An Extension of ArgoUML for the Incorporation of Profiles

Jane Pryor, Edgardo Belloni, Claudia Marcos

ISISTAN Research Institute, Facultad de Ciencias Exactas, UNICEN
Paraje Arroyo Seco, B7001BBO Tandil, Argentina

Email: [jpryor, ebelloni, cmarcos]@exa.unicen.edu.ar

Tel/Fax: + 54–2293–440362/3 http://www.exa.unicen.edu.ar/~isistan/

Abstract. The Unified Modeling Language (UML) is a language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. UML supports the most typical software modeling problems; however, due to the diversity of software development domains, there may be occasions when the model requires elements or notations not provided by standard UML. This paper presents an extension to UML by defining new profiles for different application domains: one for agent-oriented systems and another for aspect-oriented development. The ArgoUML tool has been extended to support the definition of new profiles for the modeling of different application domains, including those mentioned.

.**Keywords**: UML profiles, UML tools, aspect-oriented applications, mobile-agent systems.

Workshop: Ingeniería de Software y Base de Datos (WISBD)

Tema: Ingeniería de Software

1. Introduction

The UML (Unified Modeling Language) [RJB99] is a language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. These artifacts can be seen from two different perspectives: the UML definitions themselves, and how they are used to build software applications. The first perspective refers to the formal specification of UML, that is, the semantics, the notation, etc. of each of its elements. The second perspective refers to the fact that the designer's choice of models and diagrams influences how the problem is approached and how the corresponding solution is shown.

The vocabulary of the UML consists of three kinds of building blocks: things, relationships, and diagrams. Things are the abstractions that are first-class citizens in a model; relationships tie these elements together; diagrams group interesting collections of things. Diagrams are the graphical representation of a set of elements, and they are used to visualize a system from different perspectives.

The elements that make up the UML diagrams have a well-defined semantics that is represented by a meta-model. Each element provided by UML has its corresponding class in the meta-model.

A UML diagram, such as a class diagram, does not usually contain sufficient information in order to describe all the aspects that are relevant to its specification. It is therefore necessary to describe additional restrictions with respect to the objects in the model. In order to express unambiguous restrictions, OCL has been developed. OCL - Object Constraint Language - is a formal language that consists of expressions, easy to read and write, and which has no collateral or secondary effects on the model.

UML supports the typical software modeling problems. However, due to the diversity of software development domains, there may be occasions when the model will require elements or notations not provided by standard UML. Quite frequently, users may also need to add non-semantic information to their models. These needs may be satisfied by UML through the extension mechanisms the language provides. The user may add new types of modeling elements with different semantics, characteristics, and notation, to those specified in the metamodel. On the other hand, new information may be added to the existent UML modeling elements.

The extension mechanisms provided by UML are *constraints*, *stereotypes*, *tag definitions*, and *tagged values*. When an element is stereotyped, UML is actually being extended by creating a new building block just like an existing one, but with its own special properties (each stereotype may provide its own set of tagged values), semantics (each stereotype may provide its own constraints), and notation (each stereotype may provide its own icon). These extension mechanisms may be used in an isolated fashion or in coherent sets that have been defined for a purpose or specific domain. A *Profile* is a packet stereotype that contains one or more (related) extensions to the standard semantics of UML. Profiles may contain stereotypes, tag definitions and restrictions; they may also contain data types to be used by the tag definitions.

A variety of new technologies and approaches for the development of software are not currently supported by UML diagrams. Additionally, the existent tools do not provide flexible and simple mechanisms that permit the incorporation of new modeling elements and the handling of profiles. Also, most of these tools are not open source.

ArgoUML [ArgoUML] is an open-source tool that we have extended in order to support the definition of new UML profiles. By means of this extension, new profiles may be defined to support many different types of applications.

This paper describes how multi-agent systems and aspect-oriented applications can be supported by the UML notation. The new UML elements needed for these applications are described by a profile. The ArgoUML tool has been extended so as to support the definition of new UML profiles, and also the dynamic updating of the metamodel.

Sections 2 and 3 introduce the concepts of multi-agent systems and aspect-oriented software development, respectively. Section 4 describes the main characteristics of the extension of the ArgoUML tool. Section 5 presents the profile corresponding to the mobile-agent systems and section 6 the profile for aspect-oriented development. The remaining section presents the conclusions.

1. Multi-Agent Systems

An agent is a software component allocated in a run-time environment to assist or represent someone or something, and it acts by itself with some degree of autonomy [Bra97; Jen98; Nwa96]. Other than this autonomy, there are several additional attributes by which it is possible to characterize a software agent, such as reactivity, pro-activity, social ability, adaptability, and mobility.

An agent-based application can have only one agent, but frequently these applications have several agents interacting and collaborating among themselves, becoming a multi-agent system. In this context, it is possible to identify a multi-agent system as having at least two different kinds of agents, each of them belonging to one of the above five agent categories. Moreover, hybrid agents can also be involved in this kind of system.

In the last years, the software development based on mobile agents has received widespread attention because it introduces a new development paradigm for widely distributed and heterogeneous systems. Certainly, agent mobility [Whi97; FPV98; KT98] – i.e., the software agents' capability of moving across the Internet while they are acting on behalf of a user – is being recognized as one of the most promising features for the development of Internet applications. The mobility of an agent involves the transfer of its code, data, and possibly its execution state, towards the environments or sites where the resources it needs to access, or where other agents it needs to meet, are allocated. Such capability is particularly interesting when an agent makes sporadic use of a valuable shared resource. Efficiency, for instance, can be improved by moving agents performing queries over a large database to the host of the database itself. Additionally, response time and availability would improve when performing interactions over network links subject to long delays or interruptions of service.

2. Aspect-Oriented Applications

Aspect-oriented software development [AOSD02] is a new and well-known approach to the separation of concerns in software engineering [Dij76]. AOSD emerged in order to provide explicit support for modularizing the design decisions that overlap or crosscut the functional decomposition of a system. As these features crosscut the primary functionality of the application, their code is spread throughout the basic functional components. In AOSD, crosscutting concerns are encapsulated in separate modules called aspects, whose code is woven into the functional components of the system at predetermined point-cuts. Consequently, AOSD improves software quality attributes such as maintainability, legibility and reuse.

3. ArgoUML for the Support of Profiles

In order to extend UML with the two above-mentioned concepts with profiles and any other profiles that may be defined, it would be necessary to build a tool that supports the UML meta-model and the handling of profiles. This would allow developers of software for non-traditional domains or processes to extend the UML meta-model according to their needs, creating profiles by configuring new ones or extending and composing existent ones.

ArgoUML is an open source tool, implemented in Java and based on the UML metamodel. This tool was extended in order to permit the definition, configuration, composition, and administration of UML profiles, therefore supporting the modeling of software developed using novel approaches or techniques, or for non-traditional domains.

The editing environment provides the following features:

- The definition of profiles of a particular domain for modeling applications, e.g., profiles for modeling agent and aspect-oriented software.
- The incorporation of new modeling elements to a new or existent profile semantically defined by the UML meta-model.
- The support of a visual mechanism for the specification of new elements in OCL, as a complement to the meta-model description.
- The definition of new profiles or the composition of existent ones in order to satisfy the modeling requirements of a new application domain.

Therefore, the extended tool provides, on the one hand, the concepts, characteristics, and elements of the UML diagrams, for use in typical development domains. On the other hand, it offers the necessary mechanisms for the incorporation of new elements to the diagrams provided by UML, and for the definition of new diagrams with their semantics, to be used in different application domains. The tool also supports the formal documentation of these new elements, such that the metamodel is updated and maintains its consistency and so the tool is dynamically updated without the need to modify its code.

In order to model concepts and technologies that are not supported by UML, the developer must study the modeling requirements of these different concepts, so as to identify what UML is lacking. The developer must then define if and how these modeling requirements will be supported

by new concepts to be added to the diagrams provided by UML, or by defining new diagrams. And lastly, s/he may, using the CASE tool, build, document and visualize the application.

4. Mobile-Agent Systems

Typically, an agent-based application involves several agents that interact and communicate between themselves, playing different roles. A mobile-agent application additionally involves mobile agents and a number of computational environments – a.k.a. mobile-agent systems or servers – where agent execution can take place and where different services and resources are provided.

With respect to mobility – i.e., the agent's ability to move around some computer network, and a.k.a. migration or navigation – there are two kinds of agents: mobile and ***stationary*** (static or non-mobile) ***agents***. In this context, ***mobile agents*** are active entities that can move from a mobile-agent system to another one, in order to meet other agents or to access to services and resources provided there, while they are acting on behalf of someone or something. On the other hand, ***resources*** represent non-autonomous entities, such as files, objects, databases, programs or external applications, which can be used and shared by several mobile agents.

A ***mobile-agent system*** is primarily responsible for executing agent code through protected execution environments – a.k.a. execution places. A mobile-agent system is also responsible for providing other support features, such as agent persistence, security, and primitive operations to agent programmers, e.g., those that allow agents to migrate, communicate among them, access local resources, etc. Moreover, modern mobile-agent systems provide standard interfaces for interoperability with other mobile-agent systems, accepting agents implemented in different programming languages.

The execution places (or simply ***places***) represent logical locations, provided by mobile-agent systems, where agents execute, meet and communicate with other agents, and manipulate some local resources. In general, agents, mobile-agent systems, places and resources are uniquely identified by a name or an electronic address.

One o more mobile-agent systems can exist in a node. A node represents a hardware infrastructure on which both agents and mobile-agents are actually (physically) executed. Typically, a node is a computer, but this term includes other computing devices, such as personal digital assistants (PDA) or mobile phones. Finally, a logical network of places materializes a ***region***. The places in the region are associated to the same authority, e.g., an organization or administrative domain. The region registers each agent that is currently hosted by a place provided by a mobile-agent system of the region. If an agent moves to another place, this information is updated.

MAM-UML Profile

Recently, we have been working on an approach to improve the standard UML concerning the specification of relevant analysis, design or implementation aspects of mobile agents [BM03]. This approach is materialized by a coherent and comprehensive set of views and models, which extends UML by contributing to the analysis and design phases of mobile-agent applications

development. This set represents a preliminary UML profile, named MAM-UML – it stands for Mobile-Agent Modeling with UML, that currently includes views to model organizational, life cycle, interaction and mobility aspects of mobile-agents applications. These MAM-UML views are summarized below.

The Organizational view describes the entities, and their structural relationships, involved in a mobile-agent application. This view addresses the static logical view of an application in the mobile agent paradigm.

The Life-cycle view describes the itineraries, states and activities of each kind of mobile agent, during its life cycle. The life-cycle view includes three models in order to specify that; a model of itinerary, a model of execution states, and a model of activities.

The Interaction view describes which, why, when and how agents, roles, and mobile-agents systems need to communicate, by specifying interactions among these entities, their protocols and coordination mechanisms, from temporal and spatial perspectives. This view complements the life-cycle view in order to address the dynamic view of a mobile-agent application.

The Mobility view includes models for the specification of code and execution state mobility, as well as, the mechanisms used for data space management. The model of code and execution state describes where, why and when an agent moves, as well as, what it is moved – i.e., code, data or execution state – during the migration. In general, this view addresses the static physical view of a mobile-agent application.

Following, because of space problem, the profile of one of these views is presented.

MAM-UML Profile: Stereotypes defined for the Organizational View			
Stereotype	**Base Class**	**Tagged Values / Constraints**	**Description**
Mobile agent	Active Class	Identifier: In order to be uniquely recognized. Location: The current location place where a mobile-agent is executing. Authority: Whom a mobile-agent represents. State: The current execution state of a mobile agent. Clone: It indicates whether a mobile-agent is a clone.	Specifies autonomous entities that can move (transferring its code, data, and possibly its execution state) from a mobile-agent system to another one, while they are acting on behalf of someone or something.
Stationary-agent	Active Class	Identifier Authority	Specifies autonomous entities that model static or non-mobile agents.
Role	Class		Specifies external features of an agent in a particular context.
Resource	Class	Identifier Location Clone Type = {transferable or not transferable} It determines whether a resource can be migrated over the network or not. ApplicationRequirement = {free or fixed}	Specifies non-autonomous entities, such as files or objects that can be used and shared by several (mobile or stationary) agents.

		According to applications requirements, transferable resource instances can be classified as *free* or *fixed*. The former can be migrated, while the latter are permanently associated with a particular location.	
Mobile-agent system	Package	Identifier Authority: Whom a mobile-agent system accepts as an officially legal authority. System type: It indicates whether the system is complaint to a standard interoperability interface, such as those defined by OMG or FIPA. Language: It specifies the programming language that is used by a mobile-agent system for the coding of mobile agents.	Specifies a package that contains model elements which specify a reusable architecture for a mobile-agent application. It acts as a dock station accepting agents and providing native resources for them.
Place	Package	Identifier Authority	Specifies protected execution environments, provided by mobile-agent systems, where agents execute, meet and communicate with other agents, and manipulate some local resources.
Region	Package	Identifier Authority	Represents a logical network of places provided by mobile-agent system.
Home	Dependency relationship	Constraint: {The source must be a mobile agent and the target must be a place}	Specifies the origin execution place of a mobile agent.
Acquaintance	Association relationship	Constraint: {The source and the target must be agents}	Specifies the existence of at least one interaction involving the entities concerned.

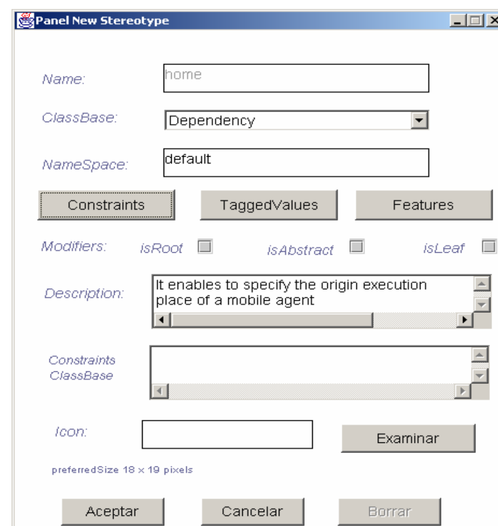

Figure 1. Profile definition with ArgoUML

5. Aspect-Oriented Software Development

A standard modeling notation for Aspect-Oriented Software Development would allow developers of these types of applications to use models that support all the concepts needed for aspect-oriented systems.

The current UML model does not provide the elements that are necessary to represent the different concepts involved in the analysis and design phases of AOSD. These elements include the aspects themselves, the point-cuts where the aspects actually cross-cut the other components, and attributes related to the actual correspondence between aspects and components. As it stands, UML users would have to use the actual code or textual documentation, to reflect and understand the aspect-oriented application.

AO-UML Profile

We have developed a visual environment for the specification and construction of aspect-oriented applications [PM03]. This visual environment leads the user through the tasks of specifying all the components involved in the development of this kind of system, and it automatically generates the code.

Through our experience with aspect-oriented developments, we have also extended the traditional aspect components and relationships in order to facilitate the adaptability and re-use of these applications.

The AO-UML profile that has been defined using ArgoUML has included all the terms necessary for the development of aspect-oriented applications. The following elements and relationships included in the AO profile will allow the user to accurately model aspect-oriented systems through the different development phases, and that will correspond to the implemented code.

An ***aspect-class***, which in our visual environment is specified as a metaobject class, is a class which encapsulates the state and behaviour of those components which cross-cut the functionality of the basic components of an application. Aspects usually implement concerns such as exception handling, synchronisation, performance optimisations, logging, tracing, and resource sharing. They traditionally pertain to what are known as the non-functional requirements of a system, though more recent aspect-oriented developments make use of this technology for the implementation of some functional requirements in order to improve an application´s flexibility.

A ***plane*** encapsulates a set of aspects that carry out a specific functionality, such as the logging of a system. Planes ease the handling of these groups of components, their reuse, their interaction with the rest of the system, their maintainability, the definition of conflicts, and so on.

Aspects cross-cut other components or aspects of a system. The point at which their behaviours interact is known as ***point-cuts*** (or join-points). There are different types of correspondence or associations between cross-cutting components, and different technologies or aspect-oriented environments support different types of associations and point-cuts.

We have identified four different strategies for associating aspects with base-level objects or other aspects. The type of association defines the ***point-cut***, i.e., where the thread of control at the base or functional level of an application is redirected to the aspect [Mar01]:

- ***Class association***, when all methods and objects of a specified class are associated to an aspect.
- ***Method association***, when only some and not all methods of a class are to be associated to an aspect.
- ***Object association***, when it is only necessary for some particular objects of a class to be associated to an aspect.
- ***Object-method association***, when only a particular method of a specified object of a class is associated to an aspect.

Additionally, the activation of the aspect can take place ***before*** and/or ***after*** the intercepted method. When defining an association, the designer specifies when the aspect is to be activated with respect to the intercepted method: only before, only after, or around (before and after). He/she may also indicate that the invocation of the original method be suppressed.

Components may be associated to more than one other component or aspect, with each of the latter carrying out a different function. ***Conflicts*** may arise if two or more aspects associated to the same object compete for activation. The runtime definition and handling of conflicts between components would control and avoid unpredictable or conflictive behaviour.

The actions to be taken when a conflict is detected depend on the characteristics of the application to be developed. In some cases it will be necessary to indicate a specific order of execution of the crosscutting concerns involved; in others, the detection of a conflict may require that one or both of the aspects not be activated. We have identified six different categories of activation policies to be implemented when a conflict is detected: ***inOrder***, ***reverseOrder***, ***optional***, ***exclusive***, ***null***, and ***context dependent***. ***InOrder*** is the type of conflict in which the aspects are activated in the order specified by the developer. ***ReverseOrder*** is when the aspects are activated in the reverse order to that established by the developer. ***Optional*** is for those cases when it may be necessary for the system itself to decide which aspect to activate. ***Exclusive*** is when only one particular conflicting aspect is to be executed. ***Null*** conflict is for those cases in which neither of the aspects is to be activated. Finally, ***Context Dependent*** is used when none of the above cases satisfy the desired solution to the conflict: the developer adds the code that specifies the activation of the aspects, which may include an evaluation of the current context of the system.

The following table summarizes the first approach to the proposed stereotypes and their characteristics.

AO-UML Profile: Stereotypes defined for the Aspect-Oriented Systems			
Stereotype	**Base Class**	**Tagged Values**	**Description**
Aspect-Class	Class		Specifies the orthogonal behavior for the application. Encapsulates the state and behavior of those components that cross-cut the functionality of the basic components of an application
Plane	Package		Groups aspects with the same goal but different strategies.
Point-cut	Association	Type: [class point-cut, method	Defines the point-cuts of the system; i.e. where the

		point-cut, object point-cut, object-method point-cut]	crosscutting takes place.
Before	Operation		The aspect component carries out its functionality before the application one
After	Operation		The aspect component carries out its functionality after the application one
Around	Operation		Part of the aspect component functionality is carried out before the application component, and the other part is developed after the application.
Conflict	Association	Category: [inOrder, reverseOrder, optional, exclusive, null, context dependent]	Specifies the type of conflict and its resolution, when two aspect classes compete for activation.

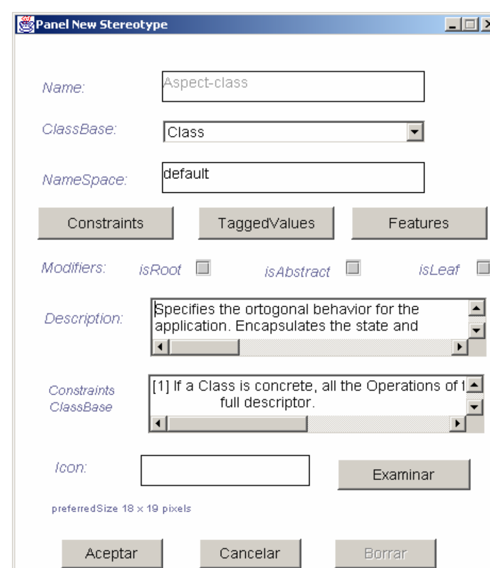

Figure 2. Aspect-class definition

6. Conclusions

This paper presents an extension to the Unified Modeling Language (UML) through the definition of new profiles, in order to support the use of this modeling language in different (non-typical) software application domains.

The modeling requirements for the development of multi-agent and aspect-oriented systems have been studied. UML has been extended with the corresponding elements and profiles to cover the different needs and characteristics of these domains.

The ArgoUML tool has been modified so as to support the definition of new UML profiles and the dynamic updating of the metamodel. The profiles for multi-agent and aspect-oriented software have been developed and added to the extended tool, which can then be used to model, visualize, build, and document any applications developed with these new technologies.

7. References

[AOSD02] 1st. International Conference on Aspect-Oriented Software Development. Enschede. Gregor Kiczales, ed., (ACM Press, The Netherlands, 2002).

[ArgoUML] At www.argouml.tigris.org

[BM03] Belloni, E. and Marcos, C. *Modeling of Mobile-Agent Applications with UML*. In Proceedings of the Fourth Argentine Symposium on Software Engineering (ASSE´2003). 32 JAIIO (Jornadas Argentinas de Informática e Investigación Operativa), Buenos Aires, Argentina. September 2003. ISSN 1666-1141, Volumen 32.

[Bra97] Bradshaw J. *An Introduction to Software Agents, in Software Agents*, Bradshaw, J.M. (ed.), Cambridge, MA: MIT Press, (1997) 3 – 46

[Dij76] Dijkstra E. *A Discipline of Programming*. (Prentice Hall, 1976).

[FPV98] Fuggetta A., Picco G. and Vigna G.: *Understanding Code Mobility*. IEEE Transactions on Software Engineering, Vol. 24, No. 5, pp. 342-361, 1998.

[Jen98] Jennings N., and Wooldridge M. *Applications of Intelligent Agents*. In "Agent Technology: Foundations, Applications, and Markets", Springer-Verlag, eds Nicholas R. Jennings and Michael J. Wooldridge - pages 3--28, (1998)

[KT98] Karnik N. and Tripathi A. R.: *Design Issues in Mobile Agent Programming Systems*. IEEE Concurrency, Vol. 6, No. 3, July-September 1998.

[LO98] Lange D. and Oshima M.: *Programming and Deploying Java Mobile Agents with Aglets*. Addisson-Wesley Longman, Reading Mass. 1998.

[Mar01] Marcos, C.: *Patrones de Diseño como Entidades de Primera Clase*. PhD. Degree Dissertation. Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN), Facultad de Ciencias Exactas, ISISTAN Research Institute, April 2001.

[Nwa96] Nwana H. *Software Agents: An Overview*. Knowledge Engineering Review, Vol. 11 No 2 - (1996) 205 – 244

[PM03] Pryor, J. & Marcos, C. *Solving Conflicts in Aspect-Oriented Applications*. In Proceedings of the Fourth Argentine Symposium on Software Engineering (ASSE´2003), 32 JAIIO (Jornadas Argentinas de Informática e Investigación Operativa), Buenos Aires, Argentina. September 2003.

[RJB99] Rumbaugh, J., Jacobson, I., and Booch G. *The Unified Modeling Language. Reference Manual*. Addison-Wesley, 1999.

[Whi97] White J.: *Mobile Agents. In Software Agents*, J. M. Bradshaw (Ed.), pp. 437-472. AAAI/MIT Press, 1997.