An agent-based layered middleware as tool integration*

Flavio Corradini
Dipartimento di Informatica
Università di L'Aquila
67100 L'Aquila, Italy
flavio@di.univaq.it

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

Emanuela Merelli
Dip.to di Mat. e Informatica
Università di Camerino
62032 Camerino, Italy
emanuela.merelli@unicam.it

ABSTRACT

Applications based on coordination and orchestration of human/system activities are more and more present in our nowadays life. They often need the coherent composition of components, services and data from heterogeneous sources. Moreover, the increasing number of (often conflicting) attributes makes the design of these complex applications very difficult.

We tackle the problem of tool integration by means of an agent-based layered middleware that supports the execution of the above mentioned applications. In order to control heterogeneity, the middleware associates, as a first layer of integration, "wrapper agents" to the tools we integrate. Wrapper agents may present tool services in a standard format, convert data in a coherent way or interfaces for interoperability - depending on the nature of the wrapped tools. On the top of wrapper agents reside the so-called "user agents" that bridge the gap between applications, described as workflow of activities, and tools. Summing up, workflows expressed in a graphical notation guarantee transparency and user-friendliness at the user level. Agent-based technologies guarantee interoperability and coordination of activities. Wrapper agents and the software infrastructure guarantee support for gluing tools.

An application of our tool integration environment is presented in the Bioinformatics application domain. We are currently investigating its applicability to other interesting domains.

1. INTRODUCTION

Tool integration is a very complex activity due to, for instance, heterogeneity of manipulated data, incompatible interfaces and uncoordinated services. In most cases, tools are black-box systems that do not allow any change in the tool itself.

We simplify tool integration by means of two abstraction levels. The first one is implemented by wrapping each tool with a Wrapper Agent (WA). A WA provides both the same set of services of the wrapped tool and the set of features to interoperate with other agents of the system (by means of an Agent Communication Language [8]). The second one is implemented by a set of utilities that make it possible to compile a user application workflow into a pool of agents supporting the workflow's activities execution [5].

This two-steps approach guarantees to users an high level of abstraction meaning that users can describe their own applications by assuming that issues like integration and heterogeneity have already been overcome at lower levels of abstraction.

We have previously developed an agent platform [10] supporting the coordinated execution of mobile agents [4], now we are enhancing the platform with workflow facilities [5, 6]. The workflow specification is given by a UML Activity Diagram [7] that composes primitive activities (such as activities of a specific application domain), supplied activities (such as activities supported by a WA) and user-defined activities (such as activities defined by the user as combination of primitive, supplied or other user-defined activities). By the workflow specification, we produce an agent-level workflow and a pool of agents (Workflow Executors) that execute all activities that are specified in the agent-level workflow. The agent-level workflow is a refinement (automatically derived) by the user specified workflow. Workflow Executors (WEs) constitute the glue layer of the proposed system, by which integration and interaction of activities, supplied by WAs, are made effectively possible. The Figure 1 shows the entities that participate in the execution of a workflow.

The rest of the paper is organized as follows. Section 2 describes an agent-based environment for tools integration, while Section 3 describes the environment in the bioinformatics application domain. Section 4 reports some concluding remarks.

2. THE AGENT-SUPPORTED TOOL INTEGRATION ENVIRONMENT

We propose an agent-supported tool integration environment that is the result of our experience with agent technologies in several application domains, such as quality control on manufacturing [3] and biological information extraction [10]. In both cases, we realized that is really needed to publish and integrate services, and more in general tools, provided by third-party companies. Moreover, users are not friendly enough with agent technologies and, hence, the user layer must abstract from any detail behind this technology.

* *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'.*

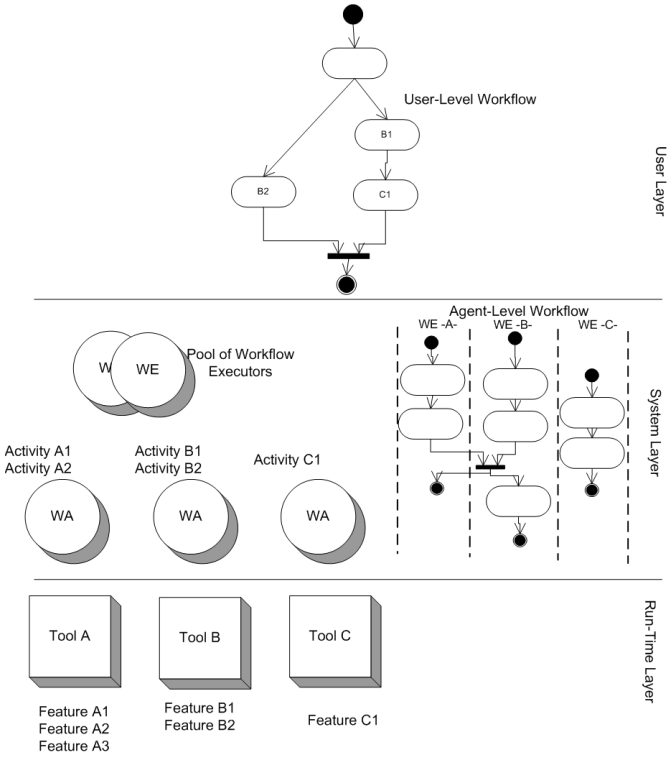

Figure 1: Agent-Supported Tool Integration System

For these reasons, we designed a three layered software architecture (Figure 1) where each layer has a different main actor: workflows at the user layer, agents at the system layer and tools at the run-time layers. Each layers is split in two conceptual levels (the application and its infrastructure) as shown in Figure 2.

User Application Workflow Workflow Management	User Layer
Application Agents Application Agents Management	System Layer
Service Agents Core	Run-Time Layer

Figure 2: The Software Architecture of the Agent-based Middleware

Applications at the *User Layer* are defined by workflows. While a workflow is executing, the user has not the exact perception of what is happening in the underlying layers because the entities that the user manages are activities, control flow arrows and synchronization bars (we reuse the formalism of UML Activity Diagram). Activities can be configured by specifying parameters, and can produce output values. For this purpose we extend the Activity Diagram notation by using UML Notes to specify *input parameters*, *output parameters*, *configuration parameters*, and *global variables*. Input parameters are specified in a note connected to the activity by an ingoing dashed arrows, for example, MaxSequences="10" states that the formal parameter MaxSequences has value 10. Output parameters are specified as an ordered set of variables written in a note connected to the corresponding activity by an outgoing dashed arrow. Configuration parameters are specified similarly to input parameters, but they are contained in a note connected to the activity by a continuous ingoing arrow; the value of a configuration parameter is always the name of a *configuration note*. Configuration notes are disconnected from any element of the workflow and they contain a set of parameters with an assigned value. Finally, there is a special note that is not connected with other entities, which contains the label "GLOBAL VARIABLES"; this note is used to declare variables that are used in the workflow specification. Our intuitive way to extend the UML notation has been partially inspired from [11].

The Workflow Management layer provides an environment supporting workflow specifications with CASE tools, IDE for editing specifications, grammar checkers, and so on.

The *System Layer* is populated with WEs that execute workflows specified by users. The activity that a WE must perform can be of two different types: primitive or supplied. In the case of a primitive activity, the WE contains the implementation of the activity that is directly executable. In the case of a supplied activity, the WE must request to a WA implementing the activity to perform it. User-defined activities

do not exist in the system layer because they are a combination of primitive and supplied activities; when agents are generated, every user-defined activity is always decomposed to an aggregation of primitive and supplied activities. The agent generation is performed at run-time from the workflow specification by the system compiler. The compiler generates the pool of agents by a two steps procedure. In the first step, the user-level workflow specification is mapped to an agent-level workflow specification. In the second step, a pool of agents that implements the agent-level workflow specification is generated.

The mapping from the user- to the agent-level workflow specification is supported by the User-Level Activity Database (ULAD). The ULAD contains the specification of each user-level activity consisting on a workflow of agent-level activities. Both input and configuration parameters of the user-level activity are mapped to input parameters of specific activities in the agent-level workflow. Global variables of the activity are mapped to agents' status variables. The notation used for agent-level workflow is the UML Activity Diagram-like notation used for user-level workflows extended with two main elements: swimlanes and agent creation activities (see for instance Figure 4). A swimlane encloses activities executed by a single agent, in this way the whole agent-level workflow is partitioned in several single agent workflows. The agent creation activity is a special activity that has the effect to create a new agent; it is denoted with an outgoing arrow crossing the swimlane of the agent that creates the new agent, which ends on the first node of the workflow of the new agent. This arrow may have a note attached, every value contained in the attached note is a value transmitted to the new agent. Each agent of the agent-level workflow has got an identifier that is used to refer to it when necessary. The agent-level workflow does not contain any configuration note, because parameters contained in the user-level configuration notes are mapped to input parameter of activities. Further details and an example of the mapping process are provided in Section 3.

In the second step, the compiler must generate agents corresponding to swimlanes of the agent-level workflow specification. The agent generation is supported by a Database of Skeletons (DoS), e.g., a database containing agent executable code that can be configured to build the proper agent. The skeletons differentiate each other because are "empty" implementation of different roles, i.e., coordinator agent, travelling agent and data collector agent. Moreover, the User-Level Activity Implementations Database (ULAID) provides the compiler with the implementation of each activity contained in the agent-level workflow. To obtain an agent that effectively behaves as written in the corresponding swimlane, the activity implementations must be opportunely composed and organized. In our system, it is possible to compose activities by a fixed set of operators that are inspired from the Jade's behaviors [2]. These operators include simple (i.e., sequential composition, parallel composition and iteration) and complex behaviors (i.e., conditional composition and conditional repetition). Number and type of operators assure that any specification can be implemented. The agent that must be activated is obtained by composing: the agent skeleton selected from the DoS, the dynamically created behavior and any necessary status variables. Status variables are declared in the agent-level specification and are the minimal set of variables that the agent needs to complete its task. Agent skeletons contained in the DoS implement special capabilities that facilitate the insertion of both the assembled behavior and the status variables.

At the end of the two steps process the whole pool of agents is created. We are investigating about the possibility to include optimization in the compiler. The compiler we are defining has the characteristic to be both *context-aware* and *opportunistic*, in fact it takes advantage of the particular configuration that the system has got in a certain moment when generating agents. In the case there are no information available (worst case), the compiler behaves as a traditional compiler, but in the case it is possible to know the current status of the system, it effectively uses this information to properly configure agents. Useful information at a certain time can be the load status of some places (in such case these places could be ignored by generated agents), the type of information that are maintained in each place (in such case it is possible to prioritize the order of the visits), agents that are running in the different places (in such case it is possible to contact an existing agent instead of generating a new one), and so on.

The *Run-Time Layer*, at the bottom of the architecture, provides the needed support for tools deployment and agent execution. This layer provides primitives for communication, security, mobility and other low level functionalities.

3. A CASE STUDY: TOOL INTEGRATION IN THE BIOINFORMATIC DOMAIN

As example of agent-based middleware for tool integration we consider the context where a bioscientist carries out his/her daily experiments. 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, and it is difficult to exploit available data from all sources [9]. Usually, bioscientists must collect information and integrate different tools such as BLAST, FASTA and all those provided by web sites [1] (NCBI, EBI, EnsEMBL, SRS, ...) to gain evidence of the correlations between cause and effect in their experiments. In such a context, the wide distribution of information and the heterogeneity of the data sources, besides the availability of a variable range of bioinformatics tools make difficult for a bioscientist to manually collect, select, elaborate and integrate meaningful information. Let us consider the case of a bioscientist, knowing only a genetic sequence (i.e. the order with which the nucleotides occurred), that tries to construct a stable structure of a new protein by using molecular modelling and homology technique[2]. The technique consists in selecting a set of proteins

[1]Even a workshop has been organized on bioinformatics tools and Websites http://pga.lbl.gov/Workshop/Feb2003/webTools.html

[2]In theory, by knowing the sequence of a protein (the order with which the amino acids occurred) is possible to infer its functional properties, a milestone for the cancer's cure.

considered more similar to the new one, searching for their crystallographic structure (by using either known protein structures or existing scientific papers as possible sources), and finally trying to construct the crystallographic structure of the new protein by using the crystallographic structure of the selected proteins. We concentrate on the first two steps that are difficult tasks to be accomplished, especially when the search starts from the only knowledge of a genetic sequence (DNA or m-RNA). In our case study, WAs create a uniform XML-based interaction framework by integrating several data formats (FASTA, ASN1,...) and by coordinating several tools (BLAST, FASTA programs, ...).

Figure 3 shows the workflow presented in the previous paragraph. The bioscientist,through the graphical interface, chooses the "select similar sequences" as initial activity. This activity has three parameters: the size of the result (in the example 10), the type of the sequence (in our the example m-RNA) that must be taken into account during the search process and the sequence (in our example `MEEP...DSD`) that must be used for matching similarity. The result of the activity is a set of sequences. The activity has a configuration parameter that is `SIMILARITY`. The `SIMILARITY` configuration note contains information related to the tool that must be used to check for similarity (in the example BLASTn) and the repositories that can be accessed to look for sequences (in the example GenBank). After the first activity is performed, the workflow is split into two branches that are executed simultaneously. In one case, the workflow continues by using the sequences to locate PDB structures and then by locating crystallographic structures from PDB structures. On the other case, sequences are used to locate related articles, and finally articles are used to extract crystallographic structures from the SWISS-PROT repository. Results from both branches are integrated by the "Union" activity and finally, the whole package is uploaded by FTP to the bioscientist repository.

The compilation process works by performing the mapping as described in Section 2; Figure 4 contains the result of the compilation process. To highlight how the mapping activity is performed we enclosed the set of agent-level activities corresponding to the same user-level activity in a dashed rectangle (the rectangle contains also a label with the name of the activity that generates it). The compiler performs several optimizations, for example the "find crystallographic structures from sequences" activity does not contain the "move to" activity that otherwise must be generated, because the compiler recognizes that the agent is just running on the target place.

Inter-agent communication exploits message passing. It has to noted that some complex coordination/synchronization activity is not explicitly represented but left to implementation details. For instance, *Agent1* and *Agent2* in Figure 4 send a message to *SynchAgent1* by means of the activity "SendSynchMSG", while the *SynchAgent1* receives the message by the "Synch Agents" activity. The underlying protocol is not explicitly shown in the diagram as, in general, complex communication protocols. The "Synch Agents" activity, indeed, would need several actions (receive the first message, receive the second message, notify both agents).

Finally, WEs are generated from the agent-level specification of Figure 4. WEs interact with local WAs and orchestrate tools as specified in the workflow. To solve the data integration problem we implemented AIXO (Any Input XML Output) [1] a generalized XML wrapper. WAs use AIXO to transform data to a common schema. A WA behaves like a pipe of three steps: (1) the WA accesses data and extracts information in its native data format (in the example FASTA format) by special purpose access methods; (2) the WA maps data to a "rough" XML document; (3) the WA maps the "rough" XML document to an XML document that complies to a common schema. These three operations correspond to three main issues that are successively addressed by the WA. In the first step, the WA solves the problem of the heterogeneity in the access methods because it embeds the technology to interact with the wrapped tool; in the second step, the WA solves the problem of the syntactical heterogeneity of data by mapping (with standard rules depending only from the source data type) the result of the previous step to an XML representation (see [1] for details); and finally, the WA addresses the semantic heterogeneity issue by transforming the structure of the data. The resulting XML document is then embedded into ACL messages and is sent to WE by a suitable agent protocol. Referring to Figure 5, we provide a portion of the result given by the WA that embeds the SWISS-PROT's BLASTp.

```
<?xml version="1.0" standalone="no" ?>
- <SwissProt_entries>
- <SwissProt_Entry>
  <ID entry_name="P53_HUMAN" data_class="STANDARD" molecule_type="PRT" sequence_lenght="393" />
  <AC number="P04637" />
  <AC number="Q16848" />
  <AC number="Q9UBI2" />
  <DT date="13-AUG-1987" release="05" type="Created" />
  <DT date="01-MAR-1989" release="10" type="Last sequence update" />
  <DT date="28-FEB-2003" release="41" type="Last annotation update" />
  <DE>Cellular tumor antigen p53 (Tumor suppressor p53) (Phosphoprotein p53) (Antigen NY-CO-13).</DE>
  <GN>TP53 OR P53.</GN>
  <OS>Homo sapiens (Human).</OS>
  <OC>Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi; Mammalia; Eutheria; Primates; Catarrhini; Hominidae; Homo.</OC>
  <OX>NCBI_TaxID=9606;</OX>
- <RN num="1">
  <RP>SEQUENCE FROM N.A.</RP>
  <RX Bibliographic_db="MEDLINE" id="85230577" />
  <RX Bibliographic_db="PUBMED" id="4006916" />
  <RA>Zakut-Houri R., Bienz-Tadmor B., Givol D., Oren M.;</RA>
  <RT>"Human p53 cellular tumor antigen: cDNA sequence and expression in COS cells.";</RT>
  <RL>EMBO J. 4:1251-1255(1985).</RL>
  </RN>
- <RN num="2">
  <RP>SEQUENCE FROM N.A.</RP>
  <RX Bibliographic_db="MEDLINE" id="87064416" />
  <RX Bibliographic_db="PUBMED" id="2946935" />
  <RA>Lamb P., Crawford L.;</RA>
  <RT>"Characterization of the human p53 gene.";</RT>
  <RL>Mol. Cell. Biol. 6:1379-1385(1986).</RL>
  </RN>
...
```

Figure 5: XML data extracted from the SWISS-PROT

4. CONCLUSIONS

The construction of environments addressing tool integration has the promising perspective to create homogeneous frameworks exploiting benefits of composition and reuse of third-party software. We suggested an agent-based approach that takes advantage of flexibility and wide applicability of agent technologies to enhance tool integration also in complex distributed environments. The integration problem leads to the design and the implementation of AIXO, a three steps approach that provides a general way to address the problem to wrap both tools' services and data gathered from tools. The system design based on the separation of concerns principle and the implementation based on the component technology guarantee exploitation of both modularity and reuse. Our experimentation is successful in the biological application domain and we plan to extend the approach to other domains.

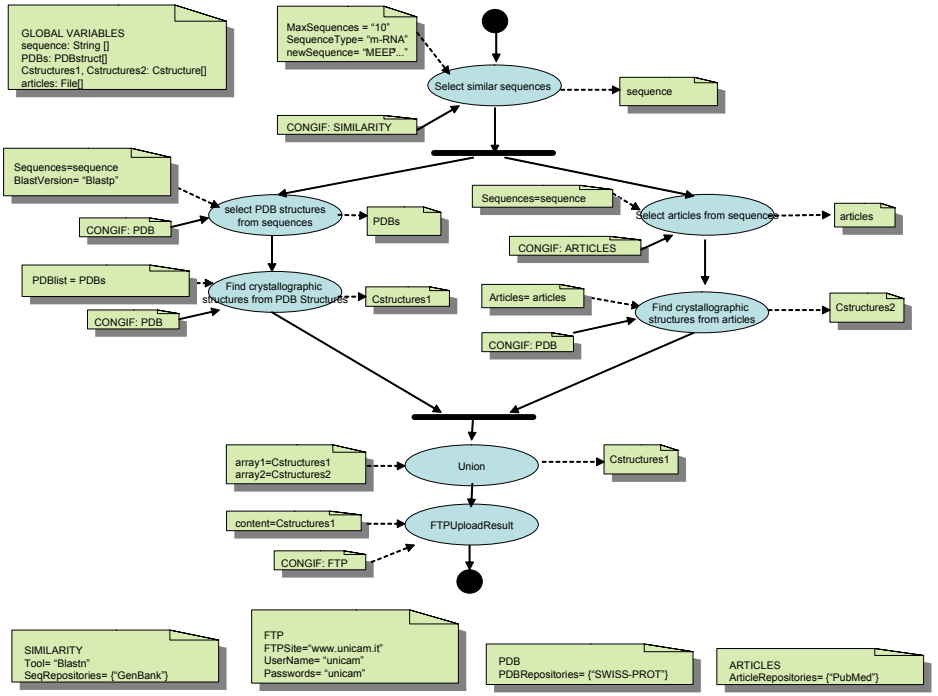

Figure 3: User-Level BioWorkflow

5. REFERENCES

[1] E. Bartocci, L. Mariani, and E. Merelli. An XML view of the "world". In *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS'03)*, pages 19–27, Angers, France, April 2003.

[2] 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*, pages 97–108, April 1999.

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

[4] G. Cabri, L. Leonardi, and F. Zambonelli. MARS: A programmable coordination architecture for mobile agents. *IEEE Internet Computing*, 4(4):26–35, 2000.

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

[6] F. Corradini, E. Merelli, A. Pierantonio, and M. Placidi. Workflow as composition of domain-specific languages. Technical Report TR04, Dipartimento di Informatica, Università di L'Aquila, 2003.

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

[8] FIPA-ACL. FIPA97 specification, part 2: Agent communication language. Specification, FIPA, October 1997.

[9] D. Frishman, K. Heumann, A. Lesk, and H.-W. Mewes. Comprehensive, comprehensible, distributed and intelligent databases: current status. *Bioinformatics*, 14(7):551–561, 1998.

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

[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.

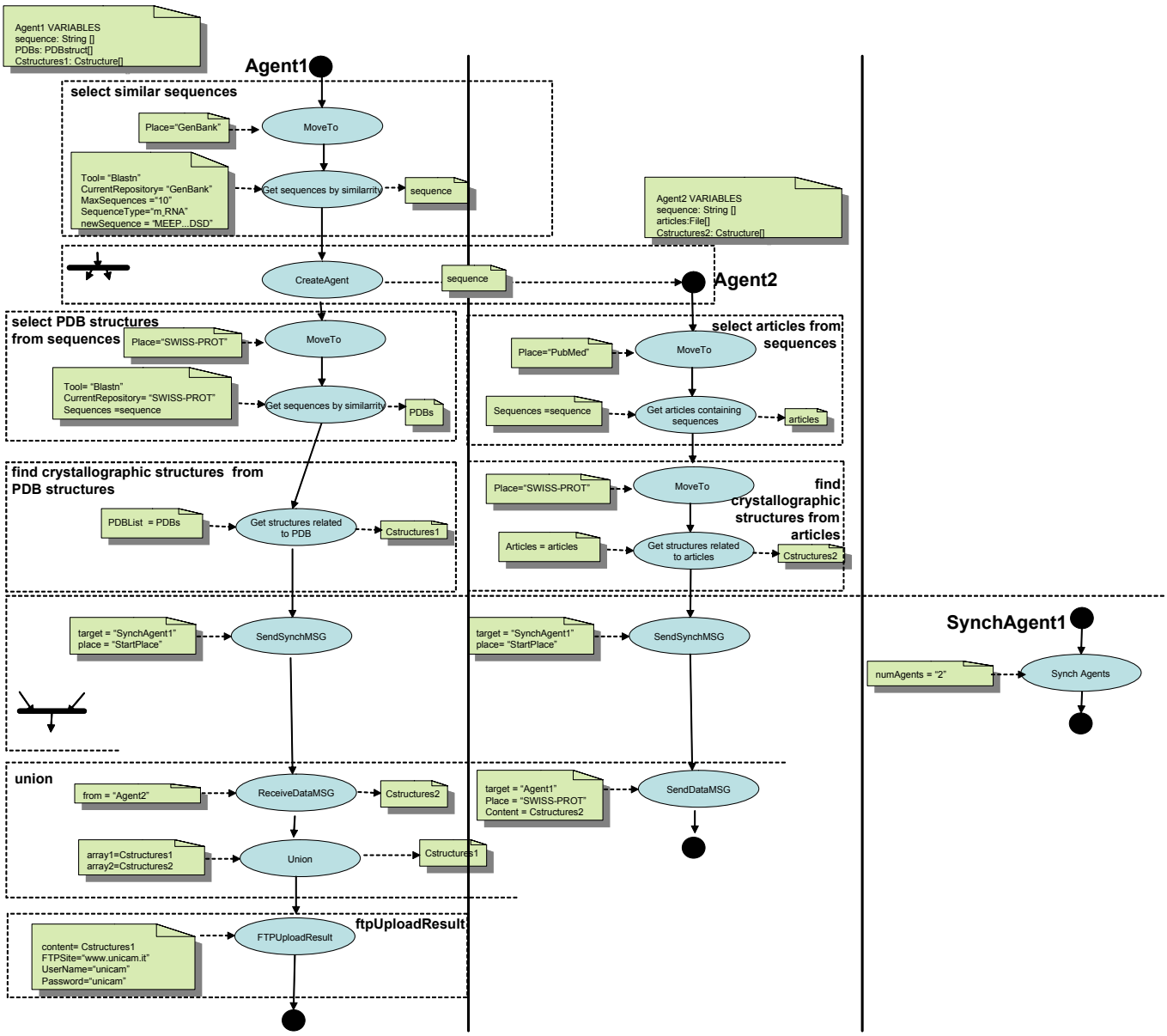

Figure 4: Agent-Level Bioworkflow