Model driven design and implementation of activity-based applications in Hermes

Ezio Bartocci*, Flavio Corradini*, Emanuela Merelli*, Leonardo Vito*
* Dipartimento di Matematica e Informatica,
Università di Camerino,
Via Madonna delle Carceri, 62032 Camerino, Italy
{*name.surname*}@unicam.it

***Abstract*— Hermes is an agent-based middleware structured as a component-based and 3-layered software architecture. Hermes provides an integrated, flexible programming environment for design and execution of activity-based applications in distributed environments. By using workflow technology, it supports even a non expert user programmer in the model driven design and implementation of a domain specific application. In this paper, after a description of Hermes software architecture, we provide a simple demo in biological domain and we show some real case studies in which Hermes has been validated.**

I. Introduction

Hermes [9] is an agent-based middleware, 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. Agents can also developed directly by an expert user using directly the Application Programming Interface (API) provided by Hermes middleware. 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 [6] and the other in bioinformatics [13]. 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. This paper is organized as follows. Section II describes the Hermes Software Architecture. Section III provides a simple demo in biological domain. In Section IV, we present several projects in which Hermes middleware has been adopted. We conclude in Section V.

II. Hermes Software Architecture

Hermes is structured as a component-based, agent-oriented system with a 3-layer software architecture shown in Figure 1: user layer, system layer and run-time layer. At the user layer, it allows designers to specify their application as a workflow of activities using the graphical notation. 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 components to support agent mobility and communication. The main difference between the run-time layer and the system layer is how agents function in each. ServiceAgents in the run-time layer are localized to one platform to interface with the local execution environment. UserAgents 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 that agent dies.

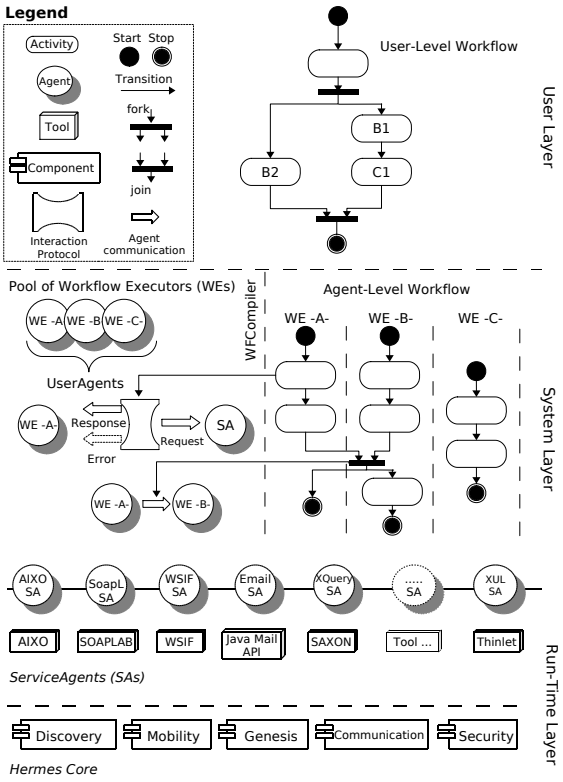

Fig. 1. Hermes Software Architecture

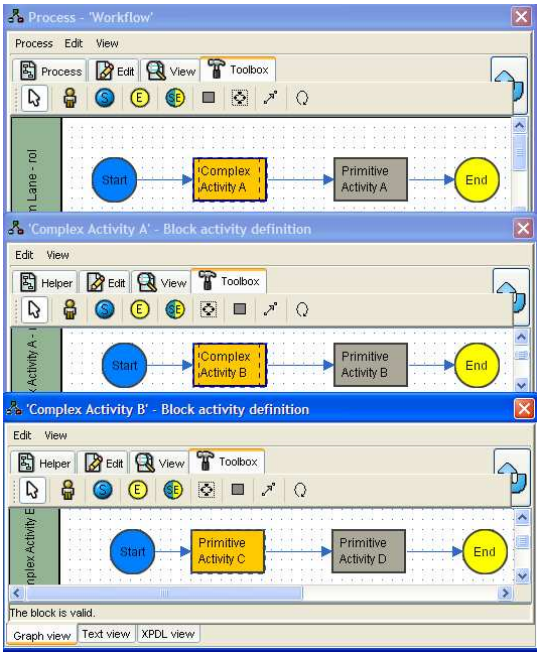

Fig. 2. Specification of Complex/Primitive activities in JaWE

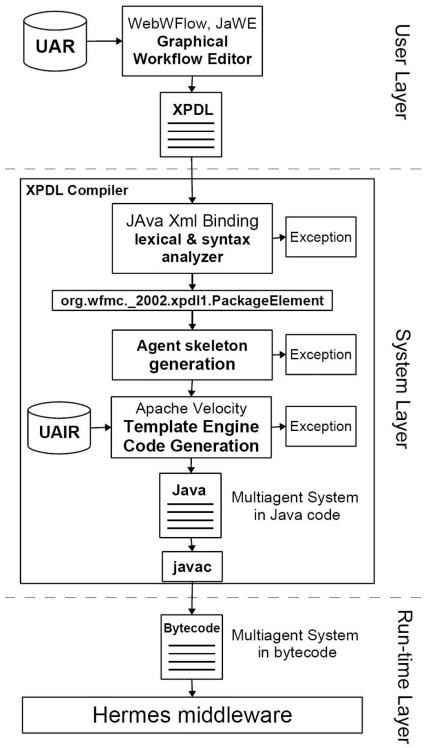

Fig. 3. Outline of workflow compilation process in Hermes

Furthermore, for security UserAgents can access a local resource only by interacting with ServiceAgent that is the "guard" of the resource. It follows a detailed description of the main components and functionalities of each layer.

A. User Layer

The user layer is based on workflow technology and provides to users a set of programs for interacting with the worklow 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. Hermes provides two editors, one is a plugin of the stand-alone JaWE [10] editor and the other is WebWFlow, a web-based editor. Both editors enable the specification of workflows by using XML Process Definition Language (XPDL) [14] a standard provided by the WfMC [12]. 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. Workflow editors enable the composition of both primitive and complex activities. A primitive activity is an activity that can be directly executed. Users can specify primitive activity without knowing the real implementation. A complex activity is an activity that must be specified before it can be used; as Figure 2 shows the specification of a complex activity could be a workflow of complex and/or 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.

B. System Layer

System Layer, on the middle architecture, provides the needed environment to map a user-level workflow into a set of primitive activities. The execution of these latter is coordinated by suitable model, they implement the activities at user level and embed implementation details abstracted from the execution environment. These primitive activities are implemented by autonomous software entities UserAgent able to react to the environment changes where they are executed. A compiler generates a pool of user mobile agents from the workflow specification. Due to the lack of space, workflow compilation process shown in Figure 3 will not discussed here and we refer to [4] for further details.

C. Run-time Layer

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 already described, the overall structure of the system is very complex, it supports abstract specifications that are mapped into a complex distributed and coordinated

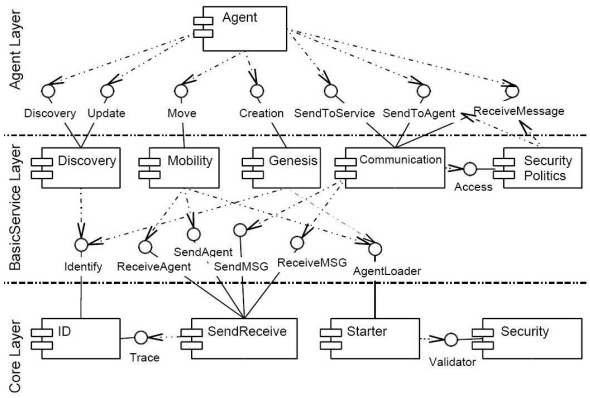

Fig. 4. 3-Layered Architecture of Hermes Mobile Computing Platform.

flows of activities over a large-scale distributed system. In order to master this complexity and to support the reusability of existing artefact during the development of a middleware system for a specific application domain, we designed Hermes kernel following a component-based [7] approach. Figure 4 shows the main components placed in the 3-Layered Architecture of Hermes Mobile Computing Platform. It follows a detailed description of components belonged to each layer.

1) Core Layer: It is the lowest layer of the architecture 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*. The *ID* component, implements general identity management functions by managing a repository containing information about locally generated agents. This repository is accessed whenever we want to know the current position of an agent. The *ID* component is also responsible for the creation of the identifiers to be associated to new agents. These identifiers contain information about birthplace, date and time of the agent's creation. Agent localization is simplified by information contained directly in the *ID*, such as birth place. In fact, the birth place of an agent hosts information about agent's current location. A second important feature of the Core is the *SendReceive* component. This component implements low level inter-platform communication by sending and receiving messages and agents. By using traceability services offered by the *ID* component, *SendReceive* can easily update or retrieve the exact position of a specific user agent. The *Starter* component processes any request for agent creation. This particular component, in fact, take 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 *BasicService* layer. The *Security* component, as mentioned above, checks for the presence of malicious code or manipulations within agent code.

2) BasicService Layer: This layer has five main components: *Discovery*, *Mobility*, *Genesis*, *Communication* and *Security Politics*. The *Discovery* component searches and detects service agents. When a user agents wants to communicate with a service, it will ask the *Discovery* for the right identifier to use as the messages's receiver. The service detection strategy can be implemented in several ways; for example by a fixed taxonomy or by an UDDI [5], commonly used in WebServices application domain. The mobility component enables the movement of code across platforms [11], it implements the interface used by the *Agent* component 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 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.

3) Agent Layer: The *Agent* Layer is the upper layer of the mobile platform, the *Agent* Layer, contains all service and user agents. This component has not any interface, but it has only several dependencies upon the *BasicService* Layer. The *Agent* component provides a general abstract *Agent* class. *UserAgent* and *UserAgent* classes extend this abstract class. ServiceAgent consists of agents enabling access to local resources such data and tools. User agents execute complex tasks and implement part of the logic of the application. Java programmers can also develop

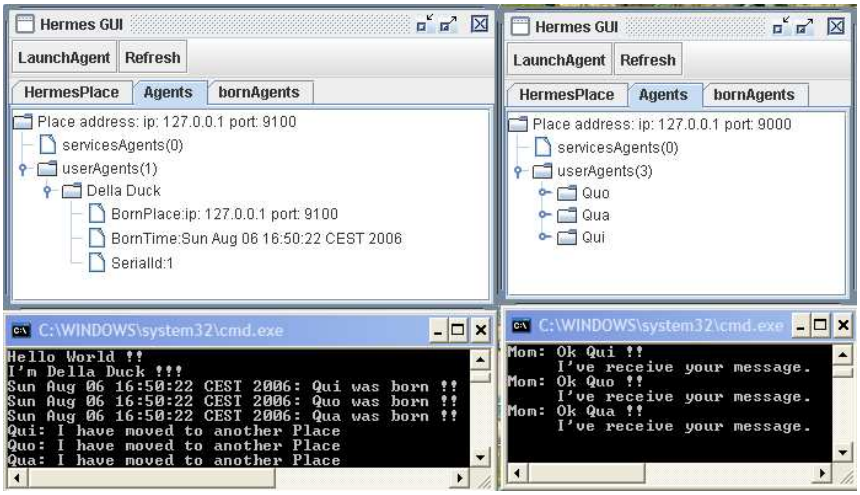

Fig. 5. Final result produced by McDuckAgent.java

UserAgents by using the API provided by Hermes Mobile Computing Library. Listing 1 shows a simple demo. A MkDuckAgent called *"Della Duck"* creates three sons *Qui*, *Quo* and *Qua* -lines 24 to 40- by cloning itself. After clonation each new agent start its behaviour calling *"afterCloning"* as initial method.

```
1  package samples;
2  import hermesV2.*;
3  import hermesV2.agent.*;
4
5  public class McDuckAgent extends UserAgent {
6
7    public McDuckAgent(String agentName) {
8      super("Della Duck");
9    }
10
11   public void init() {
12     reception(); //I enable the reception
13                  //of messages for the father
14
15     System.out.println("Hello World !!");
16     System.out.println("I'm Della Duck !!!");
17
18     Identificator temp=null,son1=null,
19                   son2=null,son3=null;
20
21     /*afterCloning is the first method
22       called after the clonation*/
23
24     try {
25       son1 = clone("afterCloning", "Qui");
26       System.out.println(new Date(
27                System.currentTimeMillis()) +
28                ": Qui was born !!");
29       son2 = clone("afterCloning", "Quo");
30       System.out.println(new Date(
31                System.currentTimeMillis()) +
32                ": Quo was born !!");
33       son3 = clone("afterCloning", "Qua");
34       System.out.println(new Date(
35                System.currentTimeMillis()) +
36                ": Qua was born !!");
37     } catch (CloneException ce) {
38       System.out.println(ce);
39     }
40     Message m0=null,m1=null,m2=null,m3=null;
41     while (!(m1!=null && m2!=null &&
42            m3!=null)){
43       m0 = getMessageSynch();
44         temp = m0.getSenderAgentId();
45         if (son1.equals(temp)) m1 = m0;
46         if (son2.equals(temp)) m2 = m0;
47         if (son3.equals(temp)) m3 = m0;
48         System.out.println((String)m0.getObject());
49     }
50     /*The mother replies to sons*/
51     Identificator myId = getIdentificator();
52     m1 = new Message(myId, son1,
53                  "Mom: Ok Qui !! \n      "+
54                  "I've receive your message.");
55     m2 = new Message(myId, son2,
56                  "Mom: Ok Quo !! \n      "+
57                  "I've receive your message.");
58     m3 = new Message(myId, son3,
59                  "Mom: Ok Qua !! \n      "+
60                  "I've receive your message.");
61     try {
62       sendMessageToUserAgent(m1);
63       sendMessageToUserAgent(m2);
64       sendMessageToUserAgent(m3);
65     } catch (CommunicationException ce) {
66       System.out.println(ce.getMessage());
67     }
68   }
69
70   public void afterCloning() {
71     Identificator myId = getIdentificator();
72     PlaceAddress  myBPA = myId.getBornPlaceAddress();
73     int           myBPAPort = myBPA.getPort();
74     try {
75       int port = (myBPAPort == 9100) ? 9000 : 9100;
76
77       PlaceAddress  myMPA =
78               new PlaceAddress(myBPA.getIp(), port);
79       this.move(myMPA, "afterMoving");
80     } catch (MigrationException me) {
81       System.out.println("MigrationException" + me);
82     }
83   }
84
85   public void afterMoving() {
86     reception(); //I enable the reception
87                  //of messages for the son
88
89     Identificator myId = getIdentificator();
90     Identificator mother = getFatherIdentificator();
91     Message m = null;
92
93     try {
94         m = new Message(myId, mother,
95                 getAgentName() +
96                 ": I have moved to another Place");
97         sendMessageToUserAgent(m);
98     } catch (CommunicationException ce) {
99       System.out.println(ce.getMessage());
100    }
101    m = getMessageSynch(mother);
102    System.out.println((String)m.getObject());
103  }
104 }
```

Listing 1. McDuckAgent.java

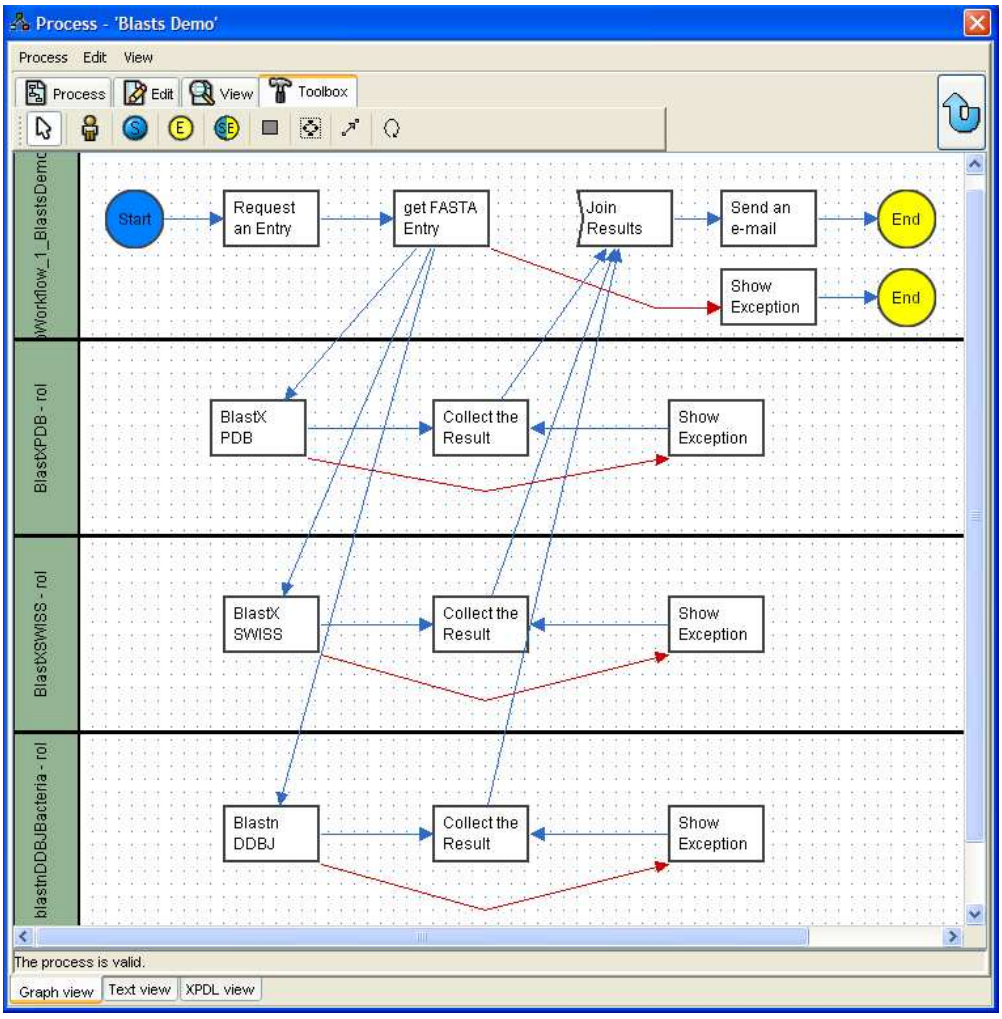

Fig. 6. Multiple blast workflow

By using *"move"* method -line 79- Qui, Quo amd Qua migrate to a *Place* different from where they were born. When they arrive in the new *Place* each one call the *"afterMoving"* -line 85- method. Then they notify to their mom their moving by using *"sendMessageToUserAgent"* -line 97- and *"getMessageSynch"* -line 101- methods. Figure 5 shows the final results.

D. Software requirements

One of the main features of Hermes middleware is its scalability. The present version, HermesV2, is a pure Java application whose kernel requires about 120KB of memory and interoperates across a systems ranging from microprocessors to very power workstations. The Hermes Mobile Computing Platform is available under LGPL on Sourgeforge [1] Web Site.

III. Model driven design and implementation of activity-based applications: a demo

In the present post-genomic era, biological information sources are crammed with information gathered from results of experiments performed in laboratories around the world, i.e., sequence alignments, hybridization data analysis or proteins interrelations. The amount of available information is constantly increasing, its wide distribution and the heterogeneity of the sources make difficult for bioscientists to manually collect and integrate information. In this section we present a demo of Hermes in the biological domain. In our example we want to find similar DNA sequences to a given one in several databases using Basic Local Alignment Search Tool [2] (BLAST). In particular, in this demo we want to compare, using BLAST, the nucleotide sequence in FASTA format of a given entry identificator with the sequences contained in the following databases:

- Protein Data Bank (PDB) [3]
- SWISS-PROT [4]
- DDBJ [5]

[1] http://sourceforge.net/projects/hermes-project

[2] http://www.ncbi.nlm.nih.gov/BLAST/
[3] http://www.rcsb.org/pdb/
[4] http://www.ebi.ac.uk/swissprot/
[5] http://www.ddbj.nig.ac.jp/

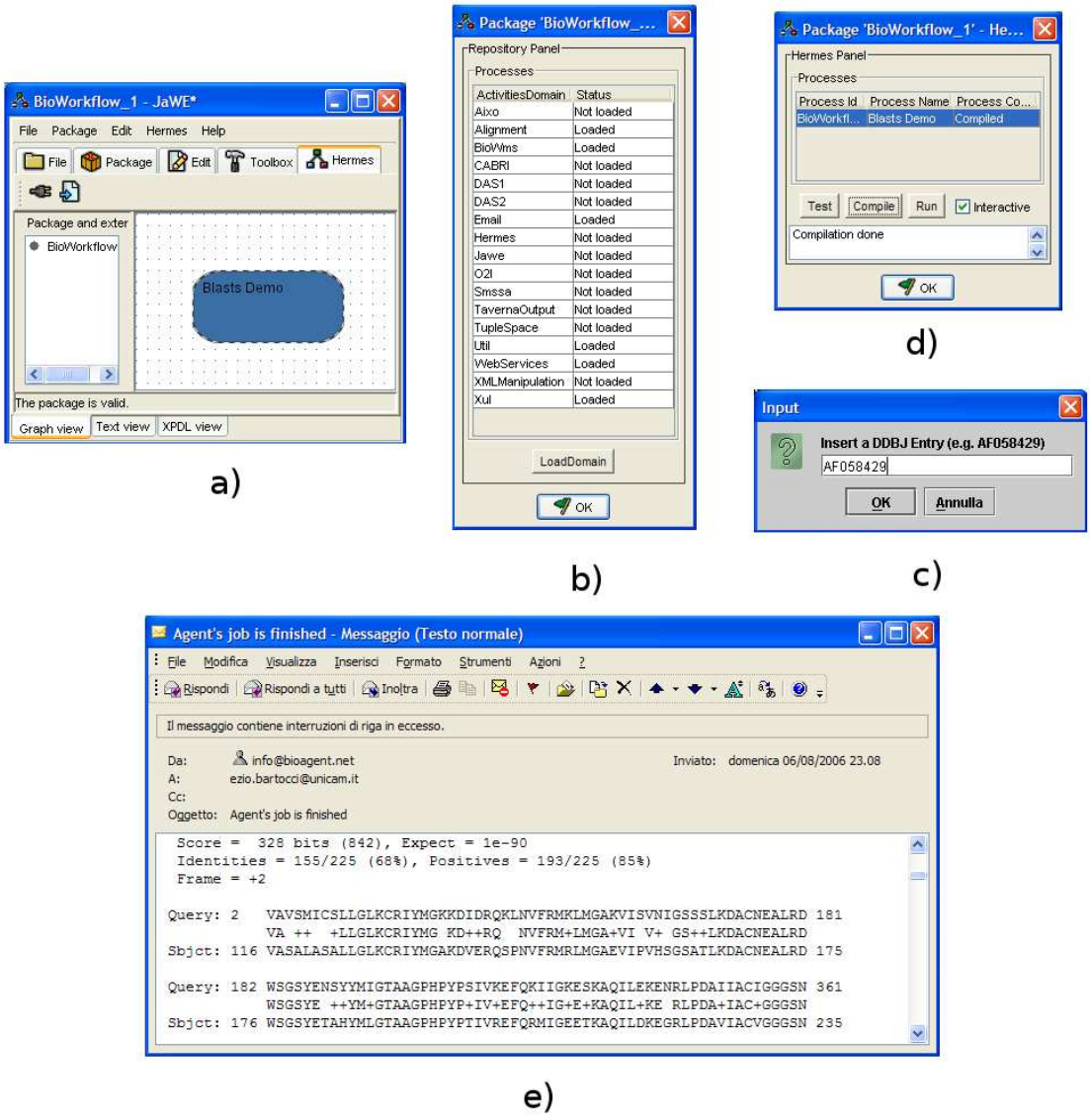

Fig. 7. Multiple blast compilation and execution

The access to these databases is guaranteed by a set of Web Services. By using workflow editors a bioscientist can specify the logic order, as Figure 6 shows, of a set of domain-specific activities without knowing the related implementation details. Each rectangle is an activity and each swimlane represents a UserAgent. As Figure 7-b shows, user can exploit a set of previous defined domain-specific activities by importing the proper library.

BlastnDDBJ Agent, *BlastXSWISS* Agent and *BlastX-PDB* Agent receive the nucleotide sequence from the *BlastDemo* Agent and throught an interation with WSIF ServiceAgent, they compare the received sequence with sequences in each database using BLAST. If no exceptions occur, *BlastDemo* Agent join partial results and send the final document to user by email throught an interaction with the Email ServiceAgent. After saving this specification, you can reload -Figure 7-a-, compile -Figure 7-d- and execute -Figure 7-c and 7-e - the workflow previous defined.

IV. Some Case studies

The Hermes middleware has been validated in several projects. It follows a brief case study description of Hermes application in some of them.

A. SI.CO.M project

In the SI.CO.M [6] project we have developed a prototype based on Hermes middleware for the traceability of ichthyic products. Generally the product is traced throught the updating of databases distributed along the main sites of the weaving factory. This approach is not efficient because trace a faulty batch of products requires to query all databases, usually with an heterogeneous schema, of all sites interested in the production process. The proposed

[6]http://sicom.cs.unicam.it/

solution with the prototype named "TraceFish", exploiting the agent-based technology, allows to move automatically the information about a single batch from a site to another of the weaving factory overcoming the limits of the classical client/server approach.

B. O2I Project

The Oncology over Internet (O2I) [7] project is aimed to develop a framework to support searching, retrieving and filtering information from Internet for oncology research and clinics. Hermes in the context of O2I project is called Bioagent [8], it supports the the design and execution of user workflows involving access to literature, mutation and cell lines databases.

C. LITBIO Project

The main objective of the Laboratory of Interdisciplinary Technologies in Bioinformatics (LITBIO) [9] is to create infrastructure capable of supporting challenging international research and to develop new bioinformatics analysis strategies apply to biomedical and biotechnological data. To satisfy the most bioinformaticians needs we have proposed a multilayer architecture [2] based on Hermes middleware. At the user layer, it is intended to support in-silico experiments, resource discovery and biological systems simulation. The pivot of the architecture is a component called Resourceome [8], which keeps an alive index of resources in the bioinformatics domain using a specific ontology of resource information. A Workflow Management System, called BioWMS [3], provides a web-based interface to define in-silico experiments as workflows of complex and primitives activities. High level concepts concerning activities and data could be indexed in the Resourceome, that also dynamically supports workflow enactment, providing the related resources available at runtime. ORION [1], a multiagent system, is a proposed framework for modelling and engineering complex systems. The agent-oriented approach allows to describe the behavior of the individual components and the rules governing their interactions. The agents also provide, as middleware, the necessary flexibility to support data and distributed applications. A GRID infrastructure allows a transparent access to the high performance computing resources required, for example in the biological systems simulation.

ACKNOWLEDGMENT

This work is supported by the Investment Funds for Basic Research (MIUR-FIRB) project Laboratory of Interdisciplinary Technologies in Bioinformatics (LITBIO).

V. CONCLUSION

As the demo presented shows, Hermes middleware provides an integrated, flexible programming environment, whose user can easily configure for its application domain. 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 simple way. 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. By using workflow as suitable technology to hide distribution and on mobile agents as flexible implementation strategy of workflow in a distributed environment, Hermes allows even to a not expert programmer a model driven design and implementation of a domain specific activity-based application.

[7] http://www.o2i.it/

[8] http://www.bioagent.net/

[9] http://www.litbio.org/

REFERENCES

[1] M. Angeletti, A. Baldoncini, N. Cannata, F. Corradini, R. Culmone, C. Forcato, M. Mattioni, E. Merelli, and R. Piergallini. Orion: A spatial multi agent system framework for computational cellular dynamics of metabolic pathways. In *Proceedings of Bioinformatics ITalian Society (BITS) Meeting*, Bolgna, Italy, 2006.

[2] E. Bartocci, D. Cacciagrano, N. Cannata, F. Corradini, E. Merelli, and L. Milanesi. A GRID-based multilayer architecture for bioinformatics. In *Proceedings of NETTAB'06 Network Tools and Applications in Biology*, Santa Margherita di Pula, Cagliari, Italy, 2006.

[3] E. Bartocci, F. Corradini, and E. Merelli. BioWMS: A web based workflow management system for bioinformatics. In *Proceedings of Bioinformatics ITalian Society (BITS) Meeting*, Bologna, Italy, 2006.

[4] E. Bartocci, F. Corradini, and E. Merelli. Building a multiagent system from a user workflow specification. In *Proceedings of Workshop From Objects to Agents - WOA*, 2006.

[5] 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.

[6] 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.

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

[8] N. Cannata, E. Merelli, and R. B. Altman. Time to organize the bioinformatics resourceome. *PLoS Comput Biol.*, 1(7):e76, 2005.

[9] F. Corradini and E. Merelli. Hermes: agent-based middleware for mobile computing. In *Mobile Computing*, volume 3465, pages 234–270. LNCS, 2005.

[10] Enhydra. Jawe. http://jawe.enhydra.org/, 2003.

[11] A. Fuggetta, G. Picco, and G. Vigna. Understanding code mobility. *IEEE Transaction of Software Engineering*, 24(5):352–361, May 1998.

[12] D. Hollingsworth. The Workflow Reference Model, January 1995.

[13] E. Merelli, R. Culmone, and L. Mariani. Bioagent: a mobile agent system for bioscientists. In *NETTAB Workshop on Agents Nd Bioinformatics*, Bologna, July 2002.

[14] WfMC. Xml process definition language (xpdl). WfMC standard, W3C, October 2005.