OPCATeam – Collaborative Business Process Modeling with OPM

Dov Dori, Dizza Beimel, and Eran Toch

Technion, Israel Institute of Technology, Haifa, Israel
dori@ie, dizza@tx, erant@tx{.technion.ac.il}

Abstract. While collaboration has become a basic requirement for many development environments, solutions for collaborative modeling are far from being satisfact1ory. OPCATeam, which relies on Object-Process Methodology (OPM), provides a collaborative modeling environment that can fit generic modeling purposes. OPM, a holistic, bi-modal visual and textual approach to the study and development of systems, integrates the object-oriented and process-oriented paradigms into a single frame of reference. This characteristic, combined with refinement and abstraction mechanisms, makes OPM ideal for business process modeling. OPCATeam features multi-user Client-Server architecture. The server holds a single OPM model for each system in a central repository. OPCATeam has three access permission levels: workgroup, OPM model, and diagram. The diagram permission, which is unique to OPM, aims to reduce the number of conflicts between concurrent updates and preventing modelers from affecting shared elements while allowing them to refine these elements. Users can simultaneously update the model through the clients according to their access permissions. The detailed design implementation is currently being tested.

1 Introduction

Collaborative design occurs "when a product is designed through the collaborative efforts of many designers" [1]. Collaborative modeling, which applies to a subset of these efforts, focuses on the architecting and design of processes and systems using a formal modeling methodology. This paper defines the requirements from a collaborative modeling environment, specifies architecture for this purpose that is based on Object-Process Methodology (OPM) [2], and describes OPCATeam – an application of these principles.

Collaborative modeling concepts have been known and implemented in such fields as business processes, systems modeling, CAD/CAM, software development, and ontology engineering. While applications in these fields operate under different conditions for different purposes, they do share a set of common requirements insofar as collaboration is concerned. Three guidelines help evaluate, compare and define collaborative system modeling solutions to the following common set of problems:

- *Concurrency*: The environment should allow team members to work on a shared system at the same time, based on a single integrated and consistent model that

- describes it, throughout the development process. The model should be available to all the members in real-time, enabling them to get the most up-to-date view of the system.
- *Communication*: The environment should enable multi-way communication among the team members regardless of their physical whereabouts.
- *Security*: The environment should allow secure development, protecting the model under construction from unauthorized external entities and unauthorized changes by modelers.

OPCATeam, which relies on OPM, provides a collaborative modeling environment that can be used for a large variety of modeling purposes. OPM is a holistic, bimodal approach to the study, development and evolution of systems whose single model is represented both visually and in natural language. OPM integrates the object-oriented and process-oriented paradigms into a single frame of reference. Combined with elaborate built-in refinement and abstraction mechanisms, this structure-behavior combination in one model makes OPM ideal for business process modeling. An interesting application of OPM [24] is a generic reverse engineering process that captures the available alternatives at different application levels of an Enterprise Resource Planning (ERP) system. This is an example of a complex system for which the option of working in a collaborative environment is most beneficial.

Since an OPM model consists of a set of interrelated Object-Process Diagrams (OPDs), the main challenge of concurrent OPM-based collaborative development is maintaining the integrity of the OPM model that is manipulated by more than one modeler at the same time. Entities (objects or processes) in one OPD can be refined in a new OPD that contains their detailed descriptions. When an entity is refined, other entities that were directly connected to it in the source, abstract OPD are brought into the newly created OPD, and the modeler can add entities such as sub-processes inside an in-zoomed process and drag the links from the process to these sub processes. When more than one modeler is specifying details of the same entity they are bound to contradict. Moreover, since OPDs can share common entities, when two or more modelers work concurrently on two OPDs that share the same entity, each change in a common entity can potentially influence other OPDs.

Fig. 1 illustrates a simple example of an integrity maintenance problem. Two modelers are working on two different OPDs (SD1 and SD2), which are refined from a common OPD (SD). All three OPDs share a common entity (Object X), the type of which was determine in SD as “char[50]”. Modeler A wishes to change the type of object X in SD1 to “date,” while modeler B whishes to changes the type of object X in SD2 to “time.” Since both modelers are working on the same model at the same time, they compromise the integrity of the OPM model. To avoid such situations, a method for maintaining the integrity and completeness of an OPD set needs to be developed. An important requirement of the collaborative system development environment is support of standard development processes, such as the spiral model. In such iterative development processes, each step in the process is based on refinement and modification of the output of the last step [3]. The architecture of our collaborative development environment takes advantage of OPM’s built in refinement mechanism. As noted, security is a major challenge in multi-user environments.

Indeed, one of our goals is to protect the OPM model from unauthorized changes. We would like to ensure that the development of the OPM model is coherent with the organizational structure and authorizations of the development team.

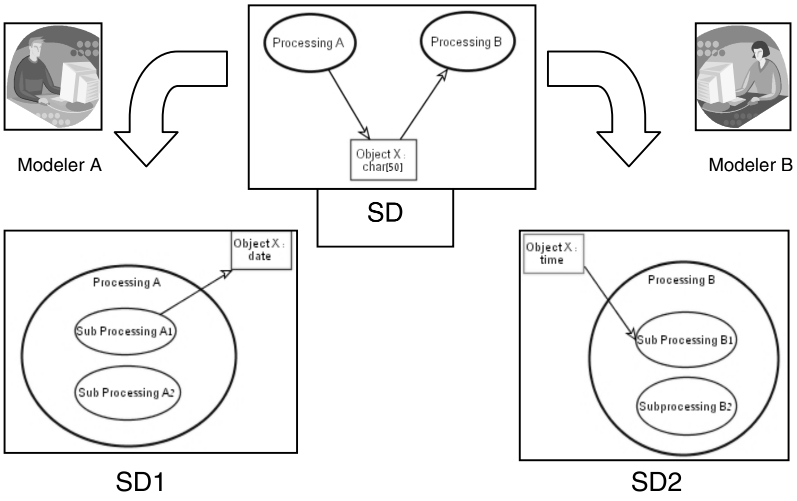

Fig. 1. An integrity problem created by concurrent OPM model development

OPCATeam has multi-user Client-Server architecture. The server holds, maintains and controls in a central repository a single OPM model for each business process or system. Users can simultaneously update the models through the clients according to their access permissions. Each client of OPCATeam sends update messages to the server, where the messages are synchronized and updated in the central repository. Communication services provide complementary infrastructure to the collaborative environment. An access control module enables organizations to implement a development process in a secure and moderated environment using the refinement features of OPM. The access control module enables users to define access permissions at three levels: Workgroup access level, OPM model access level and OPD access level. The Workgroup and OPM model are standard access control levels. They restrict access to resources for individual users or user groups. The third is a fine-grained OPD level that controls permissions to access individual OPDs. The creator of the OPD (the user who created the OPD) can grant viewing, editing, and refining permissions to other users. The viewing permission prevents modelers from being able to change the contents of an OPD. The editing permission grants full editing privileges to all the OPD elements, except those inherited from an ancestor OPD. The refining permission enables a modeler to refine a thing (object or process) in an OPD without changing its "signature," i.e., the various links that are attached to it, including inputs, outputs, enabling and event links. This permission type, which is unique to OPM, helps reduce the number of conflicts between concurrent updates, preventing modelers from introducing contradictions into the model while still allowing them to refine common entities. Using the refinement permission type is one way to solve the problem exemplified in Figure 1. If the permissions of both modeler

A and modeler B are set to "Refine" on OPD SD, then both modelers can refine the processes and the connected objects, but they cannot commit to changes that may affect each other.

This paper specifies an implementation-independent collaboration model that is embedded in OPM and takes advantage of refinement ability.

2 Background

According to Webster dictionary, to collaborate means “To work jointly with others or together especially in an intellectual endeavor.” In the context of information technology, collaboration takes place whenever humans work together to accomplish a common goal or compatible goals using one or more computer applications. The collaboration concept is normally associated with groupware technology, which, according to [4], is designed to facilitate the work of groups. This technology may be used to communicate, cooperate, coordinate, solve problems, compete, or negotiate. During the past two decades, many organizations have been considering electronic collaboration of distributed teams as a means to achieve higher productivity and improve the quality of their work products. To this end, various collaboration technologies have been introduced to provide solutions in the areas of electronic communication, coordination, and content sharing. CSCW (Computer Supported Cooperative Work) was one of the first such technologies to encourage research collaboration projects. IBM Lotus Notes [5], for example, is one of these early research efforts results. In spite of such significant efforts, groupware products have failed to deliver anything more than marginal improvements to existing email and document management solutions. The current software industry offers significant variety of collaboration products in a number of domains. These include joint activity tools (e.g., audio communication, instant messaging, and content sharing tools) like NetMeeting [6], collaborative electronic presentations and meetings like Lotus Sametime [7], collaboration activities in ERP systems like SAP [8], and collaborative content management such as Documentum [9].

In this paper we focus on collaboration in the domain of formal engineering artifacts, which includes business process modeling, systems modeling, anthologies, CAD/CAM and software coding. A new approach to the capturing of Business Process Models, [10], is a collaborative business process modeling tool that combines Web discussion forums with MS VISIO drawing tool. However, the asynchronous tool’s working mode potentially hinders its collaborative aspect.

Prominent systems for collaborative software coding include CVS [11], a large-scale open-source project, which provides a team of developers with a user-friendly, simple collaboration environment. TeamSCOPE [12] is a groupware solution that interfaces with development environments. An experiment that tested the effectiveness of TeamSCOPE concluded that features such as member list and chat improve teamwork efficiency.

Collaborative ontology engineering tools take a different approach. OntoEdit [13] uses client-server architecture to support a concurrent collaborative engineering process. The model under construction is duplicated through the client programs, and a locking mechanism enforces the model integrity. The users can lock a subtree of the ontology (which is specified as a tree) and edit it without interference. This approach could be implemented in a simple manner only under the assumption that there is no interconnectivity between the subtrees. Ontology classes may have relations between them, but in this system, working on a class assumes that it does not influence any related class. As explained, OPM cannot operate under this assumption, and must therefore employ a different approach.

In the field of systems modeling, Poseidon for UML [14] is in a process of upgrading to team-support edition. According to company announcements, this edition will include version control, multi-user support, and client-server architecture. Other UML-based tools, such as Rational Rose [15] and Cittera [16], base their collaborative features on standard version control software. Concurrency is achieved by breaking the system modules into separate files, which are then handled through the customary check-in/check-out mechanism. SoftDdoc [17] is an example of a distributed model management system that supports collaborative software development whose model descriptions are shared and managed through a middleware.

Extensive research and numerous projects, surveyed in [1], concern collaboration in the CAD/CAM domain. Eight future research opportunities were identified, including collaborative conceptual design modeling and data sharing. Another project [18] for computer-aided sequential control design tool deals with collaborative modeling problems and is based on client-server architecture. During a collaboration session, only the user who "owns" a virtual token can modify the design, while others can only view it. This solution is missing the concurrency of teamwork, which is one of the basic building blocks of collaborative work. An interesting facet of collaboration is the social aspect, which relates to the influence of electronic collaboration on the team's relationship. The big challenge for managers is to recognize that both software and personal interactions contribute to successful collaboration [19].

3 Object-Process Methodology

Object-Process Methodology (OPM) is a holistic approach to the study and development of systems, which integrates the object-oriented and process-oriented paradigms into a single frame of reference. Structure and behavior, the two major aspects that each system exhibits, co-exist in the same OPM model without highlighting one at the expense of suppressing the other. Most interesting and challenging systems are those in which structure and behavior are highly intertwined and hard to separate. Due to structure-behavior integration, OPM provides a solid basis for modeling complex systems.

The elements of the OPM ontology are *entities* and *links*. Entities are of three types: *objects*, *process*es (which are *things*) and *states*. These are the basic building blocks of any system expressed in OPM. *Objects* are (physical or informatical) things that exist, while *processes* are things that transform objects. *Links* can be structural or procedural. *Structural links* express static relations between pairs of entities. Aggregation, generalization, characterization, and instantiation are the four fundamental structural relations. *Procedural links* connect entities (objects, processes, and states) to describe the behavior of a system. The behavior is manifested in three major ways: (1) processes can transform (generate, consume, or change the state of) objects; (2) objects can enable processes without being transformed by them; and (3) objects can trigger events that (at least potentially, if some conditions are met) invoke processes. OPM can be represented by two equivalent modalities: visual and lingual. The visual formalism is defines as a set of inter-related Object-Process Diagrams (OPDs), constitute the graphical representation of the designated model. In this article we focus on the visual representation. Because of the fact that the visual and lingual have a bi-directional mapping, all the assumptions and conclusions related to the visual representation apply to the lingual representation as well.

Three built-in refinement/abstraction mechanisms are built into OPM. They enable presenting the system elements at various detail levels without losing the comprehension of the system as a whole. In-zooming and out-zooming are one pair of refinement and abstraction mechanisms, respectively, which can be applied to entities (objects, processes and states). Zooming into an entity decreases the distance of viewing it such that lower-level elements enclosed within the entity become visible. Conversely, zooming out of a refined entity increases the distance of viewing it, such that a set of low-level elements that are enclosed within it become invisible.

Unfolding and folding are a second pair of refinement and abstraction mechanisms that can be applied on things – objects or processes. Unfolding reveals a set of low-level entities that are hierarchically below a relatively higher-level thing. The hierarchy is with respect to one or more structural links. The result of unfolding is a tree, the root of which is the thing being unfolding. Linked to the root are the things that are exposed as the result of the unfolding. Conversely, folding is applied to the tree, from which the set of unfolding entities is removed, leaving just the root.

4 OPCAT: Object-Process Case Tool

OPCAT[1] (Object-Process CASE Tool) [21] is an integrated system engineering environment that supports OPM-based system development and evolution. OPCAT has been under continuous development as an academic project since 1996. Designed to eventually support the entire system development lifecycle through OPM, OPCAT supports a bimodal graphical-textual view of the system under development, enabling increased OPM accessibility to heterogeneously skilled users engaged in the system development process. The environment already provides for many phases of the system lifecycle, including system specification, automatic analysis and design documentation generation, code generation into Java [22] and potentially any programming language, generation of various UML diagrams, including class, use-case, collaboration and Statecharts, and animated simulation of the OPM model. A major function not currently supported by OPCAT is collaborative concurrent development of a single system by different teams of users. The architecture of this collaborative OPCAT version, called OPCATeam, is the focus of this work.

[1] OPCAT can be freely downloaded from http://www.ObjectProcess.org

Definitions of Key Concepts

Element: A basic building block of any system expressed in OPM, which can be an entity or a link.

Entity: An element, which is not a link, which can be a thing or a state.

Thing: An object or a process.

Object: A thing that can exist for some time physically or logically.

Process: A thing that transforms an object by creating it or consuming it or changing its state (i.e., affecting it).

Link: a connector between two entities.

Structural link: A link denoting a persistent relation between objects.

Procedural Link: A link between a process and the object it transforms or a state of that object

OPM model: The OPD set that completely specifies a business process or system along with its OPL script, i.e., the set of all the OPDs that together specify the entire system, each with its corresponding OPL paragraph. In OPCAT, the information of an OPM model is saved in a single XML file.

System Map: A directed hypergraph in which each OPD in the OPM model is a node and each edge is directed from a node to another node in which one of the things is refined.

Scaling: refinement/abstraction, which can be in-zooming/out-zooming or unfolding/folding.

Consistency: A Boolean attribute of an OPM model denoting coherence and lack of any specification contradiction across the various OPDs in the OPD set of an OPM model.

Integrity: A Boolean attribute of an OPM model denoting completeness and lack of dangling elements in the database of the OPM model.[2]

5 Problem Specification

Since OPDs in the OPD set are interconnected in nature, a major challenge of collaborative OPM development is provision of reliable parallel collaborative OPM modeling. Since OPDs can share common entities (objects, processes or states), each change in a common entity potentially influences other OPDs. Therefore, a method for maintaining the integrity of an OPD set must be developed.

Following OPM conventions [2] we label OPDs hierarchically by SD (for System Diagram, the root), SD1, SD2, SD1.1, SD2.3.1, etc. A *refinement relation* between OPDs from SD1 to SD2 exists if and only if SD1 contains a refined (in-zoomed or

[2] A similar requirement in the database domain is referred to as "referential integrity."

unfolded) version of the same entity in SD2. A *commonality relation* between OPDs SD1 and SD2 exists if and only if SD1 and SD2 share at least one common entity.

In order to analyze the integrity problem, we extend the definition of the System Map hypergraph SM [2], in which each node represents an OPD. The edges of the augmented SM are of two kinds of labeled edges: directed and undirected. A directed edge represents a refinement (in-zooming or unfolding) relation, such that the edge is directed from the source OPD, in which the refined entity (object or process) is more abstract, to the destination OPD, in which that thing is more refined. The label indicates (1) the entity that is being refined and (2) the refinement type (in-zooming or unfolding).

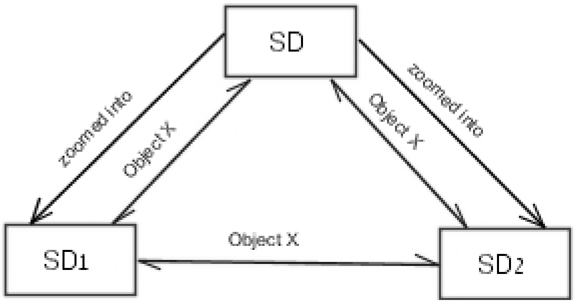

Fig. 2. An example of an augmented System Map

Fig. 2 illustrates an Augmented System Map, in which SD1 contains a refined version of an entity E1 (object or process) that appears in SD in its abstracted version. SD2 contains a refined version of another entity E2, which is also represented in SD in its abstracted version. In our example, the directed edges represent a refinement (in-zooming or unfolding) relation between SD to SD1 and between SD to SD2. Since a common entity (Object X) exists in SD, SD1 and SD2, an edge labeled “Object X” connects each one of them to the two other nodes. The first group of directed edges forms a DAG. The second group of undirected edges may create a clique, as it does in our example. The formation of a clique depends on the appearance of the entities in the various OPDs.

Having created the Augmented System Map hypergraph, few kinds of complexity queries can be asked. A typical query for integrity checking is to find all the references to an object X that has just been updated by one team member. Fig. 2 illustrates that the response to this query can range from a simple graph to a clique. In this work we focus on a solution for evolving systems with OPM in a collaborative environment, Complexity analysis of such queries is a topic for a separate research.

Security and Access Control

Collaborative modeling requires addressing two security aspects. The first is protecting the model from inspection and changes by unauthorized entities, be they innocent or malicious. This aspect is applicable to various types of systems and the collaborative nature of the modeling environment per se does not complicate the problem any further. The second aspect, which is specific to collaborative environments, relates to the enforcement of regulations that team members must follow in order to maintain the integrity of the OPM model under development. To ensure this, the team must be coordinated and the model must be protected from damage caused by mistakes and uncoordinated changes by team members.

This regulation enforcement aspect is related to two characteristics of the system development process: organizational structure of the development team and the nature of the development process.

To understand the development team organizational structure aspect, consider a typical scenario of a team consisting of a team leader and two modelers. Each modeler is responsible for modeling one subsystem, while the team leader is responsible for the integration of the two subsystem models into a coherent system model. We require that our collaborative environment restrict each modeler to the subsystems she/he is responsible for and provide the team leader with tools to control the access permissions of his team members.

Regarding the nature of the development process, we note that many modern development processes are iterative in the sense that outputs of one development stage serve as inputs for the next stage. Furthermore, artifacts created in some stage can be refined and modified to produce a more concrete artifact in the next stage downstream. OPM's built-in abstraction/refinement mechanisms can naturally support this type of development process. In a collaborative environment, we require that only authorized team members who are responsible for a specific development stage will be able to refine artifacts created at that stage. For example, the domain expert representing the customer may define higher-level artifacts, mainly requirements, while the system architect may refine these artifacts, but not change them without the domain expert's approval.

6 OPCATeam Architecture

To realize our goal of creating a multi-user collaborative OPM-based system evolution environment, our proposed architecture, whose OPM model is presented in Fig. 3, tackles the aforementioned challenges of concurrent modeling, communications and access control.

Our solution is based on a client-server multi user environment architecture, in which a central server provides collaborative services to the OPCATeam clients. The Client-server architecture optimizes the workload distribution between the clients and the server: The server handles issues that are centralized in nature, while the client contributes visual and logical services that already exist in the single-user version of OPCAT. The OPCATeam client wraps the current single-user OPCAT implementation, offering a user interface for the services provided by the server. This way, any improved new version of the single-user OPCAT will automatically be incorporated into the collaborative environment. The server has three main modules:

- The **Model Manager** module handles concurrent development of OPM models using a central repository and a concurrent update mechanism. This mechanism allows simultaneous user updates to a single OPM model, which is shared by one or more authorized users, while its perfection is maintained at all times. The module includes a version control function that logs updates and enables revision control. It uses records of update history to ensure control over the collaborative development process.

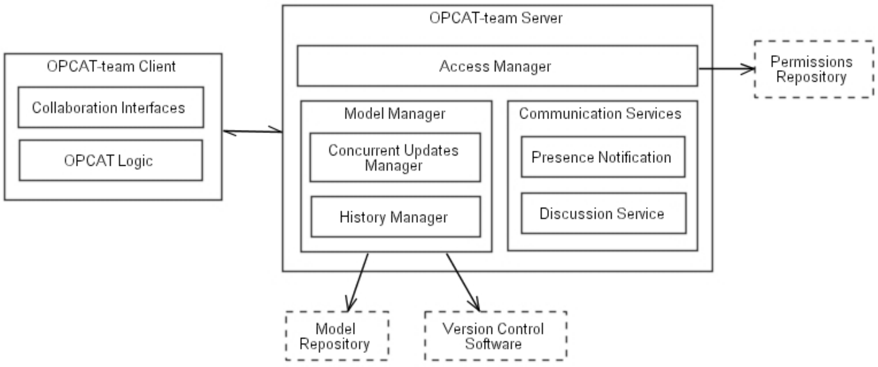

Fig. 3. OPCATeam architecture overview

- The **Access Manager** module controls access and restricts changes for reliable and secure collaborative development process. The permissions, managed by a central repository, impose access restrictions on the workgroup (a shared set of OPM models), on the entire OPM model, and on the OPM System Map.
- The **Communication Services** module is a set of communication applications, such as synchronized discussion (chat) option and a presence notification window.

Our solution is based on the existence of a **single** OPM model that is controlled and maintained by the server. Each OPD model is specified by its OPD set and saved in an XML format. OPCATeam users start a (potentially collaborative) session by downloading a copy of the OPM model to their local client workstations. Throughout their work, collaborating system modelers can send update messages, which are commensurable with their permissions, to the OPM model that resides on the server. The updates are synchronized and handled in the central OPM model repository.

The server inserts these synchronized updates into a queue and performs the following steps:

1. The server optionally rechecks the client permission to request the current handled update to boost the security level of the OPM model.
2. The server adds the authorized update to the OPM model.
3. The server checks the integrity of the OPM model immediately after introducing the change and performs any necessary adaptations of the OPM model database to maintain its perfection.
4. If the integrity of the OPM model database can be maintained, the server commits the update. After the commit execution, the updated model is accessible to all the authorized clients. Users can then initiate a request for the server to refresh their views and to inspect the changes introduced in the meantime by their collaborating team peers. If the integrity of the OPM model database cannot be maintained (e.g., a client has requested to update an object that no

longer exists in the OPM model) the server rejects the update, and sends a corresponding message to the client, specifying the reasons for the rejection. This process prevents imperfection problems. The module utilizes version control software that provides logging of updates, revision control, and rollback abilities. This enables system architects to manage and inspect the development process, and to implement utilities that may become standard in collaborative OPM-based system development. As noted, the server holds a single, most recently updated OPM model. Since the model is saved on the server, recovery from client crashing is simple. The only action the client needs to take is requesting a fresh copy of the OPM model from the server. However, the user does lose all the changes made since the last save operation (which can be either manual or automatic).

Access control enables users to define access permissions at three levels. The shallow access level is the *Workgroup access level,* the intermediate one is the *OPM model access level*, and the deep access level is the *OPD access level*. The first two levels follow the standard in many development environments, while the third is OPM-specific. The permissions are set individually at the user level and can be changed in real-time by an authorized users. *Workgroup* stands for a group of OPM models that a team of modelers takes an interest in. The workgroup access level exhibits the Boolean permissions *Create OPM model*, *View workgroup*, and *Admin* (which gives the user the permission to grant workgroup permission to unauthorized users). By default, every user can create a new workgroup, in which case he is defined as the *workgroup creator*, and gets the admin permission.

The OPM Model access level exhibits the Boolean permissions *View OPM model* (which gives the user the permission to view the OPDs in the OPD tree), *Commit OPM model* (which gives the user the permission to save the model in the Version Control Software repository), and *Admin*. The user who created the OPM model is defined as the *OPM model creator* and gets the admin permission. The corresponding workgroup administrator may add more Administrators to an OPM model.

The OPD access level exhibits the Boolean permissions *view OPD* (which gives the user the permission to view the OPD but not to update it), *Edit OPD* (which gives the user the permission to edit elements in his OPD, except for elements that were inherited from an ancestor OPD), *refine OPD* (which gives the user the permission to refine any OPM entity in his OPD by in-zooming or unfolding) and *Admin* (which gives the user the permission to grant OPD permissions to unauthorized users.) The user that created the OPD is defined as the *OPD creator*, and gets the admin permission. The OPM model Admin may add more Administrators to this OPD.

OPCATeam has at least one administrator, who is authorized to add or disable Administrators to a workgroup, disable users, etc. Fig. 4 illustrates the structure of the Access Control, and followed by the OPL that OPCAT generated. Fig. 5 illustrates the dynamic zoomed in server process for new OPD creation: a new OPD request message arrives to the server, which checks the user permission to apply for it. In case the checking results success, a new OPD is created, the user is acknowledged and the server updates the OPM Model.

The access control module helps reduce update conflicts. As noted, an OPM-model consists of a set of OPDs. The user can create, update or delete OPDs according to his permissions. Thus, in a restricted configuration of OPCATeam, conflicts can be totally prevented, while in other configurations, conflicts may occur, and will be handled by the server.

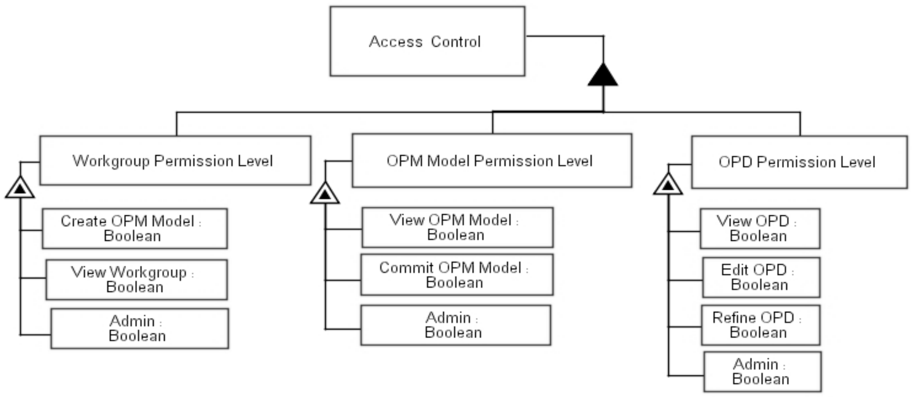

Fig. 4. Access control structure and OPL generation

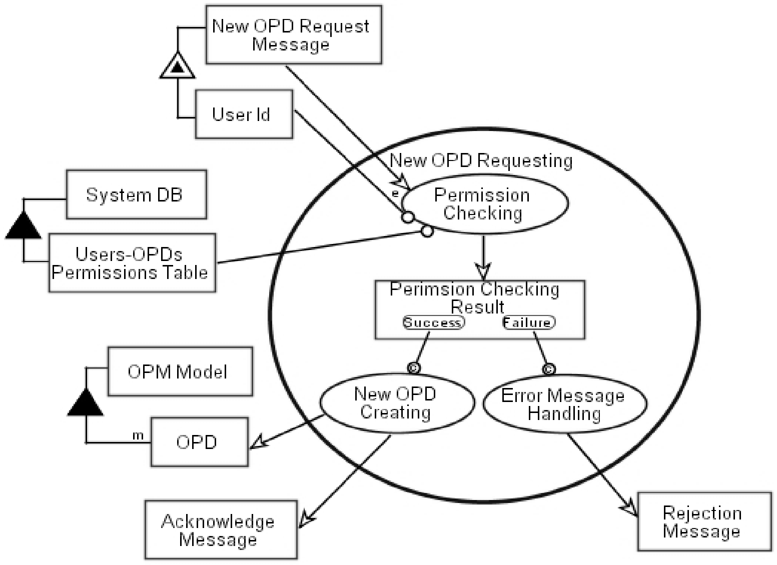

Fig. 5. A dynamic zoomed in server process for new OPD creation

Only one user, the *OPD creator*, is authorized to grant permissions to other users, but more than one user might have Edit permission to a certain OPD. The Edit permission allows the users to update, delete or create new OPD elements (except for elements that were inherited from an ancestor OPD). In such case, a conflict might occur if, for example, two clients request to update an OPM entity with two different attribute values. However, all client requests arrive at the server, which inserts them into a FIFO queue. The requests are handled one after the other, the sequence is legal, and therefore these two requests do not create any conflict. Another example is when two requests arrive at the server, the first request demands to delete an object, while the second one demands to update one of its attribute values. In this situation, the server accepts the first request, deletes the object, and updates the OPM model. The second request now results in violation of the model's integrity, so the server rejects it.

As these examples show, the server has appropriate tools to handle such cases, but conflicts can be prevented if the access control mechanism is defined in a more restricted way, i.e. by grant edit OPD permission to one user only at a time, or by replacing the Edit OPD permission with refine OPD permission, thereby preventing conflicts from occurring. Our design provides a host of access control mechanisms, letting the organization to select the set of restrictions that fits its needs. Following is an example of integrity maintenance by the server.

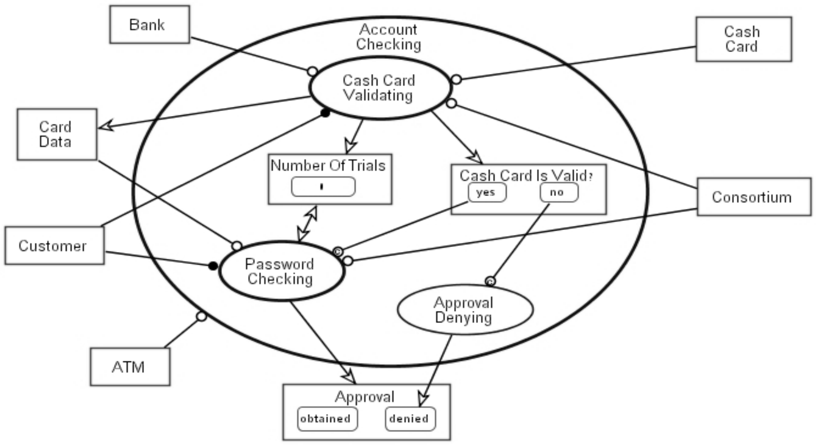

Fig. 6. The initial OPD describing an Account Checking process within an ATM system

Fig. 6 illustrates a zoomed-in process for account checking, which is an OPD taken from a description of an ATM system [2]. Fig. 7 illustrates the OPD after the server has committed an authorized user request to delete the object "**Consortium**". Fig. 8 illustrates the server rejection of an attempt to add an object that **Consortium** exhibits since **Consortium** does not exist any more in the OPM model.

The OPCATeam client is based on OPCAT (Object-Process CASE Tool) [21] and includes four major functions. The first two are inherited from the OPCAT, while the last two are OPCATeam-specific, designed to meet the collaborative system requirements. The first function is the visual support of draing and manipulating OPDs. The client offers the user a high-level and friendly graphical user interface that enables the user to model his required system fastly and clearly. The second function is the logical support in OPM. The client software prevents the user from performing illegal modeling actions that do not obey the methodology rules. The third function supplies the user with an interface to various standard collaboration utilities like server communication, chat, and presence window, while the fourth function supports the access control mechanism.

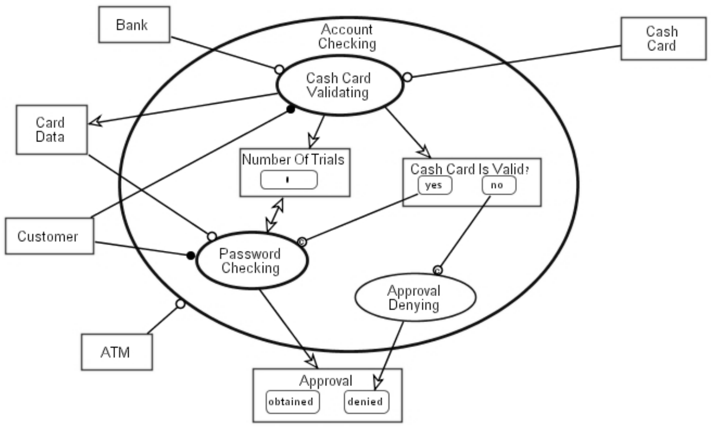

Fig. 7. The server accepts deletion of the object **Consortium** and updates the OPM Model.

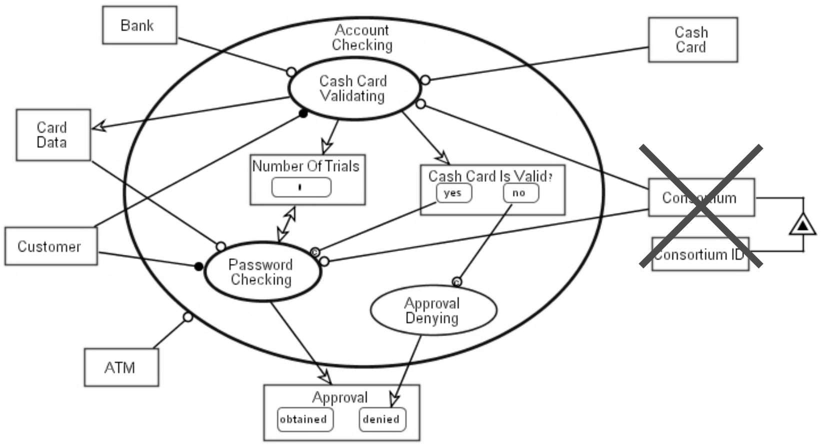

Fig. 8. The server rejects request to update a non-existing object

7 Implementation

The implementation of OPCATeam is divided into two phases. The first phase takes full advantage of OPCAT. OPCAT is able to get and return only a complete OPM model. Therefore, access control at this phase is limited to the workgroup and OPM model permission levels only. The more elaborate access control for individual OPDs will be implemented in the second phase. In the first phase, then, an additional type of permission, *Edit OPM model*, is defined. This permission temporarily replaces the Edit OPD permission. A user with Edit OPM model permission on a specific OPM model is equivalent to a user with Edit OPD permission to all the OPD-set of this OPM model. To speed up the development, users are able to work simultaneously on the same basic OPM model, but each one of them works in a separate *Session*. For example, two authorized users can start modeling from the same basic OPM model at the same time, but in two different sessions. This inevitably yields two different versions of the same OPM model, which need to be rejoined using a specific merge utility that eliminates potential lack of integrity. The goal of the second phase is to support the OPD permission level. To this end, two major functions have to be implemented. The first is relevant to the OPCATeam client granularity, which needs to be replaced by the OPD granularity. The second function is relevant to the server ability to get a partial OPM model and use it to update the complete OPM model white maintaining its integrity.

8 Conclusions

The OPCATeam architecture delivers system modeling features that meet the requirements of modern collaborative environments. It is concurrent, allowing teams of modelers to design a shared OPM model. The client-server paradigm enables real time modeling while eliminating risks of losing the model perfection. Furthermore, the architecture takes care of the interconnectivity characteristic of OPM, which poses a special challenge for collaborative OPM modeling. Additional advantages include security, central logging, and backup facilities. A disadvantage of the architecture lies in the fact that the server is the bottleneck of the system, potentially creating scalability and performance problems. Another disadvantage is that while engaged in OPM system development, users need to be connected to the server to allow for online updates and concurrent sessions. Augmenting OPCAT with the ability to add parts of the OPM model incrementally will remove this restriction as users working offline will be able to upload their updates to the model and the merge utility will take care of perfection constraints.

Our access control approach caters to the characteristics of OPM. Thus, for example, the introduction of the *refine* permission type, in addition to the standard *view* and *edit* ones, is unique to OPM. The *refine* permission allows users to refine entities generated by other users. Many established engineering disciplines apply refinement in large-scale projects or product development, but so far, software engineering and system modeling have made only limited use of this important principle. Our approach opens the door to full-scale adoption and application of refinement activities. Future research and development is planned to incorporate into the OPCATeam architecture new modules, such as workflow and peer-to-peer management. Orthogonally, principles applied in this work can be put to work in standard development approaches that use UML [23] and other modeling methodologies. We anticipate, however, that this will be more difficult since UML does not have built-in abstraction-refinement mechanisms like OPM.

References

1. Wang, L., Shen, W., Xie, H., Neelamkavil, J., and Pardasani, A., Collaborative conceptual design – state of the art and future trends. Computer Aided Design 34, 2002.
2. Dori, D. Object-Process Methodology – A Holistic Systems Paradigm. Springer Verlag, Berlin, New York, 2002.
3. Potok, T. E., Extensions to the spiral model to support joint development of complex software systems. Proceedings of the 30th Annual Southeast Regional Conference, ACM Press, 2002.
4. Usability First web site – http://www.usabilityfirst.com/groupware/intro.txl
5. IBM, Lotus Notes, http://www.lotus.com/products/product4.nsf/wdocs/noteshomepage
6. Microsoft, NetMeeting, http://www.microsoft.com/windows/netmeeting/
7. IBM, Lotus Sametime, http://www-1.ibm.com/servers/eserver/iseries/sametime/
8. SAP, http://www.sap.com/
9. Documentum, http://www.documentum.com
10. Kazanis, P. and Ginige, A. Asynchronous collaborative business process modeling through a web forum, Seventh Annual CollECTeR Conference on Electronic Commerce. Melbourne, VIC, Australia, in association with ACIS 2002.
11. CVS, http://www.cvshome.org/
12. Steinfield, C., Jang, C., and Pfaff, B., Supporting virtual team collaboration: the TeamSCOPE system. Proceedings of the international ACM SIGGROUP conference on supporting group work. ACM Press, 1999.
13. Sure, Y., Erdmann, M., Angele, J., Staab, S. Studer R., and Wenke, D. OntoEdit: Collaborative ontology engineering for the Semantic Web. Proceedings of the First International Semantic Web Conference 2002 (ISWC 2002), LNCS 2342, pp. 221-235, Springer 2002.
14. Gentleware , Poseidon for UML Enterprise Edition, http://www.gentleware.com
15. IBM, IBM Rational Rose - http://www.rational.com/products/rose/index.jsp
16. Canyon Blue Inc., Cittera UML collaborative tool. http://www.canyonblue.com/
17. Suzuki, J. and Yamamoto, Y. SoftDock: A Distributed Collaborative Platform for Model-based Software Development.
18. Yen, C., Li, W.J. and Lin, J.C., A web-based collaborative, computer-aided sequential control design tool. IEEE Control Systems Magazine, pp. 0272-1708, 2003.
19. Goldstein, H. Collaboration Nation, IEEE Spectrum, June 2003.
21. Dori, D., Reinhartz-Berger, I., and Sturm A. (2003). OPCAT – A Bimodal Case Tool for Object-Process based system development. 5th International Conference on Enterprise Information Systems (ICEIS 2003), pp. 286-291. Software download site: http://www.ObjectProcess.org/
22. Sun Microsystems, Inc. Java 2 Platform API Speciation. http://www.java.sun.com/products/jdk/.2/docs/api
23. Unified Modeling Language (UML) http://www.uml.org/
24. Soffer, P. Golany B., and Dori, D. ERP Modeling: A Comprehensive Approach. Information Systems 28, pp. 673-690, 2003.