Hermes: Agent-based Middleware for Mobile Computing *

Flavio Corradini and Emanuela Merelli

Università di Camerino, Dipartimento di Matematica e Informatica
Camerino, 62032, Italy,
{flavio.corradini,emanuela.merelli}@unicam.it

Abstract. Hermes is a middleware system for design and execution of activity-based applications in distributed environments. It supports mobile computation as an application implementation strategy. While middleware for mobile computing has typically been developed to support physical and logical mobility, Hermes provides an integrated environment where application domain experts can focus on designing activity workflow and ignore the topological structure of the distributed environment. Generating mobile agents from a workflow specification is the responsibility of a context-aware compiler.
Hermes is structured as a component-based, agent-oriented system with a 3-layer software architecture. It can be configured for specific application domains by adding domain-specific component libraries. The Hermes middleware layer, compilers, libraries, services and other developed tools together result in a very general programming environment, which has been validated in two quite disparate application domains, one in industrial control and the other in bioinformatics. In the industrial control domain, embedded systems with scarce computational resources control product lines. Mobile agents are used to trace products and support self-healing. In the bionformatics domain, mobile agents are used to support data collection and service discovery, and to simulate biological system through autonomous components interactions.

1 Introduction

Industrial production processes, the in-silico daily work of bio-scientists, and many other jobs are usually performed by executing a set of distinct, sometimes repetitive, activities [54]. Automating such an application process in a distributed environment requires coordination of these activities, but also lower level implementation support in sharing of data, localization of reliable resources, retrieval of suitable information, integration of heterogeneous tools, discovery and selection of the best available services, and mobility of computational units.

* This work was supported by the Fulbright grants, by the Center of Excellence for Research "DEWS: Architectures and Design Methodologies for Embedded Controllers, Wireless Interconnect and System-on-chip" and by Italian CIPE project "Sistemi Cooperativi Multiagente"

The application designer, whose primary expertise is in the application domain, should be free to focus on coordinating domain activities rather than being concerned with the distributed computational environment.

In the domain of production processes control, for example, supply chain management [33] has been developed mainly with workflow-oriented technology for networks of fixed distributed systems. The present need to trace products [1] and to extend the chain with customers (e.g. domestic appliances, items of clothing, food), requires flexible workflow management systems encompassing embedded systems and mobile devices (e.g. PDAs for technical assistance), and supporting code mobility (e.g. for traceability and self-healing) [41, 52].

In the bioinformatics domain, a flexible workflow management system could be used to carry out many activities whose execution environment is the Web, which is distributed and dynamic in nature, with large amounts of highly dynamic data and proliferation of (often redundant) tools. In fact, many bio-scientists aspire to automate some of the time-consuming activities to the base of wet-lab procedures, as browsing, searching and selecting resources [37, 50] so as to use flexible and expandable computational analysis and simulation tools during their in-vitro exeperiments. Advantages of moving computational "bio-instruments" over data, by delegating a mobile agent, include decentralizing execution of local activities, avoiding the warehousing of highly dynamic data, reducing network traffic, and freeing researchers from network faults and from the need to be continuously connected to a laptop. Mobile devices could also support a bio-scientist moving among different laboratories during his experiment.

Experience with these two, quite different domains suggests that applicability of Hermes-like systems is quite wide, and that many other application domains could take advantage of flexible, modular, expandable, easily configurable and scalable middleware which supports workflow management and uses mobile computation as activity implementation strategy.

Middleware technology is an emerging and promising technology that provides application designers with a high level of abstraction, hiding the complexity introduced by distribution (Figure 1)[57]. Middleware for mobile computing, in particular, is becoming a widespread technology [51, 38]. Mobile computing systems, in the sense of computing systems that can be easily moved physically and whose computing capability may be used while they are being moved, have been empowered by the diffusion of satellites and cellular technology[3].

The wide range of different developers of mobile devices has led to development of many different middleware systems, which differ in the type of computational loading (heavyweight, lightweight) of the mobile unit, the type of communication paradigm (synchronous, asynchronous) used among distributed units, and the type of context representation (transparency, awareness) provided to the mobile application. In general, a mobile system can be characterized by mobile device executing on a light computational load, by a intermittent connection with asynchronous communication, and by a dynamic context with awareness of re-

[1] European Community Directive 2001/18/EC

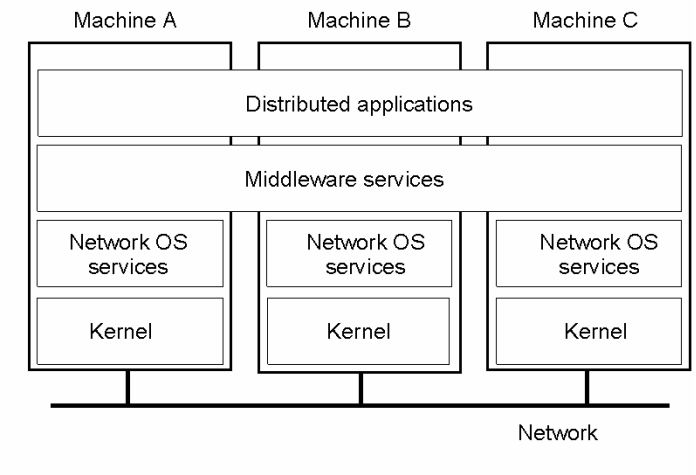

Fig. 1. A Distributed System organized as middleware [57]

source distribution. Mascolo et al [38] provide a comprehensive survey of mobile computing middleware; B'Far [3] gives an overview of principles of mobile computing. Further distinction can be found between middleware systems developed only to support physical mobility (which are traditionally application-centred [51]) and more general middleware systems [43, 45, 31, 32] developed especially to support the coordination of mobile components, most of which are based on *tuple spaces* of the Linda model [28] to support decoupled communication.

In this work, we exploit mobile agents as computational units that logically move to support execution of a distributed application. Consistent with B'Far [3] we see mobile agent particularly suitable for the following reasons:

1. *mobile agents are inherently active because of their autonomous nature*,
2. *mobile agents use less network bandwidth in comparison to RPC or RMI*,
3. *mobile agents can display better response times owing to reduced effect of network latency on the application*,
4. *mobile agents are inherently heterogeneous*,
5. *mobile agents are autonomous and asynchronous and so can deal with intermittent network connectivity gracefully*,
6. *mobile agents can adapt extremely well.*

Providing an application designer with a transparent global view of the distributed environment, with a user-friendly programming environment and executing distributed applications exploiting mobile computation, through a light

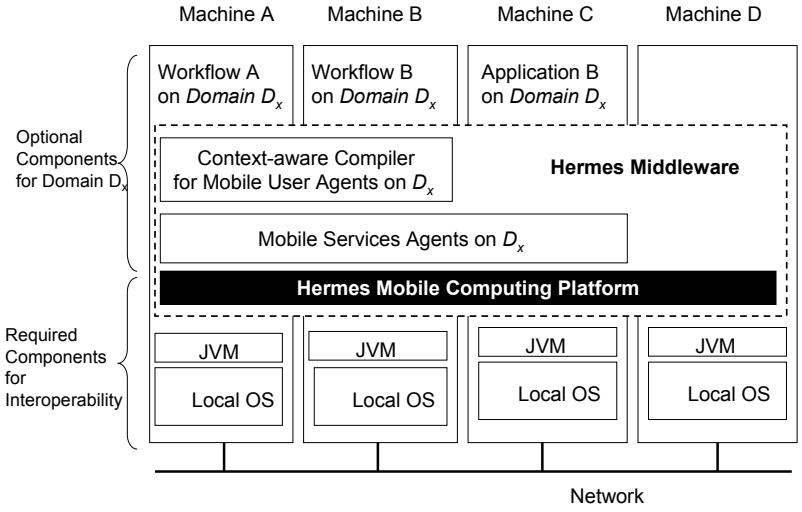

Fig. 2. A Distributed System over Hermes Middleware in heterogeneous environment. Only the Hermes mobile platform (dark layer) is required for interoperability, but additional components can be added to support workflow

and flexible mobile middleware, is the aim of Hermes [2]. Hermes is a component-based, agent-oriented system with a 3-layer software architecture [9, 15]: *user layer*, *system layer* and *run-time layer*. At the user layer, it allows designers to specify their applications as a workflow of activities using the graphical notation provided by JaWE editor [21]. At the system layer, it provides a context-aware compiler to generate a pool of user mobile agents from the workflow specification. At the run-time layer, it supports the activation of a set of specialized *service* agents, and it provides all necessary to support agent mobility.

One of the main features of Hermes middleware is its scalability. The present version, HermesV2 [29], is a pure Java application whose kernel requires about 80KB of memory and interoperates across a systems ranging from microprocessors to very power workstations (Figure 2). The lightness of its core is based on the unique class *Agent*, which assigns the basic features to each agent, including mobility. *Agent* is an abstract class, with two associated extensions *UserAgent* and *ServiceAgent* (Figure 3).

The main difference between run-time layer and system layer is how agents function in each. *ServiceAgent*s in the run-time layer are localized to one platform

[2] In Greek mythology, Hermes is the son of Zeus and Maia. He is also known as Mercury to the Romans. Hermes is Zeus's messenger, the fastest of the gods, recognizable by his winged sandals.

to interface with the local execution environment. *UserAgent*s in the system layer are workflow executors, created for a specific goal that, in theory, can be reached in a finite time by interacting with other agents, afterwards the agent die. Furthermore, for security *UserAgent*s can access a local resource only by interacting with a *ServiceAgent* that is the "guard" of the resource (Figure 4).

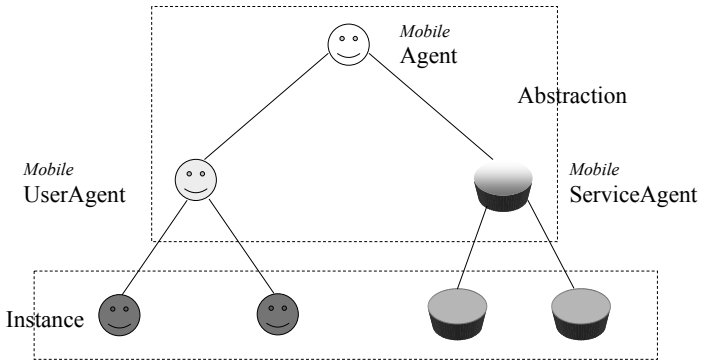

Fig. 3. Hermes agents hierarchy. The Java class "Agent" is extended with "UserAgent" which is the prototype of the the workflow executor, and with *ServiceAgent* used to interface local resources. Only *ServiceAgent* can invoke operating system functions

We can summarize that Hermes uses activity-based workflow modelling, as an high-level programming language. It uses agent-based modelling as an intermediate programming language, and it uses mobile computing as run-time support of the execution of mobile agent systems generated with respect to the functional and non-functional requirements of the distributed application.

We have also developed a set of tools particularly meaningful for an effective implementation of Hermes middleware. Among these a generalize wrapper [4] to support the extraction and the integration of heterogeneous resources, an interface to dynamically access Web Services [60], an ontology manager which supports the mapping among different resource schemas [18] and a matchmaker to discover and select services [14]. Furthermore, we have defined a mapping from UML Activity Diagram and CSP-like process algebra to allow the analysis and verification of the behaviour of the workflow designed by the user [1].
We are also working on a graphical notation to represent the mobility and execution environment of a pool of agents, its mapping to Klaim, i.e. a process calculus for mobile computing [19]. We would like to use Klaim language and Klava, i.e.

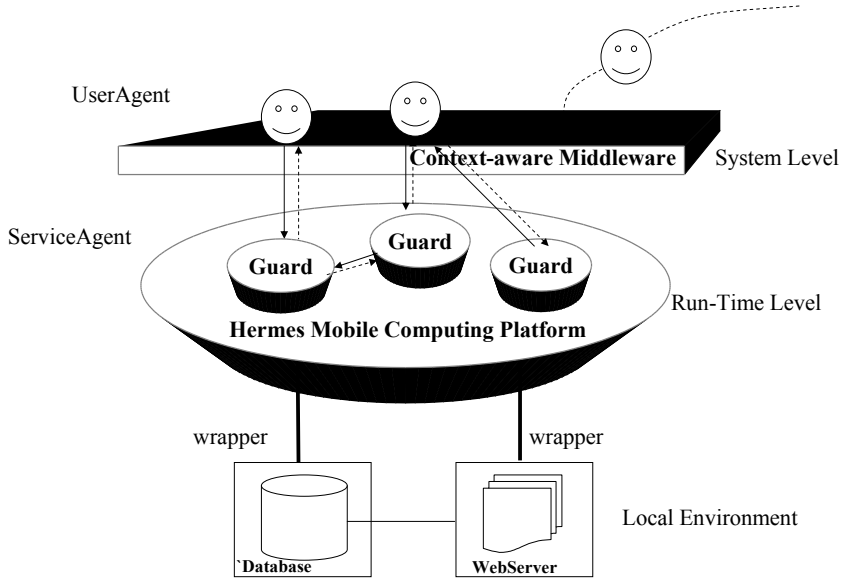

Fig. 4. Access Control in Hermes Middleware is based on different access rights given to two "Agent" extension. The instances of $ServiceAgent$ act first as a barrier towards the local resources then once the a $UserAgent$ has been identified as an interface

the implementation of Klaim in Java, into Hermes compiler to implement agent level workflow.

Recently, Hermes has been used as simulation programming environment in systems biology [17]. We have modelled and implemented a system to simulate carbohydrate oxidation of a biological cell. The course-grain approach allowed us to identify the autonomous computational units of the software system in those cellular elements that exhibit the behavior of a computational environment (cytoplasm, mythocondrial matrix, etc.). While all elements are agents whose activities were implemented for the case study, in future we aim to map the abstract machines for systems biology provided by Luca Cardelli [10] to Klaim within Hermes architecture. The agent mobility will characterize the real movement of cellular components within and through the cellular environments (compartments and membranes).

In the remainder of this tutorial, in Section 2 we set the context in which the Hermes middleware has been developed. In Section 3 we outline some of the formalism, techniques and systems have been chosen to bear the design of the global computing environment and we draw reader attention to Hermes

software architecture. Next, in Section 4 we propose some application examples taken from our experience in designing and implementing applications within specific application domains. Finally, in Section 5 we discuss future work and conclude.

2 Setting the Context

Distributed Environment *DE*

The distributed environment (DE) for mobile computing we refer to consists of collection of autonomous and mobile computational units interconnected by a communication media. It can be distributed over a wide area network (as in the bioinformatics case study), over a local area network (as in the production plant) or it can be a simulation of a distributed system (the systems biology example).

The first DE we consider is the Web. It is characterized by dynamic execution context, intermittent connection, unpredictable congestion, faulty communication, presence of security barriers, and heterogeneous, distributed and unstable resources. The second DE is characterized by permanent network connection, but it still deals with the management of heterogeneous resources. The last DE is a virtual distribution. All the mobile computation can happen within a single machine, or among homogeneous machines or heterogeneous ones (e.g. grid computing). They vary in the way they hide and manage problems deriving from the execution environment.

In our context, there are two different logical mobile computational units: at the system level, there are flexible, autonomous, pro-active [3] units, situated in a dynamic, sometimes open, unpredictable computational environment. We call them $UserAgent$s; they are created in a specific computational environment to solve problems in a certain application domain, they are coordinated by a suitable communicate model, they can move to reach a different computational environment to better fulfil the goal for which they have been created. At the run time level, there are autonomous mobile units with the special task to manage local and networks resources. We call them $ServiceAgent$s, they are units created any time a new resource becomes available in the distributed environment.

Distributed Applications *DA*

A distributed application (DA) consists of a set of coordinated activities that use distributed resources. Workflow models are useful notations of coordination to link these activities together. If we consider a workflow as a distributed program and a workflow management (WMS) as its run-time support, the functionality provided by a WMS is similar to that offered by a middleware system in a distributed environment.

[3] For proactive, we mean controlling a situation by causing something to happen rather than waiting to respond to it after it happens.

Workflow models are supported by a number of systems for business process automation and process control, but typically the model is fixed and hard-wired in the application, or configurable only through a very heavyweight customization process. In contrast, our approach makes specifying, modifying, and executing workflow a very lightweight. In the bioscience domain, for example, it is practical to develop workflow support for the varied idiosyncratic processes of individual scientists, and so free the bioscientist from from repetitive interactions with the execution environment. To the extent that workflow specifications are shared, it is also possible to incrementally support standardization of protocols and creation of a transparent analysis environment.

Workflow is specified abstractly in a graphical notation and mapped to a set of autonomous computational units (*UserAgents*) interacting through a communication medium. The mapping is achieved by a compiler that is aware not only of the contents of a library of implemented user activities but also the software and hardware environment for executing them. In our case, information available to the compiler includes available hosts and their connection topology, available services (*ServiceAgents*), the kinds of information available at different locations, and additional domain-dependent parameters. Application-independent rules for dealing with connectivity failure, service access failures, etc., are embedded in *UserAgents* and *ServiceAgents*. A user specifying workflow need not be concerned with where to search for information, in what form the information is stored, the protocols for interacting with each service, or a host of other low-level details that can be left to the context-aware compiler.

Mobile Computing

In the above described scenario, we said that user activities are mapped into system activities, by $UserAgent$s. The pool of agents must coordinate to execute user level workflow, possibly by migrating from one environment to another and coping with any of the unpredictable phenomena due to distribution. The agent mobility is supported by a light platform that characterizes the middleware.

This approach to exploiting mobile computing during the development leads to the definition of a (new methodology) to guide software development, from analysis and specification, design and validation, coding and testing, deployment and maintenance. In particular, the *analysis phase* imposes the choice of application domain (e.g. bioinformatics), identification of common user activities (e.g. sequence similarity search, functional motif search, protein analysis, etc. [54]).

Next, the *design phase* concerns specification of workflow of activities and its validation by suitable tools [1]. The *coding phase* is linked to the engineering of the layer below.

At the *system layer*, the main component is the context-aware compiler, whose engineering depends on both the application domain and the execution environment requirements. The *design phase* of the compiler relies in a two steps: step 1: the *User Level Workflow* (ULW) (Figure 6) is mapped to *Agent Level Workflow* (ALW); step 2: the ALW is coded in a pool of mobile *Workflow Executors* (WEs) the $UserAgent$s. The generation of a ALW also implies the choice of suitable coordination model, i.e the communication media used among agents.

Also this choice is conditioned by the application domain features. The first step will generate a specification of agent level workflow whose validity must be checked. Thus, tools different from those in the upper layer will be required since the mobility is also included [42, 11, 19]. Then, the *coding phase* is linked to the engineering of the layer below as well.

Implementation of system activities is based on the services offered by the *run-time layer*, including both those belonging to the kernel and those offered by the execution environment of the application domain. In the *run-time layer* the use of mobility is tied to the physical distribution of resources. In the layered architecture, mobility can play a twofold role. At *user* agent abstract levels it fulfills a modelling function while at *service* agent level it fulfills a reliability function.

In the Section 4, we describe two applications developed for two different application domains: functional testing and self healing in domestic appliance manufacturing [8]; medical bioinformatics [2, 39, 4, 5, 40] and systems biology [17]. We outline how mobility covers different and distinct aspects of the implementation in each of these domains and finally we describe a set of services and tools we have developed for Hermes.

In the next section, first we describe the functionalities of Hermes 3-layer architecture, then we describe a methodology to develop the Hermes mobile platform for an given application domain.

3 The Hermes Software Architecture

We now describe the general software architecture of Hermes, a middleware system for the design and development of distributed applications upon a mobile computing platform. This architecture has been successfully used to design an agent-based tool integration system [16]. The architecture consists of three conceptual layers as shown in Figure 5.

A *User Layer*, on the top of the architecture, where the user specifies his application as a workflow of activities with the features described above. Since our potential users may not be computer practitioners, the specification language must be simple and intuitive to use as, in most cases, graphical notations are.

A *System Layer*, on the middle of the architecture, provides the needed environment to map a user-level workflow into a set of primitive (and already implemented) activities. The execution of these latter is coordinated by suitable model, they implement the activities at the user level and embed implementation details abstracted from the execution environment (fault tolerance, for instance). These primitive activities are implemented by autonomous software entities $UserAgent$ able to react to the environment changes where they are executed. The agent-based paradigm and technology, as argued several times in the literature (see, for instance [34], and references therein), seem to be particularly suitable for designing environments populated by entities that communicate and coordinate their activities (as most of the applications of our interest are). A particular significant ingredient at this layer is the compiler that maps user

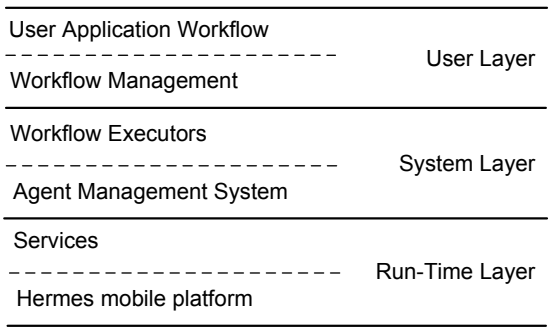

Fig. 5. The 3-layer Software Architecture for Hermes Middleware. The *User Layer* provides the editing workflow environment, the *System Layer* generate a mobile agent system to support the execution of the workflow and the *Run-Time Layer* provides all necessary to interact and move along the distributed environment

level activities into system level activities. The compiler must be aware of the available a library of implemented activities but more significantly it must be aware of the environment (software/hardware resources, knowledge, services...)

A *Run-Time Layer*, at the bottom of the architecture, provides primitives and services essential for agent mobility and resources access. The kernel is the platform for mobile computing which provides primitives for discovery, mobility, communication, and security.

As the Figure 5 shows, the three layers, User Layer, System Layer and Runtime Layer, are themselves split in two conceptual levels: - the type of application running on each layer and - the infrastructure supporting the application. At the user layer, the application is the workflow and the infrastructure is the workflow management environment (editor, model checker, ...). At the system layer, the application is a pool of running agents $UserAgent$s named *Workflow Executor*s (WEs), and the infrastructure is given by the agent management system (compiler, model checker, query optimizer, ...). Finally, at the run-time layer, the application is given by a set of services $ServiceAgent$s and the infrastructure consists of the mobile computing platform for agents mobility.

The Figure 6, moreover, presents the same architecture with the entities created at each level of abstractions: the user defines a *User-Level Workflow* (ULW) specification that is mapped to an *Agent-Level Workflow* (ALW) specification; the ALW specification is then used to generate a pool of *Workflow Executor*s (WEs) implementing all specified activities; WEs interact with distributed en-

vironment through through *serviceAgents*s (SA).

3.1 Hermes Layers Functionalities

It follows a detailed description of the main components and functionalities of each layer.

User Layer

The *user layer* is based on workflows and provides to users a set of programs for interacting with the wokflow management system. There are two main families of programs: programs for specifying, managing and reusing existing workflow specifications, and programs enabling administration and direct interaction with the workflow management system.

The *workflow editor* is the program that supports the workflows specification by composing activities in a graphical environment. The editor enables the specification of workflows complying with the WfMC reference model [30] and is implemented by using the JaWE [21] editor. Activities used in a workflow are configured by specifying input parameters and their effects are recognizable as modification of state variables or modification on the environment's status. The workflow editor enables the composition of both primitive and complex activities. A primitive activity is an activity that can be directly executed. A complex activity is an activity that must be specified before it can be used; the specification of a complex activity is a workflow of complex and simple activities. By using complex activities the specification of workflows is simplified because they enhance both hierarchical specification and reuse: we can use an already existing complex activity without caring of its specification. Users can use complex activities and stored workflows to increase productivity when specifying new workflows. Moreover, large libraries of both domain specific primitives and complex activities can be loaded to specialize the editor for a specific application domain.

Each activity can be configured with four parameters: the input data format, the output data format, the environment and its *description*. The *input data format* specifies which is the accepted input for a given activity. In similar way, the *output data format* specifies the accepted output data formats. The *environment* parameter is used to specify in which context an activity must be performed, since the same activity with the same parameters can be performed in different contexts. The environment is separated from the other input parameters because it can cause either the migration of a code or the selection of a specific implementation of the activity, while input parameters denote only data transferring. For example, consider an activity associated to the use of a specific tool implementation available in a given repository, it implies the deployment of tool on a remote site and the activation of the tool. In a similar way, consider an activity to search a given information on a given database, the activity is always the same, but its implementation is very different with respect to the target database, i.e., different authentication method, different querying interface and

different naming, hence the information on the target database is used to select the proper implementation of the activity. Finally, the activity description is used either when it is not possible to achieve transparency or when the user prefers to decide by himself where and how to execute a certain activity.

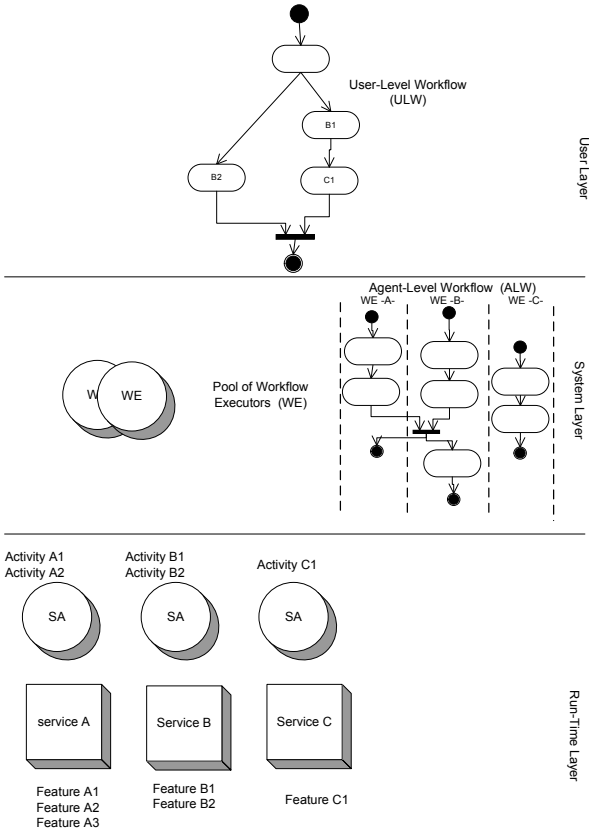

Fig. 6. Entities located at each Layer of Hermes Architecture. Any user level workflow (ULW) is mapped into an agent-level workflow (ALW) and compiled to a pool of mobile user agents, the workflow executors (WE) which interact with the service agents (SA)

System Layer

The system layer hosts WEs which are $UserAgents$ generated from the ULW specification. WEs execute and coordinate their actions to reach the fulfilment of the ULW specification. Some of the actions executed by WEs need interaction with the services (SAs); these actions correspond to operations that must be completed by interacting with a remote service.

In the case the distributed execution environment is open, the communication between agents takes place once the negotiation of communication protocol (the ontology) is successfully accomplished. By fixing an ontology, the agreement on the semantics is guaranteed, but information that can be exchanged is constrained; in fact agents can use only concepts defined in the ontology. In the case the system has defined a shared common ontology, the ontology negotiation procedure always successes.

Now we described the two phases agent generation procedure that is performed by the compiler. In the phase 1) the ULW is mapped to an ALW, and in the phase 2) the ALW is used to generate WEs. The ALW is a specification similar to the ULW, but it takes into account the existence of the agents that will execute the actions and it contains only primitive actions (actions that can be directly executed without decomposing them in workflows). Since the compiler is under development we can not provides implementation details, but only its main functionalities.

Phase 1: Mapping the ULW to the ALW The mapping from the ULW to the ALW is performed by recursively substituting activities of the user-level specification with a workflow of primitive agent-level activities. This mapping is performed by accessing to the User-Level Activity Database (ULAD) that maintains the correspondence between user-level activities and ALW. There are other rules managing technicalities of the transformation process, for example branching of the execution is translated to an agent creation activity and a join of two branches are translated to a coordination activity between multiple agents. Moreover, in the case the compiler recognizes a set of independent activities, it can distribute them among several agents to increase parallelism. The set of activities assigned to the same agent constitutes its body, therefore the result of this mapping consists on a set of workflows: one for each agent. Activities belonging to an ALW specify actions at a low-level of abstractions that can be directly executed. Messages are sent from an agent to another by using communication activities, i.e., an activity whose execution consists on sending a message to the receiver. Actually communication consists of sending and receiving single messages, in the future we want to extend this approach to definition of protocols that must be respected during inter-agent communication.

The ALW specifies all entities involved in the execution of a workflow, thus the constraint of spatial and temporal coupling communication can be respected since the compiler knows exactly when communication takes place and which are both receivers and senders.

The compiler can optimize the ALW by applying heuristics based on parameters issued to the compiler, e.g., the compiler can try to minimize the consumed bandwidth, minimize number of generated agents, minimize number of generated messages, maximize parallel execution of activities, and check for deadlock freeness. In addition to general purpose analysis, the compiler can check specific properties on the ALW, such as verifying that the shipping procedure of a specific item begins only after the purchase is completed. Actual prototype of the compiler implements part of these features.

Phase 2: Mapping the ALW to WEs In the second step, the compiler concretely generates agents from the ALW specification. To achieve this result, the compiler uses the User-level Activity Implementation Database (ULAID) and the Database of Skeletons (DoS). The ULAID stores the implementation of the agent-level activities and the DoS stores "empty" implementation of agents (the skeletons).

A skeleton is a role-specific implementation of an agent that does not contain any behaviour, e.g., a skeleton of a traveller agent can be a lightweight implementation of an agent limiting bandwidth consumption. Particular system properties can be obtained by proper choice of skeletons, e.g., limited bandwidth consumption. The concrete WE is obtained by plugging the specified behaviour into the skeleton. In particular, the compiler behaves following these steps:

- A complex behavior `CB` is generated by composing as specified in the ALW the implementation of each activity contained in the ULAID.
- The compiler analyzes the `CB` and derives all state variables that will be necessary to complete its execution.
- A state entity `SE` is generated by aggregating all state variables
- A proper skeleton is selected from the DoS. The WE is created by plugging both the complex behaviour `CB` and state entity `SE` in the selected skeleton.
- The previous steps are repeated for all WEs that must be created.
- Finally, execution starts.

Actually, we are implementing the WE generation procedure by using an implementation of the skeletons that dynamically load the compiled complex behavior and the state variables at start-up by dynamic binding. Instead of generating compiled WEs, it is possible to use skeletons behaving as interpreters of ALW specifications. In such case, the WE is obtained by associating the skeleton to the ALW specification. WEs of the former type are small, i.e., WEs contain only the code for the execution of the activities, and fast, i.e., instructions can be directly executed; while WEs of the latter type are large, i.e., they implement a complete interpreter, and slower i.e., instructions must be interpreted, but they exploit the ability to dynamically modify their behavior at run-time. The organization of our system enables the use of both type of agents. Actually, we are implementing the compiler producing compiled agents, but we plan also to investigate interpretation and dynamic adaptability.

Run-Time Layer

As already described, the overall structure of the system is very complex, it supports abstract specifications that are mapped into a complex distributed and coordinated flows of activities over a large-scale distributed system. In order to master this complexity, and support the transparency of the computing distribution by using mobile computation, the run time system provides a set of active services *ServiceAgent*s to allow a secure resources access and a mobile platform to support the agent mobility. The agent mobility is performed through mobile code environment that besides mobile code, supports also security, fault-tolerance, communication, and resource management and discovery.

More in detail, *ServiceAgent*s provide access to services. When a *UserAgent* migrates and arrives in a different platform, it can query the *YellowPageService* to gain information about services offered in the platform and then it communicates with *ServiceAgent*s to gain the information it needs. This paradigm simplifies the interactions enabling the use of an agent communication language, e.g. KQML [23] or Fipa ACL [25], as a unified way to communicate with other agents, services or resources.

A detail description of the Run-Time Layer components is given in the next section.

3.2 Hermes Mobile Middleware and its Engineering

We now describe a practical approach in developing of a modular and reusable agent-based middleware, in particular the Run-Time support of Hermes software architecture. We show the flexibility of Hermes middleware and how the followed component-based approach supports the reusability of existing artefact during the development of a middleware system for a specific application domain. As we already highlighted in the previous sections, agent-based systems are complexes [34], the development involving distribution, mobility, communication and security problems. The adoption of layered software architecture allows to master this complexity and enhances security because the interactions occurring among different layers can be monitored and filtered. In order to give flexibility to the Hermes middleware, we decided to adopt a layers plus components strategy, in fact each layer is designed as an aggregation of components.

We think that this point of view is a natural and effective approach to middleware construction and, more generally, to the development of complex systems. In the following paragraphs we give some hints of design of the Hermes kernel the detailed description can be found in [9]. We have chosen UML as architecture description language because is widely accepted in both the academic and industrial worlds as a reference language for system design.

The Hermes kernel can be described by three components, placed in a 3-layered software architecture as shown in Figure 7. Notice that this software architecture is different from that shown in Figure 5 because this last one highlights the hierarchical dependencies among system software components, for example the *agent* component in Figure 5 is unique while in Figure 6 has two distinct functional roles of *UserAgent* and *ServiceAgent*.

The *Core layer* role is similar to the kernel of an operating system, it implements the basic features of a mobile code platform, such as communication protocols, code traceability and security. The *Core layer* is essentially free of any system strategy.

The *BasicServices layer* extends the *core* features by providing services that directly support the agents activities, e.g., agent mobility and agent communication implemented on top of inter-platform communication. The *BasicServices layer* contains system strategies, but does not implement any feature of the application domain.

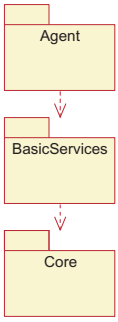

Fig. 7. 3-Layered Architecture of Hermes Mobile Computing Platform. The core supports identification, communication, loading and security; BasicServices supports discovery, mobility, creation, communication and security; Agent supports *User* agents and *Service* agents

The *Agent layer* is the container of all service agent and user agents of the application domain. The *BasicServices layer* is always present in any place, so that minimum support to agent execution is guaranteed.

Core Layer The *Core* layer is the lowest layer of the architecture (Figure 8) and contains base functions of the system, such as the implementation of the inter-platform communication protocols and agent management functions. This layer is composed of four components: *ID*, *SendReceive*, *Starter* and *Security*.

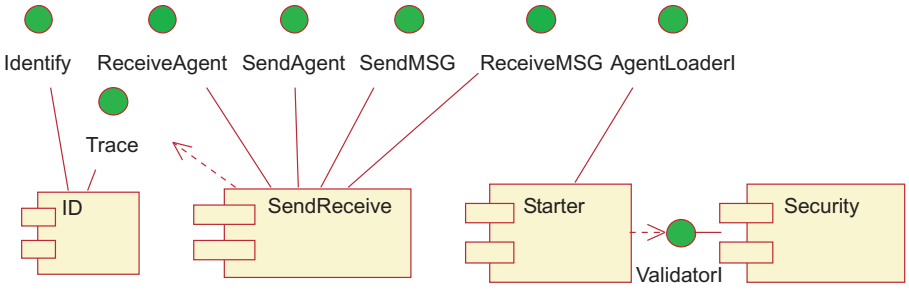

Fig. 8. The Core Layer. It supports identification, communication, loading and security

To give an idea of how the design phase has been made we describe the components belonging to the Core Layer.
The ID component implements general identity management functions by managing a repository containing information about locally generated agents (Fig-

ure 9). This repository is accessed whenever we want to know the current position of an agent.

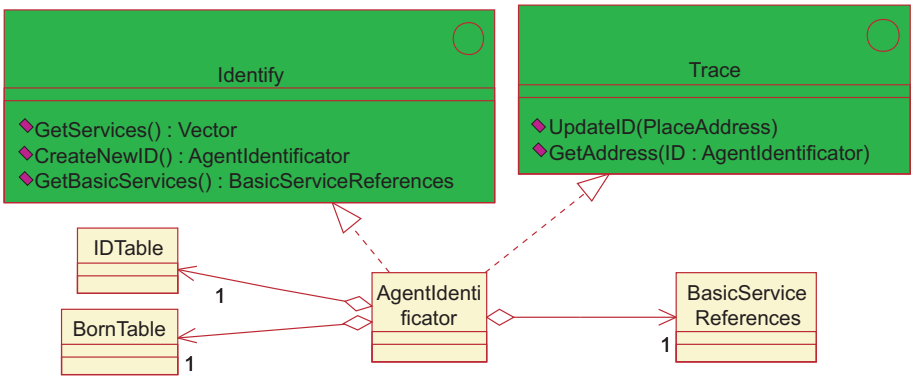

Fig. 9. ID Component

The *ID* component is also responsible for the creation of the identifiers to be associated to new agents. These identifiers contain information about the birthplace, date and time of the agent's creation. Agent localization is simplified by information contained directly in the "ID", such as the birth place. In fact, the birth place of an agent hosts information about the agent's current location.

A second important feature of the *Core* is the *SendReceive* component (Figure 10). This component implements low level inter-platform communication by

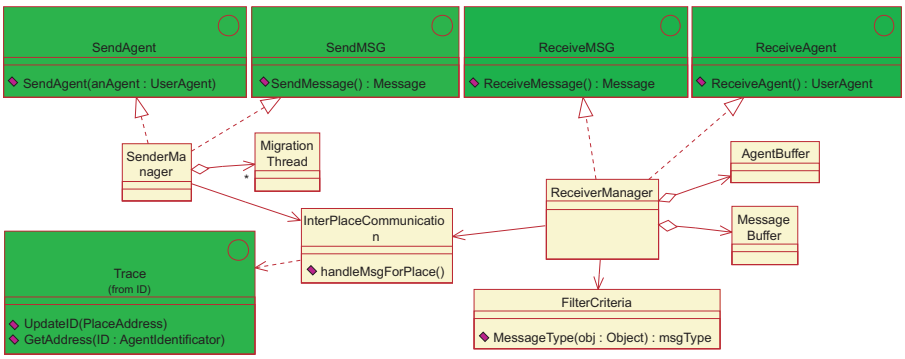

Fig. 10. SendReceive Component

sending and receiving messages and agents. By using the traceability services of-

fered by the *ID* component, *SendReceive* can easily update or retrieve the exact position of a specific *user agent*.

It is important to note that every change in the communication protocol is concealed within the *BasicService layer*. The *SendReceive* component can also send and receive agent instances. This feature is reused by the upper layer to implement agent migration.

The *Starter* component processes any request for agent creation. This particular component, in fact, takes an inactive agent (just created or migrated), and checks it for the absence of malicious or manipulated code. These agents, before activation, are dynamically linked to all basic services of the platform. During execution the agent is isolated from the *Core layer* by the *Basic Service layer*.

The *Security* component, as mentioned above, checks for the presence of malicious code or manipulations within the agent code.

Note that at this abstraction level permissions are not an issue. The code inspection concerns only dangerous agents that attempt to perform illegal operations, such as viruses.

The BasicService Layer *BasicServices layer* (Figure 11) has five main components: *Discovery*, *Mobility*, *Genesis*, *Communication* and *Security Politics*.

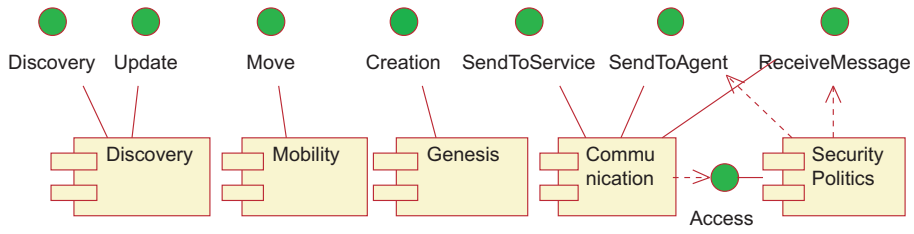

Fig. 11. BasicServices Layer

The *Discovery* component searches and detects service agents. When a user agent wants to communicate with a service, it will ask the *Discovery* for the right identifier to use as the message's receiver. The service detection strategy can be implemented in different ways; for example by a fixed taxonomy or by an UDDI [6], commonly used in the Web Services application domain.

The *Mobility* component enables the movement of code across platforms [27], it implements the interface used by the *UserAgent* and it accesses to components of the *Core layer* to send, receive and load agents. It is important to note that real communication between different locations can be achieved only through *Core*'s *SendReceive* component, and then migration is independent of the type of used transport. *Mobility* consists on copy the agent i.e. its code and its current state

and send it to the destination platform where it will be re-started in a specific point (weak mobility). The local agent is destroyed.

The *Communication* component makes possible to send and receive agent-directed messages both in an intra- and inter-platform context. Intra-platform messages are messages sent between agents and services residing in the same platform. Inter-platform messages are messages sent to agents residing in different platforms (our system does not allow for remote communication between user agents and service agents).

The agent requesting the dispatch of a message does not need to know, effectively, where the target agent is; in fact, the ID is sufficient to post correctly a message. The *Communication* component uses one of the *Security Policy*'s interfaces to ascertain whether the specific *UserAgent* or *ServiceAgent* has the right privileges for communication, if an *Agent* is not authorized to use a service, the message is destroyed.

Before accessing resources and services, an agent must authenticate itself. The identification is performed by sending a login message to a specific *ServiceAgent*, as consequence the *SecurityPolitics* component jointly with the *Communication* component intercept the message and unlock the communication. The *SecurityPolitics* component centralizes control of permissions, protects services and resources from the user agents, and provides the administrator with an easy way to manage all permissions.

The last component of the service layer is the *Genesis* component that enables agent creation. A special case of agent creation is cloning that is performed when it is necessary to create a copy of an existing agent. The two copies differ only for the agent identifier.

A special case of agent creation is cloning that is performed when it is necessary to create a copy of an existing agent. The two copies differ only for the agent identifier.

The Agent Layer The upper layer of the mobile platform, the *Agent Layer*, contains all service and user agents. This layer implements features of the agent-based workflows management system as described in Section 3.1.

This component has not any interface, but is has only several dependencies upon the *BasicService layer*. The *Agent* component contains a general abstract agent class and two inherited classes. *ServiceAgent* consists of agents enabling access to biological databases or providing algorithm. *UserAgent* represents agents created by biologists. User agents execute complex tasks and implement part of the logic of the application.

The HermesV2 Java implementation, has been completely designed and developed following this approach [29]. The middleware we have implemented is separated into several functional units (components) with mutual dependencies explicitly documented by UML diagrams.

We would like to mention that such an approach, based on layers and components, supports the generation of middleware for different domains as shown in [9].

3.3 Main Services and Tools for Hermes

In this section we describe some aspects, that have been significant for the implementation of Hermes middleware. The programming environment offered by Hermes consists of several tools both for design and execution of distributed applications. Some tools turn into *Agent* services, e.g. those that support resources access, resource localization, resource selection, schema mapping, etc. some others remain tools usable during workflows design, analysis, verification phases. Among *Service* agents we mention:

AIXO: XML Generalized Wrapper

AIXO is a tool developed to present any data source as a collection of XML documents. AIXO is flexible and modular, it allows to manage many input data sources ranging from HTML to XML, databases, flat file, CGI and command line programs. AIXO has been experimentally used on different resources in different contexts and successfully integrated as *wrapper service agent* in Hermes [4].

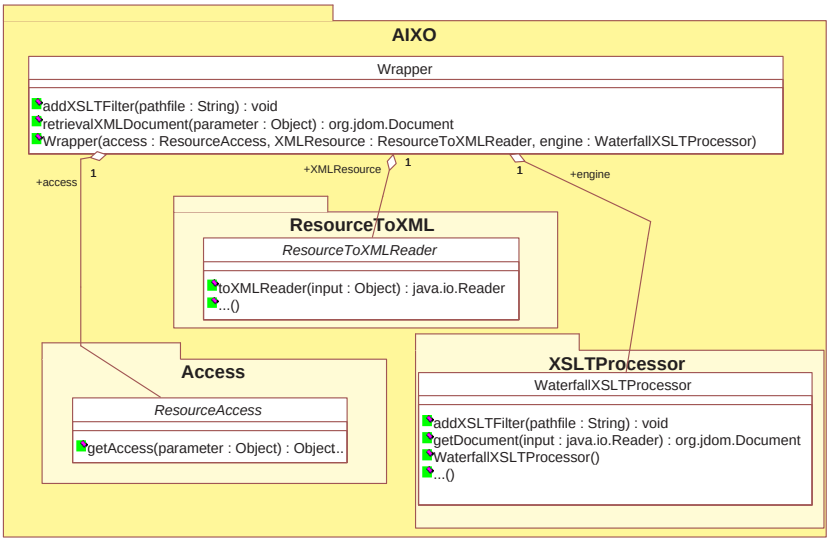

Fig. 12. AIXO architecture

The AIXO architecture is not for a specific resource or data type; rather, it is general and suitable for a wide range of resources. An AIXO *Service* agent

implementation offers a wrapper that provides an "XML view of the resource". The AIXO architecture, shown in Figure 12, is composed of three main packages: *ResourceAccess*, *ResourceToXML*, *XSLTProcessor*.

ResourceAccess manages access to the resource to be wrapped. Its implementation depends upon the communication protocol, permissions, and access policies. By using the *ResourceAccess*'s interface, data can be gathered from the resource in its native format; there is no transformation. For example, in the case of a Relational DataBases (RDB), the data obtained is contained in a "recordset".

ResourceToXML transforms data, provided by the *ResourceAccess* module, into XML. The transformation is *canonical* and independent of the data's semantics. Mapping from the original format to XML is performed considering only the data's structure. For example, in transforming a recordset to XML, the output conforms exactly to the schema of the table; in the case of a flat file, the transformation will derive its structure taking into account special characters such as tabular and white spaces. For an HTML text, the transformation extracts the document schema from the tags.

Finally, the *XSLTProcessor* applies a set of XSLT filters to the raw XML, provided by the *ResourceToXML*, to obtain the effective XML view of the resource. In this phase, the semantics of data plays an important role.

To create a concrete wrapper the *ResourceAccess* and *ResourceToXML* Java classes must be implemented and the *XSLTProcessor* must be configured using the appropriate set of XSL Transformations. Each wrapper is defined by an XML configuration file. The system automatically loads classes and initializes attributes. AIXO has been experimentally proven on different resources in different contexts [4].

AIXO *Service* agent can interact with OMSE (ontology management *Service* agent), below described, to dynamically find the mapping among resources schemas. An example of AIXO at work is given in Section 4.

WS^{2}A: a Web Service *Service* Agent

WS^{2}A is a Web Service *Service* Agent, a tool developed to access Web Services and to derive at run-time the resources access methods [60]. Briefly a Web Service is an interface which describes a set of service access methods usable through the network via XML messages. The interface hide any service implementation details.

This tool is successfully used during the research and selection process of a service that a MAS (matchmaker *Service* agent) supports. WS^{2}A is characterized by a peculiar communication among agents which allow to manipulate unknown objects at run-time. In particular, data exchanged among agents do not use messages but objects and by using JAVA reflection technique we support the manipulation of unknown data.

MSA: Matchmaker Service Agent

Service discovery is the process of localizing resources and services available in large scale open and distributed systems. In a distributed and redundant sys-

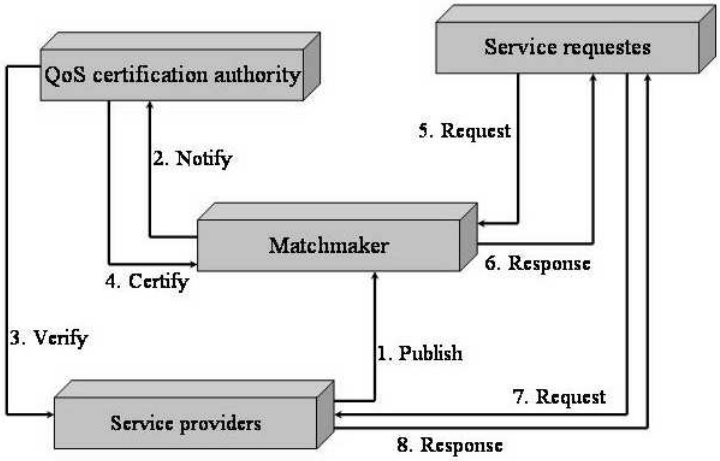

Fig. 13. Matchmaker Service Agent. Any gray box represent an agent active in the distributed environment

tem as the Web, it is necessary, beside localizing services, to filter them in order to obtain those which are best for the activities for which they have been requested. By the term *matchmaker* we mean a software entity, a *service* agent, which monitors services availability, maintains an updated file of all useful information for using services and possibly ensures a quality choice of them. We have developed a matchmaker and defined a *quality model* based on parameters that ensure the best choice of a service for a specific application domain. The communication protocol among matchmaker and other agents is given in Figure 13. A full description of the tool is provided in [14]. The quality model consists of two components, the first describes general quality aspects of the distributed computational environment where the service is offered, we have considered the Web, and the other includes quality features of the application domain. Any resources must fulfil the following requirements:

- *Aim*: the purpose for which the resource has been developed;
- *User target*: the list of hypothetical users;
- *Reliability*: the probability of successfully using a resource;
- *Feasibility*: the measurement of the easiness to access the resource;
- *Usability*: the measurement of the easiness to use the resource;
- *Originality*: the degree of correctness of the resource and its information;
- *Privacy*: the legal conditions of using the resource;
- *Updating*: the attendance of the resource updating;
- *Uptiming*: the maximum length of time between two resource failures;
- *Timing*: the daily time of resource activity;
- *Speedy*: the measurement of the execution time;
- *Browsing*: the measurement of the human easiness to find a resource;

– *Popularity*: the number of active consumers;

Each quality aspect above defined is quantitative measured on the basis of several parameters whose description if given in [49, 22]. The domain-dependent quality aspects is provided in section 4.

OMSA: Ontology Management *ServiAgent* **Agent**

The availability of automatic tools for quickly determining semantic similarity among concepts across different ontologies is useful during the processes of data retrieval and data integration, in Hermes performed by AIXO. We have developed a tool which supports the ontology management to support the mapping between domain ontology and local schema used to defines data repositories. To that purpose we have defined a similarity algorithm to compare two ontologies. The main idea is, supposing to have, in each execution environment, a shared global ontology and a local ontology, the algorithm determines similar concepts (i.e., data types, formats and terms) by computing the number of identical relationships among two concepts of different ontologies and recursively to all their derived concepts as well. The algorithm is considered an instrument that any mobile *service* agent can use to compare two ontologies, usually the application domain ontology shared at user level and that derived from the local resources schema. The detailed description of the similarity algoritm is given in [18] while an example of how the tool can be used is provided in Section 4.

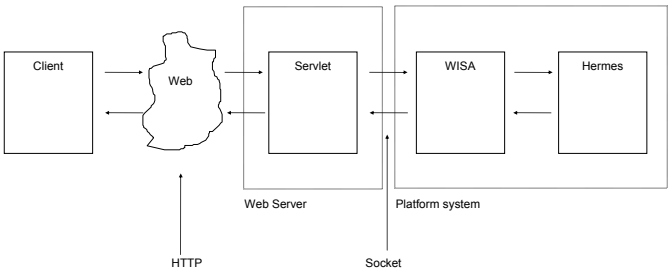

Fig. 14. WISA: Web Interface Service Agent

lightTS-SA: lightTS *Service* **agent**

lightTS-SA is a *Service* agent developed to support a coordination agents via tuple space. lightTS [48] is a Java package which provides a lightweight tuple space implementation. Light because lighTS does not support the persistence,

security and remote access, features that can be provided by the run-time support. We have used this service especially to coordinate agents that move in place where they do not know how to contact local services, but they can interact with lightTS service agent which comes between the requester and the provider of a service.

WISA: Web Interface *Service* Agent

WISA is a *Service* agent realised to support the expert programmer whose want to directly interact with Hermes at system layer. This *Service* agent has been designed to support some operations which characterize a user session: manage your personal account, create an agent, send an agent, get the output of the execution. To generalize the interface, the WISA communication protocol, described in Figure 14, does not allow the "Client" to directly communicate with WISA because the first one uses the HTTP protocol and the second one uses a protocol based on Socket and XML. To make possible the communication is needed a third component: a Web Server. The Web Server must support application server-side (Java Servlet, JSP, ASP, CGI, PHP etc.).

WfSA: Workflow Interface *Service* Agent

WfSA is a *Service* agent developed to provide an interface to end user which designs his workflow by combining the activities chosen from a give list. Note that the list of activities are those implemented at the system layer of the Hermes software architecture. The interface configured for a bioinformatics domain is give in Section 4.

Analysis and verification tools for workflow

A further aspect we have dealt with is the possibility to used an automatic tool to analyse and verify the behaviour of the workflow that a user can design. Recalling that, in Hermes, a distributed applications is a workflow of activities, designed by a graphical notation usually made by JaWE editor. We have verified that there is a correspondence between the JaWE notation and UML Activity Diagram [59]. Then we have provided a process algebra view of workflows described in terms of UML activity diagrams by defining an interpretation of activity diagrams into CSP-like process algebra terms. Similar results could be obtained if we represents the workflow by a Petri Nets. To provide Hermes with a verification tool based on CSP-process algebra to apply to user workflow, we have exploit an intermediate relational language as a bridge between activity diagrams and process algebra terms as shown in the sequel and detailed discuss in [1]. The obtained results do not only show a conceptional relationship between two different notations. The advantage of our comparison is twofold. On one hand we provide different notations for "the same" system abstraction: a textual description (process algebras terms) and a graphical notation (workflows). This can be very useful during the system life cycle. On the other hand process algebras are associated with formal semantics and this has allowed the proliferation of automatic tools for system specification and verification so that our results open the possibility to exploit such tools for the verification of workflows.

4 Application Scenarios

Scenario 1: Hermes for Bioinformatics

The scenario we refer to is related to a biological domain. In the post-genomic era, the amount of available information is constantly increasing, and it is difficult to exploit available data from all sources [26]. As an example we take the context of Oncology over Internet project [44], that aims to develop a framework to support searching, retrieving and filtering information from Internet for oncology research and clinics.

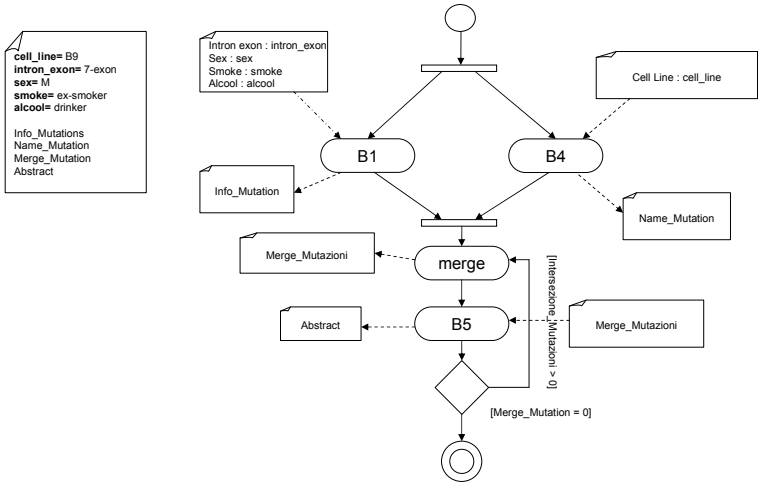

Fig. 15. Example of User Level Workflow in Bioinformatics Domain

Suppose the application domain involves the use of biological resources (micro-organisms, cell lines, mutations) that are essential for implementing a good, reproducible experiment. Established that high quality biological resources are available at some specialized centres (Biological Resources Centers:ATCC, DSMZ,) and their catalogues are available on-line and that many researchers assessing molecular biology databases often need find more information regarding resources to finally request materials.

Suppose to have three different domain each of one characterized by a set of activities as here described: *Cell Line domain*={A1: Find information about the cell line named x, A2: Find all cell lines derived from a specific tumour or

pathology, A3: Find all Cell Lines producing a specific protein, A4: Given a specific Cell Line, find all related bibliographic references A5: Given a specific Cell Line, find all information about produced proteins}, *Mutation*={ B1: Find all mutations observed in a specific intron/exon in subjects with specific sex and life habits (i.e. smokers/ drinkers), B2: Find all mutations in subjects affected by a given pathology, B3: Find all subjects affected by a tumoural pathology and with a given protein mutation, B4: Find all mutations observed by using a given cell line, B5: Given a specific mutation, find all abstracts of the correlated bibliographic references} and *Bibliographic resources*= {C1: Select all abstracts of bibliographic references, whose text includes a given term}.

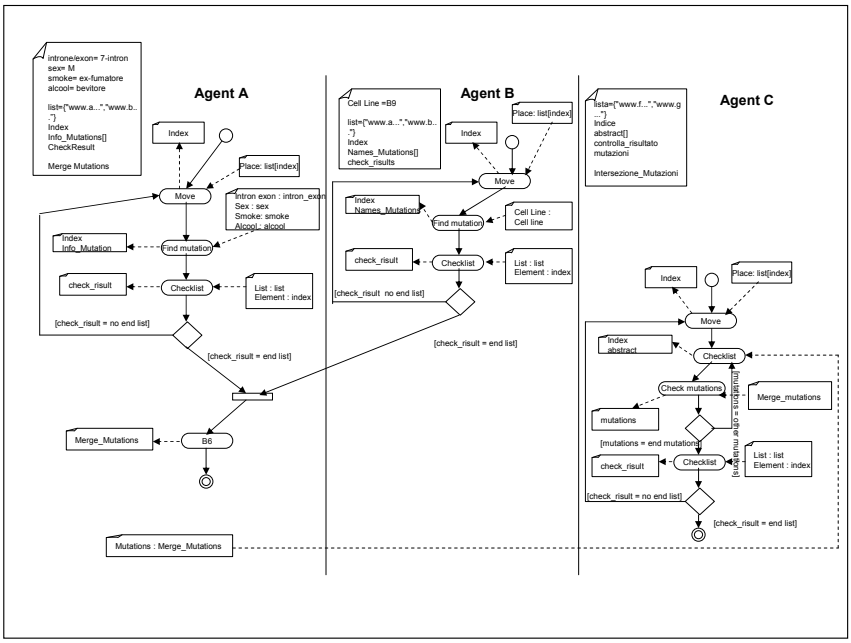

Fig. 16. An Example of Agent Level Workflow in Bioinformatics Domain

As an example consider a workflow defined to verify a *mutation* experiment by reproducing it. In particular a workflow that has a goal to retrieve abstracts from a literature databases for identifying the best *cell line* for reproducing a human TP53 mutation experiment linked to a particular tumour-habits-sex combination. Any single activity of the workflow uses bioinformatics services available on Internet in order to achieve the desired result. The user will select activities B1,B3 and B4, will provide parameters to each one: B1. Retrieve all

mutations (IDs) observed in the 7th exon in men who are ex-smokers and drinkers by searching p53 mutations database SRS implementation at IST, Genova; B4. Retrieve all mutations (IDs) observed by using B9 cell line as original resource by searching p53 mutations database SRS implemerntation at IST, Genova; B5. Retrieve all abstracts of the correlated bibliographic references, of a specific mutation ID by searching Medline. And will combine them by the workflow operators as described in Figure 15.

Hermes in the context of O2I project is called Bioagent [62], it supports the design of user workflow by the interface shown in Figure 17, i.e a Workflow *Service* Agent (WfSA). The context-aware compiler will produce the set of mobile user agents whose behaviours are described in Figure 16 and implemented by a set of activities, called use cases in the Figure 17, and stored in the knowledge base. The user get the result in XHTML.

The Figure 18 shows a typical interaction between a bioscientist and *user* agents involves the following steps:

1. a bioScientist specifies the set of activities to be performed;
2. the compiler system generates a pool of *user* agents to execute the activities;
3. *user* agents migrate and clone in order to efficiently accomplish the activities;
4. agents query resources by interacting with local *service* agents. *service* agent map the query to local schema by using AIXO which implements the abstraction layer so that agents interact only with XML documents. In the case in which an AIXO service agent has to manage different types of documents (ontologies mismatching) can interact with OMSE and use the ontology similarity algorithm previously mentioned.
5. *user* agents merge results and furnish data to the bioscientists.

In this example AIXO *Service* agent is used both to retrieve and to present resources as XML documents.

To prove the flexibility of Hermes middleware we now briefly describe a case study we have recently made [17] by using Hermes for systems biology [36], i.e. bioinformatics area which aim to understand how biological systems function. A cell consists of a large number of components interacting in a dynamic environment. The complexity of interaction among cell components and functions makes design of cell simulations a challenging task for biologists. We have used an agent-oriented methodology to design a cell components as autonomous software entities (agents) situated in an environment and communicating via high-level languages and protocols (ontologies), may be a natural approach for such models. We constructed a model of cellular components involved in the metabolic pathway of *carbohydrate oxidation*. To give an idea of approach, the Figure 19 shows the set of agents identified be autonomous part of the system while Figure 20 shows the behavior of the only one component. Note that the UML Activity Diagram described a workflow of activity which in turn is executed by a pool of mobile agents which represents small components of the cell.

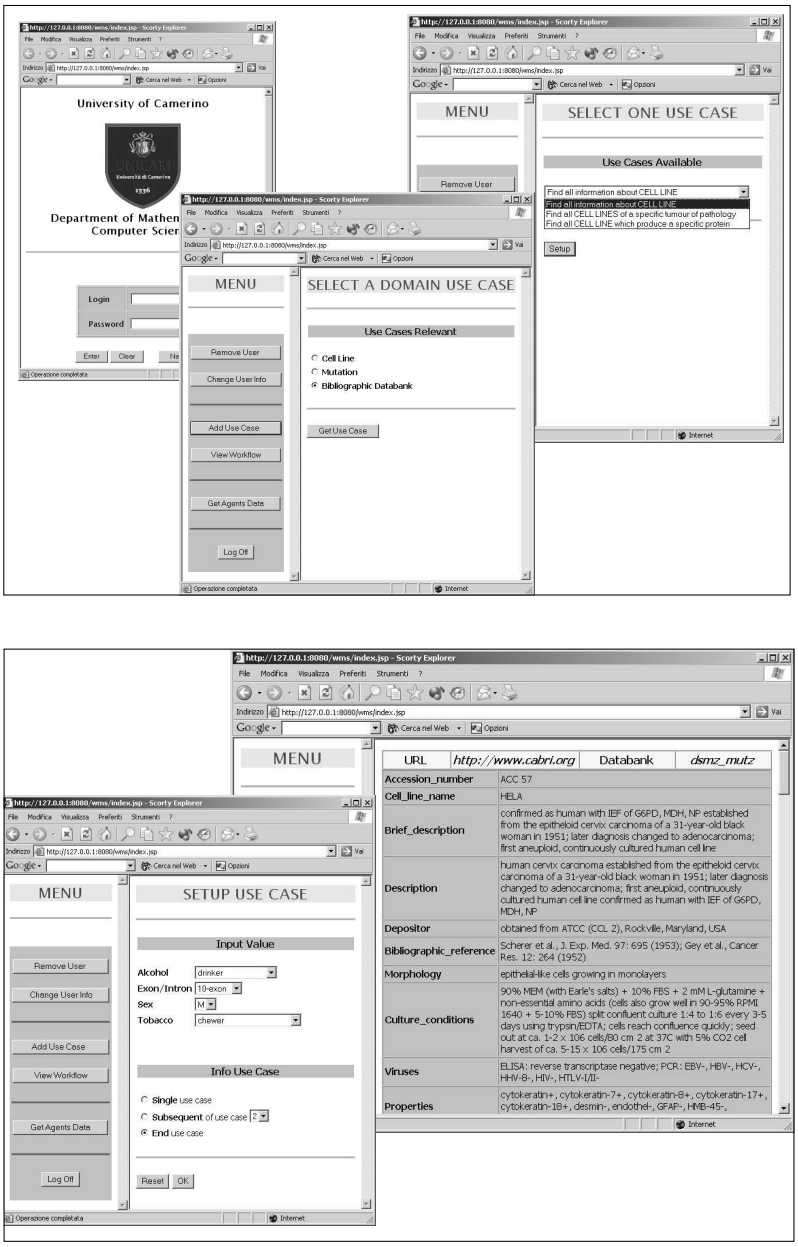

Fig. 17. User Interface for Workflow Management in Bioinformatics Domain

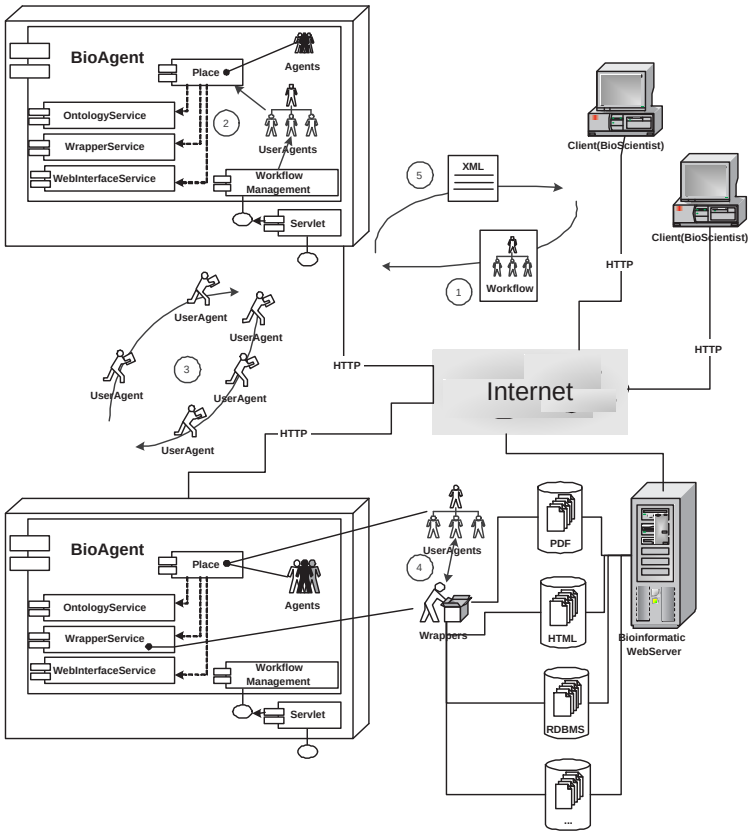

Fig. 18. Interactions Between Agents and AIXO wrappers within Bioagent[62], i.e. Hermes configured for Biologists

Scenario 2: Hermes for Industrial Control

Now, we focus on the industrial control case study, in particular the domain of quality control. In a supply chain, the actors are the suppliers and the production plans; the former usually provide both raw and semi-manufactured materials while the latter assemble the various input components to produce a final, more complex assembled product. We are interested to develop an application for the traceability of the different components and semi-manufactured products in terms of quality.

At first sight this context, geared towards quality, reflects problems with the integration of heterogeneous data. In fact, each single supplier uses his own quality control mechanisms and stores results of test in his own format. The goal is to integrate and rendered readily accessible all these data among manufacturers. It would be useful, once a defect or malfunction in the final product has been identified, to be able to trace and recover all information regarding quality that

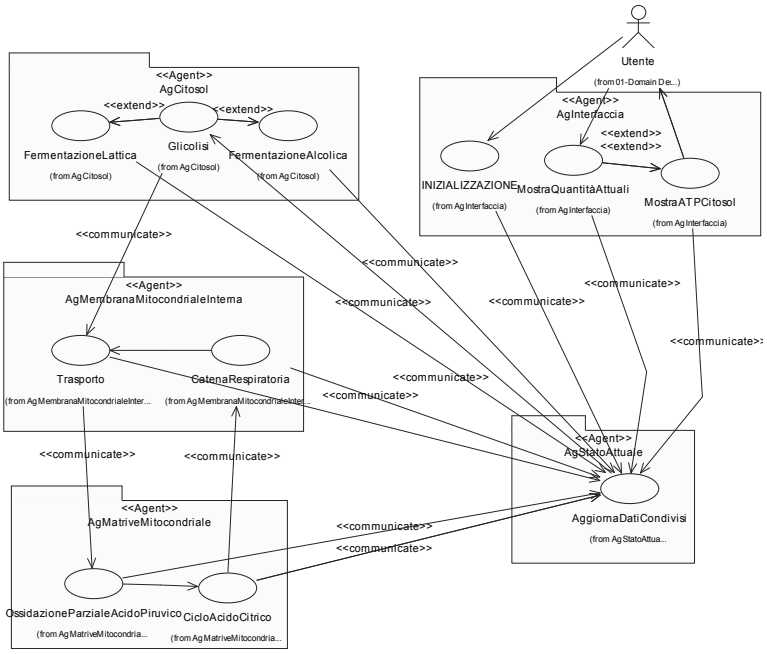

Fig. 19. The Cellular Agents Identification Diagram

has been generated by the different tests and controls on components composing the faulty product.

An agent-based system can be the technology exploiting resources and services integration in the manufacturing applicative domain, but several issues must be taken into account. Embedded systems that perform the various quality tests of the products are very heterogeneous, and data is stored in repository providing access services that differ significantly. The security issues, moreover, play a vital role all along the supply chain. In fact, both generated reports and embedded checking system must be protecting from malicious access.

The supply chain consists of federated enterprises: many suppliers, a production plant, a distribution center and a technical service center. Each enterprise is characterized by a specific role and carries out a set specific tasks in the virtual organization.

The complete set of tasks includes quality testing, performance testing, reporting on damages incurred during shipment, and reports on repairs carried out directly to the customer.

Suppose that the Production plant receives a communication of the nth fault of a washing machines family. The responsible of the plant could decide to ana-

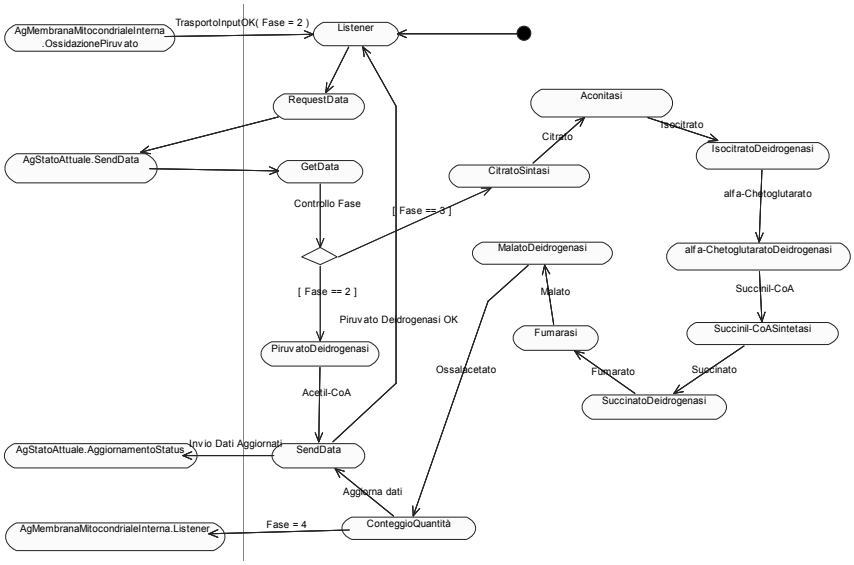

Fig. 20. An example of agent workflow, the Mitochondrial Matrix Activity Diagram

lyze the complete life-cycle testing *quality* data of the signaled washing machines family.

To that purpose he must identify any suppliers involved in the production of the washing machine and retrieve from them all distributed data regarding testing *quality* data.

Figure 21 shows a possible quality-oriented workflow which describes the human aim. The workflow consists of domain specific activities regarding any retrieval phases.

The quality-oriented workflow can be mapped into an agent-oriented workflows (Figure 22) and then compiled into a pool of agents (agent society) specialized to execute one or more activities. Among those we find *Manager Agents*, *Test Agent* and *Fragmenter Agents*. Those agents, once created, have the main features to be completely autonomous and running all the time for its goal.

Manager Agents has the goal to create the final testing report by interacting with *Test Agents* and *Fragmenter Agents*. The final testing report, created by an XML template, will include all *quality* data of the washing machine, testing reports of any single components and all defects recorded during the product's life-cycle. *Test Agent* has the goal to retrieve *quality* data for a single component by communicating with remote *Wrapper Service Agents* (running on remote site). *Fragmenter Agents* has the goal to decompose a complex domestic device

(washing machine) into a list of semi-manufactured products and raw materials (components).

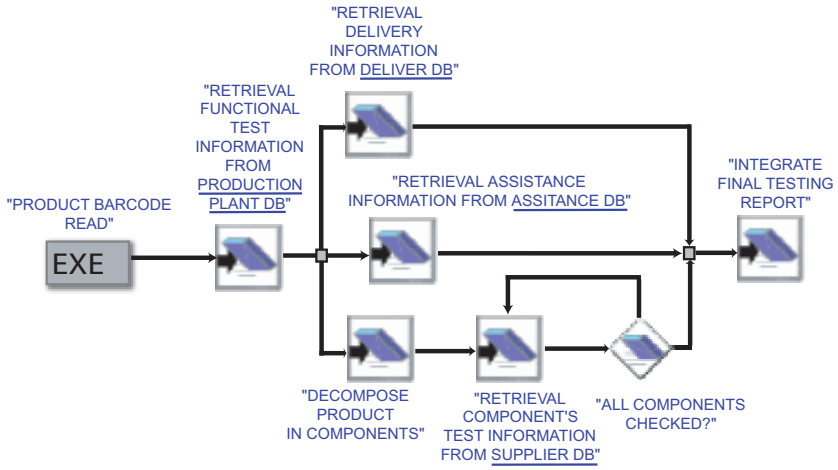

Fig. 21. Quality-oriented Workflow for the Functional Testing in the Production Control Plant

Scenario 3: Hermes for Pervasive Computing

The pervasive and ubiquitous devices are computational and control systems, located in domestic environment (domotica) and in a manufactured articles. These devices are often either masked or invisible therefore they can assist us in the shadow. In this scenario, the microcontroller is the computational system for excellence. A microcontroller is a computer system that centralizes, in a single chip, all the functionalities needed to control and manage electrical domestic appliances and automotive systems. One of the interesting characteristics of the microcontrollers is low cost that favours a quick and wide spread among many manufactured articles that surround us. A problem is the huge variety of microcontrollers offered by producers to satisfy the demand.

A great number of microcontrollers use devices like bluetooth [7], echelon [20], WLan [61], IrDA [58] that allow interactions between devices. Many enterprises provide protocols and services to allow connection between computers and devices, like SUN with Jini [35]. However, these solutions do not conciliate the computational resources of the microcontrollers with protocols flexibility, the cost and the variety of the microcontrollers.

In this scenario, we have defined a virtual machine that makes transparent the differences among microcontrollers and supports connectivity without defining new protocols and to realize a secure environment for pervasive and ubiquitous computing.

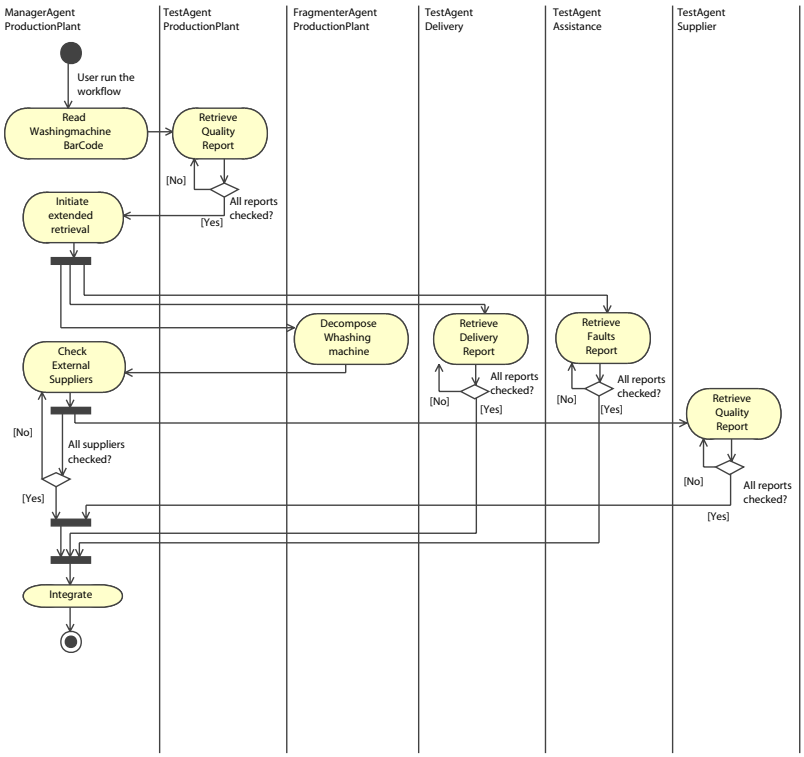

Fig. 22. Agent-oriented Workflow

To support a secure communication, and keep track of a mobile agent, we have chosen a hierarchical structure: each agent may know only the ID of its father agent (its creator) and its children agents

Clonation, *mobility* and *communication* have been identified as the kernel primitives in the microcontroller environment. Clonation allows to duplicate the code and the state of a running code. After a clonation there will be two identificable codes in execution. Mobility allows a code to move on other execution platform in proactive mode: a copy of the code and its current state is moved to the destination platform for being started from a specific point (weak mobility). Unlike the clonation, the code that performs the move primitive comes destroyed if the execution of the movement primitive succeeded. Communication directly results from the clonation primitive. After a clone operation will be created an exclusive communication channel between cloning and cloned codes. Communication (through exchange of messages) is possible only between cloning and cloned. Messages are sent-received in asynchronous-synchronous fashion.

The virtual machine relies on a calculus which describes the semantics of the minimal set of operation isolated to characterize a platform supporting mobile code [13]. The calculus for modelling mobile applications is summarized here below.

$\mathbf{A}$ is a set of basic actions, $\mathbf{A}_\tau = \mathbf{A} \cup \{\tau\}$, where τ is used to represent internal activity. $\mathcal{N}_{id}$ and $\mathcal{N}_p$ are an infinite sets of names of mobile processes and platforms,resp. $\mathcal{M}$ is an infinite set of messages.

Definition 1. (*mobile processes*)

The set $\mathbf{S}$ of sequential programs and the set $\mathbf{M}$ of mobile processes (sequential programs in execution) are generated by the following grammar:

$$S ::= \mathsf{nil} \mid \alpha.S \mid \mathsf{clone}(S).S \mid \mathsf{send}(m).S \mid \mathsf{receive}(m).S \mid \mathsf{go}(p, S)$$

where $\alpha \in \mathbf{A}_\tau$, $m \in \mathcal{M}$ and $p \in \mathcal{N}_p$. The set $\mathbf{M}$ of mobile processes (codes in execution) is generated by the following grammar:

$$M ::= \mathsf{NIL} \mid \mathsf{init}(S, \mathsf{SP}) \mid id : \{\mathsf{SP}, S, A\} \mid M_1, M_2$$

where $id \in \mathcal{N}_{id}$, $S \in \mathbf{S}$, $\mathsf{SP} \in \mathcal{P}(\mathbf{M})$ and $A \in \mathcal{P}(\mathcal{N}_{id})$, nil represents a terminated sequential program.

A process whose sequential behaviour is $\alpha.S$, $\mathsf{send}(m).S$, $\mathsf{receive}(m).S$ and $\mathsf{clone}(S_c).S$ can execute α, send and receive a message in $m \in \mathcal{M}$, clone itself and then behaves as S. $\mathsf{go}(p, S)$ instructs a process to migrate to a destination platform named p and then behaves as S.

The component-based approach, used to developed Hermes and discussed in Section 3.2, allows to create new components in the Hermes core, by reusing the existing ones. We have developed a version of Hermes which adapt its components to the hardware characteristics of the microcontrollers to guarantee the function required by middleware. As an example, the communication component can be adapted for various technologies [7, 20, 35].

We have configured the Hermes platform for running with CDC of SUN [55] on PDAs. The porting of the Hermes on one particularly compact JVM for microcontrollers (like CLCD of SUN of type KVM [56, 53]), according to [53] needs libraries for sockets, serialization and reflection. Unlike many other platforms, Hermes does not use RMI [35]. The only pre-requirement on the microcontroller is the presence of a JVM. The *core* of the platform plus AIXO service is between 120KB to 160KB. Therefore the implementation of HermesV2 over a microcontroller would supports the following functionalities:

Communications peer to peer. At most two platforms are involved in every communication. Therefore it is possible to realize it without involving other partners [12]. Every platform must only store information in order to realize communication between clonated and cloning codes currently in execution on it. This implies limited traffic of service between platforms and small tables. Substantially are draft communications to local environment.

Communications deadlock free. The communications of this model are deadlock free. According to the hierarchical structure of communication, it is impossible to establish the condition of circular wait for more that two actors. Moreover, the situation of circular wait happens only if both the actors, cloning and clonated, establish a synchronized communication. Such a situation can be easily prevented imposing that, before sending a synchronous message m, a code must control that in its own queue there is not a synchronous message sent by the receiver of the massage m. In this case the communication simply fails with exception.

Absence of timeout. Since the actor of a communication are always two codes, cloning and clonated, that may also reside on two different platforms, it is possible to determine the cause of failures in the communications. Consequently the code that sends or receives a communication can know the exact cause of the failure and always undertake appropriate operations. This is not always the case in systems where the communications is based on timeouts.

Absence of communication protocols. The communications between cloning and clonated and vice versa is not subject to protocols (ACL [24]) since the code of cloning is the same of clonated.

Security. All the requests of a code in a clonation tree (or forest of clonation trees) realize a closed system and a set of predetermined communications. The identifier produced after a clonation is only known by the cloning and it is the only handle in order to allow communication between cloning and clonated. Beyond to the communications towards the services, other shape of communication for the user code does not exist. The control of the communications allows to remove or to supply grant to the codes in execution. By removing all the communications a code becomes completely innocuous.

Correctness. Since a pool of instance relative to a code is ties at the communication network, it's possible to simulate dynamic behaviour in static background. It would be enough to eliminate from the code the primitive of mobility in order to verify the behaviour gearless of context. Moreover since the communications happen between copies of the same code is possible to verify the correctness analyzing the graph of the possible states that it can assume a code in execution [46].

5 Conclusion and future perspective

Mobile computing systems are computational systems that may be easily moved and whose computing capabilities may be used while they are moved. Several middleware have been proposed for mobile computing [47, 45] most of them focus on communication and coordination of distributed components. Indeed, we concentrate on a user not expert programmer, on workflow as suitable technology to hide distribution and on mobile agent as flexible implementation strategy of workflow in a distributed environment.

Our experience in developing applications in several application domains, convinces us on the necessity to create an integrated, flexible programming en-

vironment, whose user can easily configure for its domain. This leads to the developing of Hermes middleware. Hermes is structured as a component-based, agent-oriented, 3-layered software architecture. It can configured for specific application domains by adding domain-specific component libraries. The user can specify, modify and execute his workflow in a very lightweight.

Workflow is specified abstractly in a graphical notation and mapped to a set of autonomous computational units ($UserAgents$) interacting through a communication medium. The mapping is achieved by compiler that is aware not only of contents of a library of implemented user activities but also the software and hardware environment to executing them. In our case it includes also available services ($ServiceAgent$). A user specifying workflow need not to be concerned with where to search for information, in what form information is stored, the protocol for interacting with each services or the low level details that can be left to the context-aware compiler.

We are moving to the definition of a domain-specific mobile agent language to support as target language of the workflow compilation. We also plan to study the integration of Hermes with Klaim to allow the formal verification of agent-oriented workflow. Finally, we aim to experiment the use of the abstract machines for systems biology as one of the domain-specific language.

Acknowledgements

We wish to thank all the students have been involved, ver the last years, in the development of Hermes, among them we would like to mention Francesca, Davide, Lorenzo, Ezio, Leonardo, Marco, Chiara and Barbara. A special acknowledge is due to Rosario Culmone, Leonardo Mariani and Diego Bonura with who we have taken the most important development decisions.

We would like to thank Michal Young for valuable comments on a preliminary version of this paper.

References

[1] R. Amici, D. Cacciagrano, F. Corradini, and E. Merelli. A process algebra view of coordination models with a case study in computational biology. In *Proceedings of 1st International Workshop on Coordination and Petri Nets, PNC'04*, 2004.

[2] M. Angeletti, R. Culmone, and E. Merelli. An intelligent agent architecture for dna-microarray data integration. In *NETTAB Workshop on CORBA and XML: Towards a bioinformatics integrated network environment*, Genova, 2001.

[3] R. B ' Far. *Mobile Computing Principles*. Cambridge University Press, 2005.

[4] E. Bartocci, L. Mariani, and E. Merelli. An XML view of the "world". In *International Conference on Enterprise Information Systems, ICEIS*, pages 19–27, Angers, France, April 2003.

[5] E. Bartocci, S. Moeller, L. Todo, and E. Merelli. Integration of ensembl with bioagent. In *Abstract book of the Biocomp - Gruppo di Cooperazione in Bioinformatica*, 2004.

[6] T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen. UDDI version 3.0. Published specification, Oasis, 2002.

[7] Bluetooth. http://www.bluetooth.org.

[8] D. Bonura, F. Corradini, E. Merelli, and G. Romiti. Farmas: a MAS for extended quality workflow. In *2nd IEEE International Workshop on Theory and Practice of Open Computational Systems*. IEEE Computer Society Press, 2004.

[9] D. Bonura, L. Mariani, and E. Merelli. Designing modular agent systems. In *Proceedings of NET.Object DAYS, Erfurt*, pages 245–263, September 2003.

[10] L. Cardelli. Abstract machines of systems biology. In *Transaction on Computation System Biology, special issue for NETTAB Workshop on Model and Metaphors from Biology to Bioinformatics Tools*, Lecture Notes in Computer Science. Springer-Verlag, 2005. to appear.

[11] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):117–213, 2000.

[12] N. Carriero, D. Gelernter, and T. G. Mattson. Linda in heterogeneous computing environments. In *Proceedings of the Workshop on Heterogeneous Processing*, pages 43–46, Beverly Hills, CA, March 1992.

[13] F. Corradini, R. Culmone, and M. R. Di Berardini. Code mobility for pervasive computing. In *2nd IEEE International Workshop on Theory and Practice of Open Computational Systems*. IEEE Computer Society Press, 2004.

[14] F. Corradini, C. Ercoli, E. Merelli, and B. Re. An agent-based matchmaker. In *proceedings of WOA 2004 dagli Oggetti agli Agenti - Sistemi Complessi e Agenti Razionali*, 2004.

[15] F. Corradini, L. Mariani, and E. Merelli. A programming environment for global activity-based applications. In *proceedings of WOA 2003 dagli Oggetti agli Agenti - Sistemi Intelligenti e Computazione Pervasiva*, 2003.

[16] F. Corradini, L. Mariani, and E. Merelli. An agent-based approach to tool integration. *Journal of Software Tools Technology Transfer*, 6(3):231'244, November 2004.

[17] F. Corradini, E. Merelli, and M. Vita. A multi-agent system for modelling the oxidation of carbohydrate cellular process. In *First International Workshop On Modelling Complex Systems (MCS 2005)*, Lecture Notes in Computer Science. Springer Verlag, 2005. To appear.

[18] R. Culmone and E. Merelli. An semantic comparison of ontologies. Technical Report TR02, Dipartimento di matematica e Informatica, Universit di Camerino, 2003.
[19] R. De Nicola, G. L. Ferrari, and R. Pugliese. Klaim: A kernel language for agents interaction and mobility. *IEEE Transaction of Software Engineering*, 24(5):315–330, May 1998.
[20] Echelon. http://www.echelon.com.
[21] Enhydra. Jawe. http://jawe.enhydra.org/, 2003.
[22] C. Ercoli. Un modello di qualità per la scelta di servizi web in ambito biologico - il middleware. Master's thesis, Laurea in Informatica, Università di Camerino, a.a. 2003-2004. http://dmi.unicam.it/merelli/tesicl26/ercoli.pdf.
[23] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In N. Adam, B. Bhargava, and Y. Yesha, editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.
[24] FIPA. The foundations for intelligent physical agent. http://www.fipa.org.
[25] FIPA-ACL. FIPA97 specification, part 2: Agent communication language. Specification, FIPA, October 1997.
[26] D. Frishman, K. Heumann, A. Lesk, and H.-W. Mewes. Comprehensive, comprehensible, distributed and intelligent databases: current status. *Bioinformatics*, 14(7):551–561, 1998.
[27] A. Fuggetta, G. Picco, and G. Vigna. Understanding code mobility. *IEEE Transaction of Software Engineering*, 24(5):352–361, May 1998.
[28] D. Gelenter. Generatve communicationin linda. *ACM Computing Survey*, 7(1):80–112, 1985.
[29] HermesV2. http://hermes.cs.unicam.it.
[30] D. Hollingsworth. The Workflow Reference Model, January 1995.
[31] IBM. TSpace web page. http://www.almaden.ibm.com/cs/TSpace.
[32] Javapace. The javaspace specification web page. http://www.sun.com/jini/spec/js-spec.html.
[33] J. Jayashankar M. Swaminathan, S. Smith, and N. Sadeh. Modeling supply chain dynamics: A multiagent approach. *Decision Sciences*, 29(3), 1998.
[34] N. R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, April 2001.
[35] JINI. Jini network technology. http://wwws.sun.com/software/jini.
[36] H. Kitano. *Foundations of Systems Biology*. MIT Press, 2002.
[37] A. C. R. Martin. Can we integrate bioinformatics data on the internet? *Trends in Biotechnology*, (19):327–328, 2001. (Meeting Report).
[38] C. Mascolo, L. Capra, and W. Emmerich. Middleware for mobile computing (a survey). In E. Gregori, G. Anastasi, and S. Basagni, editors, *Neworking 2002 Tutorial Papers*, volume 2497 of *Lecture Notes in Computer Science*, pages 20–58. Springer-Verlag, 2002.
[39] E. Merelli, R. Culmone, and L. Mariani. Bioagent: a mobile agent system for bioscientists. In *NETTAB Workshop on Agents nd Bioinformtics*, Bologna, July 2002.
[40] E. Merelli, P. Romano, and L. Scortichini. A workflow service for biomedical application. In *Abstract book of the Biocomp - Gruppo di Cooperazione in Bioinformatica*, 2003.
[41] M. Merz, B. Lieberman, and W. Lamersdorf. Using mobile agent to support interorganizational workflow management. *Applied Artificial Intelligence*, 11(6):551–572, 1997.

[42] J. P. Milner, R. and D. Walker. A calculus of mobile processes, part 1-2. *Information and Computation*, 100(1):1–77, 1992.

[43] A. L. Murphy, G. P. Picco, and G.-C. Roman. Lime: A middleware for physical and logical mobility. In F. Golshani, P. Dasgupta, and W. Zhao, editors, *Proceedings of the 21st International Conference on Distributed Computing Systems*. ACM Publisher, 2001.

[44] O2I. Oncology over internet, strategic project founded by italian nationa research minestry. http://www.o2i.org.

[45] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, Sept. 1999. Special Issue: Coordination Mechanisms for Web Agents.

[46] M. Pezzé, R. N. Taylor, and M. Young. Graph models for reachability analysis of concurrent programs. *ACM Transaction on Software Engineeringn and Methodology (TOSEN)*, 4(2):171–213, 1995.

[47] G. P. Picco, A. L. Murphy, and G.-C. Roman. Lime: Linda meets mobility. In *Proceedings of the 21st International Conferece on Software Engineering (ICSE'99)*, pages 368–367, May 1999.

[48] G. P. Picco, A. L. Murphy, and G.-C. Roman. Developing mobile computing applications with lime. In *International Conference on Software Engineering archive Proceedings of the 22nd international conference on Software engineering*, pages 766–769, 2000.

[49] B. Re. Un modello di qualità per la scelta di servizi web in ambito biologico - il modello di coordinazione. Master's thesis, Laurea in Informatica, Università di Camerino, a.a. 2003-2004. http://dmi.unicam.it/merelli/tesicl26/re.pdf.

[50] R. D. Robert D. Stevens, A. J. Robinson, and C. A. Goble. mygrid: personalised bioinformatics on the information grid bioinformatics. *Bioinformatics*, (19):302 – 304, July.

[51] G.-C. Roman, G. P. Picco, and A. L.Murphy. Software engineering for mobility: A roadmap. In *The Future of Software Engineering*, pages 241–258. 2000.

[52] S. S. Mueller-Wilken, F. Wienberg, and W. Lamersdorf. On integrating mobile devices into a workflow management scenario. In I. C. Society, editor, *Proc. 11th International Workshop on Database and Expert Systems Applications (DEXA)*, pages 186–192, Hamburg, 2000.

[53] C. H. Stephan Gatzka, Th. Geithner. The kertasarie vm. In *NET.Object DAYS 2003*, pages 285–299, Erfurt, September 22-25 2003.

[54] R. Steven, C. Goble, P. Kaker, and A. Brass. A classification of tasks in bioinformatics. *Bioinformatics*, 17(2), 2001.

[55] Sun Microsystems. The CVM. http://java.sun.com/CDC.

[56] Sun Microsystems. The KVM. http://java.sun.com/clcd.

[57] A. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Printice Hall, 2002.

[58] TIDA. The infrared data association. http://www.irda.org.

[59] UML Revision Taskforce. *OMG UML Specification v. 1.4*. Object Magemement Group, 2001.

[60] L. Vito. Hermesv2 e web services. Master's thesis, Laurea in Informatica, Università di Camerino, Italy, a.a. 2003-2004. http://dmi.unicam.it/merelli/tesicl26/vito.pdf.

[61] WLAN. The working group for wlan standards. http://grouper.ieee.org/groups/802/11/.

[62] The BioAgent project. http://www.bioagent.net/.