Exploiting MOOs to Support Multiple Views of Complex Software Development Processes

Elisabetta Di Nitto Alfonso Fuggetta Vito Sodano
Giuseppe Valetto
CEFRIEL – Politecnico di Milano
Via Fucini, 2 I-20133, Milano
Tel: +39-2-239541 Fax: +39-2-23954254
e-mail: {dinitto, fuggetta}@elet.polimi.it, sodano@sia.it, valetto@cefriel.it

1 Introduction

Multi–User Dimensions (MUDs) [3], and their Object–Oriented versions (MOOs) [6], are geographically distributed, programmable client-server systems that support the cooperation of multiple users according to the *virtual environment* metaphor. In this metaphor, users are allowed to concurrently navigate in a set of "virtual" rooms. Rooms are interconnected through doors and may contain objects. Users are allowed to explore the contents of rooms, create and manipulate objects, and contact other users visiting the same room.

Since the initial development of MUD1 (1978), MOOs have been employed for entertainment and socialization within connected and distributed communities of heterogeneous users. MOOs have recently also attracted a good deal of academic attention from various viewpoints: some researchers are interested, for example, in the social implications of this kind of collaborative computing, and in general in their psychological, anthropological, and sociological connotations [7]. Moreover, there are already a significant number of applications of MOOs to promote the cooperation of dispersed members of a community, organization, or interest group [4, 5]. This is usually limited to virtual meetings and document sharing, but new extensions of the basic MOO idea offer more advanced CSCW features [9]. In particular, some recent research has pointed out that MOOs are suitable to define and support the processes according to which collaborative activities are carried out. For instance, in PROMO [8] the MOO technology is exploited to define, represent and support software processes. These efforts aim to address some of the most relevant challenges for the research work in process and workflow technologies: the creation of advanced metaphor for process representation and enactment (i.e., execution).[1] These enhancements become more and more necessary as processes are becoming increasingly complex, collaborative, and decentralized. Moreover, the number and profiles of potential users of these technologies is growing rapidly. It is therefore necessary to identify powerful means that scale both in term of easy-of-use and flexibility, and also as far as their ability to work in large geographical settings is concerned.

During the past year we have explored a specific limitation of existing process/workflow technology: they are substantially unable to describe a process according to different points of view [1]. We have decided to exploit the MOO basic technology and metaphor in order to provide users with three **coexisting and consistent views** of the same process: artifact–based, task–based, and workspace–based. In the artifact–based view, MOO rooms have been used to gather together all the artifacts related to the same product components. In the task–based view, each room stores all the artifacts pertaining to a process step. Finally, in the workspace–based view, rooms represent human agents' workspaces. Users are allowed to navigate and operate using any of these views. They can switch to a different view at any time. Moreover, any operation accomplished in a view is consistently and instantaneously reflected in the other two views. This approach has been validated by developing a prototype of a multi-view MOO that has been used to model and enact a software development process.

This paper illustrates the main achievements of our work and is structured as follows. Section 2 presents the main concepts of MOOs and their high-level ar-

[1] In this paper, we consider the terms process and workflow as synonymous [10]. Consistently, we do not make any distinction between PCEs (Process Centered Environments) and WFMSs (WorkFlow Management Systems).

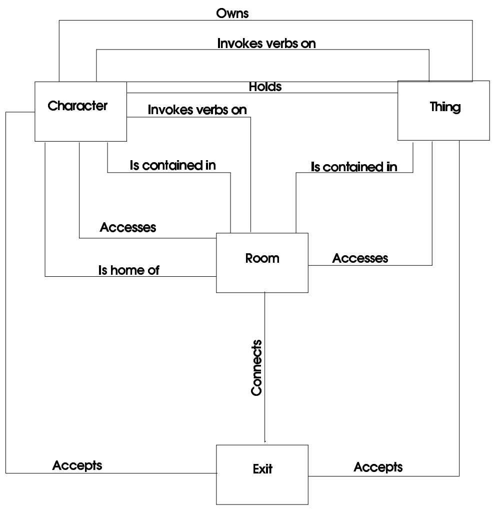

Figure 1: *Main entities of a MOO.*

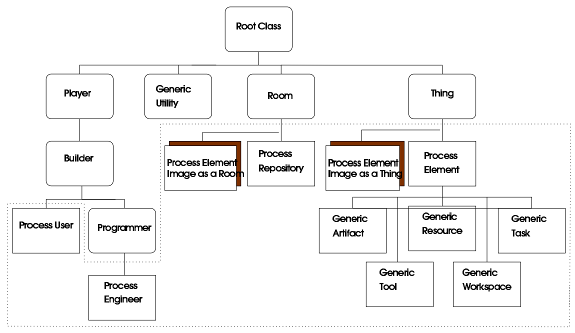

Figure 2: *The MOO Class Hierarchy and its process-oriented extension.*

chitecture. Section 3 discusses how to exploit MOOs to model processes. Moreover, it presents the three views and defines the consistency constraints that hold among them. Section 4 provides a quick overview on the implementation of our prototype and describes the test bed we have defined to validate the prototype. Finally, Section 5 provides some final considerations and illustrated our research agenda.

2 Basic concepts and architecture of MOOs

MOOs support the definition of virtual environments. In most cases, MOOs are at the same time instances and extensions of the LambdaMOO [6] system, conceived at the Xerox PARC laboratories. LambdaMOO offers all the basic characteristics and services upon which it is possible to build, and operate Object Oriented virtual environment. Its class hierarchy and its interpreted proprietary programming language allow its users to customize and enrich the environment, by providing fast prototyping facilities for cooperative programming.

MOOs are composed of *virtual rooms* connected by *exits*. Rooms may contain objects of any kind, represented by instances of the *thing* class, and can be visited by *characters*. Characters can either represent human beings or *robots*, i.e.computerized agents. Characters can hold and/or own things. Moreover, they can move across rooms. Exits may be associated with access constraints that control characters' navigation, depending on the things they hold or on the state of the rooms they leave and enter. Figure 1 shows all the mentioned MOO elements and the relationship among them.

MOOs are based on a simple client/server architecture. The server manages an object-oriented database containing the MOO elements, as well as client connection and communication. Clients enable users to access the MOO. Traditional clients are simple telnet terminals, and they visualize textual descriptions of rooms, characters and objects. However, several attempts to provide new graphical and multimedia interfaces are being carried out, as shown in [11, 12].

All the MOO objects are equipped with a set of programmable *verbs* (i.e. methods) that can be invoked either by other MOO objects or by users. MOO elements are structured in a specialization hierarchy. They are illustrated in Figure 2 by rounded–angle boxes. The `Player` class and its specializations represent types of characters, with different set of privileges and right over the environment.

Users are associated with a scope, based on their location in the MOO, that defines the objects that are visible to them and the verbs they can invoke. A MOO programmer can extend and modify the MOO objects by specializing them. It is represented within the MOO by the character type `Programmer`.

3 Exploiting MOOs to model processes

Typically, a process is described in terms of its activities, the artifacts produced and consumed during the course of the process, and the resources needed to pursue the process objectives. A typical process is composed of hundreds and even thousands of such entities. They are related through a variety of relationships that represent, for instance, the precedence among tasks or the responsibility of a human resource (see Figure 3). Representing such information is a very complex task. Powerful and effective formalisms and languages are needed, to support process comprehension and specification, and to enable an effective

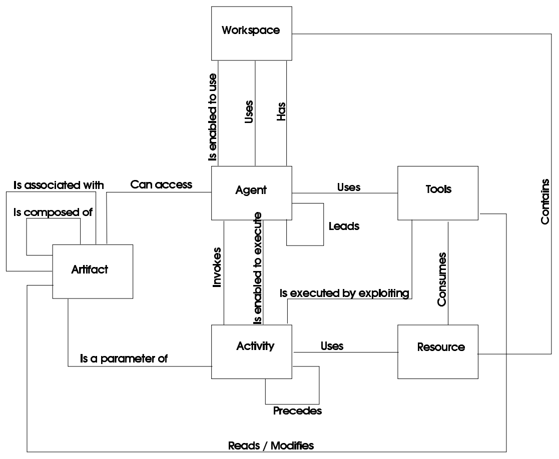

Figure 3: *Main constituents of a process.*

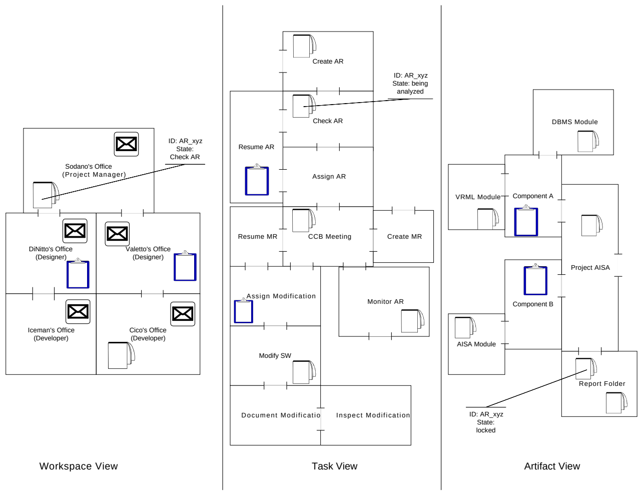

Figure 4: *A portion of the Anomaly Management MOO.*

enactment of the process.

Depending on their nature and origin, existing modeling languages (and the corresponding execution environments) tend to emphasize specific aspects of a process. For instance, notations such as Petri nets or Statecharts are typically oriented to describe the activities carried out in the process, their precedence relationship, and the artifacts used and produced during their execution. An ideal modeling language should make it possible to represent, study, and enact a process by exploiting multiple, coherent, and complementary views. Each view should represent a different "slicing" or perspective on the process, and should be able to emphasize specific facets of process performance and behavior.

We believe that there are at least three viewpoints that should be supported and that are directly related to the main process entities mentioned at the beginning of this section: tasks, artifacts, and resources. In this section, we would like to show how the MOO metaphor can be exploited to pursue this multi-view approach. For this reason we will briefly illustrate a case study to illustrate our experimentation. Then we discuss how the MOO metaphor can be used to create multiple views of the same process. Finally, we illustrate the constraints and basic mechanisms that need to be put in place the guarantee the consistency and coherence among the different views.

3.1 A simple case study

As a case study, we have selected a well–known example, documented in literature [2], and already used also in the PROMO project. This process is concerned with *Anomaly Management* activities, in the context of an industrial organization that produces telecommunication software. The users of this software system can report anomalies to a customer care center, which is in charge of generating anomaly reports (ARs). ARs are evaluated with respect to their relevance and urgency. Approved ARs are then submitted to a Change Control Board (CCB), formed by Project Managers and other domain experts. The CCB is in charge of analyzing incoming ARs and deciding the most appropriate action. Approved ARs are turned into Modification Requests (MRs) of the software product(s). MRs are assigned to Developer teams for implementation. Modifications are then reviewed and tested by the Designers in the light of the corresponding MR and AR: a Verification and Validation Report (VVR) is produced. Validated and approved software changes take part in a new software release that corrects the reported anomalies.

3.2 Three views of a process

We have identified three main views that apply to any process and that can be suitably expressed through the MOO metaphor. They have been extracted by representing the main concepts employed for process modeling as basic MOO entities. This includes finding mappings for the relationships that hold between elements in each domain.

A workspace-centered view A first metaphor is based on the idea of mapping user workspaces onto MOO rooms. According to this view, the MOO layout (i.e., the way rooms are connected by doors) illustrates the organizational structure of project teams. Each team member finds at any moments in his/her room all the artifacts, tools, and resources he/she is currently employing. A state is associated to each artifact, indicating the task according to which the artifact is currently manipulated. Exits implement constraints that reflect the organization of the team, the corresponding privileges and permissions, and the policies for task assignment. For instance, a team member is not allowed to move an artifact representing the project plan from the room of the project leader.

The leftmost side of Figure 4 shows part of the MOO layout representing the workspace view over a specific instantiation of the case study of Section 3.1. In the map, the workspaces of the project team members are represented together with the artifacts currently held by team members. For instance, Sodano's workspace contains the Anomaly Report he is currently processing.

A workspace-centered view is particularly intuitive for team members that must accomplish specific tasks. Working in virtual rooms, they can easily look at and use the tools and resources they have an hold on. However, this view does not highlight the overall structure of the process, in terms of the tasks that are currently being accomplished or scheduled and of the relationships among them.

A task-centered view The task-centered view focuses on representing tasks and the relationship among them. It is the one embraced by [8]. In this view, tasks are mapped onto MOO rooms. Each room contains the artifacts that are being used or produced in the corresponding task. It also contains the tools and resources necessary to operate the task. Exits highlight the precedence relationships between tasks and may be associated with constraints that do not allow, for instance, artifacts to be moved from a room to the next one if the preceding task (and/or the artifacts in question) are not in a proper state.

In Figure 4 we can look at the example of Section 3.1 from the task-centered viewpoint. In the current state of the process the task `Check AR` is being executed. In fact, the state of the Anomaly Report it contains specifies that it is under analysis. Notice that this Anomaly Report and the one in the Sodano's workspace are two images (see Section 4.1) of the same artifact (they have the same ID).

A task-centered view provides a complete description on the structure and accomplishments of a process; therefore, it is suitable for activities like process monitoring. On the other hand, its adoption by human executors is quite cumbersome. In fact, if users are involved in more that one task at the same time – as it is usual in most work practices – they must move back and forth across rooms to handle all of them.

An artifact–centered view The artifact–centered view is based on the idea of representing with a MOO layout the breakdown structure (PBS) of the product being developed. In this case, each node of the PBS is a room in the MOO, and the connections among nodes correspond to exits between rooms. The artifacts associated with a node in the PBS are contained as things in the corresponding room. These artifacts can either be documents (e.g. requirement specifications, or design documents), source code files, etc. When a MOO character brings an artifact with him/her, it means that he/she is locking it for his/her use.

The rightmost side of Figure 4 shows a portion of the artifact–centered view instantiated for the example of Section 3.1. Here we have a product that is composed of three main elements, components A and B and another component clustering some related reports (ARs, MRs, ...). In the Report Folder the Anomaly Report under analysis by Sodano is indicated together with its state (locked).

An artifact view, as we conceive it, shows the structural relationships and dependencies between the various parts of a software product. Therefore, it seems suitable to accommodate Configuration Management mechanisms, as well as to enforce policies for the cooperative usage of artifacts, whether composite or atomic.

3.3 Inter-view constraints

The views described above are complementary, and suit the needs of different categories of process executors. They are also strictly related: the changes performed within a view must not remain local to that view, but must be properly mirrored in the other ones. This mirroring can be accomplished if a set of *inter-view constraints* is defined and enforced. The constraints we have identified so far are related with the main operations a user can perform in each of the presented views. Some examples of these constraints are the following:

- When a character – while operating on an artifact in the workspace-centered view – changes its state, the artifact must be put in the corresponding task room in the task-centered view.

- When a character activates a task in the task-centered view, then all the artifacts located in the corresponding room must appear also in the character's workspace in the workspace-centered view.

- When a character brings an artifact with him/her from the artifact-centered view, in the workspace-centered view this artifact must appear in the character's workspace room.

4 A prototype of a MOO–based multiple view PCE

In this section we present the implementation of the three views outlined in Section 3 and the mechanisms for maintaining the coherence among the views. We also provide an example of usage of the resulting PCE, through our case study.

4.1 Implementing multiple views on top of LambdaMOO

We have exploited the programmability of LambdaMOO to implement our PCE. In particular, we have defined in the MOO database the classes that represent the basic elements of a generic process. They define the specialization hierarchy rooted by `Process Element` in Figure 2. We have also defined two new process-related types of characters, `Process User` and `Process Engineer`. The former is a generic actor of the process. The latter has process modeling capabilities and can extend and customize the hierarchy of process-related classes. Finally, we have defined a special MOO room called `Process Repository` that stores all the process-related objects. This room is made visible only to the `Process Engineer` who accesses it to modify and extend the structure of the process-related objects. `Process Users` do not operate directly on the `Process Repository` or on its content. Instead, they work on the views defined upon the above mentioned objects according to the criteria presented in Section 3. In Figure 2, all the process-related elements are enclosed by a dashed line.

In a specific view, `Process Elements` may change their appearance and structure according to the characteristics of the founding metaphor. For instance, in the task-centered view, a `Generic Task` is seen as a room containing all the other objects involved in the execution of the task. Therefore, proper mechanisms that transform objects according to predefined rules and constraints should be implemented within the MOO. Moreover, these mechanisms should guarantee that the operations performed on the views are correctly mirrored onto the actual object.

Unfortunately, MOOs, per se, do not provide any suitable mechanism for implementing this dynamic transformation. To overcome this problem, we have chosen to define the three views as sets of persistent elements of the MOO, actually stored as separate objects in the database. More in detail, for each `Process Element`, the MOO database contains three additional objects, called *image objects* that are in charge of properly representing the `Process Element` in the three different views. Image objects can be either a specialization of room or a specialization of thing, depending on the MOO element they represent in the view. Users operate on image objects, but all modifications are passed to the corresponding `Process Element`, and from it to the other images in all the other views. This simple automatic mechanism for enforcing the inter-view constraints is part of the implementation of both the `Process Elements` and the image objects.

Thus, in our approach, the various views of a process are represented simply as isolated (i.e. non-interconnected) portions of the same MOO. The mechanism to switch among views exploits the typical navigation facilities of MOOs, that under certain circumstances allow teleporting, i.e. the transfer of characters from a room to another without the need of an exit connecting the two rooms.

It is clear that our approach to multiple views does not scale up too well: the object replication can become unacceptable, in case the process repository gets particularly large; furthermore the addition of other views to the MOO–based PCE becomes a complex problem. However, it is adequate to implement some process examples that validate our approach. A more sophisticated implementation of a multiple-view PCE would require the exploitation of proper mechanisms such as the ones offered by some Object Oriented DBMSs.

4.2 Validation through the testbench process

We have proceeded to implement the case study by specializing the classes defined in the `Process Repository` of our prototype. For instance, we have created objects for ARs, MRs, etc. as well as specialized types of characters for the recognizable roles (e.g. Project Managers, Designers, Developers, etc.). Moreover, we have created the objects representing the tasks to be accomplished (modify module, inspect modification, etc.) and we have identified the relationships among those objects. Finally, for each view, we have set up the corresponding MOO layouts. Users switch among views by invoking the verb `switch view <view-name>`, implemented by any room representing a Process Element image.

In order to get the feeling of how the system works, let us consider a simple process execution scenario: a Project Manager (the character Sodano) starts working within the Task view. In particular, he/she decides to operate in the `Check AR` room, where he/she finds an image of a previously created Anomaly Report that must be looked at and analyzed for approval. To perform the relevant actions the Project Manager invokes either generic verbs provided by MOOs, that is, `enter`, `look`, etc. or verbs that are specialized by the classes written for process capture. The verb `examine`, for instance, is specialized by the Anomaly Report image, in order to display both the content and – via the invocation of the corresponding `examine` verb provided by the `Process Element` superclass – the state of the artifact. As a result of the activities of the Project Manager, the following actions may happen:

- The image of the Anomaly Report in the Workspace view is placed in the workspace of the Project Manager.
- The image of the Anomaly Report in the Artifact view is locked so that verb invocation by other players is prevented. The state information remains visible and evolves according to the analysis activities carried out by the Project Manager. In response to those activities, also the location of the artifact with respect to the PBS represented in that view may vary.
- Some tool is executed on the terminal of the Project Manager, to support the analysis of the AR.
- In case the analysis terminates with the approvation of the AR for further action, members of the CCB who are working in their own workspace are prompted with a convocation to the `CCB Meeting` task room. A copy of the AR in question is attached, to be examined and perhaps annotated. Other CCB members who are not present in their workspaces at the moment will not immediately be aware of the convocation, but will find the related artifacts in their workspaces when they visit them.

5 Conclusion and future work

We have presented in this paper some of the results of our ongoing work. We keep exploring and studying both the limitations and the advantages of exploiting MOOs as a framework for process capture that provides multiple views on a process.

We believe that our proof–of–concept prototype has served its purpose of demonstrating that the exploitation of MOOs for multi–view description and support of processes is viable, despite the clumsiness of the mainly textual user interface of these systems. With the upcoming advent of richer and user–friendly presentation facilities, working from within a MOO shall become easy and natural.

We have also observed that, notwithstanding the intrinsic familiarity of humans with the MOO metaphor, the level of user–friendliness sported by a system like the one we have depicted is highly dependent on:

- the intuitiveness of the metaphors adopted for the views;
- their accuracy with respect both the underlying application domain and the real world situation they try to mimic;
- the user understanding of the interrelationships among the views;
- the usability of the tools provided to the user for interacting and carrying out activities within the environment without drifting too far from his/her usual work practice.

With respect to the latter aspect, there is another interesting point, resulting in a further challenge: in order to support the work of the users of the environment in a natural way, a framework for the integration of external tools must be conceived. MOOs are, by their nature, eminently self–contained systems: each of them constitutes a “world” of its own and users become citizens of that world, completely immersed in it; more important, all the tools available are described, built and operating within the limits of the MOO. Opening MOOs towards the external “real” world is no easy business. Anyway, there are some possibilities: for example, PROMO suggests to exploit the MOO client–server architecture in an non orthodox fashion to provide a bridge that communicates and controls generic (legacy) applications. We intend to further explore this and other options in the future.

We plan to substantially enhance our prototype as far as the process definition mechanisms are concerned. Currently, the amount of work necessary to Process Engineers, in order to define and enact a single process instance is considerable. Moreover, all of this work must be carried out at the level of coding, i.e., programming MOO objects that represent the idiosyncrasies of the process. A consequence is also a low degree of reusability of the work done for each process case. All these drawbacks are due to the insufficient level and amount of abstraction devoted to process capture that we have been able to put on top of the generic MOO classes. Our extension to the MOO hierarchy provides only very general concepts and constructs. We are currently working towards the definition of a more extensive and exhaustive layer for process definition and support, that must derive from a highly formalized model of processes in general, as well as from a precise vision of how the views relate to the process and among themselves.

Following such a formalization effort, it will be possible to come up with a set of utilities and tools dedicated to the Process Engineers, and devoted to higher–level and more efficient process description, management and monitoring. We aim to achieve some results also in this area, that in our opinion will add a significant amount of value to a MOO–based PCE.

References

[1] V. Ambriola, R. Conradi, and A. Fuggetta. Process-Centered Software Engineering Environments. *ACM Transactions on Software Engineering Methodology*, July 1997.

[2] S. Bandinelli, A. Fuggetta, L. Lavazza, M. Loi, and G.P. Picco. Modelling and improving an industrial software process. *IEEE Transactions on Software Engineering*, May 1995.

[3] R. Bartle. Early MUD history. Technical report, Article posted to USENER group rec.games.mud, 1990. http://www.utopia.com/talent/lpb/muddex/bartle.txt.

[4] BioMOO. The biologists' virtual meeting place. http://bioinfo.weizmann.ac.il/BioMOO/.

[5] A. Bruckman and M. Resnick. The MediaMOO project: constructionism and professional community. *Convergence*, 1(1), Spring 1995.

[6] P. Curtis. LambdaMOO programmer's manual. Technical report, Xerox Palo Alto Research Center, 1996. ftp://parcftp.xerox.com/pub/MOO/LambdaMOO-1.8.0.p5.

[7] P. Curtis and D. Nichols. MUDs grow up: social virtual reality in the real world. In *The Third International Conference on Cyberspace*, XEROX Palo Alto Research Center, May 1993.

[8] J.C. Doppke, D. Heimbigner, and A.L. Wolf. Software process modeling and execution within virtual environments. *ACM Transactions on Software Engineering Methodology*, January 1998.

[9] S.M. Kaplan, G. Fitzpatrick, T. Mansfield, and W.J. Tolone. MUDdling through. In *the 20th Annual Hawaii International Conference on System Sciences*, Washington D.C., 1997. IEEE Computer Society.

[10] Amit Sheth, editor. *NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions.* May 1996.

[11] J. Tennison. WOOM. web object oriented with multiple ownership. Technical report. http://psyc.nott.ac.uk/aigr/papers/WOOM1.0/.

[12] Diversity University. Platform for experimentation with new and innovative approaches to learning. Technical report. http://www.academic.marist.edu/duwww.htm.