Executing Workflows with Mobile Agents from UML Specifications

Flávio M. Assis Silva Paulo Sérgio da Silva(*) Neima Prado dos Santos
Jorge A. Lima Santos Alfrânio T. Correia Jr. João Carlos Selch Sergio Luz
Siddhartha L. Argollo

LaSiD - Distributed Systems Laboratory
UFBA - Federal University of Bahia
Av. Adhemar de Barros, S/N - Campus de Ondina
CEP 41170-110 Salvador, BA

(*) Department of Electrical Engineering
Faculty of Engineering of Bauru
UNESP - State University of São Paulo
Caixa Postal 473 - 17001-970 - Bauru, SP

Abstract

MABflow is a workflow management system based on mobile agents under development at LaSiD/UFBA. In this system, java code for mobile agents are automatically generated from an UML specification. This code is then used during runtime to generate instances of one or more mobile agents that will cooperatively execute the workflow. This paper describes the MABflow system and process models, how migrating workflows can be specified in UML and how code is automatically generated. The paper addresses the problem of creating an appropriate methodology and tools for supporting the process of developing mobile agent-based applications. Allowing mobile agent-based workflow specifiers to model workflows using UML is a step towards this goal and is a subject very little explored by the research community. The development of the concepts presented in this paper is part of a project whose goal is to develop a reliable (fault tolerant) workflow management system based on mobile agents.

1 Introduction

Workflow management systems play an important role in today's organizations. A workflow management system controls the execution of *workflows*. There is no commonly agreed definition for workflows and different ways of classifying them exist (for example, [9]). A workflow, however, generally relates to a representation of a *business process*. The control of the execution of a workflow by a workflow management system might occur at different automatization levels (ranging from systems that strongly depend on human actions to completely automated systems), but it involves the coordination of a set of tasks that must be executed according to a set of conditions. Examples of conditions are temporal restrictions, constraints on the level and type of interleaving that tasks from the same or from different workflows might have (concurrency control), restrictions on the ordering of tasks execution, among others.

The management of workflows in so-called *open environments*, like the Internet, faces a set of challenges, due to the typical properties of such environments, such as: massive geographical distribution; the heterogeneity of the communication infrastructure; the coexistence of different administrative domains and autonomous systems; the need for supporting different types of devices with different resource constraints (such as PDA (*Personal Digital Assistants*) and desktop computers); the need for supporting disconnection of devices; among others.

For such environments, *code mobility* has been proposed as an approach to address some of the difficulties. Code mobility relates to the possibility of moving pieces of code from one place to another in a distributed environment. Among the technologies that enable code mobility are the *mobile agent systems*. A mobile agent is a self-contained software component which has the ability to migrate from a node to another in a distributed environment during its execution. When it moves, its execution is suspended in the original node, its code, data and execution state are migrated to the target node, and, after being reinstantiated at the new node, it resumes its execution.

Mobile agents are being proposed as an underlying technology for the implementation of workflow management systems due to a set of reasons, among them:

- Mobile agents provide a decentralized way of executing workflows, thereby avoiding potential bottlenecks that might arise in workflow systems with a centralized form of control (which is the traditional way of implementing workflow management systems) and accommodating more appropriately the execution of workflows to the wide geographical distribution of systems.

- Support for scalability, since the control of the workflow execution might be performed from different parts in the distributed environment.

- Good support for the interaction with human users, since a mobile agent is able to migrate to the host where a user is and interact with him/her locally, with a good quality of service (providing a good support for the interaction with human users is fundamental in workflow management systems, since workflows typically involve interactions with humans).

- Support for disconnected operation. Mobile agents carrying the execution of a workflow might migrate from a host to the network. That host might then become disconnected, while the workflow execution continues. Additionally, a mobile agent can migrate from the network to a host before that host becomes disconnected, thus making possible that a part of a workflow (being executed by that agent) can continue to be executed at that host even during disconnection (supporting disconnected operation accommodates the typical mode of operation of devices such as PDAs).

Although bringing a set of advantages, developing a workflow system based on mobile agents introduces a set of issues. Among them are issues related to the development of a suitable underlying execution infrastructure, such as: how to control the execution of mobile agents (for example, how to determine the current state of mobile agents executing a workflow, how to stop and resume the execution of mobile agents, etc.); how to support appropriately (and reliably) cooperation between a set of mobile agents; and how to achieve mobile agent fault tolerance, security and transactional behaviour. Additionally, issues related to the analysis and design of mobile agent-based workflows also arise, such as: what are appropriate languages and tools for designing and modelling mobile agent-based workflows; how to specify workflow management systems in a way to benefit from the advantages of mobility; how to integrate the development of mobile agent-based workflows with a software development process; and how to generate code for these applications.

We are developing at LaSiD/UFBA (Distributed Systems Laboratory - Federal University of Bahia) a workflow management system based on mobile agents called MAB*flow* (*Mobile Agent-based Workflow Management System*). In this system mobile agents are used to execute workflows. This system is being developed as part of a project with the same name, the goal of which is to demonstrate the benefits of using mobile agents when executing distributed applications such as workflows, to identify the fundamental issues associated with the development of such systems, and to develop appropriate solutions for them.

In the last years we have worked on some of the issues mentioned above, among them: support for fault tolerant and transactional execution of mobile agent-based applications [6, 7, 8, 19, 20]; reliable coordination of groups of cooperating agents [20, 15]; design and architecture of a mobile agent-based workflow management system [12, 2, 1]; and on the integration of the development of mobile agent systems with a software development process [10].

In this paper we describe how we are addressing the latter issue. Our approach is to build tools which can be used by workflow specifiers to model their workflows using UML (*Unified Modelling Language*) [3] and for generating automatically mobile agent code to execute them with MAB*flow*. With that, MAB*flow* workflow specifiers become able to specify workflows according to a current software engineering practice (using UML) and by using a generally known specification language (UML). Since we are relying on a standard for representing UML models as XML documents, called XMI (*XML Metadata Interchange*) [18], specifiers of MAB*flow* workflows can use any UML tool that supports this standard. In other words, any UML tool that supports XMI becomes a workflow specification tool for MAB*flow*
.

The generated mobile agent code uses interfaces and objects of a system called GMAS (*Generic Mobile Agent System*), also under development at LaSiD. GMAS does not provide itself the functionality needed for supporting the lifecycle of mobile agents, like, for example, Voyager [11] or Aglets [14]. Instead, GMAS is conceived as a layer that remains between mobile agent-based applications and specific mobile agent systems. GMAS provides a set of classes, interfaces and objects which are to be used by the agent applications. The implementation of these classes and objects is performed using a specific mobile agent system (for example, Voyager). The main goals of GMAS are: first, to hide the underlying mobile agent system being used from the applications; second, to provide generic functionality commonly needed for the development of mobile agent-based applications which is not usually found in mobile agent systems (such as user and account management, customizable user interfaces, support for disconnected operation, etc.).

The study of the aspects related to the development of mobile agent-based applications has attracted the attention of the research community recently (among other reasons, due to the fact that the effectiveness of the mobile agent application development process impacts strongly the aplicability of the technology itself). The integration of the development of mobile agent-based applications with a methodology based on UML, however, has been very little addressed [13, 16]. As discussed later in this paper, our approach complements these works.

This paper is structured as follows. In section 2 we introduce the MAB*flow* system model, the MAB*flow* process model and how mobile agents are used to execute workflows. After briefly introducing UML activity diagrams in section 3, we describe in section 4 how to specify mobile agent-based workflows with UML in MAB*flow* . In section 5 we describe how we generate mobile agent java code from UML specifications. In section 6 we discuss the related work. Finally, in section 7 we conclude the paper.

2 The MAB*flow* System

2.1 The System Model

In MAB*flow*, a workflow is executed by one or more mobile agents. From a high-level workflow specification, code for a set of mobile agents is automatically generated. The generated code implements the execution logic of the workflow. At runtime, instances of these mobile agents are created as needed to cooperatively execute the workflow. During execution, the agents migrate through the distributed environment accessing services or interacting with human users, according to the workflow specification.

Figure 1 represents components of a distributed environment, as assumed by MAB*flow*. The distributed environment is assumed to be structured in *administrative domains*. Each domain represents the environment of an autonomous organization which has its own workflows and which has control over its resources (workflows, agencies, hosts, etc.).

Each administrative domain is populated by agents, *agencies*, *MABflow (internal) servers*, and *application service providers*. From a logical point of view, an agency represents a conceptual place in the distributed environment where MAB*flow* agents might go to in order to execute part of a workflow there. From a functional point of view, an agency represents the functionality for supporting the life cycle of mobile agents (mobile agents creation, execution, transport, etc.), and functionality specific for supporting the execution of MAB*flow* workflows. The functionality for supporting the life cycle of mobile agents is provided by a mobile agent system (for example, Voyager [11] or Aglets [14]).

The MAB*flow* internal servers provide services inside an administrative domain that support the execution of MAB*flow*. Examples of these servers are: an *account server*, which maintains information and control over MAB*flow* user accounts [2]; the *thin client server*, which provides access to the system from resource constrained devices, such as PDAs [1]; and the *disconnection server*, which gives support for mobile agent execution over devices that might become disconnected [4][1].

The *application service providers* are objects that provide services in the domain of the applications. An example of such a service is a flight reservation service. Each application service provider is associated with an agency, from which it can be accessed with a good quality of service (for example, by having the agency and the service provider residing at the same host or local area network). A service is provided by making *interfaces* of objects available to agents. Each interface is composed by a set of methods, which agents can invoke to access the service. A flight reservation service, for example, can be provided by an object with an interface containing methods for specifying flight data and obtaining ticket prices for that flight.

Workflows might involve interactions with human users. For example, in a loan request workflow, in a bank scenario, the loan request might only be granted if the manager approves it. In order to enable interaction with human users, some agencies are associated with users. The agency associated with a user will contain that user's *worklist*, which is a repository of the agents that arrive at the agency to interact with the user. When an agent has a part of an workflow to be executed by a human user, it goes to that user's agency. When it arrives there, it registers itself with the user's worklist. With the aid of a graphical tool provided by MAB*flow*, the user will

[1]These services are instantiations of services provided by the GMAS system [2]

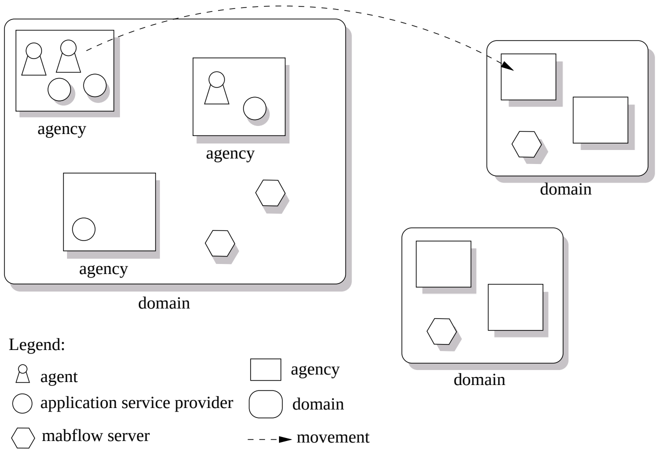

Figure 1: The MAB*flow* distributed environment

be able to manipulate his/her worklist, for example, to choose an agent to interact with (i.e., to select a workflow task).

2.2 The MAB*flow* Process Model

In this section we describe the internal structure of workflows as currently supported by MAB*flow*.

The basic component of a workflow in MAB*flow* is a task. In general terms, a task corresponds to a certain step of the workflow that is to be executed at a certain location (agency) in the distributed environment.

A workflow contains a set of tasks. Each task can be either a *generic application* task or a *subworkflow* task. A generic application task models a direct interaction with an application service provider (or a human user) through an interface. A subworkflow task models the invocation of another workflow. Since each subworkflow might have by its turn the invocation of other workflows, the MAB*flow* workflow model is a nested model.

Each task has the name of an agency associated with it. In order to execute that task, the agent executing the workflow must be at that agency. The specification of a generic application task involves defining a method of an object interface accessible from the agency associated with it and the parameters to be used during that method invocation. The specification of a subworkflow task involves specifying which workflow should be invoked and with which parameters.

A task has associated with it also a set of antecedent and successor tasks, as well as a set of execution conditions. A task is *executable* when all its corresponding antecedent tasks have been executed and if its execution conditions are evaluated true. During runtime, a scheduler executes tasks as they become executable. If more than a task is executable, one of them is elected to run.

Although MAB*flow* supports flexibility in the specification of control flows, some control flow patterns are not supported. For informations on the types of workflows not supported, the reader is referred to [12].

Data items in MAB*flow* are defined as global data. The data flow is specified by using expressions involving output parameters or global data as input parameters of tasks (generic application or subworkflow tasks).

The user of a workflow is usually interested in the execution state of a workflow executing on his/her behalf. In order to support this requirement, *checkpoints* are generated during the execution of a workflow. Checkpoints are always generated in certain points during a workflow execution (for example, before parallel tasks begin to execute). A workflow specifier, however, can force the system to generate a checkpoint after any desired task.

2.3 Executing Workflows with Mobile Agents in MAB*flow*

The execution of workflows with mobile agents in MAB*flow* takes place as follows. A mobile agent is created to start executing the workflow. When the control flow reaches a task to be executed, the agent moves to the agency associated with that task. When the agent arrives at the destination, if it is a generic application task, the agent will invoke the specified method of the specified object with the given parameters. If the task to be executed is a subworkflow task, a new agent (a child agent) is created to execute the subworkflow. If more than one subworkflow are to be executed in parallel, the agent launches a child agent to each of them. When an agent creates child agents, it remains in the current agency until all of child agents have completed their executions. When that happens, the parent agent receives the results of the executions of its child agents, combines them, updates its internal state accordingly, and continues its execution.

3 UML and Activity Diagrams

To fulfill the needs for designing and modelling systems with object orientation, Booch and Jacobson created the OOSE (Object-Oriented Software Engineering) methodology and Rumbaugh created the OMT (Object Modeling Technique) methodology. In order to establish a standard that could be used by object-oriented system developers, Booch, Jacobson and Rumbaugh combined the advantages of both methodologies and created together the UML (Unified Modelling Language) [3].

UML is a graphical language for visualizing, specifying, constructing and documenting object-oriented systems and provides a set of diagrams and notations for that end. There are nine diagrams for modelling systems in UML: class, object, use case, sequence, collaboration, statechart, activity, component and deployment diagrams. Each diagram is used to model a specific aspect of a system. For example, class diagrams are used to specify the classes, interfaces, and their relationships. The activity diagram is used to specify the flow from activity to activity within a system. This type of diagram can be used, thus, to specify workflows.

Although a complete modelling of a workflow using UML involves typically the use of more than a type of diagram, we are currently supporting only the activity diagram in MAB*flow*. Therefore, we will concentrate only on activity diagrams in this section. After introducing briefly activity diagrams in the next subsection, we introduce briefly in section 3.2 two extensibility mechanisms of UML that are used when specifying MAB*flow* workflows, *tagged values* and *stereotypes*.

3.1 Activity Diagrams

An activity diagram is one of the UML diagrams used to model the dynamic aspects of a system. An activity diagram is basically a flowchart that models the steps and flow of control of processes. An activity diagram has a name and the following main components: *activity states*, *action states*, *transitions*, *branching*, *synchronization bars (forking and joining)*, *swimlanes*, and *object flow*. Each of these components will be briefly described below:

Activity States and Action States: Activity and action states are used to represent actions (steps) in the execution of a process. An action state represents an atomic, non decomposable computation. Activity states can be further decomposed, and they are not atomic, meaning that they can be interrupted. Both an activity and an action state is represented with a symbol with horizontal top and bottom and convex sides (see Figure 2(a)). Below the horizontal line a set of actions can be specified. Each action might have associated with it a modifier. In Figure 2 the modifier *"do/"* appears, which simply introduces an action to be performed (without special handling, as can be specified with other modifiers).

Transitions: Transitions are used to represent the flow of control between action and activity states. Transitions are represented graphically by arrows (see Figure 2(b)). Transitions can have conditions associated with them (*guard conditions*), meaning that the transition can only happen if the condition is evaluated as true.

Branching: Branches are used to specify alternative paths, depending on the evaluation of associated boolean expressions. A branch is represented by a diamond symbol (see Figure 2(c)). A decision must have an input transition and two or more output transitions. A boolean expression is associated with each output transition, indicating the condition under which that output transition is to be followed.

Synchronization Bars (forking and joining): Synchronization bars are used to specify the forking and joining of parallel flows of control. *Forking* represents the division of a control flow into two or more concurrent control flows. It must have an input and two or more output transitions. *Joining* reprsents the synchronization of two or more concurrent control flows and must have two or more input transitions and a single output transition. The graphical representation of forking and joining is shown on Figure 2(d).

Swimlanes: Swimlanes are used to group together action and activity states. Each swimlane is identified by a name and represents an entity related to or responsible for the group of actions and activity states associated with it. The graphical representation of a swimlane is shown on Figure 2(e). The swimlane in this figure has two activity states associated with it.

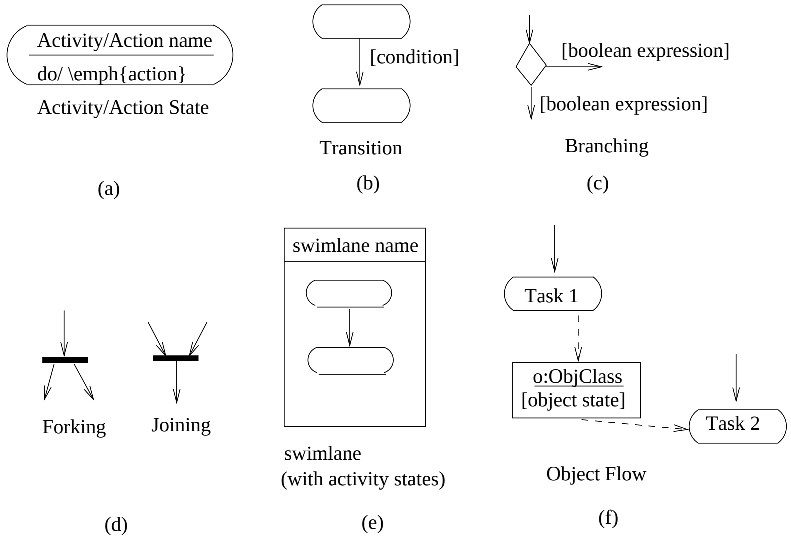

Figure 2: Some symbols of UML activity diagrams

Object Flow: Objects can be involved in the control flow associated with an activity diagram. For example, objects can be produced by certain activities and modified by others. Such objects involved in an activity diagram can be specified by placing them in the diagram and connecting them with the activity or transition that operates on them (by using so-called *dependency relationships* [3]). Graphically, an object is represented with a square (see Figure 2(f)). Additionally, attributes of the object might also be represented.

3.2 UML Extensibility Mechanisms: stereotypes and tagged values

Extensibility mechanisms make it possible to extend UML in a controlled way. These mechanisms include *stereotypes* and *tagged values*. Stereotypes extend the UML vocabulary making it possible to create new types of building blocks that derive from existing ones, but which are specific to a certain problem. Graphically, a stereotype is represented with a name between the symbols "«" and "»" and it is positioned above the name of another element.

A tagged value extends the properties of existing UML building blocks, making it possible to create new informations in the specification of the elements. The graphical representation of a tagged value is a string of caracters, between brackets, positioned below the name of another element. There is a standard tagged value called *documentation*, which applies to all elements. This tagged value specifies a comment, description or explanation of the element to which it is related.

4 Modelling Mobile Agent-based Workflows with UML

We have defined some conventions that need to be followed when modelling workflow components with UML, so that code can be automatically generated.

To make the explanation easier, the example illustrated in Figure 3 will be used. In this simple example, we model a typical travel arrangement scenario. Three agencies are directly involved, each represented by a swimlane (their URLs are shown as swimlane names). The agency *secretary.example.com* represents the agency of some company's secretary which is responsible for arranging travel plans for the company's employees. The URL *manager.example.com* represents the agency of the person at the company who is in charge of approving travel requests (one of the company's managers). The URL *210.17.143.88:9000* represents an agency at the company from which a database used to store travel funding requests can be locally accessed.

The workflow involves the following tasks. *GetTravelInfo* represents a task which will interact with the secretary (by means of a graphical interface) to get the travel information. After *GetTravelInfo* is finished, two subworkflows are started: subworkflow *ReserveHotelRoom*, which books a hotel room, and subworkflow *ReserveACar*, which reserves a car. These subworkflows might involve a set of subtasks for accomplishing their respective goals (these sub-

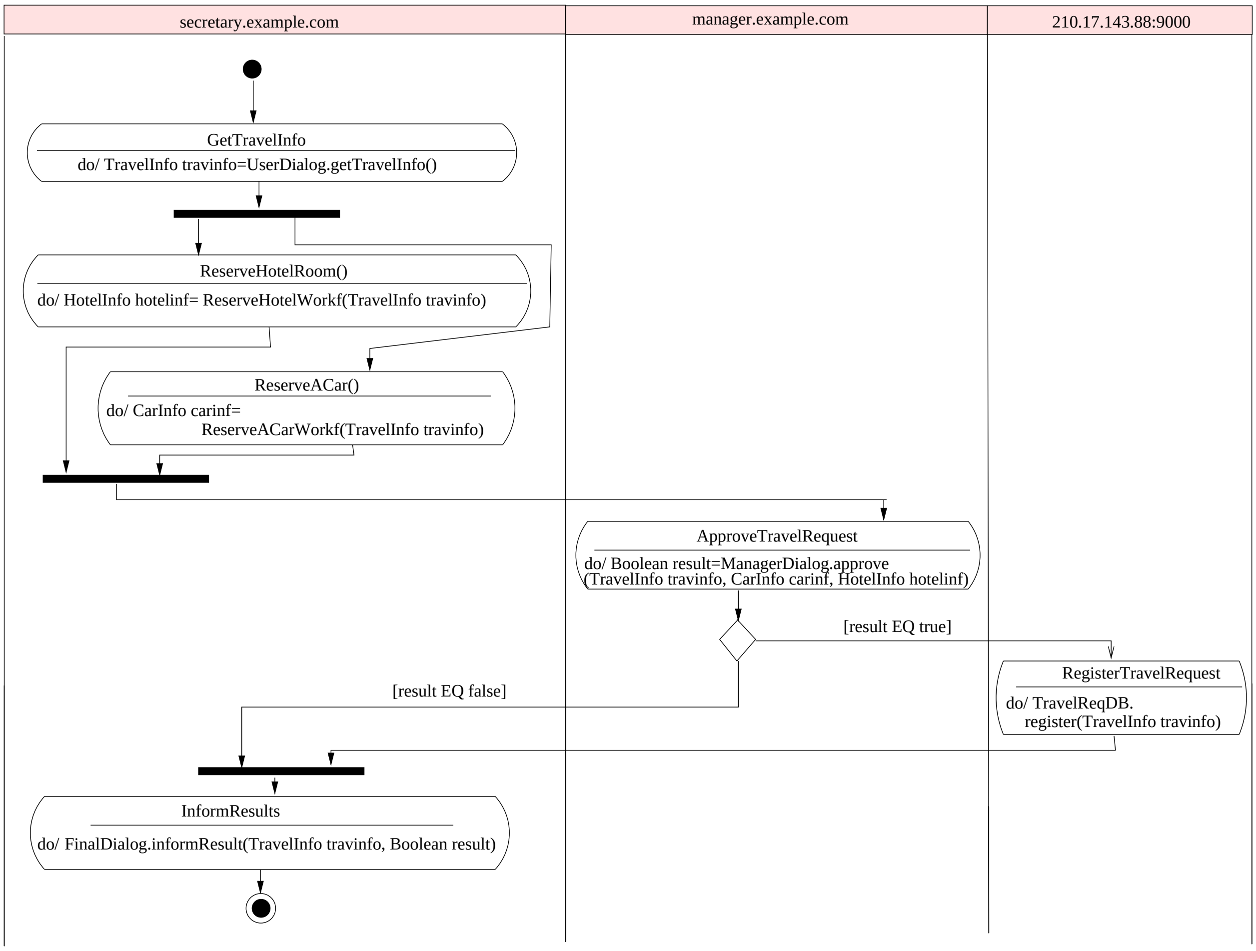

Figure 3: An example MAB*flow* workflow specified in UML

workflows are defined at other activity diagrams). After there subworkflows are finished, task *ApproveTravelRequest* must be executed. This task represents the approval of the travel request by the manager. If the request is approved, task *RegisterTravelRequest* stores a register describing the travel funding request in a database. At the end, task *InformResults* represents the interaction with the secretary to inform the results of the travel request.

According to section 2.3, this workflow would be executed by first creating a mobile agent. Since the first task is associated with the swimlane whose name is the URL of the secretary's agency, the mobile agent will migrate to that agency to execute task *GetTravelInfo*. After that task is executed, two child agents will be created, one to execute task *ReserveHotelRoom* and the other to execute task *ReserveACar*. Subworkflows are always executed by child agents. After the child agents bring the results to their parent agent, the parent agent moves to agency *manager.example.com* to execute task *ApproveTravelRequest*. After this task is carried out, the agent moves either to agency *210.17.143.88:9000* to register the travel request information in the database (task *RegisterTravelRequest*) and then to the secretary's agency (task *InformResults*) or directly to the latter, depending on if the request was approved or not. Observe that the positioning of a task in a particular swimlane defines the migration pattern of the mobile agents executing the workflow.

In the next sections we describe in more details how each component of the MAB*flow* process model is modelled in UML. We will refer to Figure 3 again to illustrate aspects of the modelling.

4.1 Modelling Workflow Meta Information

In MAB*flow* a workflow is composed by meta information (name, version identifier and additional comments) and a set of one or more tasks. The workflow metainformation is modelled using UML in the following way:

- name: The name of the workflow is represented with the name of the activity graph;
- version: The version of the workflow is represented with a stereotype of the activity graph;
- comments: Comments are represented using the documentation tagged value of the activity graph.

4.2 Modelling Tasks

A task in MAB*flow* has the following attributes: metainformation (name, version, additional comments); an identification of the agency where an agent must be when executing that task (server); an attribute indicating if checkpointing must be performed after the task is carried out (*checkpoint* attibute); a set of conditions indicating when the task can be executed (*execution conditions*); a set of antecedent and successor tasks; and the specification of a specific action to be carried out (either a generic application or the spawning of a subworkflow).

Tasks are modelled by using activity states. The task attributes are modelled in UML as follows:

- name: the name of the task is the name given to the activity state;
- version: the version is modelled using a stereotype of activity state;
- comments: represented by the documentation tagged value asssociated with the activity state;
- server: the specification of the agency where an agent must be when executing the task is done by using the names of the swimlanes. In the current version of the system, the name of the swimlane must be an URL (*Unified Resource Locator*) of an agency. That name will be the agency associated with all tasks in that swimlane;
- checkpointing: the need for doing checkpointing is specified by using an action with name *checkpoint* (*do/ checkpoint*);
- execution condition: an execution condition for a task is composed by a name, identifying a variable to be tested, a comparison operator and a name or literal and is represented by a guard condition of the transition. If there is more than one condition associated with a transition, they must be separated by commas;
- antecedents and successor tasks: the antecedents and successors of a task are modelled by using transitions between the tasks;
- actions: actions (generic application or the spawning of a subworkflow) are specified as described right below.

Although it is possible for an activity state to have many actions, in our UML workflow modelling each task must have only one action specification (either a generic application or the spawning of a subworkflow). Generic applications have a name, the specification of a method of a Java class to be invoked, a set of parameters to be passed during method invocation, and a return parameter. A subworkflow has a name and parameters.

Gereric applications and subworkflows are specified as actions of activity states. Subworkflows are identified, however, with the use of parentheses after the name of the activity. For example, in Figure 3 the task *GetTravelInfo* is represented as a generic application. In the actions part (below the line), the name of a Java class is specified (*UserDialog*) and a method of this class (*getTravelInfo*). This task is associated with the swimlane with name *secretary.example.com*. When this task is to be executed, the agent will migrate to the agency *secretary.example.com*, and will invoke the method *getTravelInfo* of an object of the class *UserDialog* made available at that agency. The method returns an object of type *TravelInfo*. The Subworkflow *ReserveHotelRoom* is also associated with the swimlane with name *secretary.example.com*. The modelling of this task includes the name of a subworkflow (*ReserveHotelWorkf()*), parameters (*travInfo*) and a return value of type *HotelInfo*. Observe that data flow through the parameters and return values of tasks. Observe additionally that the modelling of methods, classes and types follows the conventions of Java, since it is the programming language used in the underlying systems (GMAS and mobile agent systems).

The attributes of a generic application are modelled in UML as follows:

- name: the name of the generic application is the name of the task;
- method: the name of the class and the method to be invoked is represented in the action name;
- parameter: if there are parameters, these are represented after the name of the method, separated by commas.

The attributes of a subworkflow are modelled in UML as follows:

- name: it is specified as the activity name, but followed by parentheses;
- parameters: parameters are represented in the same way as for generic applications (see next section).

4.3 Modelling Parameters

An input or input/output parameter for a generic application or a subworkflow has a name, a type, an attribute indicating if it is an input, output or input/output parameter (*direction*) and an optional initial value. A parameter is modelled in UML as follows:

- name: the name of the parameter, as given by the UML specifier;
- type: the type of the parameter must be specified before the parameter name. The type is a valid Java class representing that type;
- direction of attribute (input/output): if the value of a parameter can be changed by the method, the word *out* must be specified before the type of the parameter. Otherwise, no word needs to come before the parameter type;
- initial value: if the parameter has an initial value, this value must be specified after the parameter name, preceded by an equal sign ('=') and by a literal.

The attributes of a return parameter are modelled in UML as follows:

- type: it is represented at the beginning of the name of the action, right after the *do/* string;
- name: it follows the description of the type and is followed by an equal sign ('=').

5 Generating Executable Code from UML Specifications

The process of generating executable code from UML specifications is represented in Figure 4.

The automatic code generation relies on XMI (*XML Metadata Interchange*). XMI is an OMG (*Object Management Group, Inc.*) standard for exchanging data based on XML (*Extended Markup Language*). The main goal of XMI is to serve as a standard to make possible the exchange of metadata between UML tools, allowing systems designers and modellers to share object oriented models and metadata. OMG developed a DTD (*Document Type Definition*) that represents the metamodel of UML 1.3. generated from XMI 1.0

Therefore, in MAB*flow* a workflow specifier can use any tool to write his/her workflow specification that supports the XMI standard (in this project, we have used the Rational Rose 2002 tool, with the UnisysRoseXMLTools, which exports UML specifications to XMI).

After having done the UML specification, a corresponding XML representation of the workflow in

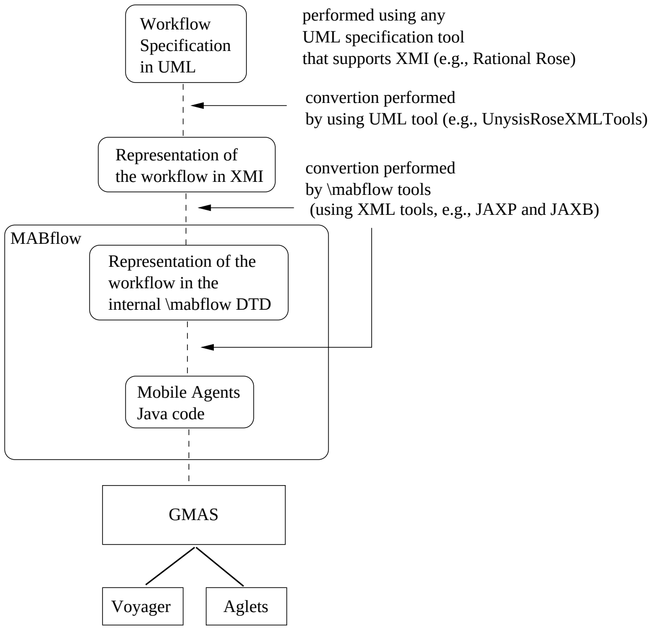

Figure 4: Automatic code generation in MAB*flow*

the XMI format is generated (using any tool desired by the workflow specifier). Then MAB*flow* tools are used to generate a corresponding XML specification of the workflow, according to an internal MAB*flow* DTD. This DTD was defined in MAB*flow* [12] to model the components of the MAB*flow* process model (see section 2.2).

From this XML representation of the workflow, mobile agent code in java is generated. Instead of generating code to a specific mobile agent system, the generated code uses only the classes, interfaces and objects provided by GMAS (see Figure 4). The GMAS is a system being developed at LaSiD that provides a set of classes, interfaces and objects that provide generic functionality needed by mobile agent applications. The functionality provided by GMAS is implemented by using specific mobile agent systems. By making the applications to see only GMAS and not directly the interfaces of mobile agent systems, we achieve independence of the development of applications from specific mobile agent systems. GMAS provides also functionality not present in many mobile agent systems that are frequently needed by applications (such as components for the management of users accounts and customizable user and management interfaces). This way, GMAS serves also as a generic customizable framework upon which mobile agent applications can be developed [2]. In Figure 4 mappings from GMAS to two mobile agent systems are shown, Voyager [11] and Aglets [14]. Mappings from GMAS to other mobile agent system can be written. Currently there exists only the mapping to Voyager.

Among the functionality provided by GMAS are the support for disconnected operation (execution of mobile agent applications on mobile devices), support for executing mobile agent control from resource constrained devices (such as PDA's and cellular phones), user account management, and customizable user interfaces.

Beyond using Rational Rose 2000 with UnysisRoseXMLTools and Voyager, the JAXP (Java API for XML Processing), and JAXB tools were used for parsing and processing XML documents [22].

6 Related Work

For the best of our knowledge, the first description of a system using mobile agents to execute workflows is [5]. After that, many other systems were developed (e.g., [17, 21, 13, 16]). The authors of these works addressed different problems, such as the construction of prototypes to show the advantages of using mobile agents for the execution of workflows, the study of architectures for such systems [17, 13, 5] and the analysis of task scheduling [21].

More closely related to ours is the work on generating mobile agent code for executing workflows automatically from XML or UML specifications (e.g., [23, 13, 16]). These works aim at providing suitable tools for managing the complexity of developing mobile agent-based workflows. Our objectives are very similar to the objectives of these projects. Currently our system supports a simple way of specifying mobile agent-based workflows with UML, but it represents a first approach to the problem. Our main midterm goal is to integrate the MAB*flow* system with the MARES platform. MARES is a platform being developed at LaSiD to support the development of transactional and fault tolerant mobile agent-based applications [20]. Integrating MAB*flow* with MARES and using the functionality of GMAS, we intend to achieve a transactional and fault-tolerant mobile agent-based workflow management system that is able to execute workflows over a widely distributed and heterogeneous environment, including support for resource constrained devices and disconnected operation.

7 Conclusion

This paper described the MAB*flow* system and concentrated on the specification of workflows using UML and on the automatic mobile agent code generation. In particular, specific conventions for modelling components of the MAB*flow* process model in UML were defined. The system relies on the XMI standard and uses XML parsing tools to generate a corresponding XML document that follows the structure of an internal MAB*flow* DTD. Finally, from this document, Java code is generated. The system is based on a system called GMAS, which makes mobile agent-based applications independent of specific mobile agent systems and which provides additional functionality not commonly provided by existing mobile agent systems. With that, any UML tool that supports the XMI standard can be used to specify MAB*flow* workflows (i.e., workflows need not be specified in a specific MAB*flow* workflow specification tool).

The paper addresses questions related to the process of developing mobile agent-based applications. Although many advances were done by the research community so far concerning infrastructural mechanisms for supporting mobile agent-based applications (such as communication mechanisms, fault tolerance approaches, approaches for security, etc.), little effort was dedicated for decreasing the complexity involved in the process of developing mobile agent-based applications. The integration of the development of such applications with a software engineering methodology of practical relevance is a step towards that goal. We consider that the results achieved with the development of the system described in this paper contribute to understand how this integration could be carried out.

Our system, as it currently is, provides a very simple solution yet, which certainly does not provide the flexibility needed for complex applications. We are currently extending the system functionality to make it more flexible and to address a broader spectrum of real application requirements. Among our ongoing work are the extension of the internal MAB*flow* DTD, the introduction of support for the definition of ontologies, and the integration of the system with the MARES platform.

References

[1] Adalberto Alves Abraão and Florisberto da Silva Santos. *Implementação de uma inteface de gerenciamento de agentes móveis sobre dispositivos móveis* (in portuguese), Monography of the Advanced Specialization Course on Distributed Systems, LaSiD - Distributed Systems Laboratory, UFBA - Federal University of Bahia, 2002.

[2] Siddhartha Lourenço Argollo. *GMAS: Um framework genérico para o desenvolvimento de aplicações baseadas em agentes móveis (in portuguese)* Monography of the Advanced Specialization Course on Distributed Systems, LaSiD - Distributed Systems Laboratory, UFBA - Federal University of Bahia, 2002.

[3] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[4] Tiago Edmundo Ribeiro Brito. *Suporte à operação desconectada no GMAS, um framework para o desenvolvimento de aplicações baseadas em agentes móveis*, Monography, Computer Science Department, Federal University of Bahia, 2003.

[5] Ting Cai, Peter A. Gloor, and Saurab Nog. Dartflow: A workflow management system on the web using transportable agents. Technical Report PCS-TR96-283, Dartmouth College, Computer Science, Hanover, NH, May 1996.

[6] Flávio Morais de Assis Silva and Sven Krause. A distributed transaction model based on mobile agents. *Lecture Notes in Computer Science*, 1219:198–209, 1997.

[7] Flávio Morais de Assis Silva and Radu Popescu-Zeletin. An approach for providing mobile agent fault tolerance. *Lecture Notes in Computer Science*, 1477:14–25, 1998.

[8] Flávio Morais de Assis Silva and Radu Popescu-Zeletin. Mobile agent-based transactions in open environments. *IEICE Transactions on Communications - IEICE/IEEE Joint Special Issue on Autonomous Decentralized Systems*, E83-B:973–987, 2000.

[9] D.Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. In A.K.Elmagarmid, editor, *Distributed and Parallel Databases*, volume 3, pages 119–153. Kluwer Academic Publishers, April 1995.

[10] Neima Prado dos Santos and Jorge Antônio Lima Santos. *Execução de workflows baseados em agentes móveis a partir de especificações em UML*, Monography of the Advanced Specialization Course on Distributed Systems, LaSiD - Distributed Systems Laboratory, UFBA - Federal University of Bahia, 2002.

[11] ObjectSpace Inc. Voyager technical overview, 1997.

[12] Alfrânio Tavares Correia Jr., Flávio Morais de Assis Silva, João Carlos Lago Selch, and Sergio Eduardo Barreto da Luz. Mabflow (mobile agent based workflow) 1.0 - manual de instalacão e utilizac ao - versão 1.0. Technical Report RI-001-04, LaSiD - Distributed Systems Laboratory, UFBA - Federal University of Bahia, 2000.

[13] Mikhail Kazakov, Habib Abdulrab, and Gilles Debarbouille. A model-driven approach for the design of mobile agent systems for concurrent engineering: MADE4CE project. *Izvestia Inzhenernih Nauk - to appear*, 2003.

[14] Danny B. Lange and Daniel T. Chang. IBM aglets workbench - programming mobile agents in java - a white paper, September 1996.

[15] Raimundo J.A. Macêdo and Flávio M. Assis Silva. Coordination of mobile processes with mobile groups. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks, DSN2002*, pages 177–186, Washington, DC, June 2002.

[16] M.B.Blake. Agent-based workflow configuration and management of on-line services. In *Proceedings of the International Conference on Electronic Commerce Research (ICECR-4)*, pages 567–588, Dallas, TX, November 2001.

[17] M.Merz, B.Liberman, and W.Lamersdorf. Using mobile agents to support interorganizational workflow management. *International Journal on Applied Artificial Intelligence*, 11(6), September 1997.

[18] OMG. XML metadata interchange (XMI) specification - version 1.2, January 2002.

[19] Flávio M. Assis Silva and Raimundo J.A. Macêdo. Reliability requirements in mobile agent systems. In *Proceedings of the Second Workshop on Tests and Fault Tolerance, IIWTF 2000*, Curitiba, Brazil, July 2000.

[20] Flávio M. Assis Silva, Raimundo J.A. Macêdo, and Ana Vitoria P. Freitas. The mares platform: Support for transactional and fault-tolerant execution of mobile agent-based applications. In *Proceedings of the Fourth Workshop on Tests and Fault Tolerance, IVWTF 2003, co-located with the XXI Brazilian Symposium on Computer Networks, SBRC2003*, Natal, RN, May 2003.

[21] Henrik Stormer. Task scheduling in agent-based workflows. In *Proceedings of the International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA)*, Wollongong, Australia, 2000.

[22] Sun Microsystems, Inc. Web services made easier. the java apis and architectures for xml - a technical white paper, October 2001.

[23] Anand Tripathi, Tanvir Ahmed, Vineet Kakani, and Shremattie Jaman. Implementing distributed workflow systems from xml specifications. Technical report, Department of Computer Science, University of Minnesota, May 2000.