Follow Me Project

Work Package L

Project Management

Final Report

ID:	DL4	Date:	28.3.99
Author(s):	M. Breu, Fast e.V. & the FollowMe Team	Status:	draft
Reviewer(s):	FollowMe Board	Distribution:	project and CEC

Change History

Document Code	Change Description	Author	Date
dl3.doc	First version, with raw input from partners	M. Breu	3.3.99
dl3.doc	First complete version	M. Breu	28.3.99
dl4.doc	Corrected Internal Renumbering according to TA, incorporated final remarks from partners	M. Breu	1.4.99

Contents

1 Executive Summary

Mobility and communication are an essential property of the modern information society. Business and personal live is changing under the emerging influence of the internet and the world wide web.

Many users are restricted to exploit the full potential of the internet as they can only perform actions while logged on. They have no ability to perform searches or interact with other services while disconnected. In addition, all information must be maintained and transported in the users' personal computer. This restricts the amount of data they can store and makes sharing and updating almost impossible.

FollowMe provides the mechanisms and infrastructure to support mobile users, their data and their objectives. It employs mobile agent technology to achieve goals without the need to constantly direct decisions. This allows users to have tasks performed while disconnected. It enables services such as stock prices and event announcements to be monitored or even have a tailored version of the daily newspaper produced.

Underpinning these agents, it the users' own mobile information space. Users are able to gain access from any location and are presented with a seamless view of their data. The FollowMe framework provides the storage and management functions which enables the position of disparate data to be transparent to the user. Information is kept securely on the network and can be automatically re-distributed to reflect the mobility and working partners of the owner.

The ability to organise, search, monitor, access and share information, irrespective of location is clearly an extremely powerful facility. This combined with agent technology makes the range of FollowMe applications potentially huge. Applications could range from distributed databases such as medical records, through to domestic goods and services such as travel agents and financial information. Possible users are mobile executives, companies with network computers and domestic users with set-top boxes.

The concepts and the infrastructure developed through the FollowMe project has manifested in the following results:

1. the component **architecture** for distributed mobile applications that includes object mobility and distribution control, a framework for autonomous agent, and user access facilities. The architecture is presented as a catalogue of design patterns for distributed applications in project deliverable DA1.3.
2. the **infrastructure prototype,** providing a complete basic architectural component framework for FollowMe application, to be integrated into marketable products for servicing mobile agents. The SW components of the infrastructure prototype are available together with the respective documentation.
3. two **pilot application** that demonstrate the architecture and the components.
4. a set of **public reports** on the architecture, user needs, implementation guide, and the pilots. An overview is presented in this document.

This document gives an overview to the results and describes the major design decisions taken during the progress of the project. It finally gives also an overview to the exploitation activities of each partner and the formal status of the deliverables.

2 Project Results

The FollowMe architectural framework organises the components into a structure that is represented by Fig. 1:

- Two basic layers that support elementary services, as mobility, distribution and persistency
- A set of components providing higher level services to the applications, as the agent framework, service deployment facilities and user access through a broad variety of device types.
- Components implementing application logic, built on top of this service infrastructure

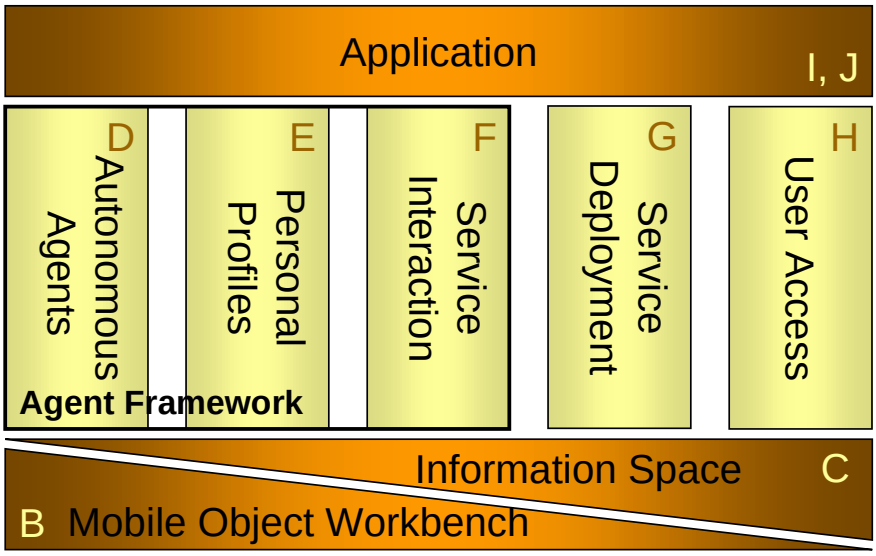


Fig. 1 *Architectural Framework Components*

The *mobile object workbench* (MOW) provides the basic platform for mobility. A mobile object resides at a "place". A place provides the minimal environment for a mobile object to live in and an infrastructure to move to other places. It also provides mechanisms for location transparent references for mobile objects. A place itself can be mobile, e.g. when it resides on a notebook, it can disconnect from the internet and reconnect on another geographical location. This is a typical case for a mobile user.

The *Information Space* component provides services to persistently store objects and retrieve them in a distributed environment. The Information Space may provide transparent replication services on objects, in order to optimise access from different locations. The information space encapsulates store and backup facilities to be able to recover mobile agents after a system crash. Both components are described in section 2.2.

During the evolution of the project it turned out quite early that the components *autonomous agents, personal profile*, and *service interaction* has to be treated as one *agent framework*. This framework is presented in section 2.3.

The *autonomous agents* are operating on top of the MOW. Autonomy means that the agent has the decision how to react on external events. The only decision a place can enforce on an agent is its destruction. Agent execute missions defined in scripts. An agent is equipped with a *profile* that holds its characterising data.

Agents interact with each other. The interaction with (non-mobile) services is provided by service interaction interfaces that act as service proxies. These proxies can be contacted by agents to query a service. An agent can also subscribe to a service, in order to be informed when a certain state change occurs.

A special type of agent is a *personal assistant*, which is directly associated to a user. The personal assistant manages a *personal profile* that holds all relevant personal information about the user. The personal profile contains simple items like the user's name and address, but also a diary of its user, i.e. a schedule via which devices a user can be contacted, in order to send him e.g. a fax or an SMS message. The persistent data components of all profiles in the FollowMe environment are implemented in XML, all agents are capable of parsing XML and all instructions on how to interpret XML data stored in the profiles are implemented using a scripting language. The user access provides service to map raw XML data into layouts appropriate for certain devices.

The s*ervice deployment* component (section 2.4) applies data mining techniques on resource consumption measures to optimise the availability of objects on different locations, e.g. by directing the replication service provided by an information space.

Finally the *user access* (section 2.5) allows a mobile agent to contact its user. Communication media could be e.g. a web-browser, a personal digital assistant (PDA), a fax, or phone. Two types of devices are distinguished:

- Passive devices: These devices will only support system output (i.e. sending a fax or a phone message), thus the agent is able to send the user regularly reports or just the request to contact him, because the agent needs user interaction. These devices are used for off-line communication with the user.
- Interactive devices (i.e. Web browsers): These devices will be used to interact with FollowMe agents, e.g. by HTML forms. Through these devices users can typically command its agents.

FollowMe has developed a set of demonstrators, which are presented in the sections 2.6 and 2.7: In co-operation with the Bavarian Citizen network ("Bürgernetze") two pilot applications were developed, a regional events notification service and a portfolio management and alarm system. Both demonstrate features for individualisation of web based services and an active presence in the internet by agent technology.

The second demonstrator, developed in co-operation with the French newspaper France Ouest, implements an individually tailorable newspaper service.

The next section presents this architecture and the interplay of the components in more detail.

2.1 The FollowMe Architecture

2.1.1 Background

FollowMe provides a component architecture for the development of distributed mobile applications. A significant part of this architecture is associated with supporting the mobile user. A mobile user is a user that is not permanently connected to the Internet and does not have a fixed home machine or location. Mobile users wish tasks to be performed while they are disconnected from the Internet and wish results to be delivered to whatever device they have available when they (re) connect. Ideally they wish to use commonly available facilities such as faxes and mobile telephones, as well as workstations and laptops, to interact with Internet services. Moreover should services require further user actions while the user is unavailable the user would like tasks to proceed autonomously by providing "sensible" defaults and taking "sensible" decisions in the absence of the user.

The purpose of the Architecture work-package in the FollowMe project (WP-A) was two-fold:-

- at the outset of the project to scope the division of work between partners into basic components with well defined interfaces and relationships; in this respect the architecture description is a reflection of the planning work done by the partners during the project proposal stage, and during the first phase of design
- as the project progressed, the architecture description was updated and refined in the light of ongoing design reviews to become a repository of design expertise assembled by the project, which we felt would be of value of others following in our steps. In doing so we looked for the re-usable elements of our overall system design.

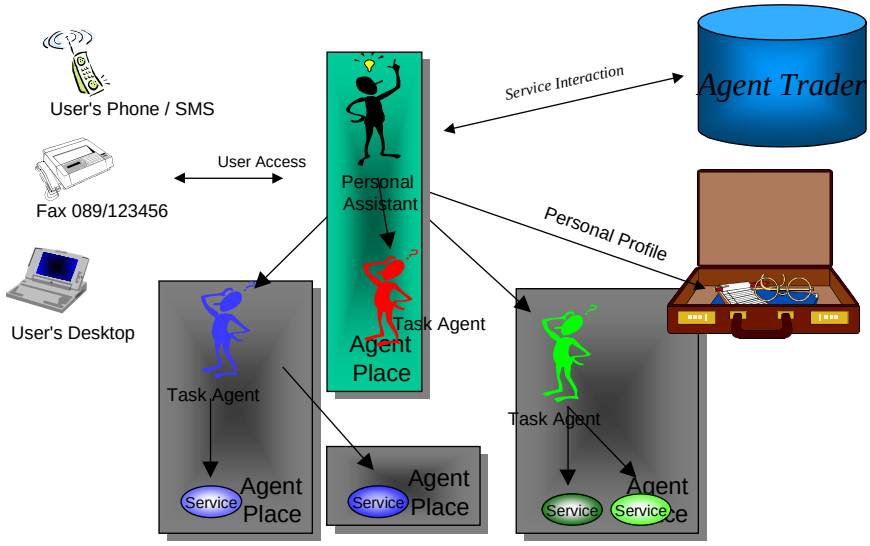
The Architecture deliverable is a report containing:

- a collection of concepts (i.e., an ontology) that is used in the description of FollowMe components and applications in other deliverables. The ontology of FollowMe was developed from the ISO/ITU Reference Model for Open Distributed Processing, The Object Management Group's CORBA specifications and Javasoft's Java specifications. The ontology includes the overall system model
- a discussion of the need for loose coupling at many levels between components in mobile agent systems and guidelines for achieving loose coupling through a consistent use of encapsulated object-based approaches to implementation.
- A series of patterns abstracting out common design principles that feature in the FollowMe components. A pattern is an approach to solving a specific kind of recurring problem in a specific context. Patterns are not simply pieces of code – they are fragments of analysis and design. Patterns are increasingly used in object-oriented systems to unify the approaches to design across levels of detail.

FollowMe augments the ODP / CORBA of distributed computing with two powerful new paradigms for information systems. The first paradigm is object mobility. Object mobility allows programmers to produce systems in which objects with state can move around a computer network. The second paradigm is agency. Agency means that programs act autonomously on behalf of users and so may make decisions when the user is not connected. Indeed one major decision that often needs to be made is how to connect to the user at a specific time to deliver a specific piece of information.

2.1.2 System Architecture

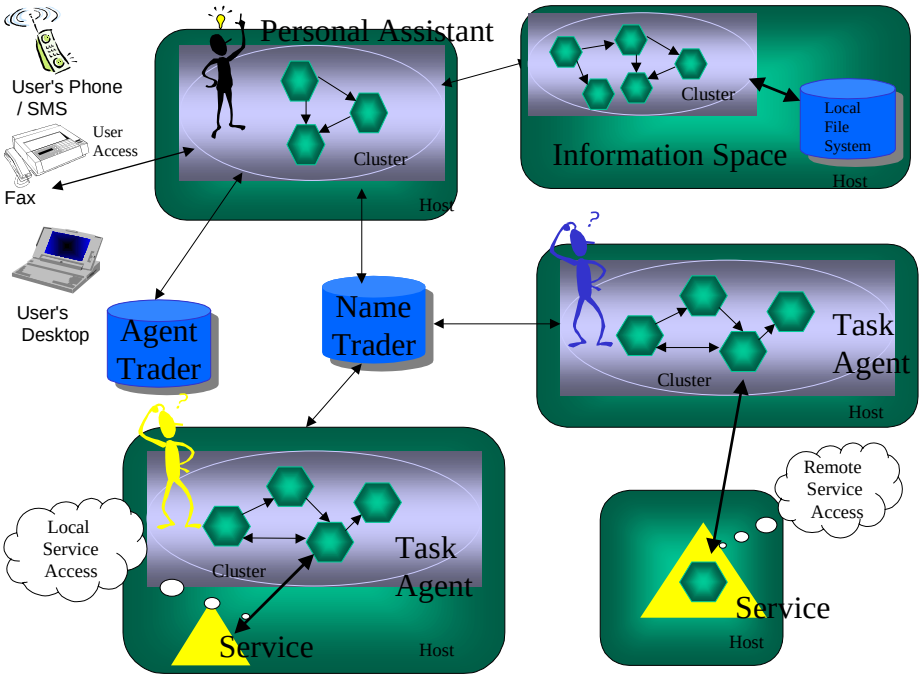
The FollowMe system architecture is best described using the scenario illustrated in the figure below.



A user wishes to locate some information on the Internet. The user does not wish to perform the search himself by browsing the web but rather wishes an agent to perform the search for him and contact him with the information when the search is complete. The user connects to his “personal assistant” agent to create a search task. The personal assistant contacts an agent trader to find appropriate search agents to perform the search task (these agents are called task agents). Generally the user will provide some specific information and the personal assistant will fill in defaults using the users “personal profile” which is a collection of information about the users preferences, contact addresses etc. The personal assistant will then act as contact point for the task agents to supply any additional information when the user logs off. The user logs off and the task agents are deployed to perform the search. The task agents may search by remotely contacting services provided at various host sites or they may move to remote sites and perform searches directly on the host, or generally from a geographically or computationally favoured location. Once the task agents wish to report back they contact the personal assistant. The assistant queries the personal profile data to discover how the user wishes to have the data delivered. This may vary according to the nature of the data (hypertext, simple text, etc.), the volume of the data, the time of day, the preference structure set up by the user between modes of available delivery etc. The personal assistant then uses the “user access” component of the agent framework to deliver the data to an appropriate choice of device in an appropriate rendering. The user access component allows the delivery of data to the user to be substantial independent of the device it is delivered on. So, for example, the use may receive a short telephone message to inform him that interactively browsable data is available the next time he has access to a browser. The message might also contain a short summary or key piece of information (e.g. the message may contain the best match to search criteria and indicate that all matches are available for viewing with a web browser when the user can get to an appropriate terminal.

The infrastructure supporting the picture above is portrayed schematically below.

The agent layer is supported by the mobile object workbench and information space infrastructure (MOW/IS). MOW/IS provide a general mobile, distributed programming model. This model implements the idea of “clusters” from ODP to realise the notion of a collocated and uniformly managed group of objects. References between clusters are location independent. All references to agents, traders, etc. are location transparent references. The personal information space is an instance of the general information space infrastructure, which itself is realised as a cluster with a management policy for persistent storage. Agent mobility is built upon the object mobility, which again is realised as a cluster with appropriate management for copying object code and state from one place to another.



2.2 The Mobile Object Workbench and the Information Space

The aim of these tasks was to show how the concepts of component-oriented system could be applied to produce efficient re-configurable, extensible middleware platforms in general and to support intelligent mobile agents and user access components resulting from tasks D and E in particular.

FlexiNet provides both a generic middleware framework and a set of engineering components to populate it. By making appropriate choices of which components are assembled within the framework a variety of different middleware facilities can be achieved including mobile objects, persistent objects, secure objects and transactional objects.

FlexiNet represents an evolutionary step from contemporary CORBA, Java Remote Method Invocation and Enterprise Java Bean (EJB) middleware. It is a full demonstration of the ANSA architectural principles at work.

FlexiNet is particularly suited to applications that are deployed in a variety of different contexts (e.g., on the Internet, in Intranets or Extranets) since its enables the infrastructure to be tailored for the specific needs of each deployment and for inter-operability between application components in different environments. Mobile intelligent agent systems are an important example of the need for these capabilities – agents may migrate between intranets, extranets and Internets with different network security and quality of service characteristics.

FlexiNet is built in Java because it provides platform portability, object-orientation for developer productivity, facilities for dynamic linking components from different sources and reflection and introspection to allow applications to discover and adjust middleware components to fit their needs. Java has an increasing role as the language of choice for distributed applications development.

2.2.1 Capabilities

The heart of FlexiNet is a remote procedure call (RPC) system for remote invocation of services implemented as interfaces on Java objects. In this respect FlexiNet is similar to CORBA and RMI. FlexiNet allows both object-by-value and interface-by-reference parameter passing. It uses introspection and dynamic code generation and linking in place of off-line stub generation.

The implementation of the RPC infrastructure is based in terms of sets of components called *binders*, which implement RPC semantics over underlying transport systems. In addition to FlexiNet specific binders, there is an IIOP binder that implements the OMG IIOP standard protocol to enable inter-working with CORBA clients and servers.

A simple Trader is provided to enable FlexiNet clients to locate FlexiNet servers.

A mobile object workbench sub-system is provided. This allows “clusters” of Java objects to move from one “place” (computer) to another. This sub-system was developed to enable support of mobile intelligent agents in the FollowMe project.

A persistent information space sub-system is provided. This allows clusters of Java objects to be removed from the execution environment and placed in storage until next activated. The persistent information space provides an “object file system” for the mobile agents of the FollowMe project.

A basic object location service is provided, principally to support tracking of mobile objects and persistent objects.

A class repository sub-system is provided which enables classes to be dynamically linked across networks and locally cached, with advanced facilities to manage name clashes that might arise in federated environments. This allows mobile agents to “drag” their infrastructure with them as the transit networks.

A visual application builder and associated infrastructure components are provided to enable transactional Enterprise Java Beans to operate in the FlexiNet environment. These facilities enable transactions to be used to enable objects to manage concurrent access and recover from failures transparently. The FlexiNet transaction infrastructure is more powerful than current EJB implementations because it includes the transaction facilities within the infrastructure rather than offloading transaction control to a database management system.

Security is provided primarily in the form of an implementation of the SSL protocol as an additional binder. This can be used both for secure access control and secure communication.

Strong encapsulation is an intrinsic feature of the FlexiNet framework. In addition, there is a design for secure carriage of mobile objects across trust boundaries.

Basic support for high availability is provided as a binder for a reliable multicast protocol between members of an object group. This implementation is currently incomplete, but includes sufficient components to give a proof of concept.

Finally a declarative configuration tool for assembling components into binders using the FlexiNet framework called “Blueprints” is provided.

2.2.2 Why a new platform?

The FlexiNet platform grew out of dissatisfaction with industrial middleware platforms and mobile object infrastructures as possible foundations for FollowMe.

We therefore begin the description of FlexiNet with a review of the limitations of existing middleware.

Generally, research middleware platforms provide application programmers with facilities for just one model for distributed programming, for example remote procedure call, or message passing or process groups. Consequently, compact, efficient and scalable implementations are often achieved. By contrast, industrial middleware platforms address the need for a ubiquitous infrastructure and provide an integrated set of capabilities including, for example, transactions, replication, authentication, privacy, auditing and others. The result is typically monolithic, inefficient and complex.

Since different applications require different combinations of middleware features, a compositional approach in which only the middleware services needed by an application need be made available is appropriate. In CORBA, for example, a set of nested choices is offered as CORBA “Object Services”. Each Object Service extends the core Object Request Broker with additional capabilities such as persistence and transactions. The benefit of the CORBA framework of Object Services is that it is comprehensive. The disadvantage is that it is unnecessarily rigid because the order in which capabilities have to be assembled is fixed and this rules out some implementation choices. Moreover, the core Object Request Broker is required to contain support for the data structures and protocols required by each Object Service whether it is used or not. Thus, in addition to causing bloat and inefficiency in the implementation, developers are forced to manage more capabilities than they necessarily need in any particular situation.

A further aspect of inflexibility comes from the use of stubs in Object Request Brokers to provide access transparent invocation. A stub converts an invocation into an untyped byte array representation to be passed on to a communications layer in the case of a remote service. Discarding language level typing and introspection facilities in this way, makes it hard to provide developer-written protocols and mechanisms that can coexist with standard stubs. Specifically, it can be difficult to tie together application level events, middleware events and communications events. For example, The Iona ORBIX Object Request Broker provides filters and transformers as a means to modify how communication events are handled. However, no conventions or data structures are defined for relating filter events (i.e., pre-stub events) to transformer events (i.e., post-stub events). Behaviour at the filter level is modelled by CORBA type codes and dynamic type checking of these has to be managed by the developer rather than delegated to the programming language.

Inherent in the design of distributed systems is the need to make appropriate trade-offs between the competing goals of abstraction and application control. Abstraction in

middleware is generally associated with distribution transparency. Abstraction/transparency makes life easier for developers by hiding the engineering details of interaction models behind a generic invocation interface (e.g., method invocation). In essence, the infrastructure manages distribution. Application control, by contrast, allows developers to optimise the infrastructure when it is beneficial to do so, for example by providing heuristics for error cases. Control requires that implementation aspects of a distribution transparency should be exposed. Unfortunately current systems either impose a 'one size for all' transparency or expose the low level 'systems' mechanisms in all their complexity.

Because of the issues outlined above, mobile agent infrastructure developers have been faced either with building their infrastructure as a stand-alone technology or as a layer above middleware. This has led to poor integration between agent facilities and distributed computing facilities, with duplication of function and impoverished computational models for agents (e.g., no type checking, no selective transparency, scalability limits).

2.2.3 FlexiNet Architecture

FlexiNet was designed 'from the ground up' to address all of the concerns described in the previous section. Whilst we wished to support interoperability with existing systems, and conform to, or extend, existing standards, these were secondary goals. FlexiNet grew out of dissatisfaction with current offerings, and a clean slate provided the opportunity to build a coherent architecture according to ANSA principles.

FlexiNet was developed to be *Java specific*. This allowed us to leverage the facilities provided by Java to provide a clean architecture and straightforward API. In particular facilities such as *objects by value*, *subclassing of arguments* and *dynamic late linking* are used in our architecture to simplify many aspects of the design, and to help make FlexiNet extensible. This choice was compatible with the World-Wide Web orientation of the FollowMe project and the goals of the other project partners.

FlexiNet was developed on NT 4.0 and Solaris using Sun's JDK1.1.7.

Where possible we reused existing concepts and principles in the design of FlexiNet:

ODP Reference Model:
The computational model from RM-ODP was uses as a basis for the programmer interface. Additionally, a number of RM-ODP engineering model concepts were reflected into the computational model. The ODP notion of interfaces and objects was used, and ODP clusters were implemented to provide encapsulation.

Java Language
We attempted to keep the FlexiNet API and remote object semantics as close to normal Java as possible. In particular, we use Java interface classes rather than IDL files, as this is more natural for a Java programmer.

Build for Change
FlexiNet was designed to be constantly upgraded and changed. We attempted to minimise the amount of 'global knowledge' and interdependencies between components, so that parts could be replaced, or new components added.

Multiple Everything
FlexiNet was designed to be component based, and to allow more than one instance of any component to co-exist. This is important, for example if a client has to speak two versions of a protocol. FlexiNet make very little use of 'static' data, as this is intrinsically restrictive.

Reflective Implementation
FlexiNet attempts to use its own mechanisms internally wherever possible. For example a FlexiNet name, passed to identify an interface, is an ordinary object, and treated as such when serialised, deserialised or otherwise manipulated. This approach allows us to change the specification of one component (for example a name) with minimum impact on other components.

2.2.4 Selective Transparency

Since FlexiNet remote method invocation has similar semantics to local method invocation, we have access transparency at the lowest level. This uniformity helps keep application code separated from 'systems' code, making it easier to move applications from one environment to another. To take control, the application programmer can inject particular mechanisms, both at runtime using an explicit binding facility and at design time by controlling the mixture of protocols and transparency components used. Resources can be managed by restricting the allocation policies for, and sizes of, resource pools assigned to selected components. This capability is described as 'selective transparency', since the developer can choose how strongly system components are tied to (and visible to) application components. Binding decisions can be taken directly by system components or handed off to third parties where this is appropriate. The former is appropriate for autonomous systems, the latter for managed infrastructures (e.g., a trusted computing base).

The RM-ODP framework identifies nine distribution transparencies. Of these, only two, *Access* and *Location* relate directly to remote invocation. In addition to these, FlexiNet protocols may provide varying degrees of *Failure*, *Replication* and *Security* transparency via meta-objects in the protocol stack. The remaining RM-ODP transparencies, namely, *Migration*, *Relocation*, *Persistence* and *Transaction*, cannot be tackled in this way. Instead, they require some notion of *encapsulation*, whereby all interactions with an object, or group of objects can be monitored and controlled. To achieve this FlexiNet implements the RM-ODP notion of a *cluster*. A cluster is the primitive RM-ODP engineering unit of encapsulation.

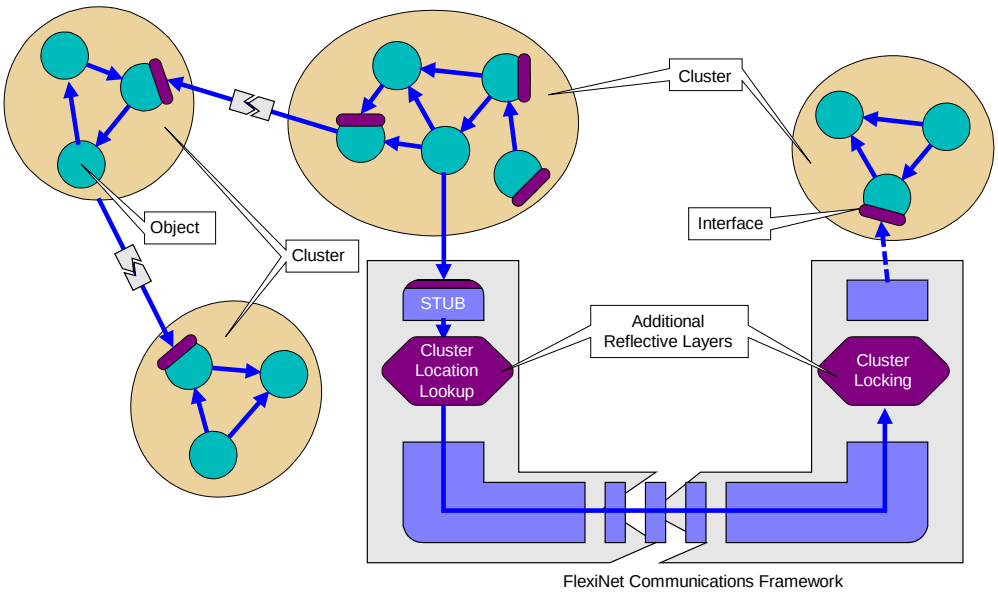


Fig. 2 *Encapsulation Using FlexiNet*

Clusters are illustrated in Fig. 2. All externally referenced interfaces in a cluster are accessed via a FlexiNet protocol stack. We arrange that when a thread in one cluster invokes a method in another cluster, we de-couple the threads so that the callee and caller cannot adversely

affect one another by blocking or thread termination. Additionally each cluster is effectively given a separate Java security manager, and class loader. Thus each cluster becomes a ‘virtual process’ that is de-coupled from all other clusters in terms of name spaces privileges, code base and management. Clusters cannot examine the internals of each other, nor may arbitrary methods on objects in one cluster be called from another without mediation by the reflective layers in the protocol stack. These co-operate with a distinguished cluster management interface associated with the cluster to provide whatever kind of distribution transparency is appropriate. In the case of a mobile object the cluster manager arranges for the atomic transfer of the cluster from one location to another, and the protocol stacks include some kind of relocation function to track objects as they move.

2.2.5 Transactions

The FlexiNet transactional framework is designed to use the EJB container model, rather than the cluster model. The motivation for this is primarily industry conformance. EJBs facilitate the reuse of standard third party components; if our transactional framework were built on a proprietary abstraction, then this advantage would be lost. Now that the EJB specification is more mature, it would be an interesting exercise to construct an EJB compliant cluster. However, if a bean were to take advantage of the less restricted environment that this would afford, it would cease to be portable to other EJB implementations.

2.2.6 Mobile Objects

Mobile Objects are an abstraction designed to support code and data mobility. In particular, they allows an executing application to ‘jump’ from one host to another. The mobile object abstractions in FlexiNet are largely transparent, however there are some coding restrictions on mobile objects, and it is not in general possible to migrate an arbitrary application.

Mobile Objects are supported in FlexiNet as a specialisation of the RM-ODP *cluster* abstraction. A cluster is a collection of objects that are managed as a whole. A *mobile cluster* is a collection of objects that may be migrated from one host to another. A cluster may be thought of as a lightweight process and a mobile cluster as a process with the ability to jump from host to host. In practice it is a process-like abstraction that we wish to be mobile, not simply a single object.

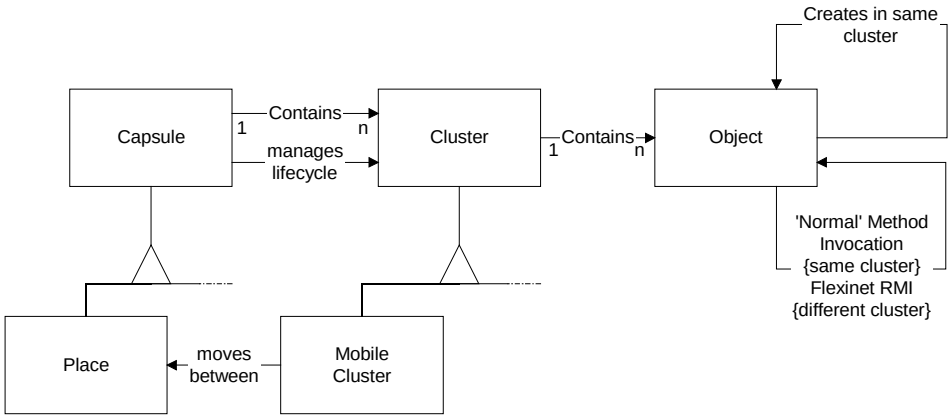


Fig. 3 *Clusters and Capsules*

2.2.7 Persistent Objects and Places

Persistent Objects are a cluster-based abstraction that allows clients to access persistent objects transparently. The persistent object itself implements one or more application specific interfaces. The client is given a reference to these, and may access the object as if it were a ‘normal’ object. The infrastructure actually stores the object on disc, and transparently reads the object in and writes it back before and after each method invocation.

Like mobile objects, persistent objects are supported by a specialisation of the *cluster* abstraction. Persistent clusters are called *storables* and are stored in Capsules called *Stores*. This is analogous to the Mobile Cluster, Place relationship (Fig. 4).

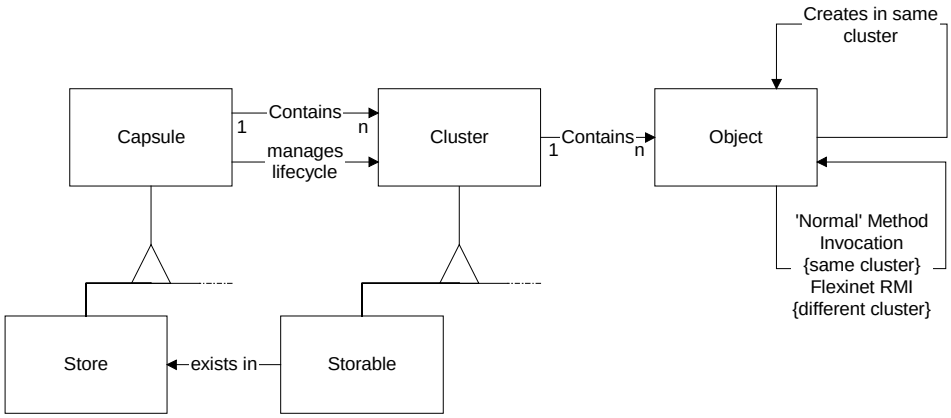


Fig. 4 *Storable Clusters*

2.2.8 Performance

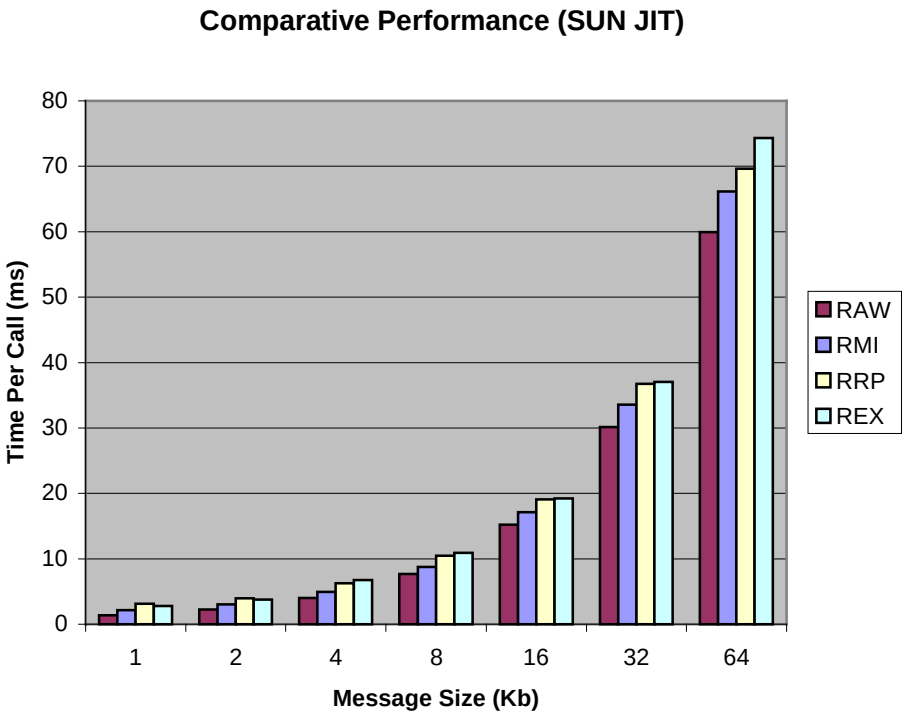
FlexiNet is a component based framework, and the protocols and abstractions that currently populate this framework were designed for modularity and reuse, rather than performance. For example, all the layers in a typical remote method invocation stack could be implemented as one module, in order to increase performance.

However, FlexiNet is fully resource controlled, and uses pools for resources such as buffers and threads, drawing on our earlier experience in C++ ORBs for real-time multi-media applications. The modularity is an advantage here, as different pool management policies may be ‘slotted in’ in order to trade off performance against resource usage.

Performance is notoriously difficult to measure. Many factors, such as a protocol’s support for failure and simultaneous access, in addition to the actual reliability of the connection, and number of simultaneous clients will all effect the achieved performance. However, to give a simple indication of the ‘raw’ performance of FlexiNet compared to other protocols we ran a series of simple tests between two machines. For these tests, four protocols were used; Sun’s RMI, FlexiNet using a TCP based protocol (RRP), FlexiNet using a UDP based protocol (REX) and a ‘raw’ TCP protocol. This latter protocol acts an indication of the inherent costs of a remote call. It uses simple Java TCP sockets and has a single threaded client and server. For all protocols, an array of bytes was used as the only argument to an invocation, and the invocation returned a void result. The results of running the protocols with different JVMs and different message sizes are shown in Fig. 5. For the record, the machines used were Pentium Pro 200s, running NT Server 4 over a 10Mbit Ethernet. To reduce the effect of class loading, compiling and TCP flow control, 100 invocations were made on each connection prior to measuring the performance.

From the graphs the following points can be noted:

- The network time dominates the total time for all calls. None of the protocols is appreciably slower than a raw socket connection.
- For large messages, the UDP protocol was slower, this is due to the need to perform UDP fragmentation. This was particularly pronounced in the Microsoft JVM/JIT.
- The FlexiNet TCP protocol is around 15-20% slower than RMI on Sun's JVM/JIT, but around 10% faster on Microsoft's JVM/JIT. This suggests that Sun's RMI and JVM have been optimised to run well with each other.



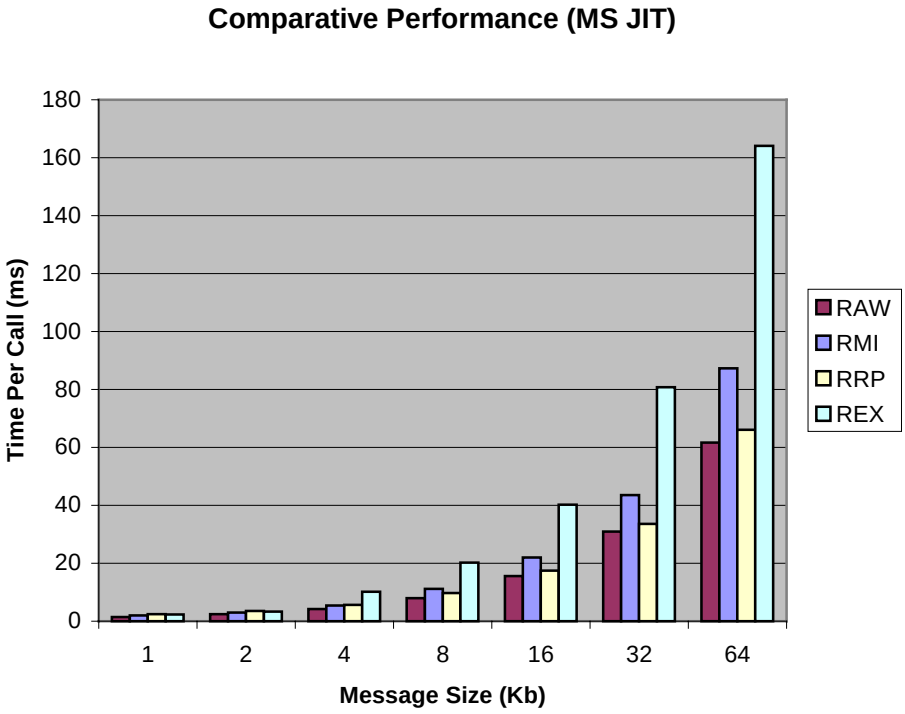


Fig. 5 *Comparative Protocol Performance*

Unlike RMI, which relies on native methods and stub compilers in order to function, FlexiNet remains 100% pure Java, with no external tools required. This makes it highly portable across Java releases and JVM implementations. Future JIT or JVM performance increases should be fully reflected in FlexiNet's performance.

2.3 Agent Framework

The Agent Framework provides a set of programming interfaces and tools for service providers and end-user programmers to construct distributed agent-based applications.

2.3.1 Autonomous Agents: personal assistants and task agents

What is an autonomous agent? Unfortunately, you will find there are as many answers to this question as there are computer scientists. It seems that computational agency is not a single technology or pattern, however almost all definitions agree that an agent is something that acts on behalf of a user to carry out some useful task with some varying degree of autonomy. Instead of asking what an agent is, we can ask "how might we use them?", and ultimately, "what are their advantages?".

One of the primary motivations for this work is mobility. Mobility of users, mobility of programs, and mobility of data. With everything on the move, how do you tie everything together? The user's first-contact with the agent framework is the personal assistant, or PA. If you are on the move, disconnected from the network, your PA will pick up your messages and keep you informed. Your PA is the hub which all your remotely running agents keep in contact with. Your PA keeps track of your information so that you or your agents can access

this data on demand. Your PA is the mission control which lets you create, monitor and sometimes kill your own agents.

Of the software delivered as part of this core package, the PA is primarily a programmatic object which can be customised to fit in with any particular application. However, for general purpose use the PA may be viewed with a stand-alone user interface based on the latest Java Swing technology. This interface offers you three views of the system.

- Location transparency for information objects is provided by the information spaces work-package. The PA lets you browse your information space, so you may, for example, access reports sent back to you by agents 'in the field'. Your information space is also the repository for your own personal profile and personal diary, your 'filofax' in the agent world.
- You may access services and agents designed to use them through the Trader. The PA lets you explore the available services and the contexts they reside in as though they were directories on a normal file-system. New task agents may be launched, simply by double-clicking on the required mission.
- The PA monitors any agents that you have previously launched, and presents this information to the user as an activity list. Simple house-keeping is possible; you may kill agents selectively, or you may even talk to running agents by clicking on their name, which brings up their 'contact' window.

The definition of what an agent does is encapsulated within the idea of a 'mission'. The mission declares all the objects and classes required by the mission. The format for missions is the eXtensible Markup Language, XML which has been adopted across FollowMe. Looking somewhat like HTML, the mission contains a number of mission components, each describing the content of a single object required by the mission. Some of these components may be passive data objects, while others may define active behaviour.

Complex agents may be custom-built by a particular service provider to suit their own services, in which case Java may be the most appropriate implementation language for these active objects. However, part of the goal of the agent framework is to empower end-users to write their own agents for their own purposes. Pre-packaged agents are good for the casual user, but more experienced users will demand more flexibility. The aim was to provide a simple programming interface that hides the underlying complexity, but none of the functionality, of the underlying mobile object workbench. Our solution is to allow agents to be programmed using the JavaScript language, which is accessible to a wide audience of end-user programmers. JavaScript commands may be written directly into the mission, and with no need to compile they may be run and tested instantly. This vision of agency makes the user the main focus of the agent framework rather than the service provider.

The agent framework provides a range of software components which can be included as part of a mission. The most important of these components are the XML support classes which provide the script-writer with full access to any XML text, including the mission itself, as though it were an object (a Document Object Model), and Document objects which may be used to build simple user interfaces (such as the agent 'contact screen') which work with the user-access work-package. With a powerful set of components, the agent programmer may regard the JavaScript as little more than 'component glue'. This DIY approach to agent construction allows users to build agents as simply as a child might build a lego toy.

2.3.2 Personal Profiles: profile and diary objects

Many services we use require a core set of information, personal information about who we are, where we live, and how we can be contacted. All too often we find ourselves entering the same information over and over again. If only that information were stored in some common and easily accessible place. In FollowMe, this personal data is stored in your personal profile located in your private information space. Your profile must be private because it contains sensitive personal information, only agents acting with your authority and with a reference to your information space can access this data and supply it to services as required.

Because the information in the profile object is primarily for sharing, we use XML as a platform independent means of sharing this structured data. Content markup languages like XML allow us to describe the content of an object in terms of its significant conceptual entities; which is distinct from the low-level, platform dependent object serializations used to transport agents from place to place.

The domain models for personal profiles and the personal diary were based on the vCard (used in Netscape Communicator) and vCalendar specifications, but are based on open standards using the eXtensible Markup Language, XML. The translation into XML followed the simple guideline that each element should be available in either unstructured human readable, or structured machine readable formats, or both. The unstructured form of an address could be printed directly on a label for example, whereas the structured machine readable version would be broken down into separate address fields, making elements like the post-code easy to extract and check.

The domain model for a personal profile contains the following elements:

- Identification properties (who am I?)
- Addressing properties (where do I live?)
- Communication properties (what's my telephone number?)
- Organisational properties (what is my job?)

Whereas the personal profile is essentially a passive information object, the personal diary is altogether different, able to raise events which rouse agents into action. The diary is composed of the following time-based elements:

- Journal - a historical record of time-stamped occurrences.
- Alarms - a single or repeating occurrence that raises a diary event.
- Events - an occurrence of a finite duration that may be associated with an alarm.
- To-Do list - an action to be performed in the future.

The personal diary provides the mechanism by which a personal assistant may relay any communications it receives, to a mobile user. If you know you will be driving on the motorway at a certain time you may want to receive traffic reports from a traffic monitoring service. Your agent is dispatched to the traffic service and you make a note of your portable messager number as a diary event along with the times you may be on the road.

Separate diaries may also be carried around by individual agents. The problems with the millenium bug pale into insignificance beside the problem of getting a mobile diary to work reliably, moving between time zones onto computers with unsynchronised clocks, without

ever dropping a scheduled event. Many long-running agents do not need to be active all the time. A diary alarm can be used to wake up the agent at the specified times.

The personal profile and diary together make up a kind of electronic ‘filofax’. Used in conjunction with the personal assistant and task-agents, they enable many interesting behaviours.

2.3.3 Service Interaction: trading & service profiles

Looking at the way agents work tells only half the story. In order for agents to be effective, they must have some way to interact with their world, and they do this via services. The service interaction work-package addresses the problems of identifying, designing and implementing services in relation to the agent framework. The main aim is to provide the basic mechanisms and tools needed by service providers and their clients, supporting the creation of the pilot applications. A secondary aim of this work is to look at additional levels of descriptive meta-data that can be represented at the service interface, and used by clients to understand the service in more depth.

While the underlying mobile object workbench provides a basic naming service for identifying well-known services essential to getting started, service interaction comes into its own when we are able to locate services on a more generic basis. The role of a Trader is to act as a broker for service providers and clients, and the point at which they meet is called the service interface. An interface is an abstraction that describes what kind of service is provided without prescribing exactly how it is done; the implementation. This flexibility allows any number of service providers to ‘tender’ for the same job. The client is able to choose among the service offers by comparing a number of public features, called properties. To ensure that the client is comparing like with like, all offers for a given service are made within a so-called context. It is this context that defines the properties these service offers should display, and is thus distinct from the service itself.

The process of declaring new contexts and defining the properties of a given service offer are simplified by the use of service profiles. The eXtensible Markup Language, XML, provides a simple way to describe all this information to the Trader. The simplest use of the service profile is to list all the contextual properties of that service as a sequence of XML elements. While this information is fine for agents written by the service provider who has full access to the design of the service itself, we want to encourage end-users to explore the full range of available services and to build their own agents to suit their own needs. To enable this kind of activity we need to make much more documentary information, or meta-data, available at the Trader; in effect using it as an interface repository. This information will provide the raw material for the creation of new clients against a service interface.

The main component of this additional meta-data defines the operations available at an interface and the types of values passed between client and service; the service signature. Service interfaces are commonly described using a standard Interface Definition Language (IDL), which provides the benefit of implementation language neutrality. We make use of the significant overlap between XML Document Type Definitions and the type structures typically provided by an IDL to come up with a scheme that fits very naturally into the scripted-XML level of the agent framework.

2.4 Service Deployment

In a global network environment such as the internet, it is difficult to anticipate the demands on a particular service. Usage can often be extremely dynamic, with many unforeseen fluctuations.

The objective of this work package was to exploit the knowledge of user profiles for providing quality of service to users. In particular, we were interested in the design and implementation of a load balancing strategy that relies on the grouping of users according to the users' profiles. A basic infrastructure to support load such balancing strategies was implemented: Tools to monitor the performance of resources, such as CPU load, or network throughput, and data mining facilities to identify groups of common patterns in user profiles.

This section gives an overview of the technical aspects of the Service Deployment work package.

2.4.1 Monitoring Tools

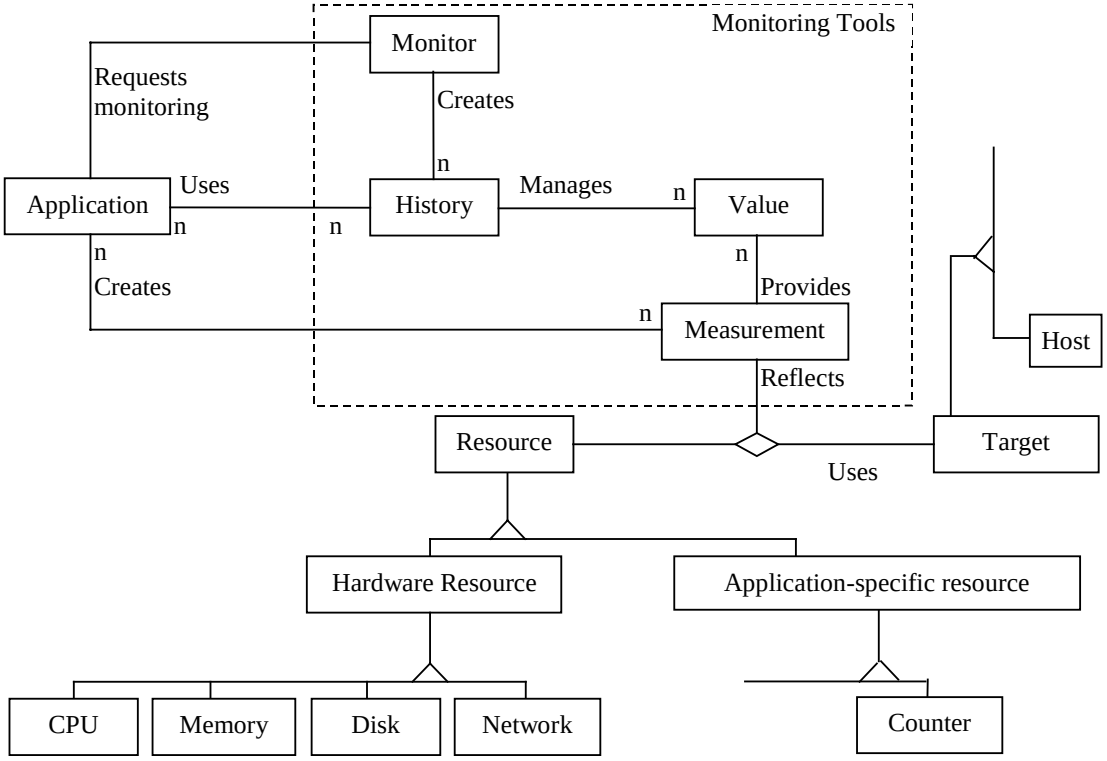


Fig. 6 *Diagram of Classes*

Fig. 6 presents a diagram of the classes used to support the software developed in the Service Deployment work package.

Resources can either be mapped on real hardware resources such as the CPU, or be application defined. The interface *Resource* defined in this work package is defining a single method that must be implemented by any implementation of that interface. This method (Value getValue()) returns upon request the current usage of the resource that is defined. The returned type might be arbitrarily complex.

The class *Target* is used to define the entity that uses a *Resource*. It can for example be a Host, an object or a cluster as defined by APM. Once the pair (*Resource*, *Target*) defined, it is

possible to define the way resource usage must be observed. The classes *BasicMeasurement* (and also *Measurement)* and *Monitor* makes this possible. A Monitor observes (periodically, aperiodically on-request or aperiodically on-threshold) the consumption of a *Resource* by a *Target* by requesting *Measurements*. The class *Measurement* encapsulates *Resource*, and may apply to the value returned by the *Resource* complex post-processing. The *BasicMeasurement* class does not apply any processing to the value obtained from the *Resource*.

Instances of the class *History* receive the values returned by (simple) *BasicMeasurement* or by (more elaborated) *Measurement*. A *History* can be viewed as a buffer in which individual measurements are appended, and on which some processing can further be applied. While *Measurement* enables a first phase of consumption processing, this processing is applied on **individual** values. The processing that may be applied on *Histories* concerns in contrast a **collection of individual values**, thereby enabling the use of aggregate operators such as Average. The post-processing that may be applied on values stored in *Histories* refers to the Class *Filter*. We have defined several built-in *Filters* like Interpolation, … The values are accumulated in *Histories* by *Monitors*. There is typically a single *Monitor* on each machine. A *Monitor* manages all the *Histories* that are currently active in a machine. *Monitors* create *Histories* during their initial interaction with the applications. Once created, the *Histories* are kept up-to-dates by the relevant *Monitor* thanks to the parameters set at creation time (like the observation frequency, …). Applications directly interact with *Histories* once created.

There are two basic ways to use *Histories* of measurements. First, they may be used in Request-based mode. In this case, an application gets values upon explicit requests. A typical request needs the resource-name and a time-range as arguments, and may return for example the average load that is placed on that resource during the specified period. Second, they may be used in Notification-based mode. In this case, an application gets values when an application-defined condition becomes true. To use a history in this mode, the application has to specify the name of the resource, a time range, and a condition. For example, the application can ask to be notified when the average load placed on that resource for the period becomes greater than a specific threshold value. *Filters* and *Histories* both provide a Request-based mode. This mode is described in a common interface called *View*.

2.4.2 Data-Mining

Our algorithm is based on associative data mining. Briefly stated associative data mining computes a set of inference rules among items stored in a database of transactions of itemset. An inference rule of the form *I -> J* means that most transactions containing the itemset *I* also contain the itemset *J*. In general, an inference rule is associated with its *support*, which gives the number of transactions verifying the rule, and its *confidence*, which gives the probability with which a transaction containing *I* will also contain *J*. A classical data-mining algorithm first computes the sets of elements that are frequent within transactions. It then identifies inference rules given the frequent sets computed first.

In our context, an item corresponds to a newspaper topic and a transaction corresponds to a user profile. First, the algorithm computes page groups of increasing size, and each group identifies a set of topics that are all accessed by a number of users exceeding the given threshold. In order to get a hierarchy of groups, the algorithm's first phase structures computed groups according to a tree structure. However, typical mining algorithms are not suited for the dynamic computation of groups as for our case. Any modification within the set of user profiles leads to re-compute the whole hierarchy.

Our algorithm extends a typical mining algorithm to exhibit a dynamic behaviour. We can therefore dynamically build the group hierarchy using recursive procedures for the addition,

removal, and modification of a user profile. By construction, a tree node is necessarily a frequent group whose level in the tree determines its size. Each node of the tree stores the number of clients whose profile contains the topics of its associated group and an additional topic (which has to be greater than all the topics of the group).

The performance evaluations presented in the Final Service Deployment Document show that using together Mining and Clustering is a way for enforcing both timeliness and scalability.

2.5 User Access

The User Access provides facilities for mobile applications to exchange information with their mobile users and vice versa. The User Access consists of a main component that provides standard interfaces for FollowMe applications to interact with users, and a set of device gateways that provide an adequate layout rendering of this information.

Main requirements were

- the support of interactive (i.e. input/output) and passive (output only) devices via common interfaces,
- the provision of a generic mark-up language to describe the contents and the layout of the rendered documents, and
- mechanisms to adapt the transmitted information and layout (i.e. the quality of service) to the performance of the system and the selected end-user device.

The design is driven by a Document Delivery Pattern. I.e. an application hands over a document to the user access for delivery. The document itself encapsulates the information, but also a description how this information is rendered to a layout suitable for a certain device.

The device gateway negotiates with the document for an appropriate layout rendering. A document provides a set of external representations of its information. If these representations are not suitable for the given device, the device gateway requests a standard representation of the information in XML together with an accompanying style sheet in XSL, and renders both to an adequate layout.

Currently device gateways for WWW-browsers, Fax, SMS, email and java-enabled devices (based on swing) are implemented.

2.5.1 Requirements addressed by the User Access Component

The User Access component provides facilities for interaction between the agent and the user (see Fig. 7). There are two main classes of requirements that are addressed by the user access component:

- Communication requirements: i.e. what devices have to be supported, what are the facilities to exchange information between the user and the agent[1]?

[1] Please note that in the following the concepts of *agent, personal assistant,* and *application* could be used synonymously. We will mainly use the term agent, in some cases personal assistant if we want to emphasise the

- Layout rendering requirements: i.e. what facilities are needed to present the information on the selected device

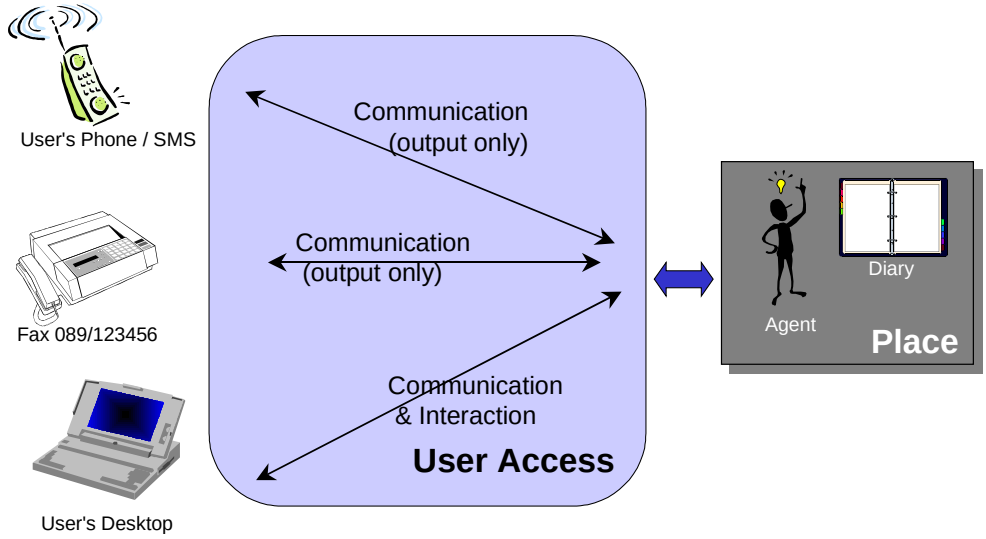


Fig. 7 *The User Access Component as a mediator between agents and devices*

2.5.1.1Communication Requirements

Users interact with their agents through a variety of different media. A set of devices servicing these media types is supported. In the following we distinguish between passive (i.e. output only) and interactive (i.e. input/output devices). The following devices should be supported

- Interactive Devices
  - web-browsers
  - Java enabled devices that support graphical output
- Passive Devices
  - fax-machines
  - audio output via phone
  - SMS (short message systems)
  - email

Additional device (types) should be pluggable into the system without the need to modify the existing agent's capabilities.

Initiation of a communication can either be triggered by an agent in order to contact its user, or vice versa.

When the user wants to contact its agent, he accesses it through an entry point provided by the user access component. The contact can e.g. be established from the user's browser. The browser contacts a server that acts as an entry point. An entry point provides two functionalities: It contacts an agent factory to create a new agent that gets in touch with its user, or it can lookup through a predefined agent directory an existing agent. In the latter case the entry point informs the agent to get in touch with its user.

personalised nature of the agent. There is no difference from the usage point of view whether the user access service is used by a (mobile) agent, or any other FlexiNet-based application.

The agent maintains connectivity information in the user's profile. When the agents wants to deliver information to its user, it has to lookup current connectivity information in the user's profile. This can be the user's business fax, or home fax, or the user's SMS gateway. The agent requests from the user access component to open a connection to this device. With this connection opened it can send to (and potentially interact with) the user.

2.5.1.2Layout Rendering Requirements

The agent may wish to send the same information to the user through different devices and media types. Thus information can have different external representations, depending on the type of device. For example if an agent wants to deliver the information that a share's stock value has exceeded some set limit, this can be rendered in various ways to the user:

- On a web browser as a web-page with sophisticated graphical layout elements
- As an email that containing a text note
- As a voice message via phone: "*Some shares in your portfolio have exceed a limit, please contact your agent*".

The knowledge of the agent about the specific characteristics of a device should be as small as possible. Thus the User Access has to provide a unique interface for different device/media types, and to bundle devices with similar layout capabilities into a device type.

To achieve this, information and layout must be separated: The information transferred is the same for all devices. The layout maps this information to an appropriate output format (see Fig. 8).

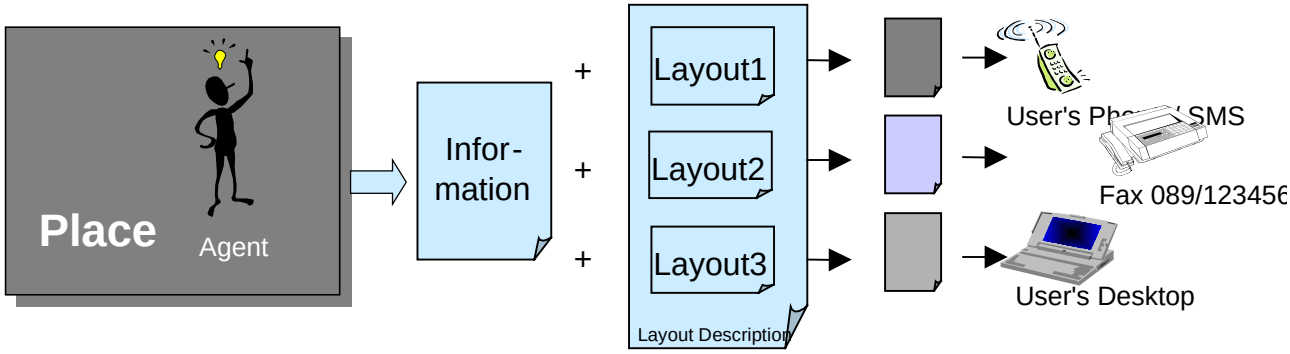


Fig. 8 *Separation of information and layout description*

The rendering into an appropriate external representation and the transmission is handled by a *device gateway*. A device gateway provides connections to a set of similar devices (e.g. a SMS device gateway, a fax device gateway, or an http device gateway). Each of these device gateways provides a certain set of capabilities.

The user access must provide mechanisms to adapt the layout description to different constraints as e.g. size of an output screen or black & white vs. colour output. The rendering of a layout can be quite resource consuming. Thus the transmission process must be separated into its two basic functionalities (see Fig. 9):

- rendering a layout by *converting* it into an appropriate representation suitable for that device, and
- *transmitting* the external representation to the end-user device.

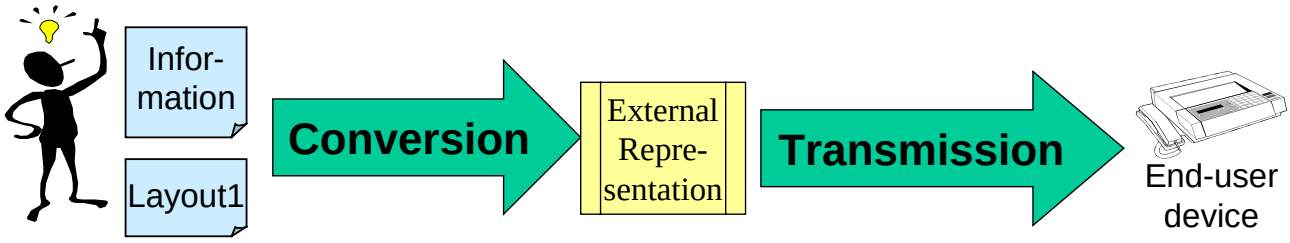


Fig. 9 *Conversion and transmission of information*

2.5.2 Design of the User Access

In the following, we describe the overall design patterns of the user access, its principles and the objects that define the service's architecture and its functionality. The interaction between User Access service and other FollowMe objects is explained in interaction diagrams.

2.5.2.1Architecture of User Access

Overview

A host, participating in a FollowMe application, can offer a set of local *devices* that support communication with the end user. Examples of such devices are fax cards, voice modems, a web-server capable to display HTML on web browsers, or an applet that acts as an (sophisticated) input/output device. These *devices* typically establish a *connection* to the user's end device. The characteristics of these connections will vary depending of the device type and the specification of the agent.

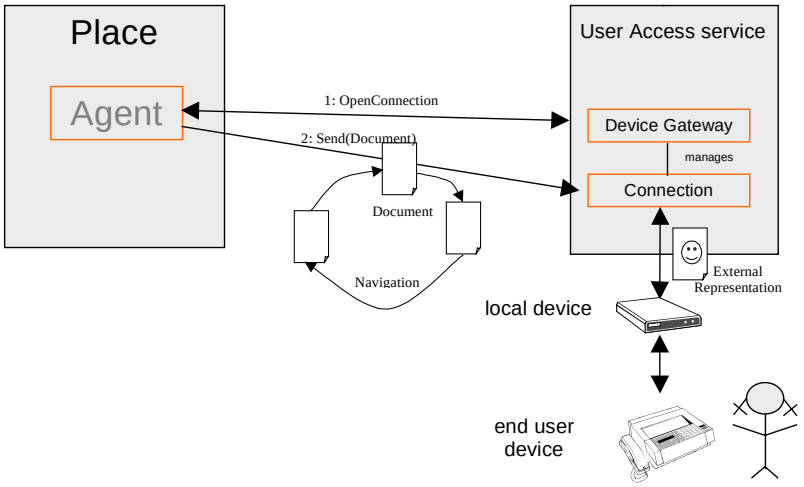


Fig. 10 *Basic communication schema between agent and end user*

Regarding this model, the main task of the UA component is to manage connections between agents and end-user devices. This includes the opening of a suitable connection, the transfer of documents, the support of navigation inside documents and the closing down of a connection. Both agents and users are able to close the connection and interrupt the communication between them at any point in time.

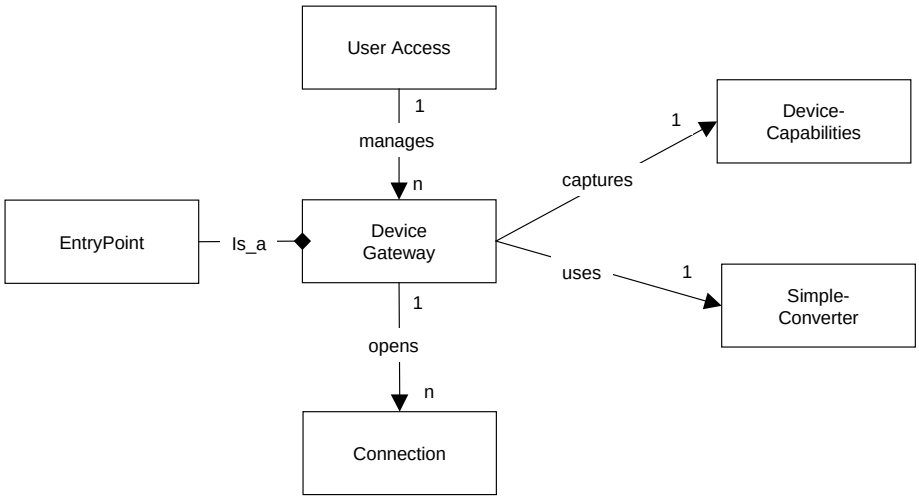


Fig. 11 *Main User Access objects*

The *User Access* is a service that manages the local devices (fax card, modem, email-service, etc,) available at a host. Each device has a *Device gateway*[2] associated. The user access service can also relay to device gateways maintained by other user access services.

Device Gateways provide facilities to open connections between the local device and the end-user devices. They hold a description of the capabilities of the device, and provide a converter from the XML format to the external representation required by the device. An Entry Point is a special device gateway installed at known locations that allow the user to request a connection to an agent.

A *connection* is a (temporary) channel to transport a set of (potentially complex) documents to the end-user's device. The transport can include conversion into appropriate external representations and transmission of referred documents (e.g. images). Also connections handle navigation through references to other documents and user responses.

There are two ways for an agent to obtain a connection. Either the agent contacts a UA service and asks it for a connection (e.g. via a fax device) to use. Or it gets a contactUser message from an entry point that contains the connection as an argument.

The connection properties, i.e. capabilities of the device, or the number of a telephone, are specified by the connectivity description that is found in the personal profile or constructed by the agent. The user access service looks for a device gateway that can support the requirements given in the connectivity description. This can be either a local device gateway or also a remote device gateway which is completely transparent for the Agent. The agent can then send documents through the connection object.

The information is encapsulated in documents. Information can have different external representations. A standard external representation supported by all device gateways is XML (see section "Document Model" below). Together with an associated layout description in XSL these representations can be translated into appropriate external representations supported by that device, as e.g. HTML, plain text.

A hook to support the Quality of Service (QoS) is provided in order to adapt the use of resources and network traffic to the performance of the system, the network, and the capabilities of the device. Via the QoS the layout of the rendered documents can be influenced. Thus the user may e.g. receive black and white data instead of coloured images

[2] Device gateways can be compared to device drivers in standard operating systems

depending on the agent/applications preferences, i.e. the agent can decide if it is preferable that the user receives a coloured image at a very low speed or if it is better to send the user just black and white pictures.

Document Model

Documents have to support three types of functionalities:

- Documents represent information sent from an agent to the end-user: The information can have quite a complex structure due to its multi-medial nature. It can contain text, pictures, audio, etc. Thus documents may refer to other documents, containing individual pieces of the information. Each document can be rendered in different external representations which are described as MIME-types (see Fig. 12).

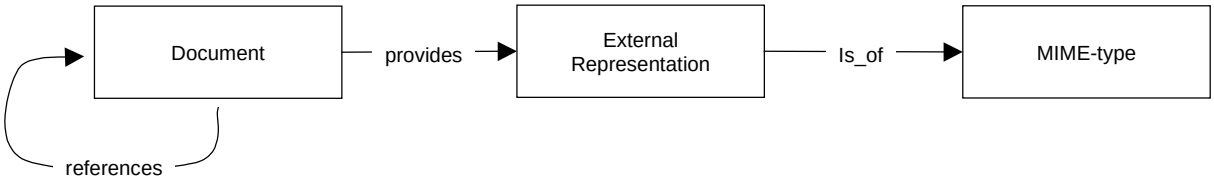


Fig. 12 *Documents*

- Documents give access to layout descriptions. Documents provide e.g. XSL documents describing the layout of the information. The user access uses this description to render the information into the appropriate layout.
- Documents handle user feed-back, i.e. they manage the reaction of a user on this document. A user can trigger feedback by two types of actions: Following links to referenced documents, and by returning a set of input parameters requested (e.g. in an HTML-form).

A document gives access to different external representations (see Fig. 13). These representations can be already precompiled, or constructed on the fly. Device Gateways provide converters[3] that convert from an external representations to another external representation, suitable for this device gateway. If a document is not able to provide the appropriate external representation for a device, the XML representation is used to derive the appropriate external representation. These external representations are described by mime types (e.g. text/xml, text/html, image/gif). The relevant method is

```
public ExternalRepresentation getExternalRepresentation(MimeType mimetype)
```

A document can refer to other documents by an (internal) name. I.e. a document can be queried to give access to another document via the method

```
public Document getReferredDocument(String name, Properties params)
```

The name can refer to the result of the filling of a form by the user. In this case params is a list of extra parameters values given by the user. Thus a document can provide its own handling facilities for user feedback.

[3] Converters implement the interface SimpleConverter. In future releases more complex converters may be defined.

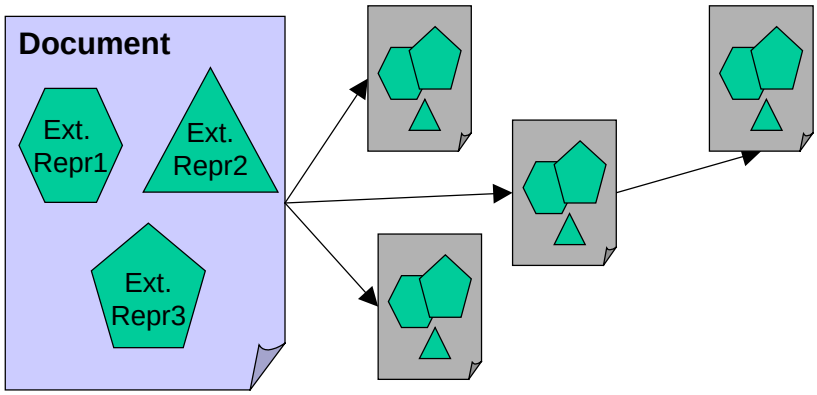


Fig. 13 *Structure of Documents*

Document is an interface, the implementation behind this interface is application depend. Thus documents can build quite a complex structure, e.g. contain subdocuments, refer to other documents.

Fig. 14 depicts the example of an issue of a newspaper. This issue is described as a document, which has two external representations in XML and HTML. These representations refer to articles as subdocuments[4] which in turn refer to documents that contain photos. These photos are represented in GIF-format.

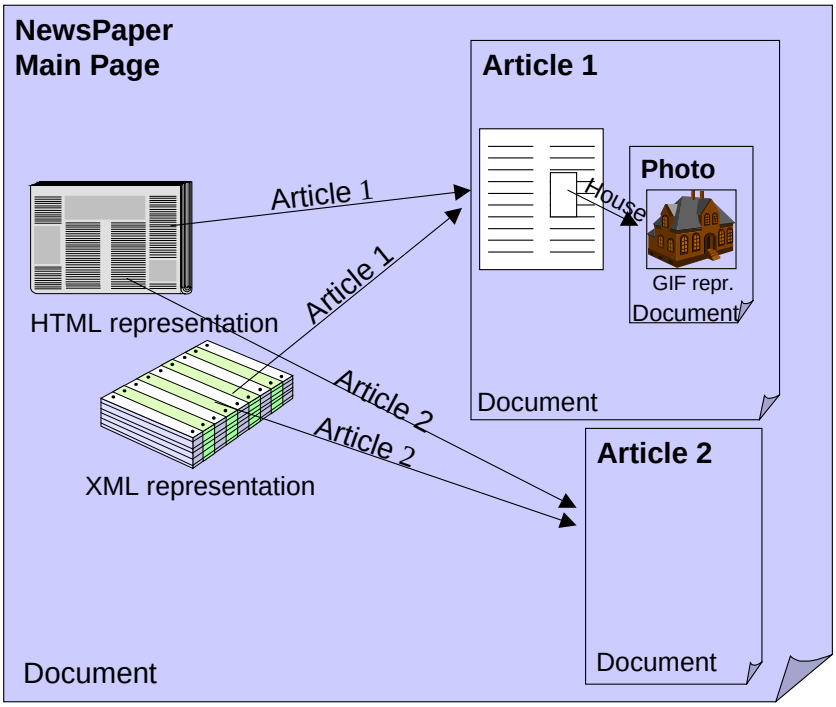


Fig. 14 *Example of a network of documents*

Interaction between Connection and Agent

The interaction between agents and connections falls into three phases:

- Obtaining a connection through an open request to the user access or a device gateway , or through the “contactUser”-message emitted by an entry point.
- Sending of documents, navigation through documents

[4] Whether these documents are subdocuments or separated documents is just a conceptual view. From the user access point of view they can be independent documents.

- Closing a connection

There are two ways to obtain a connection:

- The agent actively requests from a user access service the opening of a connection with the `openConnection(ConnectivityDescription ConDes)` method. Argument to this method is the ConnectivityDescription which describes how the user can be reached, i.e. the device type to be used, and the number. (Fig. 15)
- The user contacts an EntryPoint. The entry points are located at well known hosts for each device type through which the user can contact FollowMe, e.g. an http-address if the contact is through a browser. This EntryPoint interface is in charge of opening a Connection and it calls the method `public contactUser(Connection C)` in the interface of the agent. contactUser carries as a parameter the connection via which the user is online. The agent can then use this connection to interact with the user. (Fig. 16)

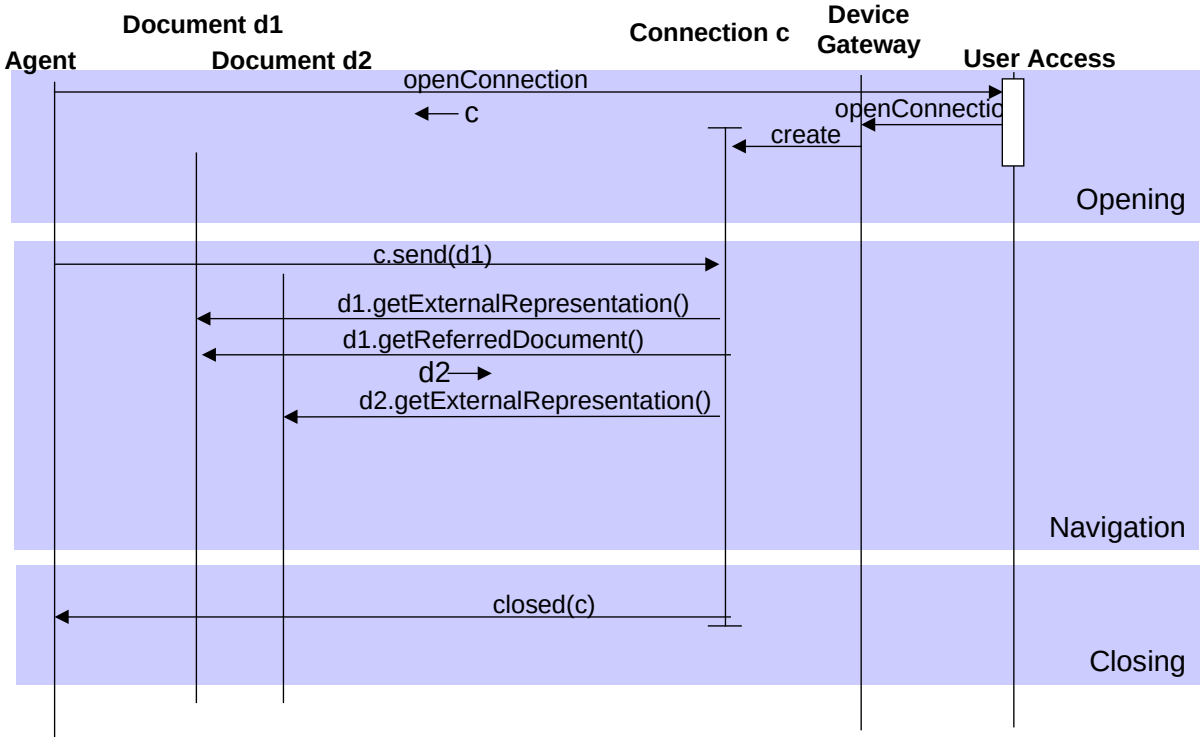


Fig. 15 *Interaction diagram: agent actively opens a connection*

`The agent uses the send` method of the connection to ask for the transmission of a document. The transmission can involve the access to referenced documents. However the agent is not directly involved in the transmission of these documents, because the documents themselves handle the access. The effect of multiple sends is device dependent. On screen oriented devices (e.g. http-devices, awt-devices) it will in open another window for each document. On text oriented devices (e.g. mail, fax) it will result in a concatenation of the documents.

The sending involves both the transformation into the correct layout and the physical transmission to the user's end-device. These two steps can be taken apart by using a SimpleConverter to render the external representation and using the method `public abstract void sendraw(ExternalRepresentation er)` to physically transmit the external representation.

A connection can be closed by the agent with the close-method. Whether this immediately closes the connection or whether the physical transmission of documents is still ongoing, is device dependent.

In order to intercept the closing of a connection by the user, or by enforcement of the user access, an agent can install a `ConnectionListener`. This listener can receive two types of events: Failure of transmission and closing of the connection.

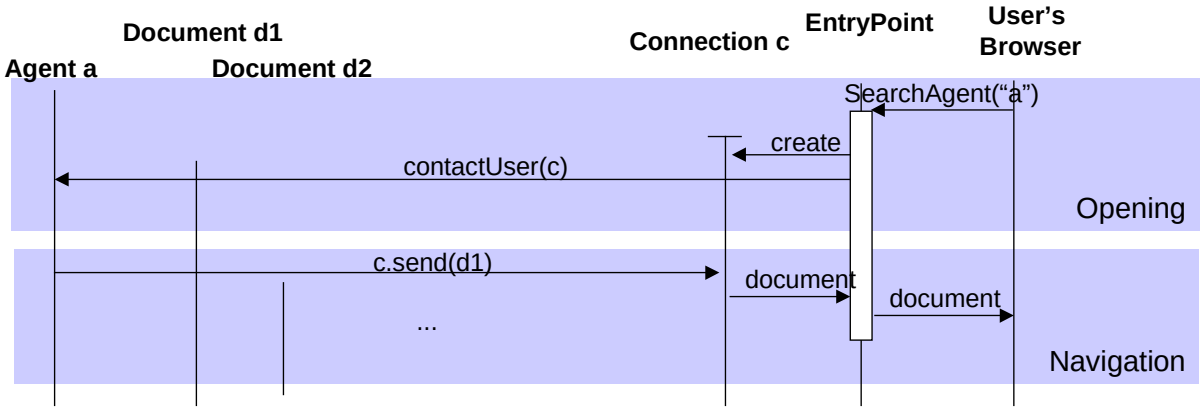


Fig. 16 *Interaction diagram: A user requests an Entry Point to search and to contact agent "a"*

Generic Mark Up Language XML/XSL

The main motivation to use a mark up language to represent data and layout was born from the requirement to have a general, common way to represent data and layout that has to be delivered to different users and devices with different capabilities like audio, html/text, etc.

Why a mark up language to solve this problem?

Some uniform way to represent structured data was needed, some way to write documents in a form that allows to describe their own grammar, i.e. describe elements and the structural relationships that those elements represent. We need extensibility, allowing agents to create their own elements representing data, managing of structured data with a high level of nested complexity, some common way to describe different views of the same data. XML emerged as a standard which provides such features. In parallel, XSL ("extended style language") emerged as the complement of XML to be used to describe in a so-called style sheet the layout of the XML data.

Both XML and XSL are W3C recommendations or W3C working draft. The language definition is not reproduced here, but can be found at [3], [2] and [1].

Use of XML/XSL by agents/applications

The document model described in 0. is capable to transmit any type of information to the end user device, provided the device gateway supports this type of information (i.e. the respective Mime-type).

The XML document can hold arbitrarily structured (textual) information. The layout of the information is defined in an XSL style sheet. XSL supports different modes. These modes are basically used to discriminate between different layout descriptions:

" "(default): for devices which can understand full HTML, e.g. Browsers

"text": for devices which can understand only text, e.g. some fax devices, e-mail

"staticHtml" for devices that support rendering of HTML-structured text and graphics like a fax

"shortText": for SMS and similar devices with limited text capabilities

"audio" : for devices which can only work with sound, like a standard phone

Via the mode-method of the QualityOfService-Interface the agent can provide an alternative mode in which the layout should be rendered. This could e.g. allow for the replacement of coloured images by textual descriptions, if the network bandwidth is low.

The connection transparently applies converters that combine the XML document and the XSL description and produces a format suitable for the end device (e.g. HTML, RTF, plain text, etc.).

If the device type is already known, the XSL style sheet can already be tailor-made for this device type. Otherwise the XSL style sheet defines a set of different layouts for all devices to which a certain piece of data may be transmitted. The different layouts are defined in different modes of the XSL layout rules.

The corresponding XSL-file is referenced in the XML-file by the <?xml-stylesheet href="..."?> tag. If no tag is given, the name "index.xsl" is assumed. After parsing the XML-file the XSL-file is requested from the document (via the referredDocument-method), and both applied to convert to the appropriate layout.

2.6 The Bavaria-Online Pilots

Customer demands nowadays move away from providing a plain web presence towards providing web-based value added information services. Thus the core objective within work package I was to evaluate whether the FollowMe architectural framework provides a suitable basis for straight forward development of web-based, user customisable information management services.

A first important result of the project had therefore been the design of the work package I application framework. The framework had been designed in close co-operation with project partners involved with FollowMe's technical work packages, especially with agent framework (section 2.3). Thus the application level architecture is also reflected in the concepts of the agent framework.

Based on the application framework two such information services have been implemented and deployed for field tests within the Bavaria Online citizens network. Feedback from pilot users and results from usage monitoring were evaluated to measure the degree of user acceptance of new services.

Application framework:

The main concept behind the application framework was derived from the following general issues related to information management infrastructures:

- The use of any database is not only defined by the sheer amount of collected data, but by the applications or services that operate on the contents of the database to provide information in form of customised results to the users of such systems.
- In order to enable the development of useful database applications, any database needs to provide a meta-model describing the structure of the offered data.
- To gain the most from largely distributed databases, the development of database applications should be de-coupled from the maintenance of the databases themselves.

That way, the contents of data sources can be re-used and re-combined when developing new applications according to the needs of information consumers.

An analysis of above issues led to the most important design decision for the WP I pilot application framework: *to de-couple the roles of service providers and content providers*. Service providers implement applications that make use of raw data offered by content providers. They define meta-models describing the data structures their applications are capable of dealing with. In order to enable the service providers' applications to make use of the data offered by a content provider, the data needs to be structured according to meta-models that form supersets of the meta- models of the service providers. This decision led straightforward to the following core axioms:

- Users of our applications will no longer (as is with the Web) address content providers to obtain information in raw or proprietary data format. Instead they will address services that provide them with already refined information according to their individual needs. The services therefore need to be customisable by the individual user.
- Services do not operate on a predefined or hard-coded set of data sources, but on specific data structures. They may use any data source available at runtime that offers relevant data as long as it offers an interface the service knows how to use.

These axioms imposed the introduction of components 'that glue things together'. On the one hand, users need to have an effective way of locating services that fit their needs. On the other hand, there need to exist mechanisms that enable services to locate relevant information sources at runtime. The appropriate concepts to fulfil these requirements are the ideas of brokers, matchmakers or – more simply – directory services.

Thus the core components of the application framework had been identified as (see Fig. 17):

- *content providers* (offering access to data-objects),
- *service providers* (providing services operating on data available from the *content providers*),
- *information consumers* (users of available services) and
- *directory services* (mediating between the other components).

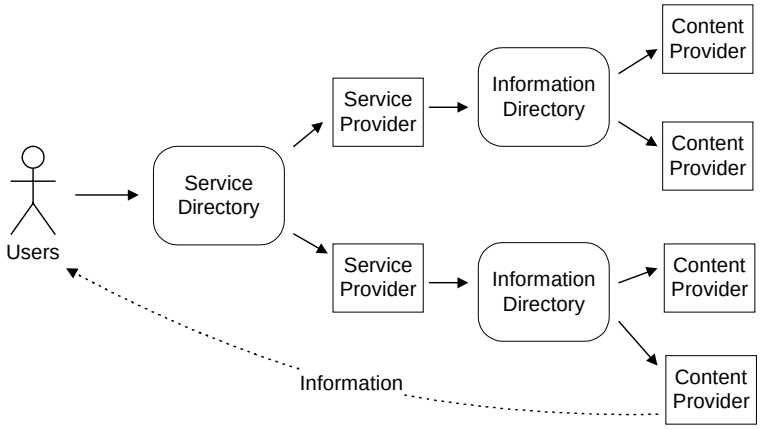


Fig. 17 *Core components of the application framework*

The overall FollowMe architectural framework (see 0) provides agent, scripting, profiling and service trader concepts to build up an agent interaction framework that enables the deployment of agent driven applications meeting the above stated needs.

In terms of the FollowMe *Agent Framework* (see 2.3) a service is composed of a component related to the information consumer (referred to as *task agent*) and a component implementing an interface to content providers (referred to as *service interaction interface*). A special user related agent (referred to as *personal assistant*) assists the user in organising the usage of services and handling personalised information.

In most client/server or network centric applications currently available on the Internet users connect to remote applications and specify their interests in form of application specific parameters. All computation is done on the server side and results are delivered to the clients. This enables application providers to gather and data-mine lots of personalised information about users of their system. In contrast, within the agent based approach of this application framework, the user's privacy is respected by hosting all user related information and computation in a location close to the user and in an environment trusted by the user (see concepts of *profiles* and *information spaces* in the FollowMe architecture documentation [4]). All agents acting on behalf of a user are instantiated on request by downloading the respective Java classes from so called *agent factories* to a user trusted environment (referred to as *FollowMe places*). This environment is located on a host with permanent online connection (i.e. a local ISP).

Another value-added feature of information services based on this application framework is the possibility to interact with the user via a variety of different device types thus enhancing user mobility. This feature builds on components developed in the User Access work package (see 2.5).

The components of the application framework are linked together as described in Fig. 18.

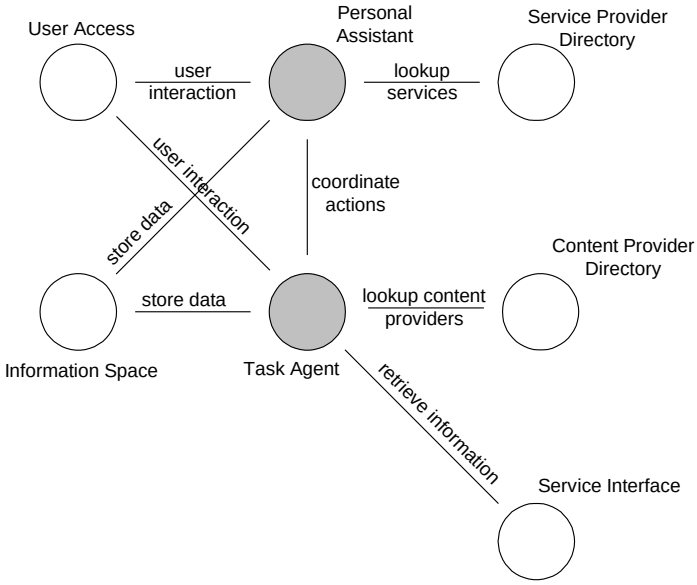


Fig. 18 *System components of the application framework*

Personal Assistant:

A *personal assistant* (*PA*) represents a single user of the FollowMe system. The role of the *PA* is to assist the user in organising the usage of services. Attached to the *PA* is the user's *personal profile*. This profile is used to store persistent data about the user. Basically user related data consists of the user's name and address, the user's system access password and a list of the services the user is currently subscribed to. Since all persistent data is stored in XML (see architecture framework), profiles can be easily extended according to the needs of the evolving system.

Besides basic persistent data about the user, the *personal profile* provides a *diary* functionality that enables the user to specify how the system may contact him at different points in time (see description of the *user access component*).

Agents act on behalf of the user while the user is offline. The system therefore provides mechanisms that enable the scheduling of task execution and reporting. This mechanism is provided by a *timer* component as part of the *personal profile*.

New users might join the system by requesting the instantiation of a new *personal assistant*. A new *PA* is created by instantiating the respective Java classes from a (remote) *personal assistant factory*.

To allow users to address their *PA* by the *Pa*'s name (i.e. *MyAgent Aladdin*), there exists a *PA directory* mapping *PA* names to object references.

Task Agent:

Task agents are agents offering application domain specific functionality. They are capable of using services with domain specific interfaces. The tasks of an agent can be described in terms of *missions* programmed in the scripting language of the FollowMe *Agent Framework* (see WP D, E and F) or coded in Java classes. *Task agents* can communicate with the *personal assistant* using the message passing protocols provided by the underlying *mobile object workbench* (see architecture framework).

Interfaces to content provider services (data-sources) are registered with a content provider directory service. The directory service may hold meta-data for each service. Based on this meta-data a task agent can optimise the selection of content providers. In example, these interface descriptions may provide information on update schedules of the *content providers*' databases (i.e. it doesn't make sense for an agent to query a *content provider* for new information every five minutes when the data sources are updated only once a day).

Attached to a *task agent* is a *task agent profile*. The profile contains user specified task parameters and scheduling information defining when to execute which tasks.

Task agents are created by downloading mission profiles and instantiating the respective Java classes from a (remote) *task agent factory* located at a *service provider* site.

Component Interaction:

As outlined in Fig. 18, the user communicates with agents via the *user access* through a variety of different device types. User interaction is required to provide the agent system with personalised instructions on how and when to execute certain tasks on behalf of the user.

Both agent types, *personal assistants* and *task agents*, may store the results of their information retrieval activities (that is data objects retrieved from *content provider* sites) in a user's *information space*. Access to stored objects is via location transparent object references. The *information space* provides access control mechanisms to ensure that data can be accessed only by authorised agents.

The activities of a user's *task agents* are co-ordinated by the *personal assistant*. The *PA* keeps a record of all active *task agents*. Tasks are triggered by the *timer* component in the *diary*.

Personal assistants link their users to the directory of available services whereas *task agents* contact *content provider directories* to determine which data sources (in form of *service interfaces*) to use. This again demonstrates the concept of separating *service providers* from *content providers* and thus freeing the user from having to deal with widely distributed raw data.

Pilot applications:

Based on above application framework two information services were implemented within work package I. After interviewing Bavaria Online users and operators about their information needs, two major characteristics of potential knowledge domains could be identified:

- Domains dealing with information with high regional focus: Regional social events, marketplaces for purchasing used items (e.g. cars, furniture, sports equipment,...), real estate business.
- Domains dealing with highly dynamic information with strong user interest for up-to-date information: Financial information services, daily news, weather information services.

Based on these characteristics we decided to build a service providing information on regional social events and a service providing share value information.

Domain of regional social events

The content providers in this domain offer information on social events with regional focus. Events advertised by the content providers might be for example local cinema programs, concerts, theatre performances, flee markets, etc. This specific domain is of interest to a majority of the Bavaria Online users and demonstrates the value-added features of our information retrieval and delivery architecture due to the following core characteristics:

- The information has high regional focus.
- The contents are provided by the Bavaria Online nodes themselves (thus getting access to the data is not a problem at all).
- Any application in this domain requires components the allow personalisation by the user.
- Automated and scheduled reporting is of great value (e.g. users once state their interest in upcoming Mozart concerts, forget about it and will be automatically informed whenever such an event is advertised).

Domain of stock share value information

Information in this domain is on constantly updated share values for stocks traded at major European and US stock exchanges. Again we summarise the core characteristics of this specific domain related to the objectives of WP I:

- The information offered is highly dynamic (constantly changing share values).
- Personalisation is required for maintaining individual portfolios.
- Automated, instant reporting is of great value (e.g. sending a SMS message to a mobile phone whenever a share value crosses a user specified limit).
- Publicly accessible content providers can be queried for the required information.

Service deployment and evaluation:

The two services were deployed in a distributed environment on hosts located at FAST and five different Bavaria Online nodes. Usage of the service by Bavaria Online users has been (and still is) continuously monitored. One month after initial deployment users were asked to provide feedback on their experiences with the new services via a web-based questionnaire.

The most important results of the evaluation of both the usage monitoring and the user feedback can summarised as follows:

- After 6 weeks of operation a total of 60 users registered with the new services. 27 % of these users contacted the system more than once in order to re-configure previously defined tasks or to schedule additional tasks.
- The 60 agents did send out 382 scheduled reports to their users. 98 % of these were sent via e-mail and only 2 % via fax devices. This result is not surprising since the first users of the services are presumably those Bavaria Online users that are the most advanced in using the internet and electronic communication. Fax delivery is likely to be more interesting for users that do not frequently connect to the internet or within application domains were instant notification (even when the user is not online and thus not able to get notified of incoming e-mails) is more relevant.
- 22 users filled out the web-based questionnaire. They in general appreciated the value-added features of the new services (e.g. automated information retrieval, scheduled report delivery and personalisation).
- When asked for recommendations for enhancements the most common request was to improve user interfaces to enhance user-friendliness and to work on extending the amount of data available from content provider sites.
- Most users explicitly stated that they want their node operators to continue to work on enhancing and extending the offered services and made reasonable recommendation for the development of similar services in additional knowledge domains.

Future plans:

As a consequence of the positive feedback from the first pilot users, the community of Bavaria Online node operators decided to continue maintenance and further development of the two implemented services and to investigate in strategies for development of additional services. FAST will therefore continue hosting service components at least until end of 1999 and will offer assistance in further development of services based on work package I's application framework.

2.7 The ETEL++ Pilot Application

This section gives an overview of some technical aspects of the second pilot application developed by INRIA and TCM, namely ETEL++. It briefly presents the main needs and presents the main connection this application has with other work-packages.

2.7.1 The Need for an XML Parser

All partners took the decision to use XML as the common language for modelling and encoding data that could cross the boundaries of software packages. From ETEL++'s point of view, the data consists of the editorial material provided by Ouest-France, that is, articles. The articles that are needed to build personalised editions are extracted from the real database at Ouest-France, converted to XML and stored in one Information Space. After this conversion, articles are analysed and meta-data resulting from the analysis is attached to articles. That meta-data reflects somehow the semantic contents of each article, that is, it consists of

thematic and geographical keywords reflecting the subjects and the areas mentioned by the body of the article.

We therefore needed to parse XML articles in order to built such meta-data. After several trials, we decided to use the freeware XML parser created by Norbert H. Mikula. We also defined a specific grammar describing the possible contents of XML articles. This grammar is used by the parser to isolate specific XML tags, and to extract the associated data.

2.7.2 The Need for Persistency and Distribution

ETEL++ is a distributed application. Its architecture consists of multiple servers interconnected by Internet, each having a local persistent storage space. The role of servers has been uniformised, that is, no server has any privileged role, and any processing task can be executed on any server. A major design concern was to abstract all the difficult issues regarding distribution. We therefore extensively used all the distribution facilities provided by APM. In addition, every single document managed by ETEL++ is made persistent, and may be copied to and from any server. We both use white box and black box objects, which appear to be very convenient abstractions for facilitating the use of persistency. All objects become persistent transparently, and the code for dealing with persistency issues is well localised, small, simple and efficient.

2.7.3 The Need for Performance Sensitive Data-Flows

A major concern of ETEL++ was to enforce Quality of Service. This can be translated in an attempt to minimise when possible the response times needed to build personalised editions. This is particularly crucial when utilising Internet in which large performance fluctuations may be experienced in an apparently random manner.

ETEL++ dynamically checks for the performance of all the servers involved in its distributed architecture, and decided upon the route articles will follow to produce the editions in an efficient way. The performance of servers is obtained by the means of the Service Deployment work package. Given the CPU and the network load of each server, each server is asked to perform a particular task, to possibly generate editions of the behalf of heavily loaded servers, or to propagate specific data to other servers.

If a CPU load factor is above a given threshold, then the corresponding machine is said to be CPU bounded. In this case, ETEL++ considers this machine has not enough CPU power to transform articles into the final representations required for user delivery. The same also applies for the network. If the available bandwidth between two machines is scarce, then it is undesirable to transfer between them a large volume of data. In this case, ETEL++ considers the machine at the end of the link to be network bounded. A machine can be solely CPU-bounded, solely network-bounded or both. In the first case, this machine will not transform documents from one media to the other, but will rather try to obtain a copy of the relevant documents after having asked another non-CPU-bounded machine to do the transformations on its behalf. In the second case, a network-bounded machine will rather get a copy of some documents, and subsequently generate all the transformed versions to match user's expectations. ETEL++ assimilates the case of a machine that is both CPU- and network-bounded to the case of machines that are solely CPU-bounded. It is also possible that a machine is not bounded at all. In this case, ETEL++ sends active documents, since this tends to minimise bandwidth consumption. Before initiating the computation of the data flow map, all servers can therefore be classified in NO-BOUNDED, CPU-BOUNDED or NETWORK-BOUNDED. This classification enables the computation of a performance sensitive data-flow

map in which each server is assigned a specific role, and in which the “journey” of each article is decided.

2.7.4 The Need for Agents

Agents and the Interaction of Services provided by UWE are of a great interest for ETEL++. Agents are utilised to obtain context-sensitive data. In particular, we use Agents to obtain the weather forecast that corresponds to the area of connection of a user. When that location of the connection changes, the content of the weather forecast is changed as well. Agents are useful in this case because they remove the complex burden of managing geographical information, and of discovering an external forecast server that is adequate. In addition, the use of Missions as defined by UWE makes possible to encode in a simple way the reactions to enforce when a discovered server is unavailable for example.

2.7.5 The Need for User-Access

The User-Access work package provided convenient ways to facilitate the production of multi-terminal editions. ETEL++ designed its hierarchy of objects in accordance to the design of User-Access, that is, specific objects implement the interfaces that offer means for coping with multiple devices.

2.7.6 Internal Data Structures

ETEL++ needs several data structures to make possible the construction of personalised editions. First, it needs multiple inverted lists that associate document references to keywords. These inverted lists enable to know what documents contains a certain keyword (thematic, geographic). All the documents are indexed by one or more inverted list.

ETEL++ needs also data structures to manage the profiles of its readers (in this case, users). A user profile contains key information about the identity of a user. It contains also data that describe in which ways the edition for a user has to be personalised. For example, a profile contains a set of keywords that describe the centre of interests of a user.

Another major data structure needed by ETEL++ is the one that keeps workload information for the performance sensitive routing of data flows.

2.8 Conclusions

The FollowMe framework was designed as a toolkit of components. Different applications may use different parts and configurations of the individual components. The common architecture guarantees the inter-working of the individual components.

With the two demonstrators the FollowMe Framework has proven to be usable for building up highly sophisticated and personalised applications.

3 Project Execution

This chapter presents for all work packages the major development steps and design decisions during the evolution of the project.

The project has adopted a model of gradual extensions to produce the results of each work package. This proved to be successful, also because it enabled the work packages to incorporate change requests from other partners. Thus a common coherent architecture could be developed.

A series of project team meetings and individual meetings where planned, in order to foster the internal communication structure. Especially in the early project phases a series of meetings took place to discuss and elaborate the mutual requirements of each work package.

All work packages delivered the planned results. The project was originally planned for 24 months and cut down to 18 month on recommendation of the EC-reviewers of the proposal. Unfortunately this cut mainly got onto account of the pilot applications which were able to develop the demonstrators, however the field evaluations must continue further to give more significant feed back..

3.1 WP A, B, and C : Architecture, Mobile Object Workbench, Information Space

The purpose of the Architecture work-package in the FollowMe project (WP-A) was two-fold:

- at the outset of the project to scope the division of work between partners into basic components with well defined interfaces and relationships; in this respect the architecture description is a reflection of the planning work done by the partners during the project proposal stage, and during the first phase of design
- as the project progressed, the architecture description was updated and refined in the light of ongoing design reviews to become a repository of design expertise assembled by the project, which we felt would be of value of others following in our steps. In doing so we looked for the re-usable elements of our overall system design.

The FollowMe project took as its foundations the ISO/ITU Reference Model for Open Distributed Processing – RM-ODP -(ISO/IEC 10746 parts 1-3, ITU Recs. X.900-903) and the Object Management Group's CORBA suite of standards.

The motivation for using RM-ODP was to provide architectural foundations for the project. This was successful in two ways. First the RM-ODP engineering model provided a template for clusters in Tasks B and C (Mobile Object Workbench / Information Space). Second in documenting the architecture, RM-ODP provided a vocabulary and structure for separating

concerns that proved useful to some extent. Most of the architectural patterns turned out to fit the RM-ODP computational model, which was sufficient for meeting the descriptive need, but gave no guidance in the identification and structuring of patterns. This information came from project design work.

The motivation for using CORBA was to inter-work with the dominant industry standard for distributed computing. FollowMe augments the ODP / CORBA of distributed computing with two powerful new paradigms for information systems. The first is object mobility. Object mobility allows programmers to produce systems in which objects with state can move around a computer network. The second paradigm is agency. Agency means that programs act autonomously on behalf of users and so may make decisions when the user is not connected. Indeed one major decision that often needs to be made is how to connect to the user at a specific time to deliver a specific piece of information.

While the project followed the ODP and CORBA architecture, the choice of Java as implementation language enabled a simpler API and more powerful to be developed than would arise by slavishly adopting the OMG Java mappings for CORBA. However the implementation does include the CORBA IIOP protocol so that it is possible to connect from a FollowMe system to an existing CORBA infrastructure. In particular, the project did not adopt the CORBA Agent Facility. Partly this was a consequence of the specification appearing after the project had committed to design decisions based on direct exploitation of Java facilities. It is commonly the case in industry for there to be optimised Java interfaces exploiting the full power of the language with a mapping to CORBA being offered for legacy integration. (For example the Java Enterprise Java Beans specification uses the CORBA IIOP protocol and Object Transaction Service in its implementation, but does not directly offer CORBA APIs to the developer).

The Mobile Object Workbench and Information Space tasks delivered an infrastructure which enables Java developers to implement objects using the full facilities of Java and transparently provide facilities for mobility and persistence. To obtain these facilities the developer must inherit from appropriate classes and respect a simple protocol for stopping and restarting objects. What this architecture does not provide is the concept of a thread of execution that can migrate from place to place. This functionality is provided by the scripting facilities produced by the Intelligent Agent task. This has worked out to be an excellent division of abstractions. A systems programmer can work at the Mobile Object Workbench / Information Space level with full Java including concurrent threads, reflection, introspection, and remote procedure call. An applications developer can work at the scripting level where the complexity of these systems issues is completely masked and high level agency facilities are provided. We contrast our result with contemporary mobile agent systems that do not provide a separate mobile object infrastructure for the system's programmer. Instead they bundle agency and mobility into a single API forcing a single transparency selection on developer, seriously limiting flexibility and extensibility.

The principal design parameters for the Mobile Object Workbench / Information Space were:

Java Language
We attempted to keep the API and remote object semantics as close to normal Java as possible. In particular, we use Java interface classes rather than IDL files, as this is more natural for a Java programmer.

Build for Change
The Workbench was designed to be constantly upgraded and changed. We attempted to minimise the amount of 'global knowledge' and interdependencies between components, so that parts could be replaced, or new components added.

Multiple Everything
The Workbench was designed to be component based, and to allow more than one instance of any component to co-exist. This is important, for example if a client has to speak two versions of a protocol. We make very little use of 'static' data, as this is intrinsically restrictive.

Reflective Implementation
We attempted to use our own mechanisms internally wherever possible. For example a name, passed to identify an interface, is an ordinary object, and treated as such when serialised, deserialised or otherwise manipulated. This approach allows us to change the specification of one component with minimum impact on other components.

These decisions proved to be highly successfully. Through four versions we added mobility, persistence, security and transactions to the basic framework proving the concept of "selective transparency" first introduced in RM-ODP.

A particular consequence of this approach was that the Information Space was delivered as a structured object store rather than a simple file system as originally proposed. While this exceeds the needs of the current applications it provides a more general mechanism that can be exploited to scale up to replicated sites and storage migration for load-balancing and latency management. This is an interaction between the Information Space and Service Deployment tasks which there hasn't been time to explore during the project.

In summary we conclude that our early architectural decisions have stood up throughout the project. We have extended the state of the art in object mobility for Java and developed a technology that can be exploited in a number of scenarios beyond the web including network management and global network application services.

3.2 WP D, E, F: The Agent Framework

3.2.1 Objectives

The Autonomous Agents (D) work package was conceived as a means for providing the essential agent behaviour for the FollowMe project. The objective was to provide means of creating agents that would exhibit autonomous behaviour and to realise this behaviour using the mobility provided by the Mobile Object Workbench (work package B). The intention was to create agents to accomplish certain tasks; hence *Task Agents*, and for these agents to be customisable to meet specific needs of a user. Since agents act on behalf of users it was clear that it was necessary to provide a means for personalising an agent and the Personal Profiles (E) work package covered the work necessary to achieve this objective. The intention was to look at existing approaches for modelling personal data and to adopt an approach consistent with the overall architecture of the project. Finally, for agents to perform some useful task it is imperative that they are able to interact with useful services. Aware of the impossibility of providing a general solution to this problem it was intended that he Service Interaction (F) work package would focus on the development of a pragmatic solution based on Trading.

3.2.2 Emergence of an Agent Framework concept

From an early stage in the project it was realised that the results from the three UWE work packages could usefully be combined into a single software deliverable which provided an *Agent Framework*; a collection of software components that facilitated the deployment of autonomous, personalisable, mobile agents.

The Agent Framework adopted the weak agency model of Wooldridge and Jennings[]. Whilst agents in FollowMe are capable of autonomous, social, and responsive behaviour they lack the rather stronger capability of pro-active (or goal seeking) behaviour. However, the implementation of the framework is sufficiently open to allow components which implement learning behaviour to be incorporated.

3.2.3 Agent Framework Components

The main components of the Agent Framework to emerge from the design process were

- Task Agent
- Agent Place
- Personal Assistant
- Personal Profile
- Diary
- Service Profile
- Trader

The implementations are based on the Mobile Object Workbench and the Information Space from work packages B and C and the interfaces for interaction with the Personal Assistant, which acts on behalf of the Task Agents, are provided by User Access from work package H.

A complete installation of the Agent Framework therefore requires the Mobile Object Workbench, Information Space and User Access. Migration of Task Agents between Agent Places on different host machines in response to machine loading would require the use of Service Deployment from work package G.

Whilst the Agent Framework was designed to be a collection of components it was thought necessary to supply a certain number of utilities providing a user interface such that a programmer could start up a Trader and a Personal Assistant and actually browse, edit and launch Task Agents with no programming effort at all.

3.2.4 Principle Design Features

The most contentious design issue within the project was to implement an agent scripting language. It was felt that it would be in the spirit of enabling relatively inexperienced users the opportunity to craft their own agents. Therefore, UWE coded a complete implementation of a scripting language based on JavaScript (ECMAScript). This included the facilities to declare and use external Java classes (scripting used as *component glue*) and support for mobility through the use of a `jump()` call. The later hid a significant amount of Mobile Object Workbench detail. A complete agent within the Agent Framework consists of the agent script together with associated implementation classes (if needed) and any XML forms needed to manage user/agent communication; this combination is known as an agent Mission. In line with the rest of the project it was decided that the mission itself should be written using XML. It was also decided that the Personal Profile and Diary should use XML. The overall design of the Agent Framework benefited from the interaction with the teams developing the pilot applications from the very beginning.

3.2.5 Agent Framework Roll-Out

The Agent Framework was delivered to the project in a number of releases in an attempt to meet particular requirements of the Pilot Applications. The full details of the capabilities of each release of the framework are contained in the release notes of the software (currently v1.4.1). There have been twelve releases of the Agent Framework since the first release on the 18th August 1998; in addition there were some earlier releases of the script engine.

3.2.6 Implementation

The Agent Framework has been implemented in Java and the full source code released to the FollowMe consortium. It is doubtful whether this project could have been realised in any other language other than Java, certainly within the tight schedule imposed.

3.2.7 Project Progress

Due to a problem recruiting a third team member at the start of the project UWE were considerably pressed to meet some of the early design deadlines. In addition, the resources available in the project and its short time scale meant it was impossible to commit more UWE staff. As already discussed, the Agent Framework has been implemented using the results of work packages B, C and H and, like the Agent Framework, the software from other partners has also been delivered to the project in various releases. Therefore, the reality of delivering the Agent Framework has required managing the complex software interdependencies between three partners. Whilst the consortium has the necessary skills and discipline to ensure that the interfaces between software components have been rigorously specified the implementation has still taken longer than predicted. To give some idea of the complexity of the project the Agent Framework distribution contains 366 classes, the Mobile Object Workbench (including Information Space) 1063 classes and User Access 202 classes.

3.2.8 Project Structure

The project structure in the technical annex describes a fairly traditional waterfall development methodology with requirements, design, implementation and testing phases. Whilst this is perhaps useful for reporting purposes it has been less useful as a project plan. A spiral-like methodology (e.g. DSDM) would perhaps have been more appropriate. Rather than waiting until the design documents had been approved before starting the implementation it would have been better, with hindsight, to have started implementing as soon as some of the basic ideas had been agreed and then for the design, implementation and testing to have iterated many times over the duration of the project. This would of course have increased the number of meetings required and would also have complicated the versioning of software. The Hyperwave server hosted by FAST has been extremely useful as a project repository (despite some downtime); it would have been interesting to consider the possibility of using this server as a central repository for developing *all* the software such that version control could have been applied across the entire project.

3.3 WP G: Service Deployment

This document gives an overview of the major steps that took place during the elaboration of the Service Deployment work package.

We first worked on designing the Monitoring Tools. These tools are at the root of any deployment capability since they are used to observe the behaviour of a system, and in turn take the appropriate decisions depending on the nature of what is observed. We started to design tools that could be generic enough for being used within the project FollowMe, but also outside. The need for this king of tools has been identified for long at INRIA. Several research actions that are independent of FollowMe do study performance-related problems of distributed computing systems. Each needs some kind of monitoring, but we have developed almost only ad-hoc tools solely compatible with Unix-based systems. Coming-up with the appropriate abstraction for having a general definition of the monitoring tools was therefore crucial.

The software delivered during the summer 1998 is somehow a snapshot of this design process. We released a complete observation framework offering tools to monitor resources. Resources can be mapped onto real hardware resources, such as a CPU, or be application defined. The object-oriented approach made possible the general definition of a Resource as an Interface. Each implementation of that interface has to provide the code for some methods that return the consumption of that resource. The observation framework also contains support for defining precisely the type of monitoring that is relevant to each application,. That monitoring can be periodic or aperiodic, and in this case, it can be either request-based or notification-based. The framework also embeds several built-in Filters that transform simple, basic resource usage values into more elaborated data. We also provide facilities to add Filters.

The work scheduled during Fall 1998 was supposed to be the implementation of the lower levels of this framework, that is, the implementation of the pieces of code that are necessary to obtain from the real hardware resources their usage values. We had however an important staffing problem because the engineer that was responsible for that part left on a short notice. That implementation was therefore suspended until we could hire a new engineer. It was impossible to hire anybody before late November 1998. Once that person was hired, we faced the complex problems of integration of other software packages provided by other partners. We consequently asked that person to work on integration problems, further delaying the implementation of the lowest level of the Service Deployment work package. In additions, attempts to complete the implementations were facing the complexity and lack of clarity of the interfaces available on Windows platforms (the ones we currently use for development). It is important to say that the core of the Service Deployment is ready, that is, tools for performing periodic observation and for enabling general definition of what is a resource and how to observe it. In the second pilot application, Service Deployment tools are used in the context of a load-balancing policy that dynamically determines optimal routes for data flows depending on the usage of resources. That usage, however, is not obtained from real hardware resources, but from configuration files that are viewed by the Service Deployment as user-defined resources.

We also worked on Data-Mining issues. Data-mining used together with clustering techniques has been implemented and evaluated. It has proved to be beneficial to client-server systems in which the server side is made of cluster of computers. In this case, grouping readers having close profiles on specifics clusters increases the hit-ratio of the buffers of the servers, thereby enforcing timeliness and scalability. The main work was to devise a dynamic data-mining algorithm since typical mining algorithms are static, and can not therefore cope with the possible dynamic update users could make to their profiles.

3.4 WP H: User Access

A key objective of the project was to enable users to access the system through a variety of different media, but without losing quality of the interaction.

During the development of the project a two major decisions have to be taken:

- Which devices need to be supported?
- What is an adequate architecture to translate service in- and output to device capabilities?

During the project planing phase, java enabled devices were in the focus of the considerations for user access. However a first survey during Oct. to Dec. 97 showed that (besides java enabled browsers) no such devices were available on the market. Also a first requirements analysis triggered from the pilot applications indicated that in order to have a large user base also non java-enabled devices as e.g. fax, email, and HTML-browsers needed to be supported. Thus the decision was to develop a generic java-enabled device (the so-called swing entry point) and to extend the support also to non java-enabled devices as fax, SMS, email, etc.

These device types exhibit considerable different rendering capabilities. In order to achieve a general mechanism for translating information into a layout suitable for a certain device was needed. The choice fall on the emerging XML/XSL standard, as XML was suitable to describe arbitrary structured information and XSL was to support independent layout descriptions. XML was announced as a W3C recommendation in February 98. XSL is currently a working draft in the latest version from December 98. Tying the information and layout description to these “de-facto”-standards ensures a broad support for the information and layout description used by the user access component.

Mapping to different layout capabilities is achieved by providing different layout descriptions either in separated XSL documents, or in one XSL document with different layout modes.

Currently the following device types are supported:

- Http-Entry Point for WWW-Interfaces like Web-Browsers
- Fax Gateway for Fax-Delivery
- Email Gateway for text mail messages
- SMS Gateway for delivery of short messages to a GSM based cellular phone
- Swing Device Gateway for local document delivery on java-enabled devices
- Offline HTML Device Gateway for producing an offline readable set of HTML-documents

The user access is now integrated into the agent framework and extensively used in both pilot applications.

3.5 WP I: The Bavaria-Online Pilots

The objective of work package I (pilot application 1) within the FollowMe project was to measure the applicability of both the FollowMe architectural concepts and their

implementations by the technical work packages by designing and implementing two pilot applications and deploying and evaluating them in a real world field test.

The applications were to be deployed at five pre-selected nodes within the Bavaria Online citizens network. The first activity within work package I was therefore to discuss application demands of Bavaria Online users with the operators of the Bavaria Online nodes. To separate themselves from standard internet service providers the Bavaria Online nodes aim to offer their members access to value-added information services with regional focus.

Hence we decided to not only develop two specific information services but to provide a generalised application framework for the development of agent-based, user customisable information gathering and filtering services and to apply this framework to two specific knowledge domains. Thus requirements analysis and design documents were split into two parts: one defining the application framework, the other describing the domain specific details of the two example services.

The criteria for selection of suitable domains were threefold: user interest, availability of content providers and, last not least, potential for full exploitation of core features of the technologies to be developed in the FollowMe project. To identify the domains we interviewed Bavaria Online users as well as potential information suppliers. Based on the results of these interviews we decided to implement a service providing information on social events (e.g. cinema and theatre shows, local markets and auctions, meetings of local associations, etc.) and a user customisable share value information service.

During the requirements analysis and application design phases, we constantly interacted with the project partners providing the underlying technical infrastructure to ensure that the technology would match the requirements of our pilots. In particular we had many bilateral discussions with the designers of the FollowMe agent framework and our ideas and requirements had major influence on the design of the agent framework.

When first releases of the basic protocol layer infrastructure - the MOW and the Information Space - became available, we started with the implementation of the application specific libraries of the event notification service. Application specific functionality and object persistency were tested via simple command line interfaces.

As soon as User Access modules, device gateways and first components of the agent framework became available, we began integrating these components and implemented user interfaces in XML/XSL. A first prototype of the event notification service was then presented to the operators of the participating Bavaria Online nodes.

Implementation of the share value information system started as soon as the first prototype of the event notification system had been completed. The experiences gained during implementation of the first service greatly reduced the effort required for implementation of this second service, thus proving the value of the generalised application framework.

A first full release of the event notification system was deployed at Bavaria Online nodes and opened for public access by the Bavaria Online user community three month before the end of the project. The share value information service was deployed and published six week later.

Usage of the services was monitored via system log-files and direct feedback from users was obtained via a web-based questionnaire. The results were evaluated and discussed with Bavaria Online operators who already started planning of future enhancements and development of additional services.

3.6 WP J: ETEL++ Pilot Application

This document gives an overview of the major steps that took place during the elaboration of the second pilot application, namely ETEL++. It reflects the chronology of the processes that lead to the achievement of the application.

INRIA and TCM were having close relationship before the beginning of the FollowMe European Esprit Project. Both were involved in the conception of an electronic newspaper derived from the daily paper published by Ouest-France. FollowMe appeared as a good opportunity for experiencing with new technologies, and as a fruitful testbed in which avant-garde ideas could be validated.

One of the first design choice (global to the whole project) was an agreement of using XML as a general language for describing and encoding the data used throughout the software packages developed by each individual partner. We therefore worked together with TCM to design a schema representing the articles manipulated by ETEL++ having XML in mind. We came up with a new document model representing the articles and their dependencies. Together with this model, we implemented an article converter that consumes articles stored under a proprietary format (the one used by Ouest-France) and that produces articles following XML requirements.

This new document model fits ETEL++ requirements. These requirements, however, do not consider all the complex issues of producing a newspaper at a real scale, for real users in the field. We therefore took advantage of this design process to re-think the actual data schema used by Ouest-France (independently of ETEL++) and to improve it when this was possible. We had to analyse the complete database schema of the articles used every single day by journalists to store their articles. This database both relies on an object-oriented and a relational database management system, and complex ad-hoc bridges are required for transferring data back and forth. As a result, our original document model proposed for ETEL++ as been augmented to cope with all of the real data managed by Ouest-France. This model is currently under implementation. The design of this model was completed late 1998 (an intermediate version was ready late spring 1998).

Since the basic format used to represent articles was XML, we had to search for a tool that would allow the parsing of these articles. This parsing is necessary to build the data structures required for high level processing (like assembling several disjoint articles to build a personalised edition). After several trials, we decided to use the freeware XML parser created by Norbert H. Mikula. This parser is used (among other things) to isolate the keywords attached to each article, to extract the semantically different sections of each article (like its title, its body, etc.) and to make possible the construction of a global map where related articles are linked together (related by subjects, keywords, or by a direct reference).

We then implemented a first version of ETEL++ to test the newspaper-assembling engine that we designed on top of the intermediate document model (this engine –which is the core of the pilot application— is presented in Document DJ3). In parallel to the implementation of this first prototype (internal and not demonstrated), we conducted extensive experiments to get accustomed to the mobility and transparencies features of the MOW and of the IS provided by APM.

Once we were comfortable with the MOW+IS, we extended our prototype. This first (external) release, which corresponds to the one demonstrated in September 1998 in Rennes, was able to generate personalised editions in a distributed manner. This release, however, was not including any Service Deployment feature, nor any Agent or User-Access code.

We initiated in fall 1998 the design of the second and final version of ETEL++. This second version differs from the first one released by (i) its document model as mentioned above, (ii) extensive use of both White and Black box objects, (iii) integration of a load-balancing policy to deploy the service adequately, (iv) integration of Agents to get context sensitive data and (v) integration of User Access to support multiple terminals.

In parallel, we experienced with the Agent Framework to understand their ins and outs. We also developed a graphical tool that will show, during the demo, the flow of article between all the machines participating in the construction of the newspaper. This tool is a convenient way for illustrating the load balancing decisions, since data-flows differ depending on the workload placed on the servers.

In general, we would like to emphasise the difficulties of integrating in a prototype some software that is developed by another partner. The first obstacle is to get a clear understanding of each other's, in terms of requirements, features, capabilities, etc… The frequent and periodic meetings that took place were very helpful to increase the level of understanding. A second difficulty is to perform this integration in such a short time frame. The duration of the FollowMe project is very short with respect to the hard issues tackled by each partner. Scheduling constraints and inter-dependencies make the integration process longer and more difficult than expected. Periodic meeting were again of a great help, and it is interesting to note that during the last meetings, some code was written *live* to clarify some hard aspects of the integration. The third obstacle, which is inherent to computer programming, is to report bug and get the fixes. This last task is of course a major source of time consumption. Integration is particularly crucial in the case of ETEL++ since it has to integrate work from *all* other partners. Any delay in any part delivered had an impact of our own scheduling.

4 Exploitation activities

It is in the interest of all partners to make the component architecture available to the broad public as seedware. The partners have agreed to license the results of FollowMe to each other beyond the end of the project, but not to develop an detailed joint exploitation plan. They do not consider this an efficient exploitation path. Rather than that each partner will exploit its results individually. This section gives an overview to the exploitation activities of each partner.

4.1 Citrix Exploitation Activities

Since APM's acquisition by Citrix effort has been spent on the investigation of product opportunities within the Citrix product range. This has involved a significant amount of work by the APM project members. However the exact nature of the exploitation is commercially sensitive. If details are required APM is willing to provide a separate disclosure to the commission or reviewers in an agreed format.

4.1.1 Presentations

Citrix has actively co-developed FlexiNet/MOW with the ANSA Consortium, through

Citrix's membership of both groups. ANSA includes ICL, Fujitsu, BT, Marconi Telecoms, GEC-Marconi, France Telecom. FlexiNet/MOW presentations and workshops have been held with all of these.

The FlexiNET/MOW work was extensively reviewed by Citrix in their acquisition of APM Ltd.

4.1.2 Publications

Papers about FlexiNet and the Mobile Object Workbench were presented at:

- SIGOPS European Workshop Lisbon, Sept 98
- Middleware lake District, Sept 98 (to be republished in IEE Journal on Distributed Computing, and a volume on selected papers from the workshop)
- Mobile Agents, Stuttgart, Sept 98
- BT Technical Journal (upcoming issue)
- Contribution on FlexiNet in a book on “state-of-the-art in distributed systems research”, published by the BROADCAST working group (to appear).

4.2 FAST Exploitation Activities

FAST plans to exploit agent based technology in further customer projects. Major target groups are financial institutions and public administration. In co-operation with the Bavarian Bürgernetze further agent-based services will be developed.

4.2.1 Presentations

FAST made a number of presentations to address potential customers.

Date, Location	*Participants*	*Contents*
13.1.98, Waldkraiburg	Representatives of the Sparkasse (savings bank) and Raiffeisenbank Waldkraiburg and of the Bürgernetze	Presentation of FollowMe portfolio pilot application and discussion of the involvement and future service to bank customers
26th and 27th of June, Munich	Bayern-Online Kongress	Presentation on FollowMe, and a stall organised by the Bavarian Bürgernetze at the exhibition
7th of July, Munich	Techno-Z CEO G. Kreilinger, Braunau (Austria)	Presentation of FollowMe, discussion of a pilot installation of an active call for tender data base.
17th of July, Munich	B. Gebauer, Vice President of the Dachverband der Bürgernetze	Discussion on potential extensions and commercialisation of the pilot application in 1999.
23rd of July, 22nd of Sept., Munich	Dr. Franzen, Bavarian Ministery of Interior, H. Göttlinger, CEO of Behörden-Online	Presentation of FollowMe and discussion on potential exploitation in public administration.
17th of March 99	B. Gebauer, President of the Dachverband der Bürgernetze	Discussion of potential extensions and commercialisation of further agent-based services

4.2.2 Publications

A scientific publication for an overview to the Bavaria-online pilot application was prepared and published on the AAAI Symposion on Agents in Cyberspace, March 99.

Diverse newspaper articles were published in the Süddeutsche Zeitung.

4.2.3 Collaboration with other Projects

FAST is partner in AgentLink, an EU sponsored initiative for the co-operation of agent based projects. FAST registered for the SIG on Intelligent Information Agents and participated on the formation meeting at the 24th and 25th of September.

FAST was invited to present FollowMe on the Climate (Cluster for Intelligent Mobile Agents for Telecommunication Environments) Workshop on Mobility on 5th of May. FollowMe is registered as an associated project to the Climate initiative and participates in the Profile working group.

4.3 INRIA Exploitation Activities

INRIA has made several presentations related to the work we achieved in the context of the FollowMe Project at different places:

- 8th ACM SIGOPS European Workshop, Sintra, Portugal, September 1998, "*Introducing Contextual Objects in an Adaptive Framework for Wide-area Global Computing*", A.-M. Kermarrec, P. Couderc, M. Banatre. This presentation details some of the ideas related to the Context-Sensitive Data mentioned in Document DJ3.
- EEMA Annual Conference & Exhibition, Electronic Commerce Europe 99, Paris, June 1999, "*ETEL - A multimedia experience in a newspaper environment*". This talk presents the ETEL and ETEL++ electronic press services.

INRIA submitted to publication the following journal paper:

- "*Quality of Service and Electronic Newspaper: The Etel Solution*", B. Charpiot, J.-M. Menaud, V. Issarny, M. Banatre. Also under publication as an INRIA Technical Report. This paper investigates the use of Data-Mining in the context of an electronic press service.

One Ph. D. Thesis is under preparation, and another one is completed. Both are related to the electronic press activities at INRIA:

- Boris Charpiot. "L'extensibilité par la répartition thématique des accès à un système d'informations distribuées", Thèse de l'Université de Rennes 1, Rennes, France, December 1998.
- Frédéric Le Mouël. "Environnements adaptatif d'exécution distribué sur Internet". In preparation.

4.4 UWE/ICSC Exploitation Activities

ICSC has started dialogue with a small Bristol based company that is developing Java applications for mobile Java enabled devices with a view to investigating possible collaboration on developing products. ICSC is also talking to the Transport Research Laboratory Ltd. in order to obtain near real time road traffic information so that ICSC can look at Agent based applications for the dissemination of road congestion reports to users.

UWE is participating in the AgentLink initiative.

4.5 TCM Exploitation Activities

The major targets of TCM exploitation plan, are Ouest-France and its partners of the French SPQR (Daily Regional Press Society). Because electronic newspaper will become a necessary and complementary channel to broadcast news, the design of Etel++ has been done in order to deliver a pilot application close to concerns of electronic press publishing.

TCM has present Follow-Me and especially Etel++ to the partners of the French SPQR. The first presentation in Paris (April the 1st) was an overall description of the concepts, the second

one (13th of May) was done to address potential customers and to get their expectations in regard with their further project.

With the first prototype of Etel++, three presentations have been done:

Date, Location	*Participants*	*Contents*
23rd of September, Rennes	Ouest-France Antoine de Tarlé (vice-president) Jean-Paul Boucher (Technical Director)	Presentation of FollowMe Etel++ pilot application. Discussion about the commercial potential strength of these application in the press market
29th of September, Rennes	Atlantel (Multimedia Subsidiary of Sud-Ouest second major French regional daily newspaper after Ouest-France) Jean-Lois François (Director) Bernard Lafitedupond (Technical Director)	Presentation of FollowMe Etel++ pilot application. Discussion on potential extensions and commercialisation with their electronic newspaper.
7th of October, Rennes	Precom (Ouest-France's advertising production agency) Philippe Toulemonde(Director) Serge Fiedler (account executive)	Presentation of FollowMe Etel++ pilot application. Discussion of would be possible to show (promotional sale ...) when you move inside the Ouest-France's regional area. Potential cost of this service for advertiser

5 Deliverables

5.1 Final Reports

In the following you find a list of final reports, as defined in the technical annex of the project.

The report deliverables are organised in documents. In order to simplify the organisation of the sheer amount of deliverables, related deliverables where combined into one document.

- WP A: Will Harwood et al.: *Architecture*, (DA1.3)
- WP B: Richard Hayton, *Mobile Object Workbench v2.0*, (DB 7.4, DB 8.3)
- WP C: Douglas Donaldson, Richard Hayton, *Information Space* (DC 4, 5.2, 6.3)
- WP D, E, F: St. Battle, N. Taylor, J. Tidmus, M. Yearworth, *Agent Framework Guide* (DD5.3, DD6.4, DE5.3, DF5.3, DF6.3)
- WP G: L. Amsaleg, et al., *Service Deployment Design*, (DG3, DG4)
- WP G: L. Amsaleg, et al., *Service Deployment Final Report*, (DG 6.2)
- WP H: M. Breu, et al., *User Access Software Report*, (DH 6.3)
- WP I: H.-G. Stein, et al., *Pilot Application 1, Evaluation Report*, (DI5)
- WP J: L. Amsaleg, et al., *Pilot Application 2: Description of the Working System*, (DJ5)
- WP K: FollowMe Project Board: *Exploitation Plan*, (DK2, restricted)
- WP L: M. Breu et al., *FollowMe Final Report* (DL3)

5.2 Deliverable Status

The following list shows the status of all deliverables:

- *Internally available*: The deliverable was produced and distributed project-internally. It either serves as an internal basis for technical decisions and preparation of the further deliverables or is a software product that is distributed project-internally. Those deliverables are not handed over to the reviewers. But they can be made available on demand

- *released*: The deliverable has successfully undergone a formal project-internal review process.

Deliverable	*Name*	*Type*	*Month*	*Status*
DA1.1	Architecture Report	Report	2	Internally available
DA1.2	Architecture Report	Report	6	released
DA1.3	Architecture Report	Report	12	released
DB1	Survey	Report	1	Internally available
DB2	Requirements	Report	2	released
DB3	Design	Report	3	released
DB4	Interface Specification	Software	3	released
DB5.1	O/S Objects	Software	4	Internally available
DB5.2	O/S Objects	Software	7	released
DB6.1	Object Locator	Software	4	Internally available
DB6.2	Object Locator	Software	7	available
DB6.3	Object Locator	Software	9	released
DB7.1	Mobile Object Workbench	Software & Report	4	Internally available
DB7.2	Mobile Object Workbench	Software & Report	7	Internally available
DB7.3	Mobile Object Workbench	Software & Report	9	internally available
DB7.4	Mobile Object Workbench	Software & Report	12	released
DB8.1	Mobile Data Object	Software & Report	5	Internally available
DB8.2	Mobile Data Object	Software & Report	7	Internally available
DB8.3	Mobile Data Object	Software & Report	9	released
DC1	Requirements	Report	3	released
DC2	Design	Report	4	released
DC3	Interface Specification	Software	5	Internally available
DC4	Object Sharer	Software	9	available
DC5.1	User Authentication	Software	9	Internally available
DC5.2	User Authentication	Software	13	released
DC6.1	PIS Object	Software & Report	6	Internally available
DC6.2	PIS Object	Software & Report	9	released
DC6.3	PIS Object	Software & Report	13	released
DD1	Survey	Report	3	Internally available
DD2	Requirements	Report	4	released
DD3	Design	Report	5	released
DD4	Interface Specification	Software	6	released

Deliverable	*Name*	*Type*	*Month*	*Status*
DD5.1	Task Agent Shell	Software & Report	7	available
DD5.2	Task Agent Shell	Software & Report	10	released
DD5.3	Task Agent Shell	Software & Report	13	released
DD6.1	Personal Assistant	Software & Report	7	internally available
DD6.2	Personal Assistant	Software & Report	8	released
DD6.3	Personal Assistant	Software & Report	10	shifted to PM 13
DD6.4	Personal Assistant	Software & Report	14	released
DE1	Survey	Report	2	Internally available
DE2	Requirements	Report	3	Internally available
DE3	Design	Report	4	released
DE4	Interface Specification	Software	4	released
DE5.1	Profile Object	Software & Report	5	delivered
DE5.2	Profile Object	Software & Report	8	released
DE5.3	Profile Object	Software & Report	11	released
DF1	Survey	Report	2	Internally available
DF2	Requirements	Report	4	Internally available
DF3	Design	Report	5	released
DF4	Interface Specification	Software	6	released
DF5.1	Service Shell	Software & Report	7	available
DF5.2	Service Shell	Software & Report	10	released
DF5.3	Service Shell	Software & Report	13	released
DF6.1	Service Directory	Software & Report	7	released
DF6.2	Service Directory	Software & Report	10	released
DF6.3	Service Directory	Software & Report	13	released
DG1	Survey	Report	3	Internally available
DG2	Requirements	Report	4	Internally available
DG3	Design	Report	6	released
DG4	Interface Specification	Software	7	available
DG5	Group Profile Analyser	Software	8	available
DG6.1	Service Deployer	Software & Report	12	shifted
DG6.2	Service Deployer	Software & Report	14	released
DH1	Survey	Report	2	Internally available
DH2	Requirements	Report	3	Internally available
DH3	Design	Report	4	released
DH4	User Interface Language	Report & Software	5	released
DH5.1	Device Adapters	Software	6	internally avail.
DH5.2	Device Adapters	Software	9	available

Deliverable	*Name*	*Type*	*Month*	*Status*
DH5.3	Device Adapters	Software	13	internally available
DH6.1	User Access Module	Software & Report	6	internally avail.
DH6.2	User Access Module	Software & Report	9	released
DH6.3	User Access Module	Software & Report	13	released
DI1	Survey	Report	3	Internally available
DI2	Requirements	Report	6	released
DI3	Design & Objectives	Report	8	Internally available
DI4.1	Working system	Software	10	available
DI4.2	Working system	Software	15	released
DI5	Evaluation Report	Report	18	
DJ1	Survey	Report	3	Internally available
DJ2	Requirements	Report	6	released
DJ3	Design & Objectives	Report	8	available
DJ4.1	Working system	Software	10	available
DJ4.2	Working system	Software	15	available
DJ5	Evaluation Report	Report	18	released
DK1	Agreement on IPR	Report, External	6	available (with DL 1)
DK2	Consortium Exploitation Plan	Report, External	15	released
DL1	Consortium Contract	Contract	3	internally available
DL2	Project Progress Report	Report	6	released
DL3	Project Progress Report	Report	12	released
DL4	Final Project Report	Report	18	(this document)

Annex A

Project Meetings

The following project meetings took place during the reporting period. Minutes and/or slides are available on the project server.

Date	Location	Meeting
15th/16th Oct. 97	Munich	Project kick-off meeting (all)
29th Oct. 97	Bristol	Integration of APM and UWE work packages (APM, UWE)
30th/31st Oct. 97	Rennes	Technical meeting ETEL requirements and User Access (INRIA, TCM, FAST)
10thNov. 97	Windsor	MOW and Agent integration (APM, UWE)
17th/18th Nov. 97	Bristol	Technical project meeting (all)
11th/12th Dec. 97	Munich	Technical project meeting and board meeting (all)
3rd/4th Feb. 98	Cambridge	Design walkthrough of MOW (APM, UWE)
26th/27th Feb. 98	Rennes	Technical project meeting (all)
22nd-24th April	Cambridge	FollowMe Team Meeting and Management Board
27th-28th May	Bristol	Meeting between UWE and FAST to discuss implementation of Pilot Application (WP I) using Agent Framework
9th June	Brussels	Preparation for 1st Review
10th June	Brussels	FollowMe 1st Review
27th – 29th July	Munich	FollowMe Team Meeting
30th September – 2nd October	Rennes	FollowMe Team Meeting and Management Board
13th – 15th of January 99	Munich	FollowMe Team Meeting and Management Board

Roster of Personnel on the Project

The following staff members contributed to the project.

Company	*Name*	*Role in the Project*
APM	M. Bursell	Software Engineer: Architecture and Mobile Workbench
	D. Donaldson	Software Engineer: Architecture, MOW and Personal Information Space
	D. Franklin	Software Engineer: Architecture and Personal Information Space
	W. Harwood	Software Engineer: Architecture and MOW
	R. Hayton	Software Engineer: Architecture and MOW
	A. Herbert	Project leader at APM, Project Board , Chief Architect
	R. Chiltern	Software Engineer: MOW
	J. Cooper	Software Engineer: MOW
	M. Madsen	Software Engineer: Internal Review
	T. Ugai	Software Engineer: Security
FAST	M. Breu	FollowMe Project Manager, software engineer
	L. Gebauer	Contact Manager: Pilot Application
	R. Haggenmüller	Project Board
	H. Nandasena	Project Assistant
	S. Pöllot	Software Engineer: User Access and Pilot Application
	P. Scheideler	Software Engineer: User Access
	J. Pitadenyia	Software Engineer: User Access and Pilot Application
	A. Rajakarunana-yake	Software Engineer: Pilot Application
	R. Krutisch	Software Engineer: Pilot Application
	H.-J. Buchberger	Software Engineer: Pilot Application
	H.-G. Stein	Software Engineer, Work package co-ordinator WP J (Pilot 1)
	A. Sindermann	Software Engineer: User Access
	E. Triep	Work package co-ordinator WP H (User Access)
	R. Sembacuttiara.	Software Engineer: Version Management
	H. Köhler	Project Assistant
	S. Radspieler	Software Engineer: Pilot Application
	T. Delpagoda	Software Engineer: User Access
	F. Matulic	Software Engineer: User Access
INRIA	L. Amsaleg	Full time engineer: ETEL++
	M. Banatre and V. Issarny	Project leaders at Inria
	M. Billot	Full time engineer: WP-G (Service Deployment)
	P. Couderc	PhD student: mobility of documents
	A-M. Kermarrec	Researcher: mobility of documents
	J-P. Routeau	Engineer, helps in building the bridge between ETEL and ETEL++
	A. Chafaqi	Full Time engineer (integration tasks)
	J B. Charpiot	Ph. D Student, Investigations related to the use of Data Mining within the project.
	F. Le Mouël,	Ph. D Student, Investigations related to context-aware information retrieval.
TCM	M. Le Nouy	Engineer: Etel++.
	C. Philibert	Project leader at TCM, project board,
	B. Toullier	Engineer: Etel++,

Company	*Name*	*Role in the Project*
UWE	S. Battle	Software Engineer and Researcher: WP D
	L. Bull	Project Mentoring: WP D, E and F
	N. Taylor	Software Engineer and Research: WP F
	J. Tidmus	Software Engineer and Research: WP E
	M. Yearworth	Work Package leader for WP D, E and F, FollowMe Management Board and Project leader at UWE, Exploitation activities

References

[1] XSL: A Proposal for XSL (http://www.w3.org/TR/NOTE-XSL.html)
[2] Extensible Style Sheet Language (XSL), Version 1.0, W3C working draft, 18. Aug. 98, http://www.w3.org/TR/WD-xsl
[3] XML: Extensible Markup Language (XML) 1.0, W3C Recommendation 10-February-1998 (http://www.w3.org/TR/1998/REC-xml-19980210)
[4] Will Harwood et al.: Architecture, (DA1.3)
[5] Richard Hayton, Mobile Object Workbench v2.0, (DB 7.4, DB 8.3)
[6] Douglas Donaldson, Richard Hayton, Information Space (DC 4, 5.2, 6.3)
[7] St. Battle, N. Taylor, J. Tidmus, M. Yearworth, Agent Framework Guide (DD5.3, DD6.4, DE5.3, DF5.3, DF6.3)
[8] L. Amsaleg, M. Billot, P. Couderc, V. Issarny, A.-M. Kermarrec, M. Le Nouy, J.-P. Routeau, Service Deployment Design, (DG3, DG4)
[9] L. Amsaleg, et al., Service Deployment Final Report, (DG 6.2)
[10] M. Breu, et al., User Access Software Report, (DH 6.3)
[11] H.-G. Stein, et al., Pilot Application 1, Evaluation Report, (DI5)
[12] L. Amsaleg, et al., Pilot Application 2: Description of the Working System, (DJ5)
[13] FollowMe Project Board: Exploitation Plan, (DK2, restricted)
[14] M. Breu et al., FollowMe Final Report (DL3)