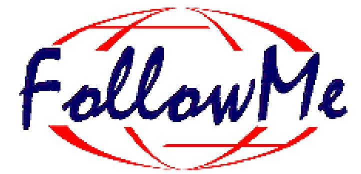


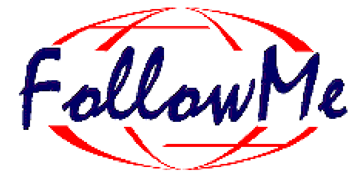
ESPRIT Project No. 25 338

Work package A

Architecture

Architecture Release 1.3

ID:	Architecture release V. 1.3	Date:	30/2/98
Author(s):	Will Harwood	Status:	released
Reviewer(s):	All	Distribution:	



Change History

Document Code	Change Description	Author	Date
Architecture Report DA1.1	Draft version of document for discussion.	Mike Bursell (APM)	23.Oct.97
	New format, completely new input	Mike Bursell, Douglas Donaldson (APM)	27.Nov.97
Architecture Report DA1.21	Complete Rewrite, references cross references and annex I missing	Will Harwood, Douglas Donaldson (APM)	20.Apr.98
Architecture Report DA1.22	Revision and Extension	Will Harwood, Douglas Donaldson (APM)	8.May.98
Architecture Report DA1.3 Draft	Complete Rewrite	Will Harwood	4 Jan 98

INTRODUCTION 1

PROJECT OVERVIEW 2

FollowMe Project Objectives 2

INTRODUCTION TO PATTERNS 9

The Basic Ideas of Patterns 9

Patterns and Ontologies 10

Patterns and ODP 10

An Object Ontology 11

PROJECT PATTERNS 14

A Project Heuristic 14

The Mobile Object Workbench/Information Space and Service Deployment Abstractions 16

The Autonomous Agents Framework and User Access Abstractions 24

The Pilot Applications 28

REFERENCES 31

Introduction

This document sets out the overall architectural framework of FollowMe. This framework consists of three parts: -

- There is a connect collection of concepts, or ontology, that defines domain of discourse of FollowMe. The ontology of FollowMe is drawn largely from the Reference Model for Open Distributed processing and as such we assume familiarity with [1] and only introduce extensions to the ontology as required by FollowMe.
- There are simple heuristics that have guided the development of FollowMe. The primary heuristic has been decoupling which is discussed in detail in the section Project Patterns.
- Finally there is the notion of Pattern [2,3,4,5]. A pattern is an approach to solving a specific kind of recurring problem in a specific context. Patterns are not pieces of code but approaches to solving problems (that may involve specific pieces of code, but again they may not). Patterns are an attempt to unify the approaches to design across or levels of detail.

Our purpose in setting out this framework is to enable re-use of the architectural ideas in future projects. As such this document does not conform to the structure of a standard scientific paper but rather is a FollowMe specific “Architecture Dictionary” to be used by the software architects of future projects.

This document has three main parts: -

- An introduction to the objectives of the project.
- An introduction to the use of patterns.
- The collection of patterns split into the three parts: -
    - The Mobile Object Workbench/Information Space and Service Deployment abstractions.
    - The Autonomous Agents Framework and User Access abstractions.
    - The Pilot Applications.

This document does not describe the designs of the work package components. Details of where to find design information from the individual work packages can be found at the end of the Project Overview (next section).

This document attempts to provide a high level view of what, overall, each work package is trying to achieve and acts as a repository for specific patterns. Theses patterns are chosen because we feel that they may be the basis of future design work on other projects.

Project Overview

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.

FollowMe Project Objectives

FollowMe opens up access to the global networks to a new class of users. These users are mobile and only connected to the network for part of their time. Moreover their point of connection may change rapidly. For example the executive may move from home, to car, to office, to airport, to hotel, in a single day. During such a day’s travel the executive will want to establish contact with his office, peruse his personalised newspaper, check his stock options and plan his weekend. Moreover he will expect to be informed of significant events in his office environment, of significant changes to his stock portfolio, of the results of searches he has set in motion for data relating to his trip etc. All this information should be delivered by the most appropriate available means. This will depend on the executive’s current location and the devices that are available at that location and on characteristics of the data itself, such as whether interactive connection is required, the sensitivity of the data, etc.

FollowMe augments the current paradigm of distributed computing as, for example, captured in the ODP reference model [1] with two powerful new paradigms for information system. 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.

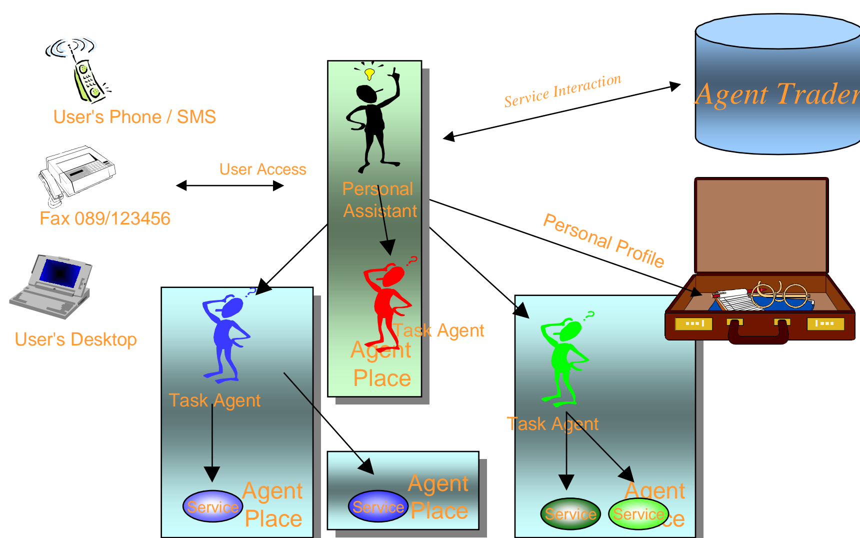


Figure 1: Agent Framework

To explain how these two paradigms interact we consider a simple applications scenario. 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 agent infrastructure for such a task is illustrated in Figure 1: Agent Framework.

This high level picture of the search task is supported extensively 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 “cluster” from ODP that 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.

The infrastructure supporting the agent picture above is schematically portrayed in Figure 2.

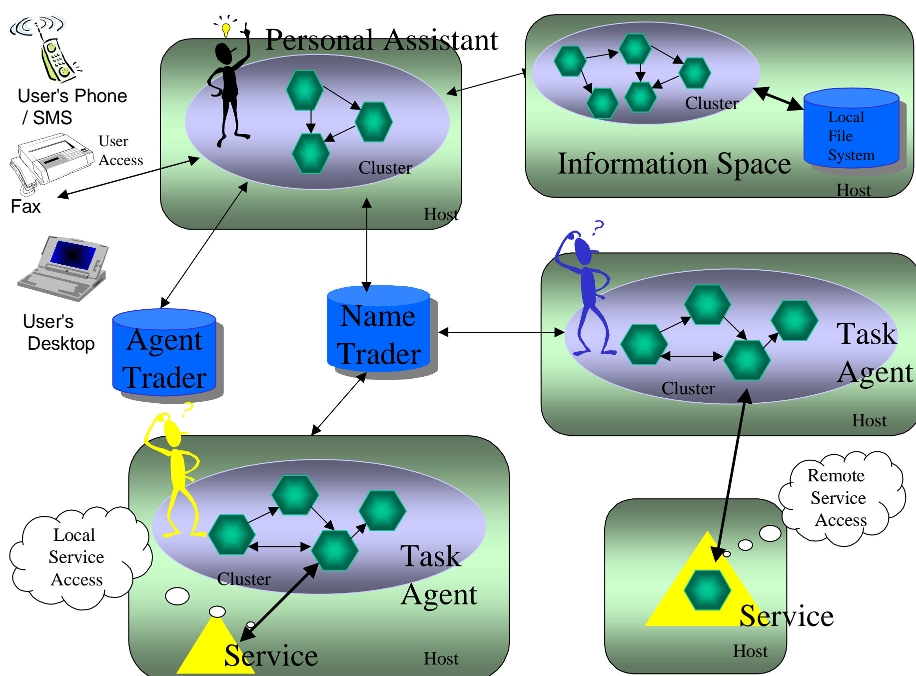


Figure 2 : MOW Infrastructure for an Application

In order to illustrate the potential of the technologies concerned, FollowMe is creating two pilot applications. The goal of these applications is to validate the approach in real-world situations and provide direct exploitation paths. The pilots are drawn from two different spheres ensuring that FollowMe technology will be generally applicable. *Ouest France*, the largest regional newspaper in France, will be the basis of one pilot, with the infrastructure of the Internet service *Bavaria Online* providing the other.

The project work has been divided into discrete units, the outputs of each being designed to be both beneficial as component parts of the project and in their own right. When combined, these component parts form a framework capable of supporting applications that meet the FollowMe goals.

The FollowMe project may be seen as three inter-related streams of activity. The first stream provides a framework for building mobile distributed applications. The second stream builds on this

work to create an "autonomous agent framework" that allows agency to be expressed by a natural and high level set of abstractions. Finally the third stream consists of the pilot applications.

The first stream consists of the work packages: -

Mobile Object Workbench

The Mobile Object Workbench (MOW) provides a platform for building distributed (code) mobile applications. In FollowMe it, together with Information Spaces (see below), provides a foundation layer for the development of new services and applications.

Information Spaces

A key requirement of the mobile user is the ability to maintain and access their own information. An information space will enable a user to access data through one consistent logical view irrespective of their, or the data's, location by providing transparency mechanisms enabling the location of data to be hidden. In addition, the system will enable applications to perform the automatic movement of subsets of the data to achieve quality of service objectives. Information spaces are an enabling technology that, for example, make it possible for employees to freely travel between company sites while still having access to their normal range of information.

Service Deployment

Anticipating the level and location of demand for a service is key to its efficient deployment. Service Deployment provides the tools for monitoring service usage and feeding that information back to interested applications. Applications may then use this information to make decisions about how and where to deploy services and data. In order to facilitate predictive load management an attempt will be made to project usage patterns from data in the Personal Profiles.

The second stream consists of the work packages: -

Autonomous Agents

The rate at which services and facilities become available on the Internet will increase. It is essential that users be provided with alternative means to locate, monitor and interact with information providers and electronic vendors. The Autonomous Agents framework automates many aspects of finding and using services, enabling users to specify objectives and have these tasks carried out while they are disconnected from the network. The project will implement a number of instances of such agents. This will validate the approach and provide a guide for other applications. FollowMe is developing the infrastructure and security mechanisms to support very large numbers of agents operating in a wide area distributed environment.

Personal Profiles

Autonomous Agents need to be able to make decisions even when they cannot directly interact with the user. FollowMe provides a profile for each user, holding a combination of facts and derived information such as their personal preferences. This will be used to direct agents and reduce the need to refer to the user for supplementary information. The work will yield a modular encapsulation of a users' profile. It explores what essential facts need to be provided and which mechanisms are required to deduce preferences. Although the pilot applications will be used to drive the requirements

of this work, the approach is, as far as possible, be generic. A modular approach enables the internal mechanisms to be easily improved or substituted.

Service Interaction

Current services on the Internet are designed for human use. However, in order for applications to monitor, interrogate and interact with these services, an alternative interface is required. FollowMe is developing a template for creating services, a directory capability for locating them and an interface library for interaction. This enables service providers to unilaterally enhance their service and provides a means to discover the services available.

User Access

This work package provides mechanisms to enable services to be presented through a range of user interface devices. It has implemented tools to support a number of common devices. However, the approach will be generic, enabling other devices to be incorporated without the requirement to modify the service or the basic mechanisms of User Access.

The third stream of activity implements prototype applications that illustrate both how the mobile user may be supported and how code and data mobility, encapsulated in mobile objects and storable objects (see below) provide a better programming paradigm for the development of these applications. This stream consists of two pilot application activities: -

Pilot 1

Pilot 1 is intended to validate the agent concept in a part of the Internet called Bavaria On-line. The Bavaria On-line project is sponsored by the Bavarian government and operates a part of the Internet. The Bavarian Citizen networks act as regional information and service providers for everyone. Servers (information brokers/service representatives) will be installed on the network, providing the following personalised services:

1. Provide information on the status of one's stock portfolio
2. Provide an information and alerting service on regional events (entertainment, politics, education, etc.).

The two applications were identified because they require different access through agents. For the first application, a user can indicate the content of his/her stock portfolio. Through a match maker (yellow pages) he accesses the single service provider which returns the information requested through the task executor (the agent). For the second application, a variety of different service providers will be available and information will need to be collected, summarised and presented to the user through the interaction of multiple agents.

Pilot 2

The ETEL electronic newspaper service that is developed by INRIA, Ouest-France, TC-multimedia, and O2 Technology (a well-known company in the database area) aims at providing a service with the following features:

- Coupled production of paper and electronic editions, i.e. production of the two versions from the same data.
- Presentation of the information that combines the advantages of both the paper version and the electronic support.
- Integrated view of newspaper-based information and links to services.

- Addressing Quality of Service (QoS) requirements (responsiveness, scalability and availability).

A first prototype of ETEL is operational since August 1996. Its architecture is based on the client-server model, the server managing the data base and readers accesses. The reader accesses the service through a dedicated interface available for PCs running Windows-95. The communication system is based on the ISDN network. The next step of the ETEL prototype, will be the integration of our solutions to ETEL QoS requirements (responsiveness).

For many reasons (accessibility, deployment, etc.), it is clear that the ETEL prototype will have to evolve to support *customisation* of the electronic newspaper content from the standpoint of both the reader's profile and the reader's geographical location. While the former type of customisation is already treated in the ETEL current prototype, the latter is not addressed due to dynamic issues as illustrated by the two following scenarios.

As a first scenario, we consider the "theatre" service provided by ETEL. In the current ETEL prototype, such a service lists movies played in pre-selected cities registered in the ETEL database. Let us now imagine that one ETEL user living in Rennes is going to travel to Cambridge (GB) for two weeks. When he/she is connecting to ETEL and accesses the theatre service from Cambridge, he/she wants to know movies played in Cambridge and not those played in Rennes (F). To implement such a facility, one solution would be to implement the theatre service as an intelligent agent that can take into account that the reader is currently in Cambridge. Then, when he/she is accessing the theatre service through ETEL, he/she gets the list of movies played in Cambridge.

The second scenario is concerned with an ETEL user, who is travelling to New York. If he/she wants to access the ETEL service each morning in order to get news from France, the best solution today is to use Internet. However due to the network traffic, he/she might have to wait a long time before reading the content of the newspaper. An alternative solution would be that the newspaper "follows" the reader in such a way that when he/she connects to ETEL, a copy of the newspaper already exists on a site located in New York. Thus, long distant accesses become similar to local ones.

The implementation of dynamic customisation raises several problems, including:

- the location of ETEL users,
- the management of user accesses from a variety of locations using diverse access points,
- the use of the agent technology including the management of agent co-operation,
- the management of data access integrating mobility aspects.

To our knowledge, there is no available technology that solves the above problems. On the other hand, the FollowMe architecture will provide the necessary support for enriching ETEL with dynamic customisation, leading to the ETEL++ newspaper service.

These example applications do not represent work that is integral to the architecture of a FollowMe system, rather they represent how components from the FollowMe kit of parts can be assembled to produce working applications. These applications are extremely important to the health of the project. They provide vital feedback from the "real-world" and indicate how deliverables need to be modified or revised to account for the needs of real applications (this has already occurred with the change in conception that has taken place in work package C which has changed so as to provide a more flexible facility for persistent information storage).

Detailed documentation on the design of the various components can be found in: -

- Mobile Object Workbench [6]

- Information Spaces [7]
- Service Deployment [8]
- Autonomous Agents [9]
- Personal Profiles [10]
- Service Interaction [11]
- User Access [12]
- Pilot 1 [13]
- Pilot 2 [14]

Introduction to Patterns

The Basic Ideas of Patterns

Design patterns are a way of talking about the abstract regularities in a software design. By conforming to a set of design patterns it is possible to achieve piecemeal growth whilst maintaining an overall design consistency between separately developed parts of a system so that they exhibit an overall regularity in structure and behaviour and fit together in a natural way. Patterns are "problem solving heuristics" presented in a context specific form. Adopting a common set of patterns within a system solving similar problems by similar means so that each part of the system "talks the same language" as other parts of the system. This means that similar problems are address in similar ways throughout the system, rather than by an ad hoc collection of solutions.

"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution" [2]. Moreover each pattern: -

- "…is the **abstraction** from a concrete form which keeps recurring in specific non-arbitrary contexts" [3],
- expresses an **encapsulation** of a separable aspect of an overall design task,
- expresses a balance or **equilibrium** between opposing requirements,
- is **open** to extension and modification,
- is **composable** with other design patterns,

Each design pattern is described by a: -

Name: A short descriptive name or phrase that is usually indicative of the solution rather than of the problem or context (see below).
Context: A description of context for the use of the pattern.
Problem: A description of the conflicting forces to be resolved and constraints placed upon the resolution, and how these interact with one another.
Solution: A description of components of a solution, their relationship to one another and the rules of construction by which the solution may be achieved. Ideally the solution for a pattern list several variants of the solution and/or ways to adjust the solution to varying circumstances. Solutions refer to other patterns in the pattern hierarchy.
Cross-reference: To the patterns this pattern reifies or which are used as components of this pattern.
Examples: Illustrations of the use of the pattern by concrete example.

Patterns are a tool for expressing structural regularities which are reused across a design. However one should be aware that the notion of structure goes beyond the simple static relationship between physical or software components and extends to structures in time (dynamics), structures in society (organisational structure) etc. That purpose of patterns is to make designs "more harmonious" by allowing the same themes to be repeated in indefinite variation throughout. Software Patterns are not to be identified with particular pieces of code or particular objects, although they may be realised by these in a particular design. Rather patterns have the flavour of interfaces and algorithms. They define what objects can do or the abstract processes by which they may do something rather than the concrete particulars.

Patterns and Ontologies

Designs do not exist in isolation, rather they are defined in terms of an ontology, a connected collection concepts that delineates a domain of discourse. In practice an ontology is open ended in that new concepts are being added along with new discoveries and inventions. Patterns define a set of problem/solution pairs within a domain of discourse. The ontology provides a set of anchor points and an extensible framework that supports a continuing and expanding dialogue about a particular class of designs.

In practice our job is to provide an ontology (or reference model) adequate for the discussion of partially connected users and mobile objects and a set of patterns for solving specific problems that arise in this context. Partially connected users arise in the context of user mobility. Users move between locations. They may either move from one location to another, and may use different machines at each location, or they may take their personal machine with them and disconnect it from one point in a network and re-connect it at another. Users may wish tasks to execute while they are disconnected from the system and results, mail etc. may accumulate for them until they re-connect. User may not have any fixed association with a "home location" or a "home machine".

Patterns and ODP

ODP is an ontology for open distributed processing design. It provides a set of basic patterns for discussing and designing open distributed systems.

The architectural aspects that will be covered in detail in this document are those that relate specifically to user and code mobility, to the use of autonomous agents and to the specific applications that have been used to demonstrate the FollowMe concept.

The ODP viewpoint model provides a way of structuring the ontology of Open distribute Systems into groups of concepts related to different aspects of a system.

The ODP viewpoints are: -

The **enterprise view** defines the intentional view of a system. Enterprise entities are intentional in that they have purpose. These entities "have responsibilities", "become obligated", "are permitted or prohibited", "have goals and commitments" etc. Enterprise relationships express the relationship between the intentions of different entities.

The **information view** defines the extensional structure of a system. The information objects represent pieces of information and the relationships represent invariant between those pieces of information.

The **computational view** defines which objects are responsible for maintaining the invariant.

The **engineering view** defines how the objects are distributed.

For each viewpoint we may speak of a corresponding **model** or ontology which defines the kind of things can be said in each viewpoint. Thus, in particular, we need to speak of the **computational model** that defines a system in terms of objects that interact through interfaces that enable the system to be distributed.

The ODP computational model defines a number of **transparencies** or abstractions that allow the application programmer to separate concerns in the program. These transparencies are realised in the engineering view.

Access transparency: which masks differences in data representation and invocation mechanisms to enable interworking between objects.
Failure transparency: which masks from an object the failure and possible recovery of other objects (or itself) to enable fault tolerance.
Location transparency: which masks the use of information about *location in space* when identifying and binding to an interface
Migration transparency: which masks from an object the ability of the system to change the location of that object.
Relocation transparency: which masks relocation of an interface from other interfaces bound to it.
Replication transparency: which masks the use of a group of mutually behaviourally compatible objects to support an interface.
Persistence transparency: which masks from an object the deactivation and reactivation of other objects (or itself).
Transaction transparency: which masks co-ordination of activities amongst a configuration of objects to achieve consistency.

In terms of the ODP ontology the extension that is required for mobile objects occurs in the computational view with respect to the notion of location transparency. FollowMe mobile objects need to violate location transparency and be partially location aware. In FollowMe this location awareness is supported by **reflecting** aspects of the ODP engineering model into the FollowMe computational model. Thus allowing application programmers to create objects which are location aware and that can change their location.

An Object Ontology

Throughout the project there is pervasive use of the “object pattern”. Since much has been written on the use of objects in software design we take objects to be an “understood” pattern. From our perspective the key feature of the object pattern is that “data” and the meaning that is assigned to “data” by operations are treated as a single, indivisible, entity.

In discussing design patterns it is useful to have a standard vocabulary for various attributes of objects or collections of objects.

Object Behaviour Concepts

Active Object: An object (or object group) is active if it has its own threads. Such an object may perform actions whether or not it is interacting with a client.
Autonomous Object: An object (or object group) is autonomous if it is self managing. That is, the object may be requested to perform certain actions by services or execution vehicles but it is up to the object to decide whether or not to perform the action. Autonomy is always limited in practice in that an execution vehicle always has recourse to lower level interfaces that allow it to operate on objects.
Persistent Object: An object (or object group) which exists beyond the lifetime of the program that uses it and beyond the lifetime of any particular execution vehicle.

Reactive Objects: An object (or object group) is reactive if it has no internal threads, i.e. the object is dormant until it acquires threads from a client via a method invocation, it performs its method and becomes dormant again upon the method return.

Object Grouping Concepts

Execution Vehicle: An execution vehicle is an (abstract/virtual) machine capable of executing a cluster.
Object Grouping: Collections of objects may be grouped together to form a "collective objects". Such collectives are defined by a common encapsulation boundary which mediates the interactions between the collective and other collectives. Access between collectives is achieved by the collective exporting interfaces for some of its members across the boundary. Object grouping is hierarchical in that collections of objects and collectives may be grouped to form a larger collective object.
Object Clustering: Behavioural components of a system are represented by collocated collections of objects called clusters. Collocated means that objects reside on the same execution vehicle. Clusters are collective objects.

Meta-Objects Concepts

Introspection: Introspection is the act of examining meta-data about an object to obtain details of the structure of the object. Use of the introspection pattern requires the creation a suitable form of meta-data and its attachment to the objects being described. In practice such meta-data is an object with a particular signature.
Meta-Objects: The meta-object pattern allows us to modify the behavior of an object (object groups) by intercepting the communications to and from an object and transforming the intercepted messages or choosing to enact some entirely new behavior. Meta-Object allows us to transparently and dynamically extend, restrict or otherwise change the behavior of an object.

Name Binding Concepts

Location Transparent Reference: Interface references between object clusters are decoupled from the actual location of the cluster. This means that an object possessing a location transparent reference to an interface may use that interface irrespective of the actual location of the object supporting that interface (although subject to other constraints such as security which may restrict the rights of objects to use the functionality offered by the interface).

Dynamic Binding: Dynamic binding is the contextually dependent binding of names to values/references.

Service Concepts

Factory: A factory is a service that manufactures objects (or object groups) of a particular kind from other objects (object groups) provided as “raw materials”. Normally a factory is reactive.
Server: A server is an object group that offers a particular set of interfaces. The behaviour offered over an interface is called a service. Servers may have internal state and may be reactive or autonomous.
Service Directory: A service directory is a Service Map where the service description is a (syntactic) name for the service.
Service Map: A service map is a service that maps between some form of service description and service interfaces.

Project Patterns

A Project Heuristic

Throughout the project there is the recurring heuristic theme of decoupling. This heuristic occurs at a high level in the applications, in which software intermediaries are used to decouple end users of information from producers of information; the idea recurs with agents that decouples “the user” from “the application” the user is running and with “user access” which decouples the form and content of information. At lower levels of abstraction the theme continues with the adoption of the ODP engineering transparencies that decouple objects from (physical) locations.

The general notion of decoupling can be broken down into two typical cases. In the first case illustrated in Figure 3 two objects that interact are contained within the same administrative or physical boundary. Decoupling is achieved by building an infrastructure to support the interaction that allows the objects to be placed in separate administrative or physical boundaries.

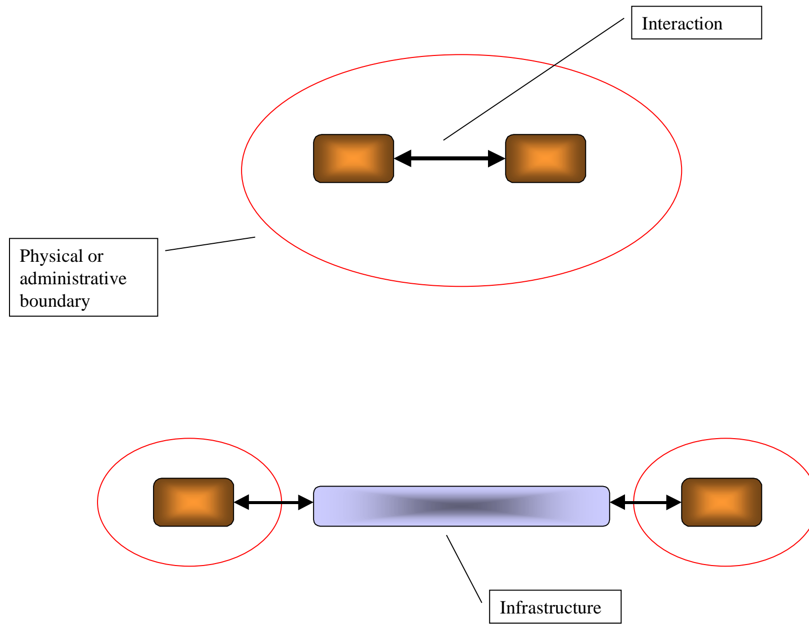


Figure 3: Decoupling of Components

The notion of boundary is quite general and may constitute the boundary imposed by a process space, a machine, a network, a physical location, a time etc.
The second form of decoupling occurs when a design in which two logically distinct concepts are jointly implemented by a single mechanism is replaced by a design in which the two logically distinct concepts are implemented by different mechanisms. This is illustrated in Figure 4: Decoupling of Implementation Mechanisms

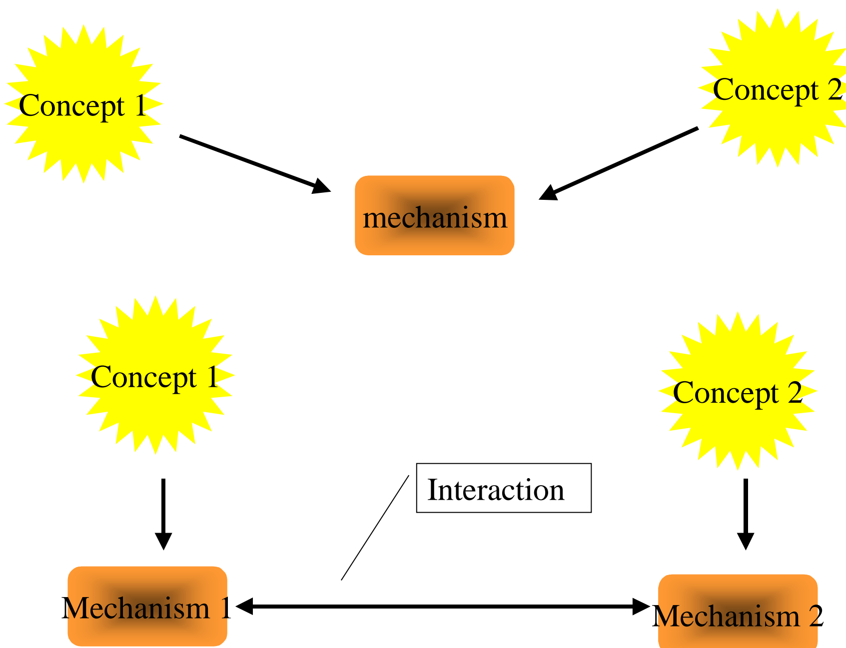


Figure 4: Decoupling of Implementation Mechanisms

The Mobile Object Workbench/Information Space and Service Deployment Abstractions

The Mobile Object Workbench and Information Space provide abstractions that support mobile and persistent objects respectively. They both build upon the notion of clusters. Clusters are collections of co-located objects that need to be managed as a group. For example, a mobile agent is a collection of objects that are required to move together between locations and a "persistent object" is a cluster that is moved between active memory and backing store. A cluster decouples objects within the cluster from objects outside the cluster in that references between objects within different cluster are "location and migration transparent". Clusters are entities in the computational viewpoint that are supported by infrastructure provided in the engineering viewpoint.

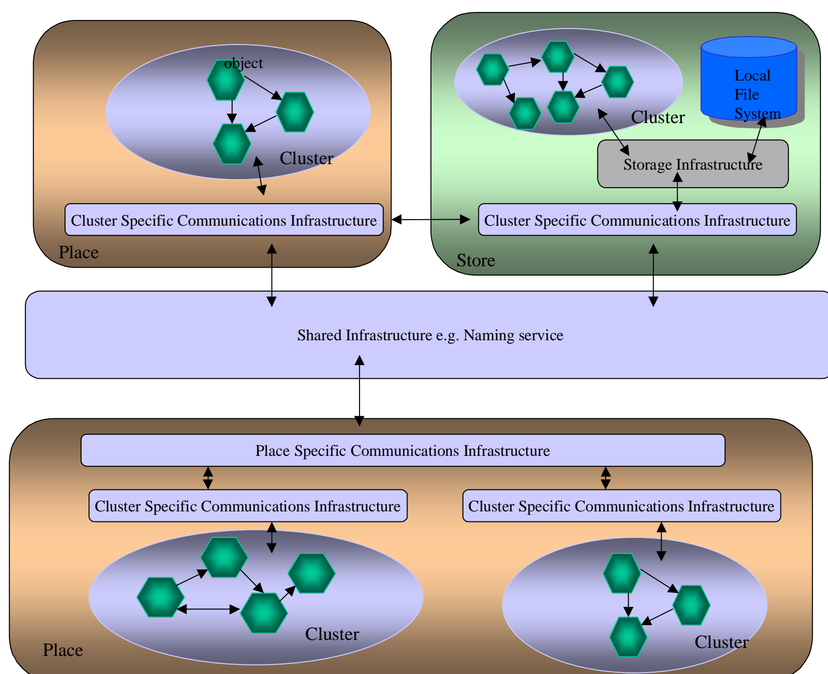


Figure 5: MOW/IS Clusters and Infrastructures

Name: **Cluster Abstraction (Computational)**
Context: Distributed Object Based Systems
Problem: A number of co-located objects need to be *managed* as a group. For example, the objects may comprise an untusted piece of code that is to be isolated from its environment, or other objects in the same location. The objects may be the components of an agent that should be migrated as a unit, or may be the components of a set that must be accessed in a controlled manner.
Solution: Clusters are a computational abstraction for 'groups' of objects. The ODP definition of cluster is "A configuration of basic engineering objects forming a single unit for the purposes of deactivation, checkpointing, reactivation, recovery and migration". We widen this definition to include groups of co-located engineering objects that need to be managed in other ways, for example for security, scheduling or version management. Cluster instances are created by *Capsules*. Different (types of) capsule may be capable of creating clusters which embody different management policies. Upon cluster creation, a capsule may (or may not) relinquish some of the management of the cluster to the

cluster itself, or to a third party. An *autonomous cluster* is one where the cluster's interaction with its environment may only be controlled from within the cluster.

Example: Example cluster abstractions used within FollowMe include *Mobile Clusters* – autonomous clusters that are capable of moving between Places (A place is a type of Capsule that supports Mobile Clusters), and *Storable Cluster* – clusters that are transparently persistent.

Name: **Clusters (Engineering)**
Context: Distributed Object Based Systems
Problem: Realisation of Computational Cluster Abstraction
Solution: Isolation is achieved by removing the (direct) use of shared resources. Each newly created cluster is therefore given its own instance of any normally shared resource, in particular, it is given its own instance of the middleware platform used to achieve distribution transparency. As the middleware platform provides its sole means of communication with other local or remote objects, the cluster is effectively isolated from them. This is sufficient to provide a cluster abstraction. To add utility to this abstraction, a mechanism is required to *reflect* the cluster's interaction with its environment, so that this may be controlled and co-ordinated. In particular, we wish to reflect communications made between objects in different clusters, which take place using the middleware platform. This reflection is directed to a *Cluster Manager*. This is an object with intimate knowledge of the environment that is used to control the clusters interaction. In particular, if invocations into the cluster may be reflected, then they may be co-ordinated to meet an isolation management policy – for example to enforce locking.
Example: The Mobile Object Workbench uses the Clusters pattern to implement FlexiNet clusters. It reflects method invocation from external clusters. This is used in the Mobile Object Workbench to isolate a cluster whilst it is preparing to move, and in the Information Space to delay an invocation whilst a cluster is retrieved from disc. In addition, *introspection* is used to enumerate exported interfaces from a cluster, to allow the MOW to map these reference to references in a newly created clone of the cluster; and to enumerate objects within the cluster, to allow the MOW and Information Space to serialise a cluster in order to copy it (to a new capsule or to disk).

Name: **Strong Encapsulation – Threads (Engineering)**
Context: Distributed Object Based Systems using Clusters
Problem: A cluster may require a high degree of isolation to allow it to be treated as a distinguished entity by the operating system or other systems outside the scope of the middleware platform providing the cluster abstraction. In particular it may be associated with a different security policy, or with a different threading priority.
Problem2: Different clusters may be mutually distrustful. In an environment where a thread within one cluster may invoke a method on an object in a second cluster it is important that the clusters cannot break the isolation between them. In particular, the caller cluster (or a third party) should not be able to leave the callee cluster in an inconsistent state by unexpectedly terminating the calling thread midway through execution of an invocation. Similarly, the callee must not be able to block a call indefinitely, unless the caller is able to recover independently.

Solution: These two problems may both be tackled by thread management. The solution relies on the provision of a Thread Group abstraction in the underlying environment, and the introduction of additional functions into the middleware platform providing the cluster abstraction. Each cluster is associated with a different thread group. Threads in this thread group may only create other threads that are in the same thread group (or in a child thread group). It is therefore possible to determine which cluster a thread belongs to. This is used as the basis of strong encapsulation. The middleware system providing the cluster abstraction contains the only means for a thread to 'escape' to a new cluster. This must be modified so that threads entering a cluster are replaced with threads in the cluster's own thread group. Thread rendezvous may be used to make this transparent. Each thread group is then linked to a particular cluster, and may be associated with different thread priority models. When a security policy needs consulting, the callee thread may be used to determine the originating cluster – even if the call is not made via the middleware platform. In Java, for example, this allows a different security manager to be associated with each cluster. The thread decoupling also solves the second problem, one cluster cannot adversely affect another by blocking or destroying a thread – each cluster has entirely separate threads, which may be created, destroyed, and timed out independently of each other.

Example: The Mobile Object Workbench uses strong encapsulation to provide mobile clusters that represent Agents.

Name: **Strong Encapsulation – Code (Engineering)**

Context: Distributed Object Based Systems using Clusters

Problem: A cluster may require a high degree of isolation to allow it to be treated as a distinguished entity by the operating system or other systems outside the scope of the middleware platform providing the cluster abstraction. In particular it may use different versions of libraries or system classes.

Solution: This solution relies on the ability to load multiple versions of a library or class at the same time. This is possible in the Java language. Each cluster is associated with a class loader, or set of class loaders. Initial objects created within the cluster are loaded using this class loader. Subsequent objects that they then create will also use this classloader automatically. By configuring the class loader appropriately, it is therefore possible to arrange that a cluster uses a particular version of a library or system class. Other clusters in the same process will be using other class loaders and will be unaffected by this. Note. The middleware system providing the cluster abstraction may itself cause the creation of new objects within the cluster. It is important that it too makes use of the cluster's class loader. Objects passed between clusters on the same machine must be fully serialised and then de-serialised; a deep copy operation would lead to a 'pollution' of the cluster/classloader abstraction.

Example: The Mobile Object Workbench uses strong encapsulation to provide mobile clusters that represent Agents.

Name: **Meaningful Names for Recovery (Engineering)**

Context: Persistent object storage

Problem: If a client process causes the creation of a persistent object in a remote location, and then fails, it is likely that the persistent object will remain, but that the (only) reference to it will be lost. There will therefore be no way of referencing the object, and, more importantly, ever reclaiming the resources it uses.

Solution: The solution is to arrange that when each persistent object is created, a reference to it is atomically created in a persistent log, or directory. This reference to the object can later be retrieved, even if all other references are lost. It is important that the object reference is identified by a name that is *meaningful*, even if the creating process immediately fails. A management process (or human user) can then recognise the name of the object and obtain a reference to it. It is essential that the object creator chooses the name – if the object store were to do this, it would have insufficient information to choose a name that was recognisable by the management process or human user. If the object store manages many clients, it must provide a mechanism to avoid the problem of different clients choosing the same meaningful name for their objects. A (directory, name) tuple may be used for this, where each client is associated with a different directory.

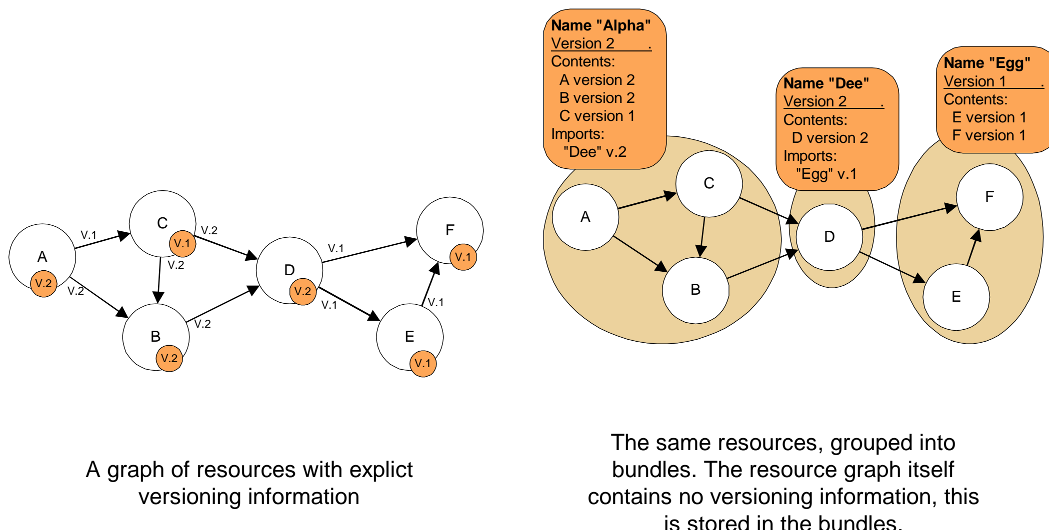
Example: In the information space, a directory hierarchy is associated with each store, to allow the meaningful naming of its contents. This directory is also used to store the names of objects copied from one store to another – as this too may result in unreferenced objects. The directory serves two additional roles, it provides a convenient directory service for application use, and it provides a management interface for 'housekeeping' of stored objects.

Name: **Bundle (Information/Engineering)**

Context: Version management of distributed objects

Problem: In a large distributed system, there may be many versions of particular sub-systems running at any point in time. In a system that supports the dynamic loading or migration of objects, then upon loading, the system must determine which version of a class or other resource to load. In addition, if that resource contains references to further resources, then the versions of those resources to be used must be selected.

Solution: The solution has two parts. Firstly, we arrange that resources are tagged with versioning information, and contain versioning information about the other resource they reference. Secondly, we arrange that when the names or identifiers of resources are passed from node to node, this versioning information is included. The potential overhead of this system is high. To reduce this we introduce the second part of the solution – bundling. Bundles are collections of resources. The resource themselves contain no version information, but the collection as a whole is annotated with version information. Likewise, each resource may contain references to arbitrary resources, and these references are not annotated. Instead, the bundle contains a list of bundles from which other resources are imported. This list is version controlled. This is illustrated in Figure 6.



A graph of resources with explict versioning information

The same resources, grouped into bundles. The resource graph itself contains no versioning information, this is stored in the bundles.

Figure 6

This 'bundling' of resources reduces the version control overhead. It also has another significant advantage; the resources themselves are not 'polluted' with versioning information. This allows bundling to be used as a *deployment time* process, and there is no need to affect the development of the distributed system itself.

Bundling allows the representation of any graph that corresponds to a 'snapshot' of resources – i.e. a graph where there is only one version of each resource. This is the only case we require for the use of bundles within FollowMe. An important decision is the *granularity* of bundles – how many resources should be placed in each bundle. Large bundles reduce the amount of additional information that must be stored for version control – but a new version of the bundle must be created each time one of its constituents is updated. Small bundles have a higher runtime overhead – but most bundles will be unaffected by a change to a single resource. Resource bundles are often used to manage code versions. For this application, an appropriate bundle size is a 'library' or other unit of code that is updated/released as a unit.

Example: Bundles are used in the Mobile Object Workbench's Class Repository to support the deployment of different versions of Java classes. Java classes are naturally collected into 'Jar' files, and in the class repository, each Jar corresponds to a bundle.

Name: **Smart Proxies**
Context: Distributed Object Systems
Problem: In a distributed object system, a reference to a remote object is represented at an engineering level by a reference a local representative or *proxy* for the remote object. In normal 'dumb' proxies, the proxy reflects method invocation by converting it into a form understood by the underlying middleware platform. This then invokes the method on the remote object by communicating with the middleware on the remote machine. For specialist use, this proxy function is too simplistic. For example a distributed application might require that a proxy represents one of a set of remote objects, and dynamically chooses which object to communicate with on a per-invocation basis (to allow replication). Another application might require that a proxy *cache* results from previous

invocations. Such functions cannot normally be provided by a middleware platform, as they are domain specific. Equally, they should not be part of the application-proper as they are engineering objects relating to distribution strategy.

Solution: The solution is to provide a mechanism for 'Smart Proxies'. These are application specific proxies. Typically, a service will decide that clients should use a smart proxy when communicating with it. It will then inform the middleware system of the proxy class to use. This is an object class, which implements the service's interface and which should be deployed on the client in place of a dumb proxy. Typically, smart proxies will also contain an ordinary (dumb) reference to the remote service. In FlexiNet, smart proxy classes must extend a distinguished engineering base class. A service may the pass a client a reference to a smart proxy, and the middleware platform will replicate the proxy onto the client machine. To the client, this is transparent, and they see only a normal remote reference.

Examples: Smart proxies are used within FlexiNet for references to the Trader. If the trader were accessed using dumb proxies, then it would be passed references to many different services, of many different interface classes. (The function of the trader is to act as a directory for services). This would lead to the trader having to load and resolve classes relating to many different applications. This is troublesome both from a security point of view (the trader must be willing to load many different classes), and from a performance point of view. To overcome this problem, a smart proxy is used to convert each interface reference into a packed byte-array format. The trader itself therefore only needs to store byte-arrays, and the security and performance issues are avoided.

Smart proxies may also be used within the ETel application to manage 'smart references' to replicated read-only objects. The proxy may then choose which replica to communicate with, based on the load at that replica, and the estimated bandwidth between the client and each replica.

Name: **Generic Proxies**

Domain: Smart proxies in distributed object computing.

Problem: Smart proxies are type specific – it is therefore necessary to provide one proxy class for each interface that is being proxied. For many applications of smart proxies, the function of the proxy is actually generic. For example a proxy may provide auditing, or locking. For these cases the programmer overhead of using smart proxies is too high.

Solution: A form of smart proxies has been designed that manage invocations in a generic way. These proxies sit below an ordinary dumb-proxy and deal with the invocation after it has been converted to a generic form. Similarly, a *generic-skeleton* may be devised to perform an analogous task on the server. This sits below the actual service object, and deals with the invocation in a generic form immediately before and after it is invoked on the service object.

Examples: Generic proxies are used within the FlexiNet framework to pass contextual information from client to server when the client is involved in a transaction. They may also be used to aid debugging, by passing client identities and vector clocks between client and server.

Name: **Generic Invocation**

Domain: Middleware engineering

Problem: Much of the processing of an invocation that is handled by a middleware platform is (or can be viewed as being) generic, in that it is the same for all invocations, regardless of

the type of the method being invoked, or the type or number of arguments or results. The standard approach to this problem is to convert the invocation in to packed byte-array form as early as possible (usually within the stub), so that is may be managed in a uniform way. This leads to a middleware system that is hard to debug, and difficult to specialise or evolve.

Solution: The solution to this problem is to use the stub to represent the invocation in a generic form by converting into an invocation object that may be manipulated by the rest of the middleware platform. The goal is to delay the conversion to byte-array form for as long as possible, to increase the proportion of the middleware that may benefit from the strong typing and mutability of the invocation object. This part of the middleware is then easier to debug, specialise and evolve. This is illustrated in Figure 7. In the Java language, construction of such an invocation object is straightforward due to the strong typing and introspection facilities provided by Java.

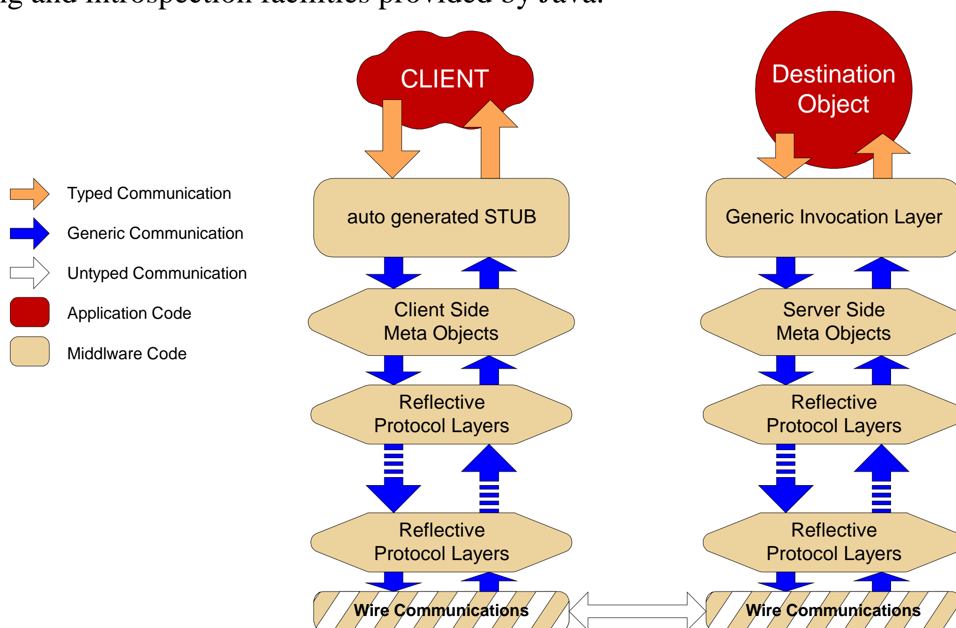


Figure 7

Example: FlexiNet, the platform upon which the Mobile Object Workbench and the Information Space are designed, uses this pattern as a key design principle.

Service Deployment

Within FollowMe services deployment provides a framework for the observation of the behaviour of resources. Resources can refer to physical resources such as a CPU, or a communication channel between two machines. Resources can also refer to high-level notions defined by applications, for example, number of readers of a newspaper or the popularity of a web page can be seen as resources. Observing the behaviour of a resource is motivated by the need to detect changes in, or evolutions of, the resources characteristics as the time goes by.

Observing the behaviour of a resource, and the possible variations, can be generalised to observing the behaviour of any kind of object. We call such observation object monitoring. The need for monitoring is common to various types of applications. Monitoring suggests a discrete observation process that can either be periodic or aperiodic. Periodic observation is typically an active process—the monitoring process uses its own thread to peek at the behaviour of the monitored object. In contrast, aperiodic observation is typically passive, since it is an on-demand process.

In its simplest form, monitoring can return a value that stands for the behaviour of the object at the time it is observed. A typical example is the periodic monitoring of a CPU, which can be translated into a load factor percentage. More generally, monitoring returns a complex object representing the behaviour of the observed object. In addition, it might be necessary to keep track of a series of observations before reacting. We call this process logging.

In the case of logging, values returned by individual observations have to be stored, the stored values are called a log. The process exploiting these observations analyses the log and takes decisions based on the global knowledge thereby acquired. Logging is a well-known technique that is widely used, in particular in the Database world.

It is possible to combine monitoring and logging: Monitoring objects can log the state of observed objects; and a separate monitoring process can use that log to watch behaviours. It should be apparent that the process of monitoring is orthogonal to the process of logging.

Name: **Monitoring**
Context: Distributed OO systems
Problem: Within a distributed object oriented system there are many different notions of resource. Generally changes in availability are not predicable by the system because either changes external to the system effect the resource or because changes within the system effect a resource in unpredictable ways. In order to allow a system to respond to resource changes, the system needs to monitor resources. Generally a resource is either a physical quantity such as CPU availability or an abstract quality such as the behaviour of an object.
Solution: An Active Object that periodically delivers values gathered from the observation of another, monitored, object. A monitor might watch over several different objects. In addition to its periodic behaviour, a monitor should support on-demand observations. A monitor is physically supported by a process running on a machine. Its interface, however, should be location independent.
Example: Performance monitor. The service deployment specialises the general notion of a monitor into a performance Monitor. Resources are monitored, as they are defined in the introduction of this text.

Name: **Logging**
Context: Data that evolves over time.
Problem: Often the data of interest is not the point value of a resource but rather the way its point values change over time. To monitor such aspect of behaviour a means of recording the point values, or some derivative data, is required.
Solution: Provide a dedicated, reactive, storage object with appropriate interfaces for storing and retrieving (e.g. getFirstLogRecord, or scanForward, scanBackward) the point data. The storage and retrieval interfaces may be distinct and may be location independent.
Example: A before-image log for a database, the histories defined for the service deployment, a check-pointing process.

Monitoring and logging may be combined to monitor derivative data. However, one should note, there are many different ways of combining these patterns e.g. a monitor may log data, an object may log data and the log is monitored etc.

The Autonomous Agents Framework and User Access Abstractions

The general function of the agent framework is to decouple the user, and the particular devices the user has access too, from the application that the user wishes to invoke. This decoupling allows the user to both change his location, and the devices which are available to him, whilst the application is running. The software intermediaries which perform the decoupling cope with the problems of: supplying default answers to input requests from the application while the user is not available; buffering results and delivering them when the user is available; and converting the material between presentation formats to deal with the variety of devices that are available to user at different times.

The general structure of the agent framework was illustrated in Figure 1. This illustration leaves out two important aspects of the agent framework. The first is the use of scripting as a technique for programming agent behaviour. The second is the service interaction framework that allows agents to discover properties of services and for services to provide their own agents to interface to a user. Although we regard both of theses as design patterns the exact articulation of them appears difficult. Both of these patterns are attempts to take the theme of decoupling further. At this stage however our understanding of these patterns is incomplete and we have not attempted to articulate them as pattern definitions.

Scripting makes writing mobile agents simpler than using the MOW directly. Generally agent scripting provides some of functionality of the MOW in a form that is more conveniently packaged for the average agent producer. The Service Interaction framework provides the means by which a service can: -

1. Advertise it’s availability to agents and users;
2. Supply agents for its use to users.

The model is: When a service starts up it registers itself with a service trader. The service describes its interfaces syntax and behaviour to the trader and may register agents that can use these interfaces. It also registers textual descriptions of these interfaces and agents that are in human readable form. An agent that wishes to discover details of the service can examine the entries for the service and either use the service directly or obtain an agent that can interface to a user to give the user access to the service. Typically a user will request his personal assistant to locate an agent to perform a particular task. The personal assistant will locate a service that can perform that task by having a conversation with a service trader. The service trader will provide a task agent that can talk to the user (via the personal assistant) to obtain required parameters etc. This is illustrated is Figure 8.

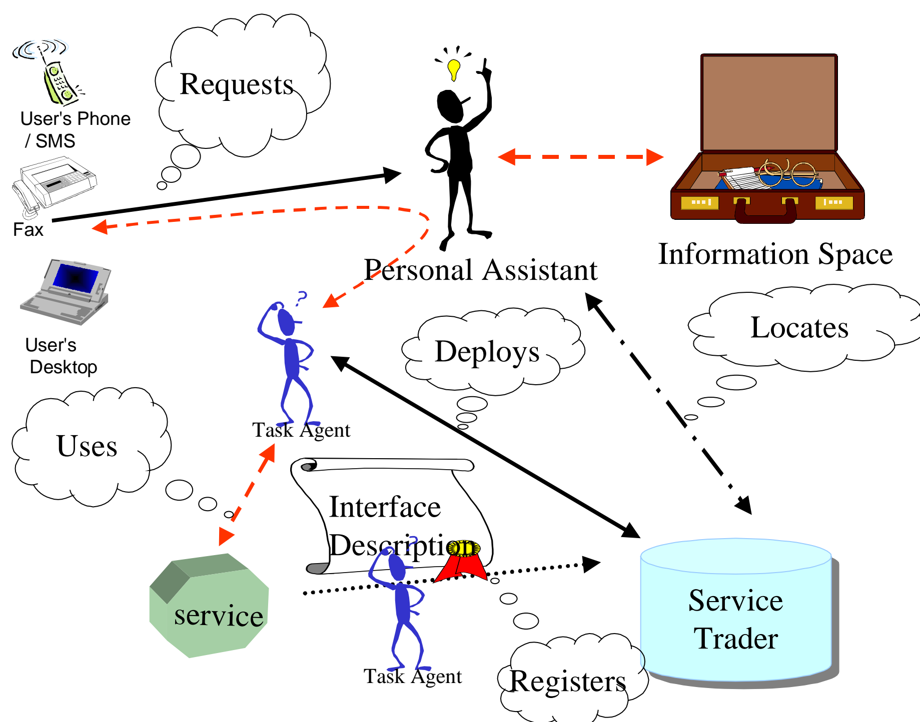


Figure 8: Service Interaction

The patterns for the Autonomous Agent Framework is described built from four simpler patterns. The first three of these patterns arise from applying the decoupling heuristic and the fourth pattern arises from the notion that we can sometimes simplify an explanation of a behaviour by changing the form of the explanation from an extensional form to an intentional form.

Name: **Spatio-Temporal Decoupling**
Context: User driven computational tasks
Problem: Users are coupled in time and space to the computing tasks they perform.
Solution: The production of computational intermediaries that act on the behalf of a user to decouple the user in time and space from the computational task. The intermediaries can answer questions on behalf of the user by user of user specific defaults, intermediaries can store and/or redirect output sent to a user and, with the use of the User Access mechanisms, can tailor output to the devices that the user has available at any given time.
Example: Personal Assistant; Task Agents.

Name: **Representation Decoupling**
Context: User driven computational tasks with mobile users.
Problem: The interaction between a user and a computational task is often strongly tied to the type of interaction/data delivery device the users is expected to have. In an environment where a user uses devices found at locations as the user moves about (rather than e.g. having a personal mobile device) the users may loose the ability to interact with the computational task because the specific device type is not available.
Solution: A partial solution is to define interaction/data delivery more abstractly so that a computation interacts/delivers data in terms of the abstract model and device rendering decisions are taken as late as possible when the particular device type binding is known.

Example: The patterns “Document Delivery” and “XML Based Document delivery”.

Name: **Complexity Decoupling**
Context: User driven computational tasks
Problem: Users fail to take advantage of the range of service available because of the complexity of the service interfaces.
Solution: The production of computational intermediaries that act on the behalf of a user to decouple the user from the complexity of the service. Such intermediaries translate between the user’s perspective of goals and the service’s perspective of means.
Examples: The Personal Assistant acts as an intermediary between the user and tasks agents and assist the user in finding and launching appropriate agents to achieve the users goals. Task Agents encapsulate task specific knowledge that sits between the user and the service that the user accesses. Service Deployment provides a framework within which a service may provide a user interaction agent to enable the user to interact with the service.

Name: **Explanatory Agency**
Context: Software Systems with Complicated behaviour
Problem: The extensional description of software becomes so complicated as to become useless in making predictions about how a piece of software will behave.
Solution: Form a description in of software in different terms, in particular in terms of its intentional attributes. That is rather than attempt to explain the behaviour of the software in terms of mechanisms the behaviour is explained in terms of such notions as goals, beliefs and desires. Such explanations allow users to form internal models of the software as “social entities” with their own agenda and thus enable users to make predictions about future states and behaviours of the software [15]. Particular models of this are the Beliefs, Desires, Intentions model (BDI [16]), the intentional models of Speech Act Theory [17] (e.g. as within JAFMAS [18]) and the interaction models of Contract Net Protocols [19]. The shift in stance means that we adopt a social description of software as interaction among self-interested agents whose rationality is bounded by computational complexity.

Name: **Autonomous Agents Framework**
Context: User driven computational tasks with complicated behaviour.
Problem: Users fail to exploit services because:

- The services are complicated to understand and use effectively.
- The use of the services requires the online presence of the user that in many cases is not practical.

Solution: Combine the patterns Complexity Decoupling, Explanatory Agency, device Decoupling and Spatio-Temporal Decoupling to produce an intermediary between the services and the user:

- that manages the complexity on behalf of the user;
- is explicable in terms of the intentional behaviour of the intermediary permits the a range of devices found at locations;
- and Spatio-Temporally decouples the user from the service.

User Access

Name: **Document Delivery**
Context: Multi-media presentation of complex information
Problem: Information needs to be delivered to users making use of a variety of devices. The details of which kind of devices are available to a user change with the time and the user's location, and indeed the user's preferences for how information is delivered may change for arbitrary reasons. The information must be rendered on these devices in an adequate quality.
Solution: A uniform model is adopted in which information can be encapsulated in an inter-linked set of documents. Each document represents a certain piece of the information. A document supports methods to render this piece of information in several external representations (graphic, text, audio, ...), depending on the capabilities of the device. The external representation may refer to other documents that maintain other linked pieces of information. Devices are encapsulated by device gateways. Device Gateways encapsulate the knowledge of the capabilities of the devices and provide Connections to the devices through which documents are rendered to the users. According to the available capabilities the most appropriate external representation of a document is selected. Interlinked information is rendered by querying the referred documents. This model allows for an information centred design. The representation of this information can be provided either at compile time, or at delivery time. The encapsulations of a variety of device types into device gateways allow a uniform interface for rendering of data.
Cross ref: A document is an object that understands how to render itself in different contexts. Particular instances of the pattern are XML based Document Delivery.

Components:
The component architecture assumes that a client object wishes to deliver something to a user. The client contacts the User Access component for the user and requests a Connection satisfying some criteria (e.g. suitable for delivering at a particular time of day and a particular volume of information), the User Access creates a Connection to be used by the client to communicate with the user and is returned a reference to the Connection together with a description of the Connection which may impose limitations on messages (e.g. message size). The client constructs a set of documents suitable for the Connection (it is assumed that it is the client that knows how best to construct, edit, précis etc. the information to be delivered). The Connection is an abstraction which hides the details of the use of real devices including any possible multiplexing, queuing etc.

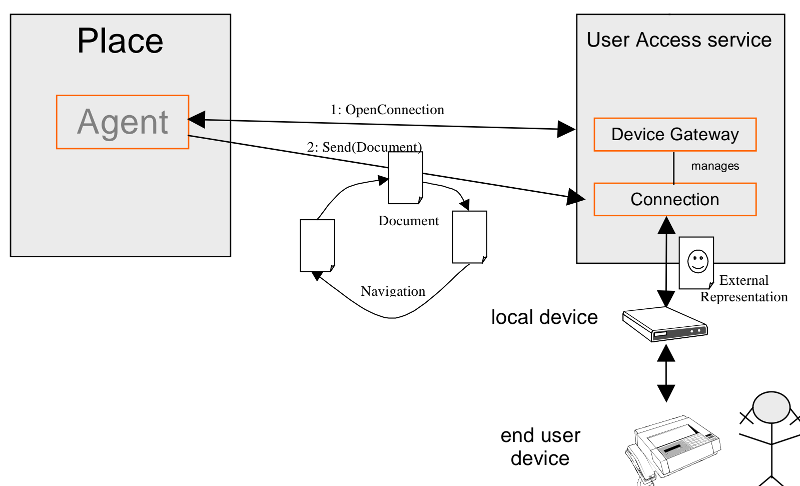


Figure 9: Document Delivery

Variations:

- Documents may be stubs. By asking them to provide the appropriate external representation the virtual device causes the deliverable to collect the actual data from a remote file store.
- Documents may be created dynamically, depending on the feedback from the user. When this method is called by a virtual device the deliverable may act as an applet and an interactive session is started with the user using the local device for computation.

Name: **XML-Based Document Delivery**
Context: Multi-medial presentation of structured information
Problem: Different devices may support different layout mechanisms to represent textual information, e.g. either as HTML for a web browser, as plain text in an email, as short message in SMS, RTF in MS Word. The same information must either be stored and exchanged in different layouts or it is stored and exchanged in one unique format and from this format the different layouts are derived.
Solution: The structured information is stored in a document, which supports the representation as XML-text. The XML-text holds a reference to an XSL document. This XSL document contains a set of layout definitions (so-called modes). Each layout definition contains a set of rules, which describe how the structured information is represented in the appropriate format.
Components:
The client sends a document to the connection. The connection requests an XML representation of the document and retrieves also the XSL layout definition as a referred document. The XML and XSL documents are given to a converter that produces the final layout in the appropriate mode.

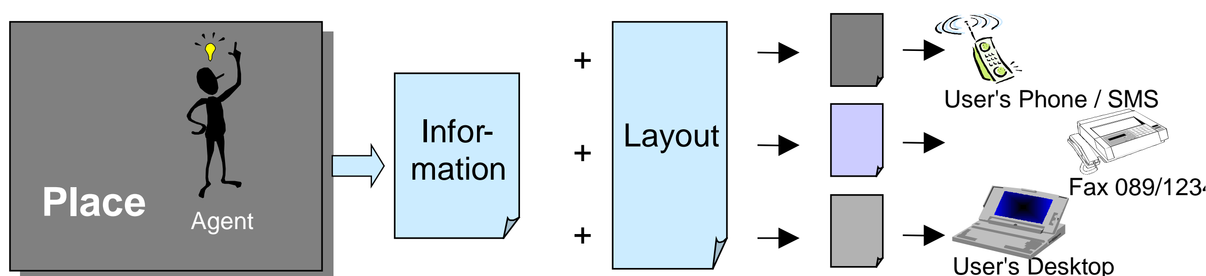


Figure 10: XML-based Document Delivery.

The Pilot Applications

The pilot applications are all concerned with delivering information to users through the use of intermediaries. The nature of the intermediary varies with the application but the general structure is capture by the pattern “Information Latticework”. Applications are deployed in a dynamic environment and component objects may find themselves subject to resource constraints beyond their control. This leads to a need for application components to monitor and respond to their environment dynamically. This aspect of applications is captured by the patterns “Reactive Policy” and “Resource Aware Adaptation”.

Name: **Information Latticework**
Context: Distributed Object Oriented Information Systems
Problem: Raw data is produced by a variety of information sources. Often this data is capable of answering high level queries from end users when equipped with an appropriate interpretation. Generally, however, the end user does not know how to navigate to this data or how to interpret his query with respect to the available data.
Solution: Provide a set of intermediary processes that understand how to interpret high level queries in terms of lower level data and how to access the raw data sources. Generally we can think of information consumers linked to raw data providers by a latticework of intermediaries that "understand" the structure of information below them (towards the providers) and how to represent this to nodes above them (towards the consumers). Intermediaries structure the raw data into "information" by constructing indexes or other meta-data that make the raw data accessible and meaningful in a specified context (context is also meta-data) and by transforming the data representation into appropriate forms to be used the "application" at the next layer in the latticework.

The links between nodes in the latticework may either be "push" or "pull" and in practice it is likely that different layers will adopt different strategies depending on the details of the application. Generally the links between the sources, intermediaries and consumers are established dynamically. However, if we consider a snapshot of the network so formed it is easily visualised as a latticework.

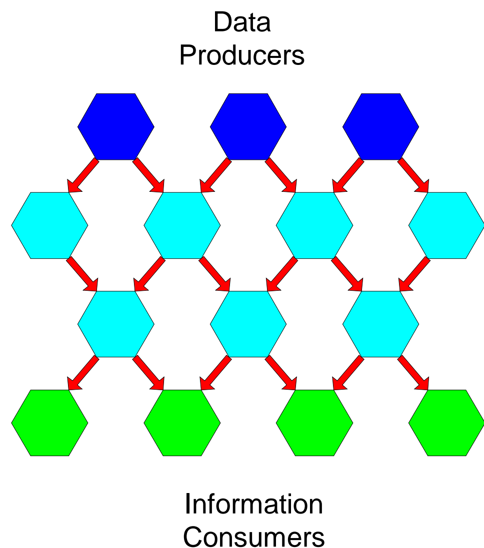


Figure 11: Information Latticework

The network has several degenerate cases that deserve names of their own. *The information funnel* is a latticework with a single consumer and many producers and intermediaries. The *information fountain* is the converse with one producer and many consumers and intermediaries. The *information chain* has one consumer and one producer and many intermediaries. Although one often pictures a latticework as regular this is by no

means always the case. Latticework intermediaries may cross connect in arbitrary fashion and there may be an arbitrary number of layers of intermediary connecting any data provider to a consumer.

Name: **Reactive Policy**
Context: Mobile Object Systems; Object Systems operating in dynamically evolving environments.
Problem: Objects perform their tasks in a context that is only partially known at the time of the object's creation. This means that it is not possible to fully tune an object's performance before it is deployed.
Solution: Objects need to actively monitor their environment, and their own performance in that environment, and tune their behaviour accordingly.
Example: Resource Aware Adaptation (below)

Name: **Resource Aware Adaptation**
Context: evolving environments
Problem: adapting the behaviour of the application according to an evolution diagnosis stemming from logging and monitoring.
Solution: Using together monitoring and logging and implementing a reactive policy.
Related: protocol objects, deployable objects as means. monitoring and logging as sources.
Uses: load balancing policy.

References

[1] ITU-T Recommendation X.901 | ISO/IEC 10746-1: Overview

- ITU-T Recommendation X.902 | ISO/IEC 10746-2: Foundations

- ITU-T Recommendation X.903 | ISO/IEC 10746-3: Architecture

- ITU-T Recommendation X.904 | ISO/IEC 10746-4: Architectural semantics

[2] A Timeless Way of Building Christopher Alexander Oxford University Press

[3] Understanding and Using Patterns in Software Development. Dirk Riehle and Heinz Zullighoven, Theory and Practice of Object Systems 2, 1, 1996, pp. 3-13

[4] CORBA Design Patterns. Thomas J. Mowbray, Raphael C. Malveau. Wiley 1997. ISBN 0-471-15882-8

[5] Design Patterns Elements of Reusable Object Oriented Software. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Addison Wesley 1995. ISBN 0-201-63361-2

[6] DB8.2: Mobile Object Workbench Deliverables incorporating DB2, DB3, DB4, DB7.2, Richard Hayton 16.04.98

[7] DC6.1 InformationSpace: Requirements, Design, Interfaces and Report incorporating DC 1, 2, 3, Douglas Donaldson 05.03.98

[8] DG3: Service Deployment, Design, L. Amsaleg, M. Billot, M. Le Nouy, 23.03.1998

[9] DD3: Autonomous Agents, Design, Nick Taylor 24/02/98

[10] DE3: Personal Profiles, Design and Interface Specification, Steve Battle, 06/02/98

[11] DF3: Service Interaction, Design, Steve Battle, 21/04/98

[12] DH3: User Access, Design and Interfaces incorporating DH4, Elcha Triep, Michael Breu[13] DI2: Pilot Application 1, Requirements, Hans-Guenter Stein, FAST e.V., Lioba Gebauer, FAST e.V., 16.03.1998

[14] DJ2: Pilot application 2, Requirements, L. Amsaleg, M. Billot, M. Le Nouy, 30.03.98

[15] The Intentional Stance, Daniel Dennet, MIT Press 1987

[16] BDI Agents from Theory to Practice, Anand S. Roa and Michael P. Georgeff, Australian Artificial Intelligence Institute, Technical Note 56, April 1995

[17] Speech Acts: An Essay on the Philosophy of Language, Cambridge University Press 196918] JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation, Deepika Chauhan, ECECS Department, University of Cincinnati, 1997

[19] Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework. Sandholm, T. and Lesser, V. 1995, First International Conference on Multiagent Systems (ICMAS-95), San Fransisco, pp. 328-335.