions for TA; can create TA from template (maybe also kill and trace by audit trails – see issue ‘ object owning’); negotiate with PA about access to parts of IS and profile for new TA; maybe responsible for backup and recovery of TA	WP B: MOW (concepts) WP D: Autonomous Agents WP I: Pilot 1 (Application specific part)
TA Directory Service (TADS)	Register/de-register domains and TA/TAF; by request, send list of available domains and services to other components; provide human readable description of services; might offer trader functionality that allows agents to look for other agents with specific interfaces;	WP F: Service Interaction (concepts of naming and traders) WP I: Pilot 1 (Application specific part)
Information Provider Service Interfaces (IP)	Allow agents to query information sources provided in a standard format (XML) by calling Java-methods provided by the IP; register/de-register at IPDS; send query results to another object/agent/IS	WP F: Service Interaction WP I: Pilot 1 (Application specific part)
IP Directory Service (IPDS)	Register/de-register IPs (domain, service name, service ID, service description); lookup IPs that match certain criteria; send lists of IPs to other objects (TA)	WP F: Service Interaction (concepts of naming and traders) WP I: Pilot 1 (Application specific part)
TIMER	Part of a host/place; mechanism, that allows agents to set alarms to be called at certain points in time	Technical WP not yet assigned !!!

Special issues

Object owning (to MOW people, Richard Hayton for backup and recovery issues):
Does every object have a dedicated owner (a user or a agent – who is the owner of an agent itself?)? Who is responsible for creating and killing objects? Who is to trace the lifecycle of an object? Who cares for object backup and restore mechanisms (i.e. a task agent might belong to the task agent factory that created it or it might belong to the personal assistant of a specific user)?

State aware objects (to MOW people):

In order to trigger actions by changes that occur to certain objects' attribute values, these objects need to be aware of their state. This should be a general feature of any object (at least data storage objects) and could be implemented using the standard Java pattern of observers and observables.

Control of system resources (to whom it might concern):
At present it is not clear to us which components will be responsible for controlling of host/place resources (i.e. when an agent or agent factory requests the creation of an information space, there must be some component granting a certain amount of system resources for this purpose).

Object authentication (to APM - security model):
In order for the system to work properly we will at least need very basic authentication methods. For a first implementation it will be sufficient, if objects can restrict access to their public interfaces to other objects that identify by their object ID (i.e. an agent wants to write to an information space; the IS manager must know, which objects/agents have access rights, ask the agent for its object ID, check the ID and then grant access).

Controlling of objects in general (to MOW people):
There seems to be the general idea that objects control themselves and are not controlled by any supervisor instance. But an object cannot create/backup/restore itself and their always needs to be mechanisms to negotiate about the use of system resources.

Issues on IS (to IS people):
Is there a need for a IS Factory or who will create it? How does cleaning up of old objects (by agents, by the user) work?

Whom to inform if things change (to whom it might concern):
If objects, that are not the owners of another object have rights to make changes to these objects, then there must be a standard mechanism to keep to owner of the object informed about all these changes.

Access to IS (to Agent and IS people):
Will TA directly communicate with the UA and the IS or will they contact the PA to mediate between these components?

Handling events and the Timer component (to MOW and others):
What standard mechanisms will the MOW support for handling system events (Java exception handling)? Will there be event broadcasting features (i.e. an object can send an event message to all its subscribers)?

Interfaces to information providers (to Service Interaction people):
What will Service Interaction provide? What infrastructure/interfaces must IPs provide to enable agents to use their services/query their information? What is domain specific and what will be provided by the Service Interaction WP?

Agents and their user interfaces (to Autonomous Agent people):
Will agents carry around their user interfaces or store them in IS or will there be libraries of UI for use by any agent?

Stratum: PA 2 – ETEL++ (WP J)

Status

The work on this stratum has only just begun. The engine needed to build the newspaper is under control. The design of the PA in terms of agents is still in its infancy.

Survey due:	December 1997
Requirements due:	February 1998
Design due:	Mai 1998
Interfaces due:	around design time…
Implementations due:	July 1998

Scope

The purpose of ETEL++ is to show the benefits of having mobility to users and data. Its goal is to unveil low-level features (mainly MOW and Service Deployment) via the manipulations of an electronic newspaper across a network.

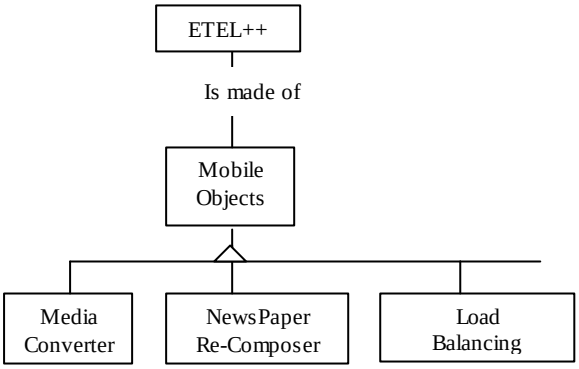
- ETEL++ will focus on:
  - Benefits of using mobility for data/objects and users,
  - Impact of Geography on delivered data,
  - Impact of user preferences on delivered data,
  - Quality of service by enforcing good response times. Means: having the data following the user.
- ETEL' s secondary focus is on the multi-terminal aspect. We plan to add multi-terminal support to our mobile PA.

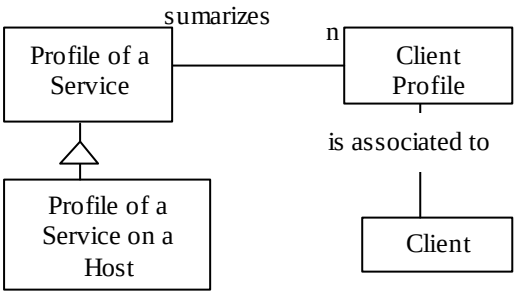
Issues

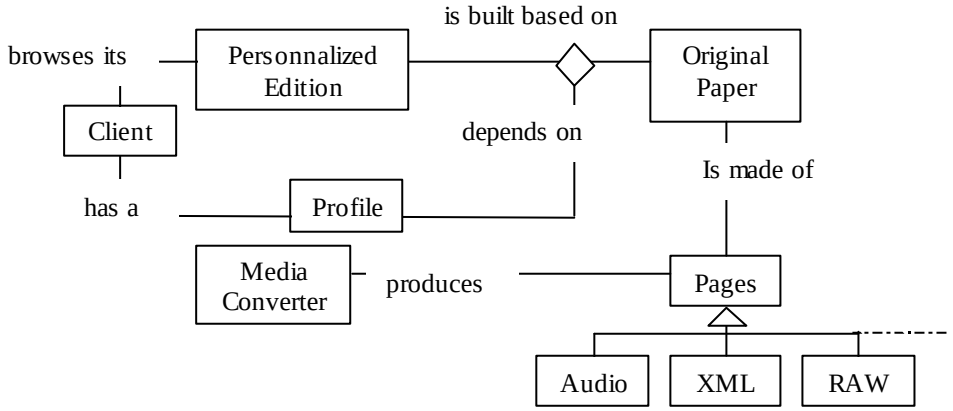
The following issues are still to be decided with respect to this stratum:

- **Connections with SI and AA:** The way ETEL++ can cooperate with AA and SI is still very vague. We have been talking about the following idea: in addition to the profiles attached to a user, there is a shared area in which every service can publish "keywords" that correspond to the user' s interests (user explicit agreement needed, however). These keywords can be latter accessed by an agent that will try to find other existing services providing data that match the published keywords.
- **UA:** Using UA as a provider of functions that takes care of the translation from one media to another is still an issue.

OMT Models







Components

ETEL++ is made of several objects that are potentially mobile. Some of the main objects so far identified (and presented in this short document) are:

- **Media Converter:** the goal of this object is to offer support for converting the data that compose an edition from one media to another. Typically, it consumes RAW data given by the journalists and produces various versions, identical in their contents but different in their type. It uses features provided by UA (see section Issue).
- **NewsPaper Re-Composer:** This object actually creates the personalised editions that fit the users requirements. It uses the profiles of users, the type of device they use, and recompose the newspaper by picking the appropriate elements from the pool of data available in multiple media (more on this given by the Figure below).
- **Load-Balancing:** This object deals with the issues related to L.B. The goal is to maximise ETEL++' s efficiency.
- Among the remaining objects not represented, we can cite:

- **Profile Manipulator:** This object interacts with the user and offers a set of features to maintain the part of the profile relevant to ETEL++.
- **User action analyser:** This object tries to get feedback from the way the users actually browse their edition. The ultimate goal is to enhance the delivery of data by refining the knowledge ETEL++ has about users.
- **Connection to the user:** This object cooperates tightly with UA in order to allow a user to have access to ETEL++.

The following figure zooms inside ETEL. More precisely, here are presented some of the main blocks implementing the engine that creates the newspaper.

The following Figure shows how ETEL computes a profile for itself, referred to "profile of a service"(again, in this case ETEL). Each of ETEL' s client has a profile that reveals how he uses ETEL. Given this set of profiles, ETEL is able to compute a "summary" that reflects the global utilisation of the service (e.g., the most popular pages, the most busy time of the day w.r.t user' s, …). This knowledge participates to the decisions taken in accordance to WP-G tools.

Functions

The ETEL++ application will provide:
- Facilities for users to create/update/maintain/… a profile describing their personal interests.
- Personalised edition for each individual.
- On-line and Off-line access to the newspaper
- What is delivered to a user is impacted by:
  - the user' s profile,
  - the device the user uses,
  - the user' s physical location,
  - possibly by the actual performance of the "elements" between the service provider and the user
- Automatic conversion of data in order to cope with different medias.

API

The API for ETEL++ is not yet precisely defined. We think about providing interfaces like the one given below, which is an example that might become completely obsolete in the future.

```
public XMLarticle getArticle(user _user, idInfo _idInfo, keywordList _keywordList) {
        // this interface returns an XML data structure containing an article.
        // Say this article is the first of the list of articles that match 100%.
        // The article is composed of a TITLE, HEADER, BODY.
        // The article has been retrieved from ETEL database and matches the keywords
        // given in the call. The caller can be any agent that exists inside the context of FollowMe.
        // To access this interface, the agent actually works on behalf of a well know and identified user.
}
```

Comments on API

ETEL will provide some interfaces that will be publicly available for accessing a restricted set of the information that compose a newspaper. Such interfaces can be used by AA and SI to gather data that are relevant to the goal the user assigned to the agents. No much work done to define such API, no deep thoughts about who can access with which rights...

Requirements on other strata

The following requirements are made by this stratum on other strata:

- **MOW:** Mandatory needs for real PUSH in addition to the traditional PULL communication mode.
- **MOW:** Mandatory needs support for Bulk-Transfer. Real-time issues have not to be considered. (i.e., no need for real-time streaming mpeg onto a screen. “a page at a time” transfer mode not entirely satisfactory.)
- **IS:** Control over actual location of physical “instances” of the logical object IS stores.
- **PP:** easy access to the Profile-Factory.
- **UA:** Support for rules that are triggered when UA adapts the layout of what has to be displayed to the actual device. The rules are provided by ETEL++.

Other comments

No other comments.