Int J Softw Tools Technol Transfer (2004) 6: 231–244 / Digital Object Identifier (DOI) 10.1007/s10009-004-0158-5

An agent-based approach to tool integration

Flavio Corradini[1], **Leonardo Mariani**[2], **Emanuela Merelli**[1]

[1] Dipartimento di Matematica e Informatica, Università di Camerino, 62032 Camerino, Italy
e-mail: {flavio.corradini,emanuela.merelli}@unicam.it
[2] Dipartimento di Informatica, Sistemistica e Comunicazione, Università di Milano Bicocca, 20126 Milano, Italy
e-mail: mariani@disco.unimib.it

Published online: 3 November 2004 – © Springer-Verlag 2004

Abstract. Tool integration is a very difficult challenge. Problems may arise at different abstraction levels and from several sources such as heterogeneity of manipulated data, incompatible interfaces, or uncoordinated services, to name just a few examples. On the other hand, applications based on the coherent composition of activities, components, services, and data from heterogeneous sources are increasingly present in our everyday lives. Consequently, tool integration takes on increasing significance.

In this paper we analyze the tool-integration problem at different abstraction levels and discuss different views on a layered software architecture that we have designed specifically for a middleware that supports the execution of distributed applications for the orchestration of human/system activities. We noticed that the agent paradigm provided a suitable technology for abstraction in tool integration. Throughout the paper, the discussion refers to a case study in the bioinformatics domain.

Keywords: Agent supported tool integration – WfMS (Workflow management system) – Data extraction and integration – Bioinformatics tools – MAS (Multiagent systems) – Data Management

1 Introduction

Many applications are built on top of software or hardware tools (or a combination of the two) that provide automatic or semiautomatic activities, components, services, or data abstractions. These kinds of applications will be increasingly present in our software products due also to the increasing number of available software libraries (often open source) that allow for software reuse and easy prototyping. Such libraries integrate the underlying tools and provide more abstract software entities (e.g., one can integrate editors, parser tools, compilers, and model checkers for the verification of correctness system properties).

In most cases, this integration process can be very difficult. The involved tools may have incompatible data formats, different interfaces, different interaction paradigms, different degrees of accessibility to internal functionalities (i.e., by API), different extension capabilities (i.e., by plug-ins), and different access rights. Standards (for data formats or communication protocols) provide an important level of confidentiality and agreement among tools and (will) certainly suggest rigorous methodologies for the tool-integration problem. At the moment, however, most tools do not comply to standards, and there is still a need for suitable methodologies and technologies for integrating existing tools (as also emerged in the ESEC/FSE 2003 International Workshop on "Tool-Integration in System Development," held in Helsinki in September 2003).

In this paper, we analyze the tool-integration problem at different abstraction levels. Since most of the above-mentioned tool-based applications naturally possess a layered software architecture, we discuss the different views on a software architecture that we have designed specifically for a middleware that supports the execution of distributed applications for the orchestration of human/system activities.

We discuss the tool integration problem at different abstraction levels: (1) service and data accessibility, (2) information management and coordination, and (3) tool orchestration. The service and accessibility level is the lowest level of abstraction and consists of a coherent environment providing data and services. This coherent environment needs then to be properly glued. This is the main aim of the information management and coordination abstraction level. At this level, information on the semantics of interfaces, communication channels, data, etc. is particularly critical for properly implementing, for instance, interoperability among tools. The tool orchestration level is the highest level of abstraction and provides a programming environment for the coordination of

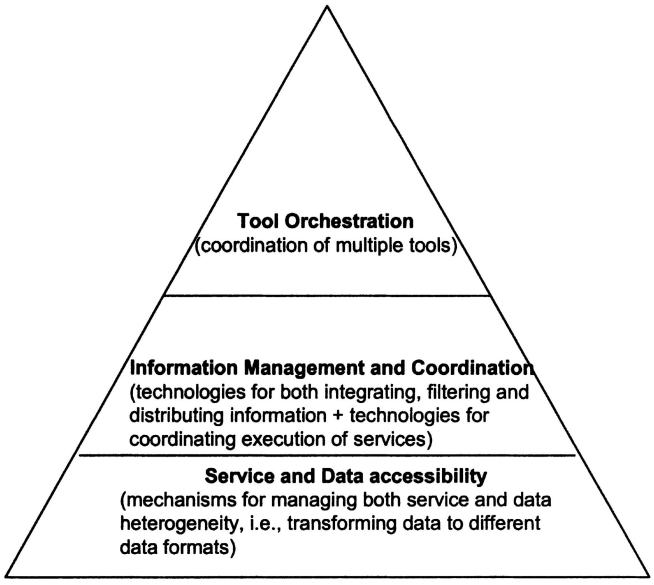

Fig. 1. Different abstraction levels in tool-integration

several heterogeneous tools. Figure 1 shows the three discussed abstraction levels.

A three-layered software architecture (Fig. 3) was proposed in [27] for a middleware that supports the execution of global activity-based applications. At the top of the architecture is the *User Layer*, which supports the specification of a workflow of activities. In the middle of the architecture is the *System Layer*, which provides the needed environment to map a user-level workflow into a pool of agents that implements a set of more primitive activities. At the bottom of the architecture is the *Run-Time Layer*, which provides the needed support for the run-time execution of system level agents.

To show the applicability of our proposed integration framework, we present empirical evaluations of the problem of Web-based tool integration. In particular, we consider the case of bioscientists that repeatedly use online data sources and computational tools to complete their experiments. Early results demonstrate an increase on reuse, simplicity, and productivity.

The rest of the paper is organized as follows. Section 2 discusses the abstraction levels in tool integration, while Sect. 3 introduces the reader to our case study. Then, Sect. 4 presents the agent-based middleware software architecture and shows how the layers suitably conform with the tool integration abstraction levels. Section 5 shows preliminary experimental results obtained from our case study. Finally, approaches to tool integration not discussed in Sect. 2 are presented in Sect. 6, while conclusions and future work are outlined in Sect. 7.

2 Abstraction levels in tool integration

Implementing a development environment composed of many integrated tools requires the management of several aspects, such as synchronization of operations, data transformations, performance tuning, and management of permissions. Our research focuses on achievement of interoperability among tools [34] and integration of functional aspects; integration of nonfunctional properties need not always be addressed explicitly and will not be discussed in the present work.

By surveying existing approaches in tool integration it is possible to recognize certain aspects that play a strategic role: data, services, information, coordination, and orchestration of tools. In fact, tools implement *services* and both consume and provide *data*. Moreover, *information* extracted from tools must be managed and activities must be *coordinated* to support the execution of complex tasks, which requires the *orchestration* of several tools. Integrating tools requires achieving *interoperability* on these aspects. To master the complexity of this goal, we analyze the tool-integration problem from different levels of abstraction, as shown in Fig. 1.

The recent ESEC/FSE 2003 Workshop on "Tool Integration in System Development" held in Helsinki 1–2 September 2003 highlights the necessity of addressing tool integration at all abstraction levels; in fact, existing technologies focus on data and service integration [46, 76, 84] (and the quality of the integration [58]), on metadata management and coordination of services [17, 51], and, finally, on tool orchestration [6, 26, 57], but they still lack a framework conceptually addressing all levels of abstractions.

Figure 1 summarizes aspects that must be addressed when integrating tools; thus developing a complete tool-integration environment corresponds to providing both a technology addressing each level and a framework enabling integration of these technologies. The clear separation of the functionality implemented at each level of the abstract architecture in Fig. 1 enables the possibility of using different technologies that we briefly survey in the following paragraphs.

Service and data accessibility. This is the lowest level of abstraction and consists of a coherent environment providing data and services. The environment interacts with requestors by referring to abstract data models and abstract functionalities that simplify interaction with tools. Abstract data can be obtained by combining multiple concrete data, in the same way as an abstract functionality can be obtained by interacting with several services implemented by tools.

Database management systems (DBMSs) are a good example of systems presenting an abstract data model, i.e., the relational model, instead of the concrete data model, i.e., the physical model. Moreover, the DBMS provides the possibility to execute abstract operations, e.g., operations specified both by queries and by invocation of stored procedures that are mapped to operations on the data.

Several other technologies referring to abstract models provide accessibility to data and services [54, 78], but

when facing tool integration it is necessary to face the additional issue of integrating potentially incompatible abstract models; in fact it is not reasonable to assume any agreement among tools.

In this scenario, self-describing platform-independent semistructured data models gain importance; in particular the XML technology [18] has attracted the attention of both researchers and practitioners. XML is a flexible and portable data format widely adopted when interoperability between applications is required. Today, XML is supported by several tools, e.g., XMLTK [8] and Xalan [7]; models, e.g., DOM [50] and SAX [20]; and languages, e.g., XSLT [23] and XPath [24]. Many resources are still non-XML, thus several XML-based wrappers have been implemented [3, 11, 56]; such wrappers provide an XML-based layer of abstraction embedding the target resource.

In the case of services, the necessity of interoperability in Web-scale environments leads to the development of the Web Services technology. A Web Service has been defined as "a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web Service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols" [82]. Several technologies are available for Web Services, i.e., dynamic discovery of services [12]. Service discovery and run-time binding has also been used in other technologies, e.g., CORBA [63] and EJB [78] dynamically discover and link distributed components, while multiagent systems use discovery services for dynamically identifying services provided by agents [41].

Information management and coordination. This is the level where extracted data can be correctly interpreted, integrated, and managed; such operations can be safely completed only by considering semantics; in fact a wrong interpretation of the data can lead to misuse.

A simple way to overcome semantic problems consists in defining a common schema of known semantics shared among all entities of the information management and coordination level. The solution based on the common schema has several drawbacks: a common schema is difficult to define and update and the semantics is only implicitly specified (it is still possible that different data consumers associate a different semantics to the same terms). Moreover, data consumers often need to work locally with their own schema despite the existence of a shared schema; therefore, it would be necessary to define mapping functions transforming data from one schema to another [53]. Writing these functions when many schema exist is a long error-prone activity. Moreover, transformations are sensible to changes on the schema; indeed, if a schema is modified, all the transformations working on the schema must be modified accordingly.

Issues related to semantics can be addressed by defining ontologies [39]. An ontology provides a formal specification of the knowledge about a domain and can be used to remove or minimize errors on data interpretation. When two entities interact referring to the same ontology, the semantic correctness of exchanged data is guaranteed. Ontologies have been used in several contexts, such as data integration [29], wrapper generation [35], and agent communication [40].

Coordination has been implemented by four main reference models: client-server, meeting-oriented, blackboard-based, and Linda-like. Client-server interactions are characterized by spatial and temporal coupling because communication requires both naming the server and synchronizing the operations. In meeting-oriented models, communication can take place only in specific points at specific times, but it is not necessary to know the identity of the receivers; therefore, it represents the case of a time coupling, but spatial uncoupling, model. The blackboard-based model implements spatially coupled but temporally uncoupled communication; indeed, interactions take place by shared data spaces where it is possible to leave messages for a specific receiver. Finally, Linda-like models are completely uncoupled because communication takes place by shared data spaces with associative retrieval mechanisms (hence the receiver is not stated in the message).

There are several different implementations of each model. Client-server-based coordination models have been exploited in various contexts, such as multiagent systems [55], peer-to-peer systems [73], and component-based systems [59]. Meeting-oriented models have been implemented in [70]. Blackboard models have been used in many systems, as presented in [36]. Finally, a proliferation of Linda-like models have been implemented, e.g., JavaSpaces [42], KLAIM [31], TuCSoN [68], and Lime [72].

In the context of tool integration, client-server coordination models [19, 43, 84] have been preferred when integrating tools from different vendors (where wide agreement on data formats and data semantics is difficult to obtain), while a solution based on the central repository [16, 53, 71] has been preferred when integrating tools of the same vendor (where wide agreement on data formats and data semantics is simple to obtain).

Tool orchestration. This is an environment supporting interaction and integration of several heterogeneous tools. Tool orchestration can be achieved with *transparency* of existing tools or with *awareness* of existing tools. In the case of transparent tool integration, the environment for orchestrating tools provides a homogeneous set of primitives and functionalities for the specification and execution of complex activities without explicitly relying on user knowledge about the tools. In the case of aware tool integration, the tool orchestration environment facilitates interoperability and integration of

tools by letting the user be aware of the tool he/she is using.

An intuitive way to specify complex activities with different levels of transparency is by workflows [49]. Several languages can be used to specify a workflow; activity diagrams [32] and petri nets [1] are two well-known examples. Workflow-based orchestration presents the advantage of providing a homogeneous environment that can hide to the user the existence of tools. An important drawback consists of the complexity of the engine executing the workflow.

On the other hand, different environments exist for awareness integration of tools [6]. These environments provide the possibility of interacting with multiple tools simplifying mutual interactions. In general, the user is aware of the tool that is actually in use, of the functionalities activated on the running tools, and of the data produced by each tool. Moreover, it is required that the user be able to interact with each single tool. ToolNet is a well-known example of this integrated environment [6].

With respect to the presented abstraction levels, our architecture addresses the integration of (heterogeneous) tools through wrapper agents based on the AIXO technology [11] at the service, multiagent systems for what it concerns the information management and coordination abstraction level and workflows for what it concerns the tool orchestration abstraction level. We actually noticed the effectiveness of the agent paradigm for tool integration. In our experience, agents turn out to be good abstractions for service and system heterogeneity [15] and for information integration [27] (see Sect. 4 for a detailed description of agent technology).

3 Motivating example: tool integration in the bioinformatics domain

Bioinformatics is an emerging scientific discipline that uses information technology to organize, analyze, and distribute biological information in order to answer complex biological questions. It involves the solution of complex biological problems using *computational tools* and systems. It also includes the collection, organization, storage, and retrieval of biological information using *databases and data management tools*. The amount of available information is constantly increasing, and it is difficult to exploit available data from all sources [44]. Many of the available data are interrelated, but it is currently difficult to identify or use those related data because different tools use different data formats and with different semantics.

The online sources of these data provide sophisticated user interfaces by which computational tools and deeply interconnected data sets of great richness can be used. Unfortunately, each interface is different, both in the subset of data presented and in organization. The lack of a standard interface for resource access leads to a huge variety of tool-integration problems.

To explore the complexity of biological applications, we focus on the problem of retrieving from gene information the three-dimensional representations of the *p53 protein*'s molecular structure.[1] According to the Web-based tutorial in [66] for the Oak Ridge National Laboratories, the problem can be solved by executing several steps that imply interaction with many tools. The proposed solution represents the state of the art for biologists that need to discover information on the p53 protein by using online resources.

The tutorial requires the biologist to interact continuously with Web sites in order to activate tools and retrieve data, to filter the retrieved data, and to use the gathered information in the further steps. By analyzing the above-mentioned tutorial [66], we distinguish ten different questions that must be answered in order to complete the tutorial and get the final result:

Q_1: What is the *general information* about *p*53?
Q_2: What is the *nucleotide sequence* of the *p53 gene*?
Q_3: What if we are either given just the *protein sequence* or the *DNA sequence* and would like to find the other *type of sequence*?
Q_4: Now that I have the sequence, is this *sequence similar* to those of *other proteins*?
Q_5: Do the *bases* in *p53 code* for any specific type of *recognition site*?
Q_6: What does this protein *look like*?
Q_7: What does a *mutated* form of this protein look like *compared to the original type* form of this protein?
Q_8: Now that we know what *proteins* are *similar* to *p*53, how similar are they with respect to sequence?
Q_9: What *other organisms* have *p*53?
Q_{10}: What are some *common mutations* found in *p*53?

Since the technicalities involved in implementing each query are similar, we simply focus on query Q_1. According to the above-mentioned tutorial, we have to perform the following steps:

1. *Start* from "Online Mendelian Inheritance in Man (OMIM)" site.
2. *Search* the OMIM *database* by entering the words "*p53 protein*".
3. *Choose* the first result "*191170 TUMOR PROTEIN p53; TP53" from among several.
4. *Retrieve* the associated Web page.
5. *Peruse* the information given previously to make a further choice. On this page, indeed, several accesses to multiple areas of research are available. The OMIM database introduces the protein domain through a lot of other general information such as:

   (a) the description of p53,

[1] The gene for the p53 tumor-suppressor protein plays a vital role in regulating cell growth. Since it is a tumor suppressor, it halts abnormal growth in normal cells and therefore prevents cancer. Mutations in this gene can cause an ineffective regulation of the cell cycle and the possibility of cancer. See [65] for further details.

(b) how it is cloned,
(c) how it works,
(d) the possible forms of gene therapy,
(e) the molecular genetics,
(f) the three-dimensional imaging, and
(g) the animal models of the protein,

from which specific information can be *extracted*.

6. *Integrate* the extracted information.

These steps consist of interactions with Web sites by either browsing or interacting with bioinformatics tools and integrating the results. We remark that today all these operations are essentially manually performed by bioscientists. We will consider in Sect. 5 this sequence of operations as a case study for both evaluating our tool-integration framework and applying our integration technology.

4 The agent-supported tool-integration environment

In our previous work, we noticed the effectiveness of the agent technology when dealing with service and system heterogeneity [15] and with information integration [27]. Moreover, other researchers have successfully addressed heterogeneity and information integration by agent technology [77, 79]. Our idea is to extend this approach to the tool-integration domain. With respect to Fig. 1, agent technology represents entities providing *information management* and *coordination*, e.g., agents can retrieve and filter information while at the same time coordinating the sending and receiving of messages. In our system, integration of the information is based on both the agent's knowledge management engine and ontology-driven interactions. In particular, communication between two agents is possible only if an ontology exists[2] that is known by both agents; this restriction guarantees the agreement on the semantics of exchanged data. Moreover, whenever an agent obtains additional information, it *integrates* the information with its personal knowledge base. Each agent is responsible for the *consistency* and the *correctness* of this operation.

Coordination is hard coded in the agents that explicitly synchronize their operations by sending and waiting for messages. Moreover, agents must be aware of the identity of receivers, and they must also know when they are going to receive a message. Therefore, interactions are both temporally and spatially coupled.

Agent technology provides additional benefits:

- Agents can exploit optimal consumption of resources by dynamically cloning, merging, dying, or passing tasks [74].
- Agents can exploit mobility; mobile agents are particularly effective in mobile scenarios, such as wireless systems [47], mobile computing [45], and ubiquitous computing [13].
- Agent technology is very effective when applied to decentralized systems such as peer-to-peer systems [67].
- Agent technology is an ideal paradigm for specifying active entities that must coordinate to perform complex tasks; thus agents are good candidates for workflow execution [21, 60].

Both *data* and *service accessibility* are obtained by implementing a wrapper agent (WA) associated to each tool. The WA receives requests from other agents and translates them to the corresponding operations that must be performed on the target tool. Requests are issued by sending Agent Communication Language (ACL) [40] messages. The WA capability of manipulating tools is enhanced by the use of the Any Input XML Output (AIXO) technology [11]. AIXO enables the agile development of XML-based wrappers and provides the possibility of interacting with the wrapped tool as it would be a resource for generating XML documents.

Finally, the *tool orchestration* capability is provided by an environment enabling the specification of workflows. A workflow specification is a natural way to describe with a high level of abstraction tasks solving complex problems. Moreover, we have recognized that several problems have been successfully solved by specifying high-level workflows [2, 4, 83]. Furthermore, an environment based on the specification of workflows can be specialized for multiple application domains by loading libraries of domain-specific activities, e.g., if the user is a biologist and needs to retrieve information about DNA sequences, she can load libraries containing procedures for Internet searches and manipulations of sequences. The workflow specified by the user is used to generate a pool of agents implementing the activities. The mapping from the workflow specification to the pool of running agents is described in Sect. 4.2.

The *programming environment* contains the implementation of the IDE supporting workflow specification and many additional utilities such as compilers and parsers. Actually, we are implementing a console for both controlling the status of the execution of the workflow and retrieving partial results.

Figure 2 presents entities created at each level of abstraction: 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 agents implementing all specified activities; agents are executed and their synchronization is supported by mechanisms implemented in the run-time

[2] The term ontology is used to refer to a set of terms of known semantics; the semantics can be formally specified.

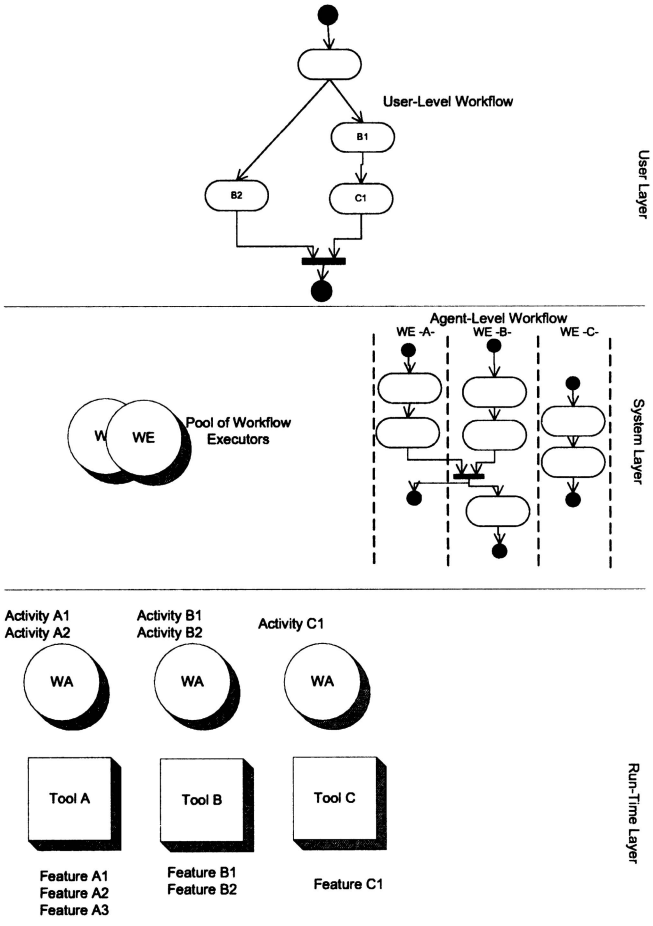

Fig. 2. Entities located at each layer

layer; finally, low-level interaction with tools is possible through the interaction of WA and AIXO wrappers.

To clarify our approach, Fig. 3 shows 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 composed of the workflow management environment (editor, engine, compilers, ...). At the system layer, the application is a group of running agents named Workflow Executors (WE), and the infrastructure is given by the agent execution environment. Finally, at the run-time layer, the application is given by WA and the infrastructure consists of the core environment supporting their execution.

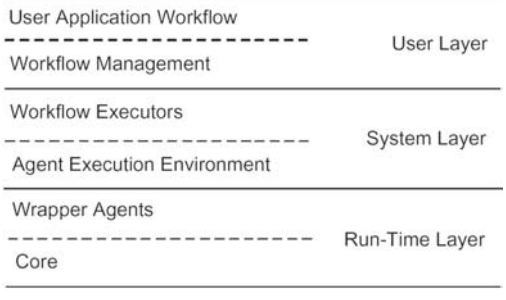

Fig. 3. Applications and infrastructure of each layer

4.1 User layer

The *user layer* is based on workflows and provides to users a set of programs for interacting with the 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 system.

The *workflow editor* is the program that allows one to specify workflows by composing activities in a graphical environment. The editor enables the specification of workflows complying with the WfMC reference model [49] and is implemented by using the JaWE [37] editor. Activities used in a workflow are configured by specifying input parameters, and their effects are recognizable as modification of state variables or modification of 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. Complex activities simplify the specification of workflows because they enhance both hierarchical specification and reuse: we can use an already existing complex activity without regard to 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 the tool. The *input data format* specifies the accepted input for a given activity. In a similar way, the *output data format* specifies the accepted output data formats. The *environment* parameter is used to specify what context an activity must be performed in since the same activity with the same parameters can be performed in different environments. The environment is separated from the other input parameters because it can cause either the migration of a tool or the selection of a specific implementation of the activity, while input parameters denote only data transfer. For example, the activity of using one of the BLAST[3] implementations in a given repository implies the deployment of BLAST on a remote site and the activation of the tool. In a similar way, the activity of searching for given information on a given database is always the same, but its implementation is very different with respect to the target database, i.e.,

[3] BLAST is a common and widely used tool in biology; more information is at `http://www.ncbi.nlm.nih.gov/BLAST/`.

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 information on the *tool* is used either when it is not possible to achieve transparency or when the user prefers to be aware of the activated tool.

We do not formally describe activities in this paper. However, we are aware that several description languages are emerging both from the workflow (XLANG, XPDL) and the Web Services (WSFL, DAML-S) communities. We are working on a formal framework where workflows can be described being compiled into pools of agents (WE).

The user can request data by interacting with the system in two main modes: offline and online. In offline interaction, the user starts the execution of a workflow and then stops interacting with the system, e.g., he/she turns off the computer. When the user needs to know the current status of the execution, he/she can use the *interaction console* for requesting analysis, monitoring the workflow execution, and retrieving partial results. In the online mode, the user starts executing the workflow and continues interacting with the system until the workflow is completed. The online mode is also supported by the interaction console. The console can be executed both as a normal computer program and within a Web interface, and in this way it is possible to interact with the system from any computer connected to the Internet. The interaction console enables a posteriori communication; thus additional parameters can be sent to agents that are waiting for information.

4.2 System layer

The system layer hosts WEs that are agents generated from the ULW specification. WEs execute and coordinate their actions to reach the fulfillment of the ULW specification. Some of the actions executed by a WE require interaction with WAs; these actions correspond to operations that must be completed by interacting with a tool.

Communication between agents takes place once the negotiation of 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. If the system has defined a shared ontology, the ontology negotiation procedure always succeeds.

We now discuss the two-phase agent-generation procedure that is performed by the compiler. In step 1 the ULW is mapped to an ALW, and in step 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 contains only primitive actions (actions that can be directly executed without decomposing them into workflows). Details of the procedure for agent generation are presented in the following paragraphs.

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 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, if 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 of 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 one agent to another by using communication activities, i.e., an activity whose execution consists of sending a message to the receiver. Actually, communication consists of sending and receiving single messages. In the future we would like to extend this approach to a definition of protocols that must be respected during interagent 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 the receivers and which the 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 the number of generated agents, minimize the 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. An actual prototype of the compiler implements part of these features.

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" implementations of agents (the skeletons).

A skeleton is a role-specific implementation of an agent that does not contain any behavior, 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 con-

crete WE is obtained by plugging the specified behavior into the skeleton. In particular, the compiler performs the following 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 for completing 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 behavior `CB` and state entity `SE` into 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 startup by dynamic binding. Instead of generating compiled WEs, it is possible to use skeletons behaving as interpreters of ALW specifications. In such cases, 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, they exploit the ability to dynamically modify their behavior at run time. The organization of our system enables the use of both types of agents. Actually, we are implementing the compiler producing compiled agents, but we also plan to investigate interpretation and dynamic adaptability.

4.3 Run-time layer

The run-time layer supports the deployment of tools and contains the implementation of mechanisms enabling service and data accessibility, e.g., wrappers.

In our system, we use WAs as concrete representations of tools that must be accessed. The WA maps requests received from WEs to a sequence of operations that must be performed on the corresponding tool. A WA can be responsible for multiple tools. In such cases, WEs perceive a unique tool, e.g., a tool for manipulating DNA sequences, but the WA interacts with multiple tools, e.g., several tools enabling different operations on DNA sequences.

The implementation of the WAs is based on the AIXO technology [11]. AIXO is a software architecture for wrapper systems tailored to provide an abstraction layer based on the XML technology for any type of resource. An AIXO wrapper is used for each tool; thus the WA percepts tools as XML producing resources.

The behavior of an AIXO wrapper is based on three steps. In the first step, the target resource is accessed and data, if any, are collected in that resource's native data format. In the second step, data are mapped to XML by taking into account only structural information, e.g., data gathered from a relational database are mapped to XML by considering the organization based on rows and columns. Finally, in the third step, a set of XSLT transformations [23] are used to map XML data to the final XML document. The final XML document complies with the data semantics required from the WA.

Moreover, the adoption of the AIXO-based technology provides the following benefits [11]:

- The component-based architecture both enables flexible wrapping of tools and minimizes the number of updates when changing the structure of the wrapped resources.
- The XML output data format enables a high degree of reusability and portability.
- AIXO is entirely based on the XML technology, so it can be easily updated to include new XML standards.

5 Experimental results

We use the motivating example presented in Sect. 3 to evaluate our technological solution to tool integration. In particular, we implemented several WAs for the biological tools involved in the tutorial, we implemented the activities and the workflows stored in, respectively, the ULAD and the ULAID, and, finally, we implemented a few skeletons.

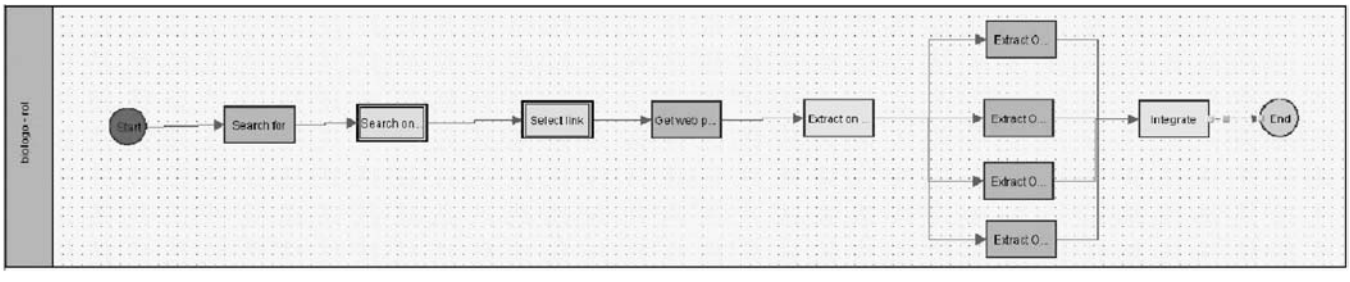

Fig. 4. JaWE workflow specification

In our experimentation, we wrote, the ULW W_1 implements the query Q_1 by the workflow editor. The workflow is represented in Fig. 4. Activities are specified with several parameters (as explained in Sect. 4) that do not have a graphical representation in JaWE; thus we report the complete list of used activity in what follows (`E` denotes the environment, `I` input data, `T` the activated tool, and `O` output data):

search on Database
E: *OMIM Database*
I: "P53, Articles"
O: ListOfLinks:JDOM

select Link
I: ListOfLinks, "1"
O: link

get web page
I: link
O: TP53WebPage

extract OMIM information
E: OMIM-ProteinWebPage
I: TP53WebPage, "description"
O: Description

extract OMIM information
E: OMIM-ProteinWebPage
I: TP53WebPage, "cloning"
O: Cloning

extract OMIM information
E: OMIM-ProteinWebPage
I: TP53WebPage, "gene function"
O: GenFunction

extract OMIM information
E: OMIM-ProteinWebPage
I: TP53WebPage, "gene therapy"
O: GeneTherapy

integrate OMIM information
I: Description, Cloning, GenFunction, GenTherapy
O: p53GenralInformation

Solving Q_1 by W_1 instead of manually executing all steps, i.e., solving problems by *workflows*, is a good practice and provides several benefits:

- W_1 can be specified by knowing only the tool-integration environment without regard to integration and tools.
- If the biologist needs more control on the underlying system, it is possible to specify the tool parameter to explicitly indicate tools that must be activated.
- W_1 can be reused without effort: the biologist loads and reexecutes the workflow with a few clicks. Moreover, W_1 can be adapted to new requirements by changing the control flow or the parameters. Even subparts of W_1 can be reused.
- The biologist monitors workflow execution by the interaction console; therefore, he/she does not need to be continuously in front of a computer.
- New tools and new resources can be included in the system, and the workflow specification takes advantage of these new sources without requiring updates.

The agent layer has been implemented by the BioAgent multiagent system [62] – a multiagent system that we have developed for biological applications. However, WEs and WAs can be implemented on top of any existing multiagent system supporting distributed communication between agents. Actually, WAs behave by mapping each received message, which represents a command, to a sequence of operations performed on available tools. Gathered results are then shipped back to the requestor by XML messages. Furthermore, actual implementation of WEs support some of the advanced functionalities introduced in Sect. 4.

In particular, WEs can *adapt* to unexpected events and *manage* "technical aspects" by behaviors built into the skeletons that were used to generate the WEs. We implemented several agents embedding some of these behaviors. For example, we used reactions to manage the situation where a mobile WE is not able to reach a given destination host. Two reaction policies are available: waiting indefinitely until migration is possible or retrying for a limited number of times and then aborting. In our experiments, adaptability was demonstrated to be useful for facing unexpected interactions with tools and unexpected states of the environment, even if these situations rarely occur. Agent adaptability can be enhanced much more by developing a larger set of behaviors and by implementing a mechanism to explicitly integrate events generated from behaviors built into skeletons with behaviors generated by the workflow. We are working to build a version of the agent architecture supporting this mechanism.

The ability of agents to *clone* and *create* other agents have proved very useful for managing distribution of tasks, while the agent coordination paradigms have proved useful for coordinating those running tasks. By using both agent coordination and reproduction capabilities we shift the focus from tool coordination to agent coordination. In fact, execution and coordination of tasks are performed by executing and coordinating WEs, which implies running tools and integrating results. Agent systems work at a higher abstraction level with respect to systems of integrated tools; thus the user benefits from this abstraction by perceiving a simpler system. In some cases, the user can necessitate deep controllability of the system despite interaction simplicity and usability, but in the case of Web-based tools, which is our case, we find it more valuable to increase automatization, simplicity, and reusability with respect to maintaining direct controllability of tools.

Decentralization of multiagent systems is naturally exploited in our system; in fact, workflow specifications define peer interactions among agents that correspond to exchanging data among tools, i.e., sending data from one WE to another often corresponds to sending data gathered from a tool to another WE, which will use the received information to interact with a new tool. Therefore, peer interactions among agents resemble peer interactions among tools.

Many tools provide only a local interface; thus, *mobility* is an important property of WEs that enables integration and distributed interaction of tools even for local tools only. This property has been used in our experiments for interacting with several command-line programs (use of the BLAST command-line program has not been reported in the presented part of the case study). Moreover, local communication, in some cases, can be used to avoid large bandwidth consumption caused by intensive exchange of messages between a remote WE and a tool.

The *decision-making* capability of agents has not been used in our case since the workflow defines most of the behavior of the running agents. Moreover, further behaviors are obtained by the compilation process and by using skeletons with built-in capabilities. Agents implementing decision-making capabilities are often "large" since they use knowledge bases, inference engines, etc. Therefore, intelligent agents do not seem very useful in our context. However, we recognize possible applications of intelligent agents in the case of tools producing large data sets. Intelligent agents can use the extracted data to increase the knowledge base and to infer facts that can then be used to make decisions. We did not exploit this possible application of agents.

Wrapping of tools has been implemented by AIXO. In the case of tools providing a Web interface, AIXO sends `HTTP` requests and collects results that are then parsed and converted to XML. In the case of databases, AIXO sends queries to the DBMS and converts responses, usually recordsets, to XML. AIXO wraps tools provid-

```
<aixo_message>
       <wrapper xml_result="JDOM" id="OMIM_Database" XQLstring="" >
          <Action value="SearchOnDatabase"/>
          <ResulType value="ListOfLink"/>
          <Parameter_access name="proteinID" value="P53"/>
          <Parameter_access name="informationType" value="Articles"/>
        </wrapper>
</aixo_message>
```

a

```
<?xml version="1.0" standalone="no" ?>
<Source>Select Entries from OMIM -- Online Mendelian Inheritance in Man</Source>
<Protein> p53 </Protein>
<OMIM_links>
    <OMIM_link>
           <MinInformation MinNumber="191170" MinType="star" />
           <Aliases><Alias> TRP53 </Alias> <Alias>P53</Alias> <Alias> TRANSFORMATION-
                    RELATED PROTEIN 53 </Alias>  </Aliasis>
           <Title>TUMOR PROTEIN p53; TP53 </Title>
           <Symbol>TP53</Symbol>
           <Locus> 17p13.1 </Locus>
           <Link>/htbin-post/Omim/dispmim?191170</Link>
    </OMIM_link>
    <OMIM_link>
           <MinInformation MinNumber="602143" MinType="star" />
           <Aliases><Alias> 53BP2  </Alias> <Alias>APOPTOSIS-STIMULATING PROTEIN OF p53,
                    2; ASPP2 </Alias> </Aliasis>
           <Title>TUMOR PROTEIN p53-BINDING PROTEIN 2; TP53BP2 </Title>
           <Symbol>TP53BP2 </Symbol>
           <Locus> 1q42.1</Locus>
           <Link>/htbin-post/Omim/dispmim?602143</Link>
    </OMIM_link>
…
</OMIN_links>
```

b

Fig. 5. a WE message to WA; **b** AIXO results

ing a command-line interface by interacting with input and output streams; the output stream is parsed and converted to XML. Finally, AIXO wraps tools providing APIs by sending requests to the interface and converting single results to XML. Some applications of AIXO are presented in [11].

In the proposed example, the WE performs the first activity *search on database* by sending a message to the WA; the message includes the wrapper to be selected, i.e., *OMIM database*, the expected output result's type, i.e., ListOfLinks, and the input parameters, i.e., "P53" and "Articles" as described in Fig. 5a. The WA selects the corresponding AIXO wrapper and, by using the access method, retrieves the list of links in OMIM format, which will be translated into an XML format as described in Fig. 5b.

In our work, ontologies have been implemented by taxonomies of terms of common intended semantics, and we use them to enable interactions among agents. This basic form of ontology has been suitable for sharing knowledge and enabling communication. However, it is also possible to move to more complex ontologies [25, 39]. Other approaches address the use of ontologies for integrating tools [28, 38, 52], but their concrete effectiveness remains an open issue. In our case, migration to a more ontology-driven context is quite simple since the agent layer supports communication based on ontologies and the workflow layer can easily integrate ontologies together with task specification, for example, by dynamically loading libraries of both terms and relations that can be used during workflow specification. Ontologies provide a way to implement automatic reasoning and inference from exchanged data, but they also require additional computational resources. Further investigation is necessary to establish costs and benefits of creating a system using complex forms of ontologies. Actually, we recognized that shared taxonomies are a good starting point and represent an acceptable tradeoff between complexity and interoperability [30].

6 Related work

Few software architectures are designed to support tool integration in a heterogeneous distributed environment; among those we mention the agent-based architecture proposed in [28] in which a software tool and information servers are encapsulated as "agents" that receive and reply to requests using both a declarative knowledge representation language and a library of formal ontologies that defines the vocabulary of various domains. ToolNet architecture [6] is based on a service-oriented approach whose main components are Services and ToolAdaptors. The service encapsulates a tool by segmenting its functionalities into basic services, whereas the adaptor is responsible for the communication between the integrated tool and the framework, its main task being to publish/provide the services supported by the specific tool. We share with ToolNet [6] the idea of viewing a tool as a collection of services that can be accessed from other tools. Moreover, the encapsulation of tools into agents that interact by using shared vocabularies is present in both the agent-based architecture in [28] and our architecture. However, our system combines the advantages of both the agent abstraction and encapsulation of tools with a set of services; in addition, our architecture provides a suitable architecture for direct tool integration and provides a coherent, homogeneous, and simplified framework where the user can transparently interact with tools by high-level workflows.

Tool registration and discovery functionalities for network tool management have been proposed by Mueller et al. in TRMS (Tool Registration and Management Service) [64]. TRMS allows the dynamic discovery of a tool using the semantic description of the desired behavior. A tool is described by a set of significant properties based on which it can be discovered in the network. This approach is mainly related to two other works dedicated to the definition of a specification and integration language, TES (Tool Encapsulation Specification) language [75] and WSDL (Web Service Description Service) [81], the latter being used for encapsulating the tool in a service. In our case, encapsulation of the service is performed by agents and tool discovery, and integration is transparently managed by both the middleware and the workflow engine.

Integration of tools and services is often performed by plugging components into existing systems to extend implemented functionalities. This type of integration requires components designed for the system where they are plugged in and requires designing the target system for supporting the addition of components. Providing a way to add a plug-in into an application is a development practice that is becoming common. A well-known example is given by Eclipse [33], which is an open source extensible development environment supporting the addition of plug-ins. Integration of functionalities by plug-ins is not considered in our approach since we address tools developed by independent parties that do not guarantee agreement on any extension mechanism.

7 Conclusions and future work

Several works have been proposed to support the workflow execution by multiagent systems [14, 22, 61], most of them focused on components, agents, and problem solving. Indeed, we concentrate on service accessibility, information management, and coordination of activity for tool integration.

The construction of environments addressing tool integration has the promising for creating homogeneous frameworks exploiting interoperability, reuse, and transparency. We presented a rationalization of the tool-integration problem consisting of a three-layered view:

(1) service and data accessibility, (2) information management and coordination, and (3) tool orchestration. We expect that future work will take advantage of this conceptual view when developing new tool-integration frameworks and when comparing existing approaches.

Moreover, we suggested an approach based on AIXO wrappers, agents, and workflows. AIXO wrappers exploit flexible XML-based wrapping of tools. The agent technology exploits management of heterogeneity and coordination of activities. Finally, the workflow technology exploits transparency with respect to complexity of existing tools. Our experience highlights the effectiveness of the integration of these technologies when dealing with large collections of tools placed in a distributed setting. Furthermore, we previously applied the proposed architecture in various contexts [15, 27], such as automation [15]; this shows the generality of the ideas explored in this paper.

We are moving toward a more formal notation for specification of activities by exploring the use of XLANG, XPDL, WSFL, and DAML-S. We also plan to perform further experiments in different application domains addressing coarse-grained tools as well, i.e., tools that are difficult to model as software modules implementing a set of activities. Emerging technologies, such as ubiquitous computing and mobile computing, provide attractive domains to test our tool-integration approach, especially for the flexibility of the agent-based technology. In the future, we expect that tool integration will consider also these domains.

Acknowledgements. We would like to thank Alessandro Ricci for valuable comments on a preliminary version of this paper.

References

1. Aalst W (1998) The application of Petri nets to workflow management. J Circuits Syst Comput 8(1):21–66
2. Abbott KR, Sarin SK (1994) Experiences with workflow management: issues for the next generation. In: Proceedings of the 1994 ACM conference on computer supported cooperative work, Chapel Hill, NC. ACM Press, New York, pp 113–120
3. Adelberg B (1998) NoDoSE a tool for semi-automatically extracting structured and semistructured data from text documents. In: Proceedings of the 1998 ACM SIGMOD international conference on management of data. ACM Press, New York, pp 283–294
4. Agostini A, de Michelis G, Grasso MA, Patriarca S (1993) Reengineering a business process with an innovative workflow management system: a case study. In: Proceedings of the conference on organizational computing systems. ACM Press, New York, pp 154–165
5. Allouche M-K, Sayettat C, Boissier O (1997) Towards a multi-agent system for the supervision of dynamic systems. In: Proceedings of the 3rd international symposium on autonomous decentralized systems (ISADS '97), Berlin, April 1997, pp 9–16
6. Altheide F, Dörfel S, Dörr H, Kanzleiter J (2003) An architecture for a sustainable tool integration. In: Proceedings of the ESEC/FSE workshop on tool integration in system development, Helsinki, September 2003
7. Apache Software Foundation (2003) Xalan-j. `http://xml.apache.org/xalan-j/`
8. Avila-Campillo I, Green TJ, Gupta A, Onizuka M, Raven D, Suciu D (2002) XMLTK: An XML toolkit for scalable XML stream processing. In: Proceedings of Programming Language Technologies for XML (PLANX), Pittsburgh, PA, October 2002
9. Barber KS, Goel A, Martin C (2000) Dynamic adaptive autonomy in multi-agent systems. J Exp Theor Artif Intell 12(2):129–147
10. Barber KS, Martin CE (2001) Dynamic reorganization of decision-making groups. In: Proceedings of the 5th international conference on autonomous agents, Montreal. ACM Press, New York, pp 513–520
11. Bartocci E, Mariani L, Merelli E (2003) An XML view of the "world". In: Proceedings of the 5th international conference on enterprise information systems (ICEIS'03), Angers, France, April 2003, pp 19–27
12. Bellwood T, Clément L, Ehnebuske D, Hately A, Hondo M, Husband YL, Januszewski K, Lee S, McKee B, Munter J, von Riegen C (2002) UDDI version 3.0. Published specification, Oasis
13. Bergenti F, Poggi A (2001) LEAP: A FIPA platform for handheld and mobile devices. In: Proceedings of the workshop on agent theories, architectures, and languages (ATAL)
14. Blake MB (2001) Agent-based workflow configuration and management of on-line services. In: Proceedings of the 4th international conference on electronic commerce research (ICECR-4), Dallas, TX
15. Bonura D, Corradini F, Merelli E, Romiti G (2004) Farmas: a MAS for extended quality workflow. In: 2nd IEEE international workshop on theory and practice of open computational systems. IEEE Press, New York
16. Boudier G, Gallo T, Minot R, Thomas I (1998) An overvie of PCTE and PCTE+. In: Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering symposium on practical software engineering environments
17. Braun P (2003) Metamodel-based integration of tools. In: Proceedings of the ESEC/FSE workshop on tool integration in system development, Helsinki, September 2003
18. Bray T, Paoli J, Sperberg-McQueen CM, Maler E, Yergenau F, Cowan J (2003) Extensible markup language (XML) 1.1. W3C proposed recommendation, World Wide Web Consortium (W3C), November 2003
19. Brown A, Carney D, Morris E, Smith D, Zarella P (1994) Principles of CASE tool integration. Oxford University Press, Oxford, UK
20. Brownell D (2002) SAX2. O'Reilly, Sebastopol, CA
21. Chen Q, Hsu M, Duyal U, Griss M (2000) Multi-agent cooperation, dynamic workflow and XML for e-commerce automation. In: Proceedings of the 4th international conference of autonomous agents (Agents 2000), Barcelona, Spain, June 2000
22. Chen Q, Hsu M, Duyal U, Griss M (2000) Multi-agent cooperation, dynamic workflow and xml for e-commerce automation. In: Proceedings of the 4th international conference of autonomous agents, Barcelona, Spain, June 2000
23. Clark J (2003) XSL transformations (XSLT) version 1.0. W3C recommendation, World Wide Web Consortium (W3C)
24. Clark J, DeRose S (1999) XML path language (XPath) version 1.0. W3C recommendation, World Wide Web Consortium (W3C)
25. Connolly D, van Harmelen F, Horrocks I, McGuinness DL, Patel-Schneider PF, Stein LA (2001) DAML+OIL reference description. W3C note, W3C, December 2001
26. Corradini F, Mariani L, Merelli E (2003) An agent-based layered middleware as tool integration. In: Proceedings of the ESEC/FSE workshop on tool integration in system development, Helsinki, September 2003
27. Corradini F, Mariani L, Merelli E (2003) A programming environment for global activity-based applications. In: Proceedings of WOA 2003 dagli oggetti agli agenti – sistemi intelligenti e computazione pervasiva
28. Cranefield S, Purvis M (1997) An agent-based architecture for software tool coordination. In: Cavedon WWL, Rao A (ed) Intelligent agent systems: theoretical and practical issues. Lecture notes in artificial intelligence, vol 1209. Springer, Berlin Heidelberg New York, pp 44–58

29. Cui Z, Jones D, O'Brien P (2002) Semantic B2B integration: issues in ontology-based approaches. SIGMOD Rec 31(1): 43–48
30. Culmone R, Rossi G, Merelli E (2002) An ontology similarity algorithm for bioagent. In: Proceedings of the NETTAB workshop on agents and bioinformatics, Bologna, Italy, July 2002
31. de Nicola R, Ferrari GL, Pugliese R (1998) Klaim: a kernel language for agents interaction and mobility. IEEE Trans Softw Eng 24(5):315–330
32. Dumas M, ter Hofstede AHM (2001) UML activity diagrams as a workflow specification language. Lecture notes in computer science, vol 2185. Springer, Berlin Heidelberg New York, pp 76–90
33. Eclipse (2003) Eclipse Platform Technical Overview. White paper. `http://www.eclipse.org`
34. ECMA (1993) Reference model for frameworks of software engineering environments. Technical Report NIST 500-211, ECMA TR/55 3rd edn
35. Embley D, Campbell D, Jiang Y, Liddle S, Ng YK, Quass D, Smith R (1999) Conceptual-model-based data extraction from multiple-record web pages. Data Knowl Eng 31(3):227–251
36. Englemore R, Morgan T (1988) Blackboard systems. Addison-Wesley, Reading, MA
37. Enhydra (2003) Jawe. `http://jawe.enhydra.org/`
38. Falbo RA, Guizzardi G, Natali ACC, Bertollo G, Ruy FF, Mian PG (2002) Towards semantic software engineering environments. In: Proceedings of the 14th international conference on software engineering and knowledge engineering. ACM Press, New York, pp 477–478
39. Fensel D (2001) Ontologies: a silver bullet for knowledge management and electronic commerce. Springer, Berlin Heidelberg New York
40. FIPA-ACL (1997) FIPA97 specification, part 2: Agent communication language. Specification, FIPA, October 1997
41. Foundation for Intelligent Physical Agents (2003) FIPA agent discovery service specification. Preliminary version 1.2e
42. Freeman E, Hupfer S, Arnold K (1999) Javaspaces principles, patterns and practice. Addison-Wesley, Reading, MA
43. Freude R, Königs A (2003) Tool integration with consistency relations and their visualization. In: Proceedings of the ESEC/FSE 2003 workshop on tool integration in system development, Helsinki, September 2003
44. Frishman D, Heumann K, Lesk A, Mewes H-W (1998) Comprehensive, comprehensible, distributed and intelligent databases: current status. Bioinformatics 14(7):551–561
45. Fuggetta A, Picco GP, Vigna G (1998) Understanding code mobility. IEEE Trans Softw Eng 24(5):352–361
46. Haase T (2003) Semi-automatic wrapper generation for a-posteriori integration. In: Proceedings of the ESEC/FSE 2003 workshop on tool integration in system development, Helsinki, September 2003
47. Hadjiefthymiades S, Matthaiou V, Merakos L (2002) Supporting the WWW in wireless communications through mobile agents. Mobile Netw Appl 7(4):305–313
48. Hexmoor H, Vaughn JT (2002) Computational adjustable autonomy for NASA personal satellite assistants. In: Proceedings of the 2002 ACM symposium on applied computing. ACM Press, New York, pp 21–26
49. Hollingsworth D (1995) The workflow reference model. Workflow Management Coalition Specification TC00-1003, Workflow Management Coalition, Winchester Hampshire, UK, January 1995
50. Hors AL, Hégaret PL, Wood L, Nicol G, Robie J, Champion M, Byrne S (2003) Document object model (DOM) level 3 core specification. W3C candidate recommendation, World Wide Web Consortium (W3C)
51. Jin D, Cordy J (2003) A service sharing approach to integrating program comprehension tools. In: Proceedings of the ESEC/FSE workshop on tool integration in system development, Helsinki, September 2003
52. Jin D, Cordy JR, Dean TR (2003) Transparent reverse engineering tool integration using a conceptual transaction adapter. In: Proceedings of the 7th European conference on software maintenance and reengineering, March 2003
53. Karsai G, Gray J (2000) Design tool integration: an excercise in semantic interoperability. In: Proceedings of IEEE engineering of computer-based systems
54. Kreger H (2001) Web services conceptual architecture (WSCA 1.0). Technical report, IBM Software Group, May 2001
55. Lange D, Oshima M (1998) Programming and deploying Java mobile agents with aglets. readings. Addison-Wesley, Reading, MA
56. Liu L, Pu C, Han W (2000) XWRAP: An XML-enabled wrapper construction system for web information sources. In: Proceedings of the international conference on data engineering (ICDE), pp 611–621
57. Margaria T, Wuebben M (2003) Tool integration in the ETI platform – review and perspectives. In: Proceedings of the ESEC/FSE workshop on tool integration in System Development, Helsinki, September 2003
58. Mariani L, Pezzè M (2003) Behavior capture and test for controlling the quality of component-based integrated systems. In: Proceedings of the ESEC/FSE workshop on tool integration in system development, Helsinki, September 2003
59. Matena V, Hapner M (1999) Enterprise javabeans™ specification. Public Draft version 1.1, Sun Microsystems
60. Meng J, Helal S (2000) An ad-hoc workflow system architecture based on mobile agents and rule-based processing. In: Proceedings of the 2000 international conference on artificial intelligence, Las Vegas, June 2000
61. Meng J, Helal S (2000) An ad-hoc workflow system architecture based on mobile agents and rule-based processing. In: Proceedings of the 2000 international conference on artificial intelligence (ICAI2000), Las Vegas
62. Merelli E, Culmone R, Mariani L (2002) Bioagent: a mobile agent system for bioscientists. In: NETTAB workshop on agents and bioinformatics Bologna, Italy, July 2002
63. Merle P (2001) Corba 3.0 new components chapters. TC Document ptc/2001-11-03, Object Management Group, November 2001
64. Mueller W, Schattkowsky T, Eikerling H, Wegner J (2003) Dynamic tool integration in heterogeneous computer networks. In: IEEE (ed) Design, automation and test in Europe conference and exhibition (DATE'03), Munich, 3–7 March 2003
65. National Library of Medicine (US) (ed) (2003) Genes and disease. Bethesda, MD
66. Leonard S (2001) A web-based tutorial written for Oak Ridge National Laboratories. Senior Thesis Project. `https://sharepoint.cisat.jmu.edu/isat/klevicca/Web/p53/Tutorial.htm`
67. Ogston E, Vassiliadis S (2002) A peer-to-peer agent auction. In: Proceedings of the 1st international joint conference on autonomous agents and multiagent systems, Bologna, Italy. ACM Press, New York, pp 151–159
68. Omicini A, Zambonelli F (1998) Coordination of mobile information agents in TuCSoN. Internet Res Electron Netw Appl Policy 8(5):400–413
69. Parunak HVD, Baker AD, Clark SJ (1998) The AARIA agent architecture: from manufacturing requirements to agent-based system design. In: Proceedings of the workshop on agent-based manufacturing, ICAA98, Minneapolis, MN, May 1998
70. Peine H (1997) Ara – agents for remote action. In: Crockayne W, Zyda M (eds) Mobile agents: explanations and examples. Prentice-Hall, Englewood Cliffs, NJ
71. Picard P (1990) SFINX: Tool integration in a PCTE based software factory. In: Proceedings of the 1st international conference on system development environments and factories, pp 219–228
72. Picco GP, Murphy AL, Roman G-C (1999) Lime: Linda meets mobility. In: Proceedings of the 21st international conference on software engineering
73. Rhea S, Wells C, Eaton P, Geels D, Zhao B, Weatherspoon H, Kubiatowicz J (2001) Maintenance-free global data storage. IEEE Internet Comput 5(5):40–49
74. Shehory O, Sycara K, Chalasani P, Jha S (1998) Agent cloning: an approach to agent mobility and resource allocation. IEEE Commun Mag 36(7):62–67
75. Specification TE (1992) Version 1.0.0, CAD framework initiative inc., Austin, TX

76. Stoeckle H, Grundy J, Hosking J (2003) Notation exchange converters for software architecture development. In: Proceedings of the ESEC/FSE workshop on tool integration in system development, Helsinki, September 2003
77. Subrahmanian VS, Bonatti P, Dix J, Eiter T, Kraus S, Ozcan F, Ross R (2000) Heterogeneous agent systems. MIT Press, Cambridge, MA
78. Sun Microsystems (2003) Java™2 platform enterprise edition specification. Final Draft v1.4, Sun Microsystems
79. Sycara K, Decker K, Pannu A, Williamson M, Zeng D (1996) Distributed intelligent agents. IEEE Expert 11(6): 36–45
80. Sycara KDK (1997) Intelligent adaptive information agents. J Intell Inf Syst 9(3):239–260
81. W3C (2002) Web services description language (WSDL) version 1.2. W3C working draft, W3C, July 2002
82. W3C Web Services Architecture Working Group (2002) Web services architecture requirements. W3C working draft, W3C, August 2002
83. Weske M, Goesmann T, Holten R, Striemer R (1999) A reference model for workflow application development processes. In: Proceedings of the international joint conference on work activities coordination and collaboration, San Francisco. ACM Press, New York, pp 1–10
84. Wilcox P, Russell C, Smith M, Smith A, Pooley R, MacKinnon L, Dewar R, Weiss D (2003) A CORBA-oriented approach to heterogeneous tool integration: Ophelia. In: Proceedings of the ESEC/FSE workshop on tool integration in system development, Helsinki, September 2003
85. Wooldridge M (2000) Reasoning about rational agents. Intelligent robotics and autonomous agents. MIT Press, Cambridge, MA