Patterns in CSCW Modeling

Marco Paludo[1,2], *Robert Burnett*[1], *Edgard Jamhour*[1]

[1] Pontifícia Universidade Católica do Paraná - PUC-PR
Mestrado em Informática Aplicada
R. Imaculada Conceição, 1155 - Prado Velho
80215-901 - Curitiba - Paraná - Brasil
Tel/fax: +55 41 330-1669

[2] Banco do Estado do Paraná S/A - Banestado
R. Máximo João Kopp, 274 - Santa Cândida
80630-900 - Curitiba - Paraná - Brasil
Tel: +55 41 351-8770

E-mail: m.paludo@sul.com.br, robert@lambda.pucpr.br, jamhour@rla01.pucpr.br

Abstract

Modeling object-oriented systems is usually presented in a great variety of methods, however few of them consider patterns and reuse issues in its structure. The purpose of this article is to present some CSCW characteristics and put them into two different approaches of system modeling. One pure object-oriented and other pattern-oriented. The focus is to stress the particularities of groupware applications, nevertheless traditional ones can also be addressed. Some object-oriented modeling characteristics are presented initially, establishing comparisons of Peter Coad's pattern-oriented methodology with others, followed by the modeling of a Project Management case study system with both approaches. The results are presented by comparing both modeling considering the static, dynamic and functional modeling, follow by the conclusions of the work.

Key-words: CSCW, Patterns, Object-Orientation, Object Modeling, Reuse

1 Introduction

This article intends to cover some software development techniques that envisage reuse in all process phases. Among existing techniques, the use of patterns is gaining special interest from the scientific community and the development industry. Programming languages have been extensively studied and enhanced from the reuse standpoint, but the initial requirement, analysis and even design stages are focused by current efforts.

The relationship between patterns and object-orientation is very close, and the current concern of the object-oriented community is no longer only with tools, techniques, notations or code; according to Ward Cunningham, in the preface to [Fow97a], when faults occur, they are normally due to lack of experience, and patterns represent such experience.

CSCW represents a new paradigm of software development, more deeply presented later, marked by an aspect that corroborates with patterns: repetition. Every CSCW applications has

some common characteristics, like the three C's[1], that are over and over used and once modeled and defined, can easily be reused independently of the domain. Patterns are proposed to work along with CSCW to achieve high layered reuse.

The presentation of patterns is made by means of a brief approach to the context in which they fit and some definitions and characteristics of the several pattern methodologies. The project management system modeling is then presented, using pure object-orientation. Next, the same system is modeled using patterns, as proposed by Peter Coad [Coa97].

This paper attempts to identify pattern utilization aspects in CSCW systems, validating them. The comparisons with the traditional process are made, mainly concerned with the processes, the diagrams and the results achieved. Lastly some conclusions are extracted from the approaches.

2 Problem Characterization

The development of object-oriented software is fully met by several methodologies developed by authors for over a decade, and is no longer a problem. Each methodology has a particular notation, a particular emphasis and a well-defined process, but no solutions or tools are provided to obtain large-scale reuse within the problem domain and of the decisions made in other projects.

An attempt to automate reuse in the higher layer of the software process is the use of frameworks[2]. Taligent, Inc proposes a hierarchy of framework elements, the root of which is the application obtained by reuse, made up of frameworks, formed by patterns that are in turn formed by abstract classes [Tab94a].

The first publications on pattern studies called them design patterns, since they were focused mainly on implementation and abstract classes. With the evolution in the attempt to achieve a higher reuse level, patterns are applied from the very beginning of the analysis, and their notation became analysis patterns, or simply patterns.

2.1 Definitions

Gamma [Gam94] suggests that design patterns capture solutions developed and evolved throughout time, in a compact and easy-to-apply form. Each design pattern systematically names, explain and assesses an important and recurring subsystem in object-oriented design.

Taligent, Inc. [Tab94a] defines design patterns as a means to identify, name and abstract common themes in object-oriented design. They capture the hidden intentions in a design, identifying objects, how they interact and the responsibilities assigned among them, thereby generating an experience foundation on which to build reusable software, acting as building blocks that may be used to structure more complex projects.

Lajoie [Laj94] states that patterns represent a mechanism to express how components interrelate, as a high-level technique to capture and express design experiences in a proper way to facilitate reuse.

---

[1] From Communication, Collaboration and Coordination.

[2] Gamma and Johnson et al [Gam94] state that frameworks implement the general development architecture: the partition into classes and objects, their responsibilities, how they help each other in the control queue, so that the designer may concentrate his efforts only on the specific aspects of their application.

2.2 Approaches

The methodology proposed by Peter Coad [Coa97] employs patterns in two different ways. One of them is the traditional development in which the software is modeled by strategies that may be seen as the development process, just complemented by patterns. The second is the use of the patterns proposed from the initial analysis stages through the design and implementation, increasing reuse in the problem domain.

By tracing a parallel with other methodologies, we have:

- The methodology [Gam94] is concerned with cataloguing 23 design patterns, so as to reach an analysis and particularly system design standardization, until a level is reached in which the design patterns nomenclature is used as the standard language of communication between development teams, or between development teams and external units, such as users and suppliers. It is one of the pioneers and its greater emphasis is on design and implementation.
- The Pattern-Oriented Software Architecture (POSA) [Bus96] proposes a large pattern relationship similar to [Gam94], but the latter is concentrated mainly on design patterns and the former is much more abstract. POSA models its patterns according to the following structure: Context, describing situations to which a pattern may be applied; Problem, capturing the essence of a recurring problem; and Solution, showing how to solve recurring problems and deal with the forces involved in it. The solution is composed of a static part, contemplating component structure and its relationship, and a dynamic part such as the behavior in execution time.
- Analysis Patterns [Fow97a] address the domain objects in a parallel way to POSA, that is, more abstract than [Gam94], but the separation of its patterns occurs as a function of the domain, such as inventory, accounting and finances.
- Hot Spot[3] is presented in [Pre95] and differs from the previous ones because it has a well-defined process. It starts with the definition of classes and objects, and later identification of hot spots. Then the framework design/redesign is carried out, rendering the classes more flexible according to the hot spots. This last phase is recurring and stops when the hot spots reach the desired fit.
- Lastly, we have the methodology from Taligent, Inc [Tab94a] and [Tal94b] suggesting a well-defined process supported by proprietary development environment tools. The difference is the proposal to built small frameworks that will in turn, form larger frameworks, widening reuse options, once both small and large frameworks may be reused.

Coad's methodology [Coa97] is one of the most complete from the patterns standpoint, since it has a strategy process, an aspect absent from example-oriented methodologies, such as [Gam94] [Bus96] [Fow97a], but it proposes 31 patterns, absent in [Pre95] [Tab94a] [Tal94b]. In view of this wide scope, this was the one selected to develop the case study and will be detailed later on.

3 CSCW application modeling

The main objective is to assess the results of modeling a system having 'Computer Supported Cooperative Work' – CSCW characteristics, from two perspectives. One of them uses the process presented by James Rumbaugh [Rum91] with UML [Fow97b][Rat97] notation

[3] The method is called hot spot driven, but hot spot does not have is literal meaning. In this case it means 'part to become flexible'.

playing the role of pure object-oriented modeling. The second employs the pattern concepts presented by Peter Coad [Coa97].

CSCW is a scientific discipline that motivates and validates groupware projects, and according to [Lot95], groupware is the intersection of three technologies in software development, depicted in figure 1 and numbered at sequence:

1. Communication: is the transmission of knowledge by means of electronic message transmission, mainly by e-mail, in a simple and efficient way.
2. Collaboration: also aimed at enhancing teamwork, it is represented by the storage of information in shared spaces, with the necessary combinations of space (location) and times. "Shared databases facilitate collaborative interaction by providing a *virtual common workplace*, with a group-centered interface that allows participants to share information and ideas". [Lot95].
3. Coordination: represented mainly by the workflow, coordination supports company policies leading people working in cooperation to achieve certain objectives. For that purpose, the coordination leads those people to complete a structured set of tasks in a particular sequence, respecting conditions and time constrains. Thus, business processes are modeled and integrated to communication and collaboration.

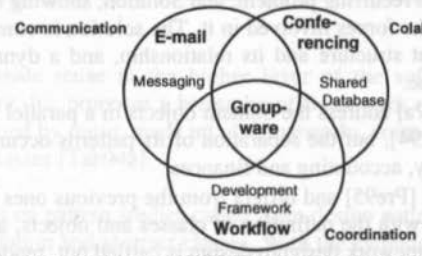

Figure 1 – Intersection of CSCW systems characteristics [Lot95].

Another proposed classification of the aspects needed to support cooperative software development is presented by [Ara97], in which communication, coordination and group memory (collaboration) have similarities with those previously presented, but a fourth perspective – perception - is introduced and defined as "fitting individual activities into the context by understanding activities performed by other people" [Ara97], allowing team members to see where their work and that of their team fit in the context.

Based on CSCW systems characteristics and on the objectives of the work, the software selected for the case study is a Project Manager intended to be generic so as to fit the several project types. Here projects are distributed to Departments as a way to control and follow-up results. They are further divided into stages, which are in turn divided into tasks. All these components have responsibles, schedules and status, which are the parameters needed to implement cooperation and communication. Collaboration takes place by controlling the access to the information in the several components, by user queries and updates that take on different roles according to the project.

4 Traditional object-oriented approach

In both approaches the emphasis is on modeling and not on implementation, that is, implementation details were disregarded, particularly in OMT[4] in which the analysis and design phases have a more clear-cut separation.

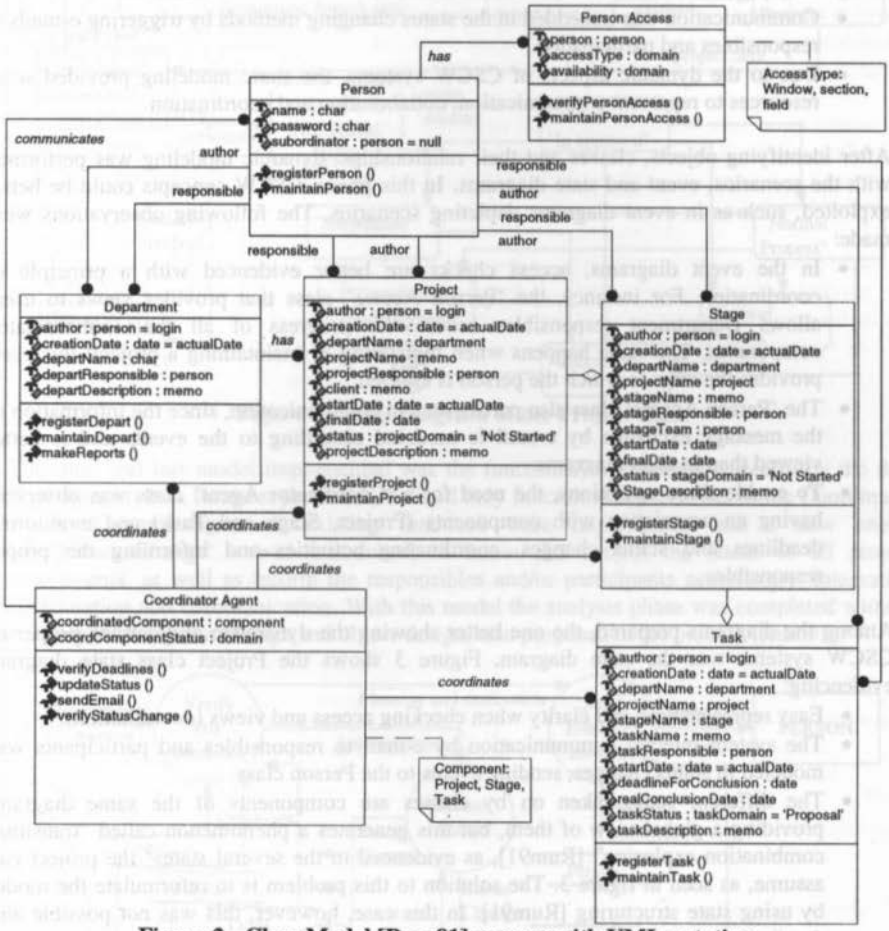

Figure 2 - Class Model [Rum91] process with UML notation.

Having as starting point the end of the requirement phase, analysis began with the static modeling phases, having as products: data dictionaries, completed, iterated and refined object model and class model. Because of the similarity between the object and class models, figure 2 shows the complete class model with the respective associations, aggregations, methods and attributes.

[4] OMT: Object Modeling Technique: object-oriented notation, analysis and design processes introduced by James Rumbaugh [Rum91].

The model presented concerns the last version, after undergoing evolution in the following phases of this process, but some particularities became evident in its conception:

- In the first iteration the Person, Department, Project, Stage and Task classes were identified. The 'Person Access' class was conceived to meet the requirement of views to implement collaboration, but is too abstract.
- Communication was imbedded in the status changing methods by triggering e-mails to responsibles and participants.
- Due to the dynamic aspects of CSCW systems, the static modeling provided scant resources to represent communication, collaboration and coordination.

After identifying objects, classes and their relationships, dynamic modeling was performed with the scenarios, event and state diagrams. In this phase, CSCW concepts could be better exploited, such as in event diagrams depicting scenarios. The following observations were made:

- In the event diagrams, access checks are better evidenced with a principle of coordination. For instance, the 'Person Access' class that provides views to users allows Department responsibles access the progress of all their subordinated components. The same happens when registering or maintaining a project: this class provides the view to which the person is entitled.
- The 'Person Access' class also participates in communication, since the information in the message exchange by e-mail is modeled according to the events and is better viewed than in class diagrams.
- To enable these operations, the need for a 'Coordinator Agent' class was observed, having an association with components (Project, Stage and Task) and monitoring deadlines and status changes, coordinating activities and informing the proper responsibles.

Among the diagrams prepared, the one better showing the dynamism and actions proper of CSCW systems was the state diagram. Figure 3 shows the Project class state diagram evidencing:

- Easy representation and clarity when checking access and views (coordination).
- The system internal communication by e-mail to responsibles and participants was modeled in status changes, sending events to the Person class.
- The different states taken on by classes are components of the same diagram, providing a global view of them, but this generates a phenomenon called "transition combination explosion" [Rum91], as evidenced in the several states[5] the project can assume, as seen in figure 3. The solution to this problem is to reformulate the model by using state structuring [Rum91]. In this case, however, this was not possible and the diagram remained hard to read.

[5] The possible project statuses are: 'new project', 'not started, 'in approval, 'normal process', 'critical process', 'completed and approved', 'cancelled', 'suspended'.

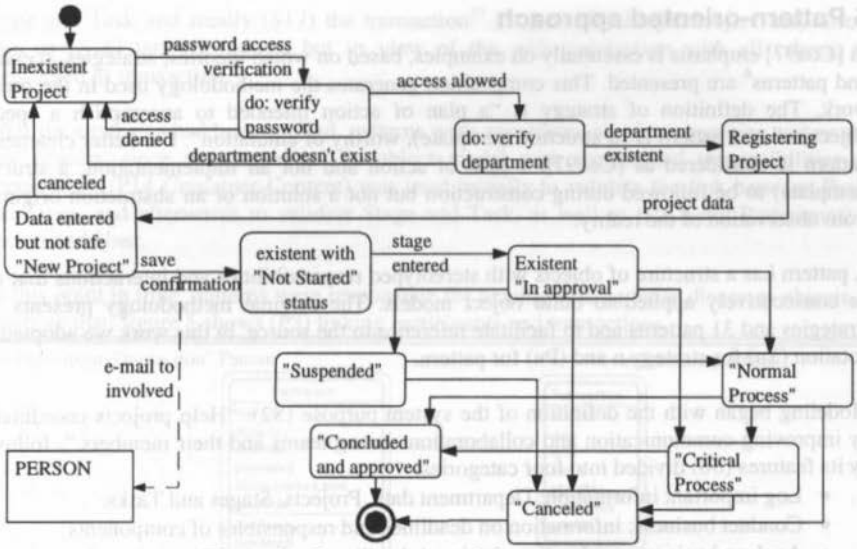

Figure 3 - State diagram of the Project class.

The third and last model implemented was the functional, represented basically by the data flow diagram that, for most processes, defined only functional dependence. In the Coordinator Agent diagram, figure 4, existing functions were implemented and a new one - 'checkStatusChange' – was introduced, to know and control the status of all project components, as well as inform the responsibles and/or participants accordingly, integrating coordination and communication. With this model the analysis phase was completed without entering in the system design and object design, also proposed in the process [Rum91].

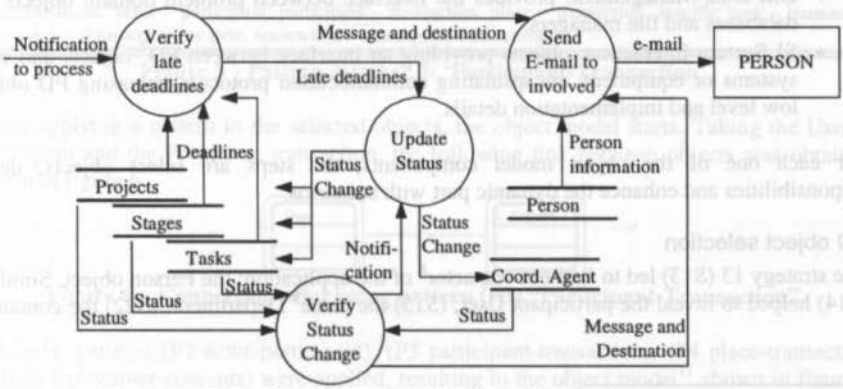

Figure 4 – Data flow diagram of the Coordinator Agent process.

5 Pattern-oriented approach

In [Coa97] emphasis is essentially on examples, based on which theories, strategies, scenarios and patterns[6] are presented. This composition generates the methodology used in the present work. The definition of strategy is "a plan of action intended to accomplish a specific objective" and pattern is "a structure (template), worthy of emulation". For better clearness, a pattern is considered as [Coa97]: a plan of action and not an implementation, a structure (template) to be followed during construction but not a solution or an abstraction originated from observation of the reality.

A pattern has a structure of objects with stereotyped responsibilities and interactions that may be consecutively applied to build object models. The original methodology presents 148 strategies and 31 patterns and to facilitate reference to the source, in this work we adopted the notation (Sn) for strategy n and (Pn) for pattern.

Modeling began with the definition of the system purpose (S2): "Help projects coordination by improving communication and collaboration among teams and their members.", followed by its features (S6) divided into four categories:

- Log important information: Department data, Projects, Stages and Tasks.
- Conduct business: information on deadlines and responsibles of components.
- Analyse business results: list of missed deadlines and calculation of results, assessing the quantity of activities per responsible.
- Interaction with systems: not applicable.

The next attempts were separated into four object categories (S25):

- PD Problem Domain: contains the objects corresponding directly to the problem domain and are technology-neutral.
- HI Human Interaction: provides the interface between the problem domain and people (usually window and report objects)
- DM Data-Management: provides the interface between problem domain objects and databases and file managers.
- SI System-Interaction: objects providing an interface between PD[7] objects and other systems or equipment, encapsulating communication protocols, releasing PD objects low level and implementation details.

For each one of the object model components, the steps are select objects, define responsibilities and enhance the dynamic part with scenarios.

PD object selection

The strategy 13 (S13) led to find out one actor[8] of the application: the Person object. Similarly (S14) helped to reveal the participant User, (S15) the place[9] Department; (S22) the containers

[6] The term adopted for this paper, for the sake of standardization, is 'pattern(s)'; in the literature there are references to: pattern(s) [Coa97] [Bus96], analysis patterns [Fow97b] [Fow97a], design patterns [Gam94], [Pre95], pattern language and object-oriented patterns.

[7] Later mentions are made with the initials PD, HI, DM, and SI.

[8] The actor might be a person, an organization or any other agent participating in one or more ways throughout time. A participant acts in a specific way, plays a role or fulfills a specific mission [Coa97].

[9] Places are physical locations where objects rest or contain other objects [Coa97].

Stage and Task and finally (S17) the transaction[10] Project. The Project object may also be seen as (S16) tangible things, but in view of the wide interaction with all others, was considered as transaction.

After the strategies had been applied, patterns were considered to select, organize and propose new objects, since they represent an object model with stereotyped responsibilities and interactions. (P14 Container-Content) was used initially to validate the link between Project and Stage, and afterwards to validate Stage and Task, as well as (P2 Actor-Participant) for Person and User.

At this point in time, patterns were responsible for helping linking those disperses objects. To illustrate that, figure 5 shows (P3) linking participants and transactions.

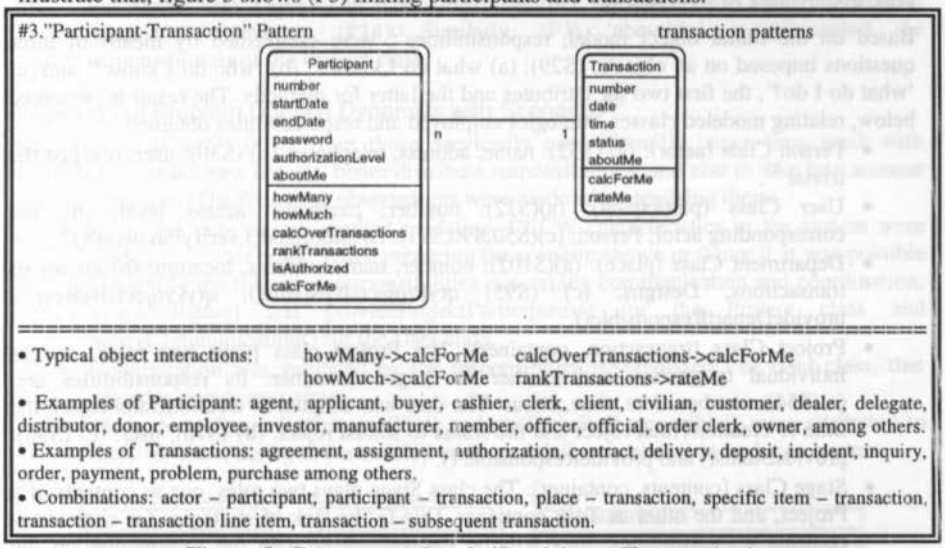

Figure 5 - Pattern number 3: 'Participant-Transaction'.

When applying a pattern to the selected objects, the object model starts. Taking the User as participant and the Project as transaction, the following link between objects was obtained, shown in figure 6.

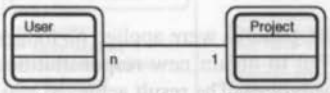

Figure 6 – Connection after using pattern (P3) "Participant-Transaction".

Similarly, patterns (P2 actor-participant), (P3 participant-transaction), (P4 place-transaction) and (P14 container-contents) were applied, resulting in the object model[11] shown in figure 7.

[10] A transaction is the *log* of some significant event, knows who are the participants and makes calculations concerning the event [Coa97].

[11] Notation: a double frame represents a class with objects and a simple frame a class having no direct correspondence to objects. Multiplicity is represented together with the object involved. In figure 9, multiplicity n shows that the User 'knows" n Designs.

Since propositions are being dealt with, a certain multiplicity was adapted to the domain in the final model.

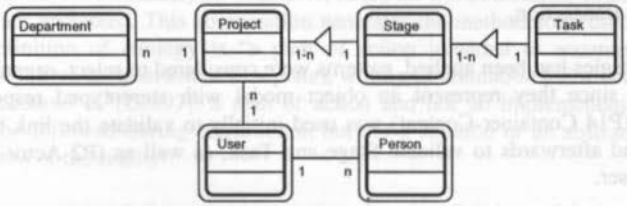

Figure 7 - Initial object model obtained using patterns.

Responsibilities of PD objects

Based on the initial object model, responsibilities[12] were established by means of three questions imposed on all objects: (S29): (a) what do I know?', (b) 'who do I know?' and (c) 'what do I do?', the first two for attributes and the latter for methods. The result is presented below, relating modeled classes, strategies employed and responsibilities obtained:

- Person Class (actor): (a) (S52): name, address, telephone; (b) (S30): user; (c): just the trivial [13].
- User Class (participant): (a)(S32): number, password, access level; (b): the corresponding actor, Person; (c)(S50,S90,S91): isAuthorized(),verifyPassword().
- Department Class (place): (a)(S102): number, name, address, location; (b) knows its transactions, Designs; (c) (S95) qtyProjectsByStatus(), qtyProjectsByResp(), provideDepartResponsible().
- Project Class (transaction, container): The Project class plays two roles, one as individual transaction and another as Stages container. Its responsibilities are: (a)(S54): number, date, time, status. The date was adapted to creationDateProject, the time to creationTimeProject and the status to statusProject. (b) (S76): user. (c) (S96): provideStatus() and provideResponsible ().
- Stage Class (contents, container): The class Stage plays two roles, one as contents of a Project, and the other as Task container. Due to the lack of strategies for containers-contents responsibilities, they will be defined solely with the information on the problem domain. (a); (b)Project, Task and User; (c) provideStageStatus() and provideStageResponsible().
- Task Class (contents): Similar to Stage. (a); (b)Stage and User; (c) provideTaskStatus() and provideTask-Responsible().

To complement responsibilities, patterns were applied no longer with the objective of defining object links and hierarchies, but to obtain new responsibilities through the class of patterns possessing stereotyped responsibilities. The result achieved was the following:

- By applying pattern (P1 Collection-Worker), called fundamental pattern and the origin of all others, and assuming Project as collection and Stage as worker, obtaining qtyStagesByStatus()[14] and qtyStagesByResponsible() for the Project class, starting from the stereotyped method 'calcOverWorkers'. For the class Stage to play its role,

[12] Responsibilities of an object: its knowledge about itself, about others, and what it is capable of doing.
[13] [Coa97] considers operations read(), save(), search(), initialize and similar ones as trivial basic services (S89 basic services) and is of the opinion that they should not be included in the object model.
[14] In later model iterations, the term 'qty' was removed because methods return not only the quantity but also information on components that meet arguments.

the existing provideStageStatus () and provideStageResponsible() were validated by the stereotyped method 'calcForMe' linked to the collection class.

- Similarly applied to Stage-Task, it generated qtyTasksByStatus() and qtyTasksByResponsible() for the Stage class starting from 'calcOverWorkers' and validated the already created methods provideTaskStatus() and provideTaskResponsible().
- Pattern (P3 Participant-transaction), when applied to User and Project classes, generated the qtyProjectByResponsible() method in the User class, by means of the stereotyped method 'calcOverTransactions'. This method communicates with the equally named one in the Department class.
- The application of (P14 Container-contents) generated name, number and description for Stages and Tasks. The other responsibilities of Stages and Tasks came from 'aboutMe' found in (P14). Similarly, (P3) 'aboutMe' complemented the responsibilities of the Project class.

Working out Problem Domain Dynamics with Scenarios[15].

Scenarios in [Coa97] differ from those previously used [Rum91], since they work with patterns to find out new objects, better distribute responsibilities, and also to take into account system dynamics. The following observations were made when applying them:

- As in the pure object-oriented modeling, CSCW characteristics in the system were evidenced at this point. When preparing the scenario shown in figure 8, it was possible to identify the following responsibilities concerning communication and coordination: monitorStatus() and provideProjectParticipants() for the Project class and notifyCommunication() for the User class.
- Collaboration was validated by the responsibility isAuthorized() of User class, that provides component views.
- Methods qtyStagesByResponsible() and qtyTasksByResponsible() were attributed to the User class, acting in partnership with provideResponsible() in each component.

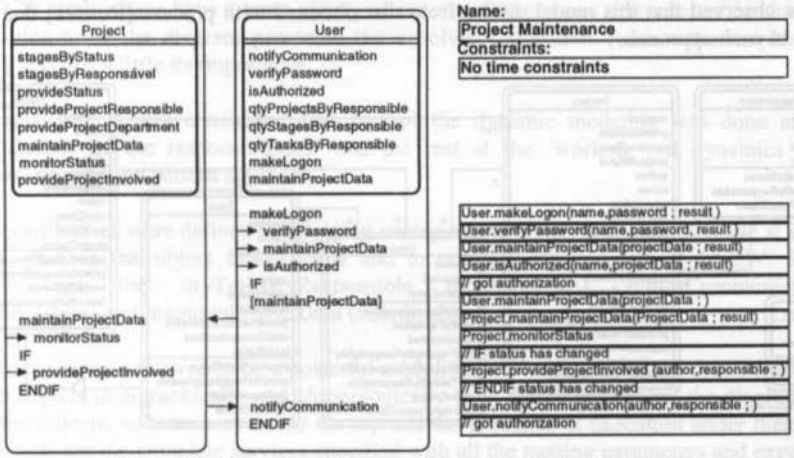

Figure 8 – Scenario of one problem domain (PD) objects.

[15] [Coa97] A scenario is a chronologically ordered sequence of interactions between objects, containing messages that objects send and receive, and the methods consequently invoked.

Trying to keep modeling in the same layer presented in section 4, not taking design into account, scenarios were developed solely for essential activities and modeling of the other components, HI, SI and DM were made in a superficial manner.

HI Objects

Human Interaction (HI) objects are typically windows and reports. To select them (S27) was employed to create windows for all PD objects, (S28) for the logon window, (S29) for configuration and (S31) for reports. In this model, relationships are described by means of attributes in classes and not graphical links, as happened with PD objects.

When modeling the scenarios of these objects, the following were obtained:

- HI scenario 'projectWindow' was subdivided into four sub-scenarios: 'prepare file', 'file items', 'perform maintenance' and 'distribute e-mail'. The maintainProjectData method is used both in the Project and User classes. According to [Coa97], the inclusion of these methods in the PD model is optional, but for illustration purposes it was included in the respective classes.
- In the 'distribute e-mail' scenario, the monitorStatus(), provideProjectParticipants() and notifyCommunication() methods were employed.

SI Objects

No SI objects were identified in this system.

DM Objects

Each data management object (DM) possesses a corresponding PD, since its objective is to ensure the persistence in non-volatile media and provide query needs by means of a collection of objects[16](S32). DM objects and their responsibilities were selected and applied to scenarios, but they did not imply in modifications to the problem domain model.
Figure 9 shows a PD object model, after being complemented throughout all process phases. It was observed that this model differs from that obtained with process [Rum91] due to the focus in each approach.

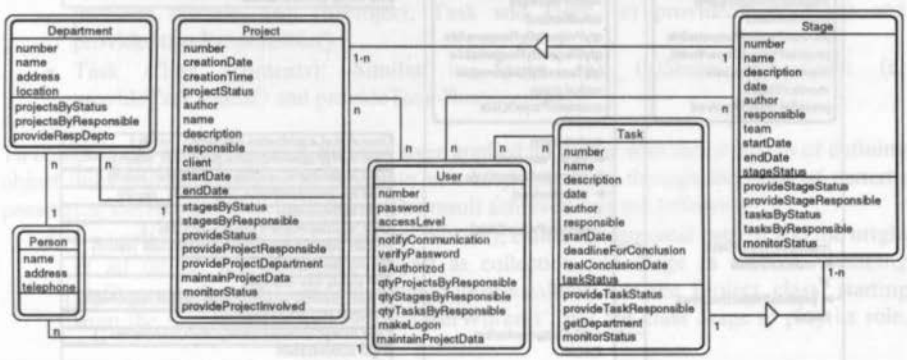

Figure 9 – Object model according to process [Coa97].

[16] Concept similar to the one presented in the structural design pattern PROXY, in [Gam94].

6 Comparisons

Coad, in [Coa97], states that its not productive to separate analysis and design models, affirming that the developer needs to be free to consider both analysis and design matters as they come up. The separation proposed is by concerns: problem domain, human interaction, system interaction and data management. In order to trace some common aspect for comparison, in this article, the OMT steps were selected as guidelines and the pattern approach deliveries were spread by them.

6.1 Static modeling

The object-oriented approach showed to be very poor to represent the communication, collaboration and coordenation features of CSCW. Some services were embedded inside others, like the 'send e-mail trigger' (communication) in the statusChange service. Other objection raised was that all the models started from scratch.

The pattern approach also suffered to cover these requirements, however the stereotyped services, attributes and relationships have helped this phase. As an example, the password/ authorizationLevel attributes of the Participant in (P2 Actor-Participant) has sketched some nuance of collaboration and startDate/endDate/calcOverTransactions in (P3 Participant-Transaction) some nuance of coordination.

At this point patterns showed more user-friendly tc achieve the proposed objectives of modeling CSCW applications.

6.2 Dynamic modeling

The dynamic analysis showed better opportunity to represent CSCW features for both approaches. The object-oriented, initially, provided the scenarios and the event diagram, proving to be a good tool for coordination and communication, with the PersonAccess class providing the views of the users and also the message exchange, modeled according to events. The state diagrams provided a clear picture of the access and views (coordination) but, seen that CSCW applications have a great variety of states and sometimes even created or changed at execution-time, the diagram presented the unsolved "transition combination explosion" anomaly and lost a little its importance.

For comparison, it was considered that part of the dynamic modeling was done at the 'identifications of the responsibilities' and the rest at the 'working out dynamics with scenarios' steps of the pattern approach.

The responsibilities were defined by applying strategies and patterns, as already made at static analysis, making the object model grow and to consider all elements of CSCW. Some examples are the qtyTasksByResponsible (collaboration), notifyCommunication (communication) and maintainProjectData (coordination).

The pattern scenario showed to be a powerful modeling tool because it shows the interactions between objects in a graphically and chronologically way. At the left side of the diagram are placed the objects, with the services at the top and the sequence of execution under them. At the right side are the complete services specified with all the passing parameters and expected results.

Coad, in [Coa97], considers the use of state transition diagrams inadvisable and suggests encapsulating state-dependency within an object, meaning that the object responds differently depending on its state. Regarding to state attributes, the objective is to include 'applicable states' in its descriptions and to state services, include 'pre/postconditions, trigger and terminate' states.

6.3 Functional modeling

The OMT approach treats the functional model as the results of a process without specifying how or when they were made. The constraints are also present at it. When the data flow diagrams were made, some methods turned out to be necessary like checkStatusChange at the Coordinator Agent class, however the contribution for the whole project was scant.

The pattern approach doesn't treat the functional aspect separated and argues that "if a service is so complex that you need a data-flow diagram to describe it, you've got a partitioning problem..." [Coa97]. To deal with functional matters, the pattern scenarios are provided with constraint fields and with some control structure that can be inserted at the execution section. The control structures are If/Else/Endif, While/Endwhile, Do/Enddo, Case/ Endcase and Start_Task /Stop_Task. Another feature to supply functional modeling is the possibility to create hierarchical scenarios, linking the more with the less detailed ones.

6.4 General observations

The object-oriented approach as is, is suitable for CSCW modeling because it can address most of its characteristics, mainly at the dynamic analysis, but the process doesn't assist the developer much with the components of the application. The observation done is that the developers build the models from scratch, what causes to waste unnecessary efforts. In addition to this, the possibilities to reuse the models for future projects are rare because too many conditions must be the same in order to succeed.

One of the appeals of patterns is that the developer starts the modeling with some footsteps to follow, taking advantage of the stereotyped services, attributes and relationship. This improves the time-to-market and quality of models.

The reuse approach of patterns is that systems can have similarities with 'problem domain' characteristics and different 'human interfaces' or vice-versa. The same for the other components: 'data management' and 'system interaction'. Partitioning the models this way allows the system to be conceived with the "design-for-reuse" [Ygl98] feature, where the focus is not just the actual project, but the advantage others would take of these models. This represents an advantage over the reuse obtained with pure object-oriented, leveraging the speed of CSCW application modeling that is marked by a great number of common points between projects.

Some of these points, for example, can be User and Project, which will probably be part of many groupware applications[17]. The developer can directly apply the pattern (P3 Participant-Transaction) or reuse the object model result conceived at this article. Other examples are Person-User (P2 Actor-Participant) and Department-Project (P4 Place-Transaction).

[17] Some examples of services that may also be found in other projects are: monitorStatus(), provideProject-Participants() and notifyCommunication().

One drawback of the pattern approach is the absence of state diagrams, widely used by many object-oriented methods. However, if the state issue is not well addressed by the patterns approach, a traditional state diagram can be added to complete the description of the problem. This need was not present in this case study because some attributes and services (e.g. provideStatus, monitorStatus) were enough to describe the states of the CSCW problem.

One clear bottleneck of patterns is the deep knowledge one must have of the existing patterns catalogue and structures, and even of those proposed for a specific domain. The lack of this knowledge can lead the developer to conceive ineffective models, not achieving the reuse objectives.

7 Conclusions

The development of object-oriented software points towards a greater application reuse in all process phases. The use of patterns, though originated in the 80's, now arises great interest in academic and business circles, particularly in the object-oriented development, where its concepts may be fully applied, achieving better results.

This paper presents the aspects of object-oriented modeling in two different ways: one as a purely object-oriented process and the other proposing the use of patterns. In both it is seen that the CSCW system characteristics are better evidenced in the phases concerned with the system dynamics, even causing changes in the static modeling to provide the proper support to these characteristics.

During the presentation of the Project Management system, observations on each phase are made, mainly concerning communication, coordination and collaboration, considered here as basic components of CSCW systems. The benefits of applying patterns throughout the whole project, allowing large-scale reuse in the higher layers of the process, are demonstrated.

Further research on patterns in the CSCW area will proceed with the implementation of the Project Management system, completing the whole process life cycle. The objective is to create a number of patterns specially addressing reuse of CSCW applications. It is hoped that the use of CSCW patterns results in more efficient development of a large family of CSCW applications. Another future effort may be applying formal metrics in CSCW applications developed with patterns, in order to accurately quantify the benefits in terms of time and quality.

References

[Ara97] Araujo, R.; Dias, M.; Borges, M., "Computer Added Cooperative Software Development: Classifications and Proposals", In Proceedings of XI Brazilian Symposium on Software Engineering XI, Fortaleza-Brazil, 1997.

[Bus96] Buschmann, F; Meunier, R.; Rohnert, H.; Sommerland, P.; Stal, M., "Pattern-Oriented Software Architecture: A System of Patterns", John Wiley & Sons, Chichester, 1996.

[Coa91a] Coad, P.; Yourdon, E., "Object-Oriented Design", Yourdon Press, 1991.

[Coa91b] Coad, P.; Yourdon, E., "Object-Oriented Analysis", Yourdon Press, 1991.

[Coa97] Coad, P.; Mayfield, M., "Object Models: Strategies, Patterns & Applications - Second Edition", Yourdon Press, New Jersey, 1997.

[Fow97a] Fowler, M., "Analysis Patterns - Reusable object models", Addison-Wesley, Menlo Park, 1997.

[Fow97b] Fowler, M.; Scott, K., "UML Distilled. Applying the Standard Object Modeling Language", Addison-Wesley, Reading, 1997.

[Gam94] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, 1994.

[Laj94] Lajoie, R.; Keller, R., "Design and Reuse in Object-Oriented Frameworks: Patterns, Contracts, and Motifs in Concert.", In Proceedings of 62nd Congress of the Association Canadienne Française pour l'avancement des Sciences (ACFAS), Montreal, 1994.

[Lot95] Lotus Development Corporation. "Groupware: Communication, Collaboration and Coordination", Cambridge, 1995.

[Pre95] Pree, W., "Design Patterns for Object-Oriented Software Development", Addison-Wesley, Wokingham, 1995.

[Rat97] Rational Software Corporation, "UML v 1.1 Notation Guide", www.rational.com, 1997.

[Rum91] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Frederick, E.; Loresen, W., "Object-Oriented Modeling and Design", 1991.

[Tab94a] Taligent, Inc, "Building Object-Oriented Frameworks", Taligent, Inc White Paper, 1994. http://www.taligent.com.

[Tal94b] Taligent, Inc, "Leveraging Object-Oriented Frameworks", Taligent, Inc White Paper, 1994. http://www.taligent.com.

[Ygl98] Yglesias, K, "IBM's Reuse Programs: Knowledge Management and Software Reuse", In Proceedings of Fifth International Conference on Software Reuse (ICSR5), Victoria, 1998.