A Programming Environment for Global Activity-based Applications

Flavio Corradini
Dipartimento di Informatica,
Università di L'Aquila
67010 L'Aquila, Italy
Email: flavio@di.univaq.it

Leonardo Mariani
DISCo
Università di Milano Bicocca
20126 Milano, Italy
Email: mariani@disco.unimib.it

Emanuela Merelli
Dipartimento di Matematica e Informatica,
Università di Camerino
62032 Camerino, Italy
Email: emanuela.merelli@unicam.it

***Abstract*— This paper focuses on large-scale distributed systems that can be modeled as workflows of activities sharing resources, knowledge, know-how and services. We propose a programming environment for such global activity-based applications the execution of which relies on the agent-technology. Methods and emerging technologies will be discussed from the user level applications to the run-time support. The programming environment we propose is the result of our experience in designing and implementing applications within specific application domains such as controlling industrial platforms and bioinformatics.**

I. Introduction

Nowadays computer systems are more and more complex. They can be geographically distributed, present features of dynamic topology, real-time and heterogeneity, fault-tolerance, mobility and service discovery. To manage complexity in such systems new methodologies, methods and tools for their analysis, design and implementation have been provided. Among others we mention component-based design and programming [1], ubiquitous computing [2], agent-based systems [3] and mobile code [4].

This paper focuses on large-scale distributed systems that can be modeled as workflows of activities that share resources, knowledge, know-how and services. Though many such features can be abstracted away at the workflow application level, the infrastructure that supports this kind of applications must anyway support (very often) distributed interactions, service discovery, fault tolerance and security. Distributed interactions are necessary to make possible communication between partners located in different places. Fault tolerance is needed to let complex, distributed systems working even in faulty situations such as site failures or disconnections. Service discovery is needed to take efficiently advantage of resources or system capabilities. Finally, security allows to prevent malicious users and servers to perform dangerous actions such as mitigate the diffusion of false information.

Most of these attributes are often strictly needed in certain kinds of systems and make a system very difficult to design. A deep understanding of the application domain can certainly help in reducing problems. Simpler techniques, technologies and implementation strengths can be suggested by a proper analysis of the domain itself. The application domain, moreover, suggests the set of primitive activities that can be composed to build workflows. Besides the above attributes, the applications we refer to are, hence, domain-specific applications.

We now describe a general programming environment for global activity-based applications as described above. This is the result of our experience in designing and implementing applications for controlling industrial platforms [5] and for searching data within in the bioinformatics domain [6]). We have abstracted a general architecture for the design of our applications based on three conceptual layers:

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

A *System Layer*, on the middle of the architecture, which provides the needed environment to map a user-level workflow into a set of more primitive (and already implemented) activities. These latter cooperate to implement the activities at the user level and embed implementation details abstracted at the user level (fault tolerance, for instance). These primitive activities follow the agent-based technology since, as argued several times in the literature (see, for instance [7], and references therein), the agent paradigm seems 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 level activities into system level activities. The compiler must be aware of the available a library of implemented activity 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, that provides the needed support for the run-time execution

This work was supported by the Center of Excellence for Research 'DEWS: Architectures and Design Methodologies for Embedded Controllers, Wireless Interconnect and System-on-chip', by the MURST strategic project 'Oncology Over Internet' and by the MURST project 'Sahara: Software Architectures for Heterogeneous Access Networks infrastructures'.

of system level agents; namely, primitives for coordination, communication, security, mobility and other implementation details.

Summing up, the whole architecture for programming our applications is that shown in Figure 1, where the three intro-

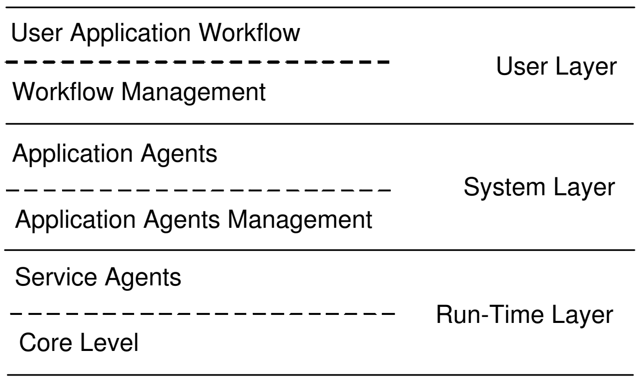

Fig. 1. A Software Architecture for Global Activity-based Applications

duced layers, User Layer, System Layer and Run-time Layer, are themselves split in two conceptual levels (the application and its infrastructure). On the top of the architecture, the User Application Workflows layer, reside our applications; namely our user workflow applications. The underlying Workflow Management Layer provides the needed environment to support workflow application specifications. It includes, for instance, CASE tools and graphical environments for editing workflows. The User Application Workflows level and Workflow Management level constitute the User Layer.

The pool of agents that implements user applications reside in the Application Agents layer, below the Workflow Management layer. The behavior of such objects is still described in a (lower level) workflow notation. The Application Agents Management contains a (context-aware) compiler and libraries of suitable agents that implement primitive user activities. Applications Agents level and Applications Agents Management constitute the System Layer.

At the bottom of the architecture we find the Service Agent layer where agents located in sites provide access to services available in those sites. Then the Core provides the infrastructure necessary to execute and support agent execution and other implementation details. The Service Agents and the Core constitute the Run-time Layer.

The rest of the paper is organized as follows. Sections II, III, IV describe in more details the User Layer, System Layer and Run-time Layer, respectively. To fix intuition we make use of a running example that, actually, identifies a specific class of applications where our programming environment can be exploited (different from those presented in Section V). Section VI contrasts our approach to existing ones, and Section V summarizes our experience on specific applications and discuss concluding remarks and further work.

II. User Layer

The applications we address in this paper are based on workflows, called User Level Workflow (ULW), describing the control flow of the activities that must be performed to reach a specific goal. A general user of this system is not a computer expert, so he/she needs an environment with characteristics of user-friendliness, usability and simplicity. A graphical notation can provide a suitable environment for the composition and configuration of activities that are contained in a domain-specific library.

There are several formalisms (notations) supporting the visual composition of activities. These formalisms, often based on UML and Petri Nets (see, for instance, [8], [9], [10] and references therein), allow for the sequential composition of activities, their distribution, coordination, cooperation . . . Given the wide number of different proposals (often offering not yet stable or partial solutions), in this introductory paper we do not faithfully follow one of these formalisms but, instead, we try to explain our ideas by resorting to a simple and intuitive notation. We believe that this way makes the paper readable and appealing also to non-experts in specific research areas. However, those readers that are familiar with UML can find in our workflow descriptions some strong similarities with UML Activity Diagrams. We intend to use the emerging notation AUML [11] (always based on UML) for the aspects of agent coordination and agent communication protocol in future, more detailed, descriptions of this paper.

Together with activities some state information is often needed that in our notation take the form of notes attached with activities; one on the right denoting the parameters in input to the activity and one on the left report the state variables affected by the performed activity. The whole set of state variables are defined in a stand-alone note at the beginning of the diagram.

As one would expect, the activities that a user can compose can be either primitive activities or composite activities. These latter can be already described in a workflow notation and typically denote frequently used workflows or user personalized workflows. Moreover, the available activities are, of course, customized for a specific application domain. This contributes to let our applications domain-specific.

The definition of a ULW proceeds by choosing basic activities on existing libraries and connecting such actions according to operators of the language. These operators should have a clear and intuitive meaning even for non-experts (this is the reasons why we used UML-like activity diagrams). The user, then, sets opportunely the input and output notes to configure the application, e.g. the name of the book that we would like to buy or username and password enabling the access to a service.

Figure 2 shows an example of a user workflow describing the activities needed to buy a DVD recorder. Predefined state variables are described in the stand-alone note on the top-left corner. The flow of the activities begins with the concurrently searching for best price of the device and checking the bank account. These two activities are then synchronized to verify if a sufficient amount of money in the account is available. In the positive case, the purchase is possible. Otherwise, a notification email is sent. It is worth of noting that most details such as security, implementation details, coordination policy, etc. are abstracted away at this abstraction level creating a very simplified environment for the user.

III. System Layer

A User Level Workflow (ULW) is compiled into a System Level Workflow (SLW), that now specifies the behavior of application agents; namely, the activities that a pool of user agents (also called MOWE, for Mobile and Opportunistic Workflow Executors) must perform for user level activities by migrating from a site to another to retrieve information, communicate each other, coordinate activities, etc. The compiler, then, produces two outputs: a SLW and a pool of MOWEs that implements the SLW. Each MOWE (actually the "skeleton" of each MOWE) resides in a domain-specific library that the compiler uses to implement the primitive activities used to specify the ULW.

SLW, as ULW, are specified with activity diagrams notation. In the former case, however, swimlanes are used to emphasize the fact that each user level activity can be implemented by a pool of coordinated agents; where, in particular, each swimlane denotes the actions executed by each agent. A possible SLW, output of the compilation of the ULW in Figure 2 is shown on Figure 3.

The scope of a state variable is local to a single agent and hence limited to a swimlane. When the input note is positioned in a different swimlane from the one containing the activity state, this means that a message from the note's owner agent must be received. We also use two types of synchronization bars: inter-agent synchronization when multiple agents must synchronize, and intra-agent synchronization when a single agent must coordinate its activities.

The mapping is performed by substituting each user level activity to a workflow of system level activity with a case-by-case approach.

It is quite difficult to produce an efficient pool of agents without having specific information on the environment; in fact the agent implementation would have to face a wide range of possible situations. Our approach is to exploit a *context-aware compiler*. A compiler is context aware if it knows the environment and takes advantage of this knowledge to produce goal-directed agents. In our case the compiler knows information such as hosts making part of the system, the type of existing services, the type of information stored on each place, the topology of the system and other useful parameters depending on the application domain. During the compilation some of these parameters may change, for this reason the compiler creates opportunistic agents. An opportunistic agent may face an unpredicted situation and still solve its task. Opportunism is achieved by embedding a set of rules into the MOWE that allows the agent to deal with situations that are not explicitly specified in the user level workflow but that can happen such as site or connectivity failure, login failures, service accesses failures, etc.

Then, by using a context-aware compiler the user can completely ignore where to search information, in which form the information is stored, how he/she can interacts with each service, and the number of interesting hosts. All these information are managed by the context-aware compiler that maps an application workflow (a ULW) to an (context-aware) agent-based workflow specification.

The execution of MOWE is supported by a run-time support of agents that implement security, fault tolerance, mobility, distributed communication and (easy) access to services as discussed in the next section.

IV. 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, a (two) layered architecture is considered, where at the top there is a:

- *Service agent layer*, that provides a set of agents offering services in the place where they are located (service agents do not migrate among sites), while at the bottom there is a,
- *Core* that provides all the infrastructure necessary to execute and support both user and service agents, in particular security, fault-tolerance, communication, mobility and resource management.

More in detail, the service agents provide access to services. When a user agent migrates and arrives in a place it can query the yellow page service to gain information about services offered in the place and then it communicates with the service agents to gain the information it needs. This paradigm simplifies the interactions enabling the use of an agent communication language, e.g. KQML [12] or Fipa ACL [13], as a unified way to communicate with other agents, services or resources.

Regarding the core layer it has to be noticed that most of the attributes above mentioned are already present in JADE [14], a FIPA Compliant agent platform. Then, in principle, we can take JADE as a core layer for our applications, though, in the past we had to develop the core from scratch in order to take advantage of particular design decisions suggested by the specific domain.

V. Applications of the General Architecture

We propose a programming environment for global activity-based applications; namely, large-scale distributed systems that can be modeled as workflows of activities sharing resources, knowledge, know-how and services. The typical scenario is the following. A user specifies his/her application in a workflow of activities taken from available libraries of activities tied to a specific application domain. The user then invokes a compiler that translates the user application into a pool of agents committed to the execution of the user activities. The

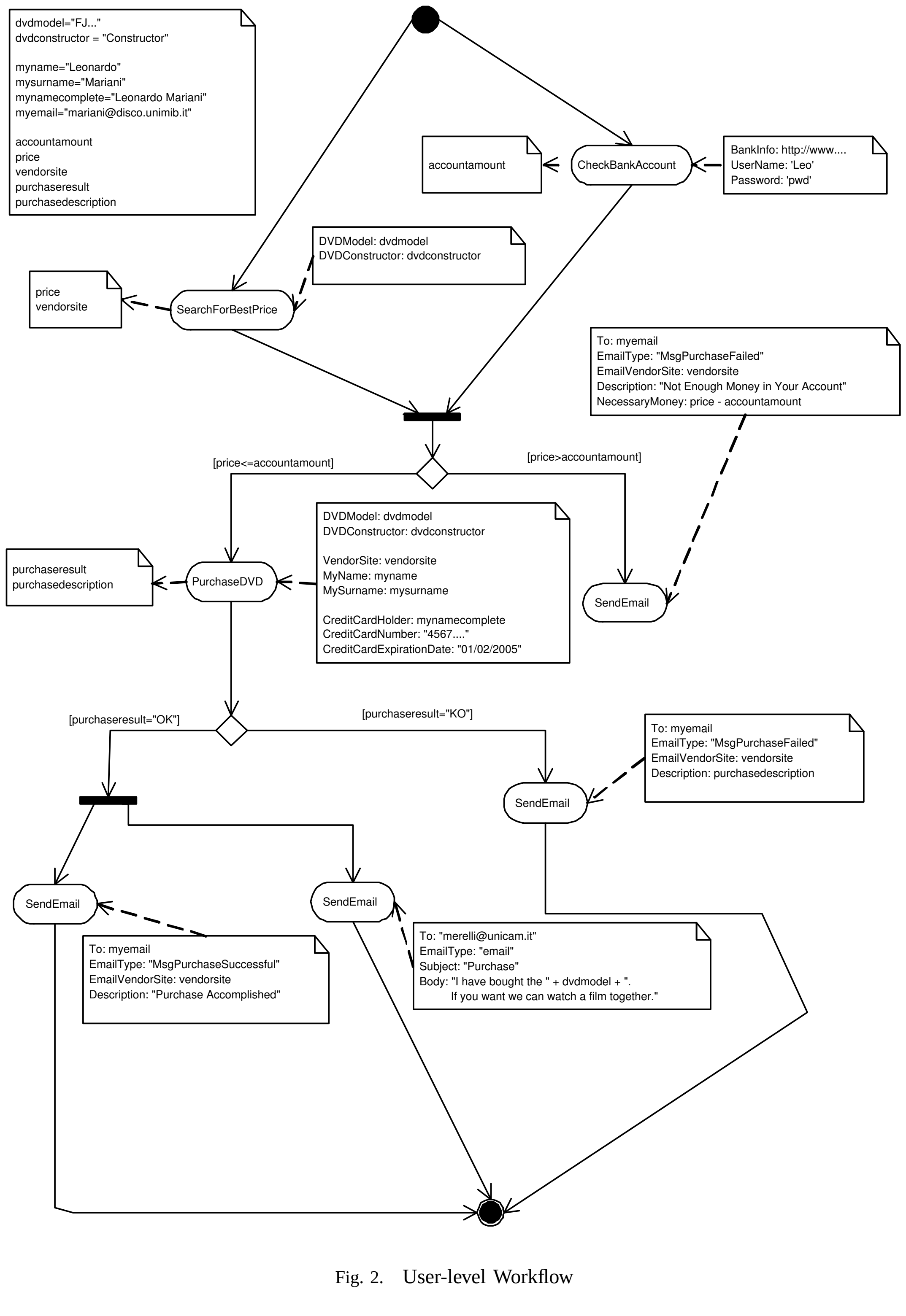
dvdmodel="FJ..."
dvdconstructor = "Constructor"
myname="Leonardo"
mysurname="Mariani"
mynamecomplete="Leonardo Mariani"
myemail="mariani@disco.unimib.it"
accountamount
price
vendorsite
purchaseresult
purchasedescription
accountamount
CheckBankAccount
BankInfo: http://www....
UserName: 'Leo'
Password: 'pwd'
DVDModel: dvdmodel
DVDConstructor: dvdconstructor
price
vendorsite
SearchForBestPrice
To: myemail
EmailType: "MsgPurchaseFailed"
EmailVendorSite: vendorsite
Description: "Not Enough Money in Your Account"
NecessaryMoney: price - accountamount
[price<=accountamount]
[price>accountamount]
DVDModel: dvdmodel
DVDConstructor: dvdconstructor
VendorSite: vendorsite
MyName: myname
MySurname: mysurname
CreditCardHolder: mynamecomplete
CreditCardNumber: "4567...."
CreditCardExpirationDate: "01/02/2005"
purchaseresult
purchasedescription
PurchaseDVD
SendEmail
[purchaseresult="OK"]
[purchaseresult="KO"]
To: myemail
EmailType: "MsgPurchaseFailed"
EmailVendorSite: vendorsite
Description: purchasedescription
SendEmail
SendEmail
SendEmail
To: myemail
EmailType: "MsgPurchaseSuccessful"
EmailVendorSite: vendorsite
Description: "Purchase Accomplished"
To: "merelli@unicam.it"
EmailType: "email"
Subject: "Purchase"
Body: "I have bought the " + dvdmodel + ".
If you want we can watch a film together."

Fig. 2. User-level Workflow

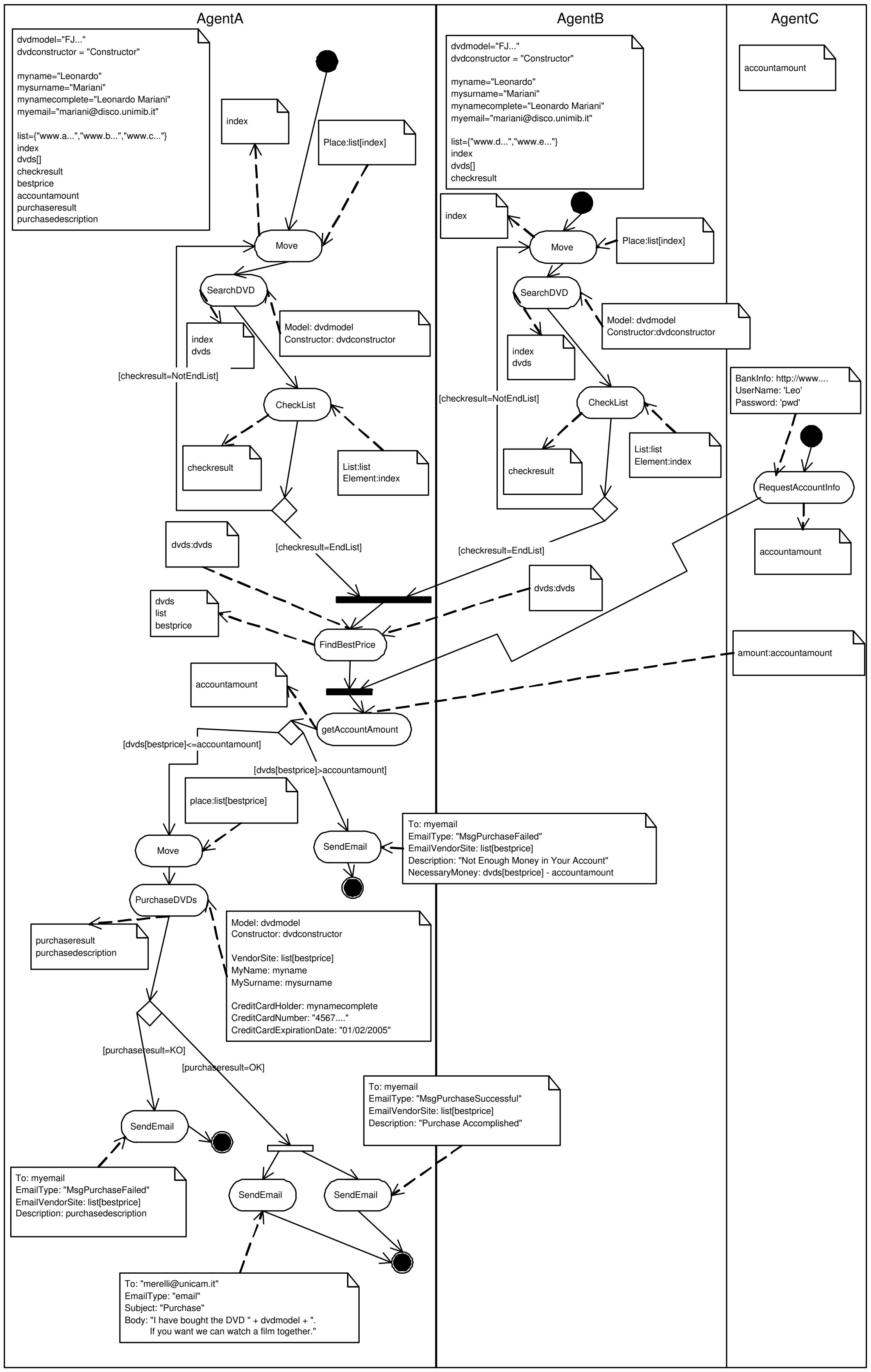

Fig. 3. Agent-level Workflow

compiler transports its knowledge of the environment into the agents in such a way that these latter may assume opportunistic behaviors in cases of exceptions during execution. The agent technology is well suited for developing complex, distributed systems [7], and our architecture can be adapted to exploit other technologies. Indeed, we aim at experimenting also components in place of agents.

The execution environment of the pool of application agents is provided by a run-time support that implements primitives for agent coordination and communication, agent mobility and security, fault-tolerance, services and resources.

In the rest of this section we shortly describe two instances of the presented environment: one in the industrial domain, the other in bioinformatics. We shortly describe the different domains and the roles of the involved agents.

A. Product Quality through Information Integration in Industrial Platforms

The production process is usually performed by a set of activities associated to different actors. In the case of a supply chain, the actors are the suppliers and the production plans. The suppliers provide both row and semi-manufactured components, while the production plans assemble the components to produce the final product. Once a defect or malfunction in assembling the final product is identified, all the information regarding the quality tests of any component should be retrieved. Quality control within a supply chain is a complex activity, information integration is difficult and access to all information simultaneously is, in most cases, impossible. In this setting our workflow application support the quality control through the traceability of the different components and semi-manufactured products [5]. User agents support the execution of workflow activities; they retrieve component's test information from the suppliers and integrate data. Service agents allow the access to company internal data repositories. In fact each single supplier may be using quality control mechanisms different from other suppliers, and generating different data as result from specific tests. The resulting distributed system is based on a network of hosts (the suppliers) each one allowing the access to local information through proper agents of our middleware.

Product traceability during the product production process is also an interesting application of our architecture. This is on going research.

This research is carried out thanks to a project funded by Loccioni Group [15] and by a project SICOM "SIstemi COoperativi Multiagente" co-funded by CIPE-Regione Marche and Università di Camerino.

B. Retrieving and Integrating Data in Bioinformatics

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. Users who possess their own algorithm, but have no data to gain evidence of the correlation between cause and effect would have the possibility to move and execute their code to data sources. Existing services offer no easy or efficient solution to this problem. For example, Basic Local Alignment Search Tool (BLAST) [16] offers a set of services (BLASTp, BLASTn ...) to allow sequences comparison, but it doesn't support the execution of correlated activities of an experiment. In the Bioagent project [1] we have provided a framework to define the bioscientist experiment as a workflow of activities[6]. For which, a pool of mobile user agents is created to completely decentralize local tasks processing, and free bioscientists from the need to continuous interact with remote services. As an example of workflow application, suppose to look for a new protein structure; the bioinformatician (biochemestrian) that make the experiment, first selects a m-RNA sequence for a known protein, then by using $BLASTn$ in the genomic database GenBank looks for the first 10 sequences with the highest score. Afterwards, he looks for the crystallographic structure (the PDB files) of the associated proteins. This search is made in two different data collections SwissProt and PubMed. In SwissProt he uses $BLASTp$ to extract the first 5 sequences with the highest score, and for each article he extracts the references to the Protein Data Bank. In PubMed, for any protein (or sequence) he extracts all the articles written on this sequence, by selecting only those for which a crystallographic structure is available (option: Structured Links). Each workflow's activity is described by a set of tasks supported by a pool of user agents; each user agent coordinates the collection and integration of retrieved information by moving to remote data source and by cooperating with local service agents.

This research is carried out thanks to the MURST strategic project 'Oncology Over Internet'.

VI. Related Work

The complexity of the applications we are addressing, though tight to specific application domains, needs an involved engineering and the usage of suitable technologies. Our work aims at giving a general architecture that provides guidelines for designing global activity-based applications and suggests possible technologies. All the applications of this kind we have developed falls within this setting.

Similarities to this work can be found in the suggested technologies. Workflows, that cover a wide range of distributed applications and form a nice specification language (presented as Activity Diagrams, Petri Nets or similar), have a wide literature (see WfMC reference model [17] and references therein). The agent technology, as a means to implement workflows has been suggested, for instance, by Chang and Scott [18] and by Mertz et al. [19]. It has to be said, however, that most papers propose to implement WfMS with multi-agent systems where agents are stationary [20], [21], [22], to

[1]www.bioagent.net

support to Web Services [23], or specialized in manufacturing [24]. Some agent mobility is present in the AWA architecture by Stormer [25] where, agents of different types, form a work team oriented to execute a workflow. None of these works, however, present and discuss the needs of having a context-aware compiler to translate an abstract workflow into a specialized mobile pool of agents trained to cooperate in an opportunist way during their execution.

Another work with similar aims as ours has been recently proposed by Ricci et al. [26]; they use the center space as communication model and TuCSoN coordination laws as workflow definition language. Tuple space enhances uncoupled interactions in space and time. A Programming environment has been proposed also in [27], where it is possible to define applications based on cooperating agents. For the kind of applications we are interested in (and also because an agent is typically aware of the pool of agents with which it has to coordinate activities) we decide to rely on message passing communication. This also allows us to base our middleware for global activity-based applications on JADE, a FIPA Compliant agent platform.

VII. Concluding Remarks and Further Work

The programming environment presented involves two keywords that deserve to be mentioned once again: Workflows and Agent Run-time support. Both of them posses guide lines by well-established coalitions; namely, the Workflow Management Coalition [28], and the already mentioned Foundation for Intelligent Physical Agents also known as FIPA [29], respectively. Any developed tool fitting the proposed guide lines can be adopted in our (implementing) architecture so that, hopefully, our proposal can be though as an effective integration of the ideas by the two coalitions.

As already said, the presented architecture turns out to be adaptable to different application domains. The case studies in Section VII provide some evidence on its generality. Following the programming environment presented in the current paper, we are developing a similar workbench for a class of applications more and more frequent; those related to microcontrollers and embedded systems in general.

Acknowledgments

We would like to thank Rosario Culmone for useful discussions on the subject of this work. Diego Bonura and the anonymous referees of WOA'03 are thanked for their valuable comments and suggestions that have certainly improved a previous version of this paper.

References

[1] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

[2] S. Consolvo, L. Arnstein, and B. R. Franza, "User study techniques in the design and evaluation of a ubicomp environment," in *UniComp 2002: Ubiquitous Computing*, ser. Lecture Notes in Computer Science, G. Borriello and L. Holmquist, Eds., vol. 2498. Göteborg, Sweden: Springer-Verlag, September 2002, pp. 73–90.

[3] M. D'Inverno, M. Luck, M. Fisher, and C. Preist, Eds., *Foundations and Applications of Multi-Agent Systems*, ser. Hot Topics, vol. 2403. Lecture Notes in Computer Science, 2002.

[4] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Transaction on Software Engineering*, vol. 24, no. 5, pp. 352–361, May 1998.

[5] D. Bonura, F. Corradini, E. Merelli, and G. Romiti, "Farmas: a MAS for extended quality workflow," Dipartimento di Informatica, Università di L'Aquila, Tech. Rep. TR05-2003, 2002.

[6] E. Merelli, R. Culmone, and L. Mariani, "Bioagent: a mobile agent system for bioscientists," in *NETTAB Workshop on Agents nd Bioinformtics*, Bologna, July 2002.

[7] N. R. Jennings, "An agent-based approach for building complex software systems," *Communications of the ACM*, vol. 44, no. 4, pp. 35–41, 2001.

[8] W. Aalst, "The Application of Petri Nets to Workflow Management," *The Journal of Circuits, Systems and Computers*, vol. 8, no. 1, pp. 21–66, 1998.

[9] M. Dumas and A. H. M. ter Hofstede, "UML activity diagrams as a workflow specification language," *Lecture Notes in Computer Science*, vol. 2185, pp. 76–90, 2001.

[10] W. Aalst, A. P. Barros, A. Hofstede, and B. Kiepuszewski, "Advanced workflow patterns," in *7th International Conference on Cooperative Information Systems (CoopIS 2000)*, ser. Lecture Notes in Computer Science, O. Etzion and P. Scheuermann, Eds., vol. 1901. Berlin: Springer-Verlag, 2000.

[11] J. Odell, H. Parunak, and B. Bauer, "Extending uml for agents," in *Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence.*, 2000.

[12] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an Agent Communication Language," in *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, N. Adam, B. Bhargava, and Y. Yesha, Eds. Gaithersburg, MD, USA: ACM Press, 1994, pp. 456–463.

[13] Foundation for Intelligent Physical Agents, "FIPA97 specification, part 2: Agent communication language, Specification," October 1997.

[14] F. Bellifemine, A. Poggi, and G. Rimassi, "JADE: A FIPA-compliant agent framework," in *Proceedings of the Practical Applications of Intelligent Agents and Multi-Agents*, April 1999, pp. 97–108.

[15] Loccioni Group, http://www.loccioni.com.

[16] BLAST: Basic Local Alignment Search Tool , http://www.ncbi.nlm.nih.gov/BLAST/BLASThome.html.

[17] WfMC, "The workflow management coalition," http://www.wfmc.org.

[18] J. W. Chang and C. T. Scott, "Agent - based workflow: TRP support environment (TSR)," *Computer Networks and ISDN Systems*, vol. 28, no. 1501, 1996.

[19] M. Merz, B. Liberman, and W. Lamersdorf, "Using mobile agent to support inter-organizational workflow management," *Applied Artificial Intelligence*, vol. 11, no. 6, pp. 551–572, 1997.

[20] I. Hawryszkiewycz and J. Debenham, "A workflow system based on agents," in *Database and expert systems applications*. Springer-Verlag, 1998, vol. 17, pp. 688–.

[21] A. Wang, R. Conradi, and C. Liu, "Integrating workflow with interacting agents to support cooperative software engineering," in *Software Engineering and Applications*, 2000.

[22] N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, and B. Odgers, "Autonomous agents for business process management," *Int. Journal of Applied Artificial Intelligence*, vol. 14, no. 2, pp. 145–189, 2000.

[23] M. B. Blake, "An agent-based cross-organizational workflow architecture in support of web," in *Proceedings of the International Conference on Electronic Commerce*, 2001, pp. 567–588.

[24] R. Salice, E. Montaldo, M. Coccoli, M. Paolucci, and A. Boccalatte, "Agent-based architecture for workflow management in manufactoring," in *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science and Education on the Internet*, L'Aquila, Italy, August 2000.

[25] H. Stormer, "A flexible agent-workflow system," in *Workshop on Agent-Based Approaches to B2B, at the Fifth International Conference on Autonomous Agents*, 2001.

[26] A. Ricci, E. Denti, and A. Omicini, "Agent coordination infrastructures for virtual enterprises and workflow management," in *Cooperative Information Agents V*, ser. LNCS, M. Klusch and F. Zambonelli, Eds., vol. 2182. Springer-Verlag, 2001, pp. 235–246.

[27] M. Becht, T. Gurzki, J. Klarmann, and M. Muscholl, "ROPE: Role oriented programming environment for multiagent systems," in *Conference on Cooperative Information Systems*, 1999, pp. 325–333.

[28] The Workflow Management Coalition, http://www.wfmc.org.

[29] FIPA: Foundation for Intelligent Physical Agents , http://www.fipa.org.