Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'92), Vancouver, ACM Press, pp. 359-376, October 1992.

Object-Oriented System Modeling with OMT

Bernd Bruegge, Jim Blythe, Jeffrey Jackson and Jeff Shufelt
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA. 15213

Abstract

We describe our experience with the object-oriented design methodology OMT [19] applied to a project in an undergraduate software engineering course at Carnegie Mellon University. The project involved 30 students previously unfamiliar with object-oriented modeling. They designed and implemented a system of 125 classes (27,000 lines of C++ and C code) in the relatively short time of 15 weeks. We describe the overall structure of the project and the system model, the usefulness of OMT and its impact on communication, and discuss some of the problems encountered during the development of the system.

1 Introduction

In this paper we describe the use of a promising object-oriented modeling technique applied to a moderately complex software system. The system was built in the context of an undergraduate software engineering course at Carnegie Mellon University. The course is offered to junior and senior undergraduate students with a wide range of programming capabilities but, in general, no previous exposure to object-oriented design.

One of the goals of the course is to teach the students how to work together in a team to develop and deliver a single large system to an external client. The emphasis on teamwork is becoming the preferred paradigm for teaching project-oriented software engineering courses [14][18][21]. The project style and size in these courses range from small problems carried out by teams of 3-6 students [15], to single projects with an external client that require the participation of the entire class. Successful single project courses with 15 and more students have been reported [6][22].

With the growing popularity of our course, we were facing the problem of maintaining the single project idea with a large number of students (30 or more). Finding topics for such a large team is not a problem, but the combined effect of dealing with beginners and the potential communication overhead makes the manageability of such a project questionable. Several alternatives such as cutting the class size down or partitioning the class into several medium-sized projects were considered briefly. However, we are convinced that a single project course has three main advantages over a course with several small project teams: first, the students are given a much more realistic exposure to the problems of teamwork and project organization; second, they have to analyze and design a problem which is so complex that it cannot be done by a single student in the given time frame; and third, they have to deal with integration and delivery problems in a realistic way. We therefore decided to go with a single project course and to address the communication and management issues which would result.

The most important goal of our course is to have the students appreciate the problems of analysis and design of a large system. Given such a large group, can we reach every project member in a balanced manner? Design is inherently difficult, but we believe that the time has come for analysis and design to be taught to, and understood by, a wider audience than the usual few super-designers.

In pursuit of this goal, one of the central issues in a software engineering project course is the choice of the software development methodology. What language is used to express a design? Is it supported by tools? Is it possible to keep a design consistent and communicate it to others? Does it lead to a successful implementation? In our courses we have been constantly experimenting with different answers to and levels of importance of each of these questions. In the fall of 1989, for example, we used object-based design without any formal design tools at all [6]. Instead we asked our students to reuse existing software facilities as much as possible to create a bridge between electronic mail and facsimile transmission provided by a special fax board running on a personal computer. The students succeeded in developing a running prototype, *Workstation Fax*, which they demonstrated in a formal presentation and acceptance test at the client's site. The prototype consisted of 10,000 lines of C code and was used by the client as the basis of the design for an Email-to-FAX system that is now used on a daily basis on campus.

In the spring of 1991 we became more ambitious. The students had to build an interactive shared data base, *Interactive Maps,* providing a user with tools for navigation through the CMU campus at different levels, from the level of building down to individual classrooms or offices. It is also possible to obtain schedules and visual information about the occupants of a room. In this course we required the use of a formal specification language [10] and a CASE tool, Software Through Pictures (StP) from Interactive Development Environments [12]. We asked the students to describe the requirements analysis and design with a mixture of two design approaches, data flow-oriented design [26] with the SA/SD (structured analysis/structured design) tools available in StP, and object-based design using abstract data types [17]. Data flow-oriented design has been successfully applied to strictly sequential informations systems, but is not as useful for the design of interactive systems. For example, the students were not able to express the user interface for *Interactive Maps* in a structured design notation. Another problem with the SA/SD development methodology is its inability to express the results of the analysis and design phase with a uniform notation. We found it difficult and time-consuming to convince the students that it was necessary to change not only the terminology but also the model when moving from analysis to design.

We therefore started looking for a method that can be used for consistent modeling during the whole development of the system. Object-oriented methodologies looked especially promising to us, but these methodologies are currently appearing at a fast rate and not much experience had been reported in a teaching environment. Werth [23] describes a project course with project teams of 5-6 students using the MPW/MacApp programming environment. The course was given high marks by the students, but no experience in the use of analysis or design tools is reported. A comparison of recent object-oriented methods [1] bases the comparison on features such as concepts, models, and support of the process concept, but not on the actual experience with the methodologies. From the methods discussed, we selected the Object Modeling Technique (OMT) because of its strong support for analysis and design as well as implementation and its support by a textbook [19] and a tool, OMTool [20]. We did not choose the other methods for various reasons. Wirfs-Brock's methodology [24] also looked very attractive but is not supported by a tool. Buhr's methodology [5] is tied to Ada packages. Most of the students entering our course don't have extensive programming experience in Ada and, while the same could be said for C++, we felt the use of Ada would lead to a steeper learning curve. HOOD [11] still relies on SA/SD for the requirements analysis. Booch's methodology [2] looked more powerful than OMT, but we chose OMT because it seemed to strike the right balance between simplicity and expressive power, which implied that it would be teachable within a semester and yet still be useful as a design tool for a moderately complex project[1].

The other problems we faced by admitting a large number of students to the project were project management related. Many instructors object to the single project course because they feel it is too difficult to manage a single large team; in particular, increased communication problems arise. Students are taking other classes (typically 3-4 additional classes during the semester) and the instructors are usually involved in research projects. However, such time constraints are not limited to an academic setting; they apply to projects outside the classroom as well. Examples include software developers working on several projects simultaneously, or working only part-time on a single project. To facilitate communication between "part-timers" we encouraged the students to make heavy use of electronic bulletin boards. Bulletin boards have proven to be very powerful communication tools that add significantly to the ability to realize single project courses.

1. Booch has recently introduced an extended notation of the Booch method which includes a subset ("Booch lite") [3]. The extended notation comes with a tool called ROSE.

In the following we report on our experience with OMT in developing a moderately large system, *Interactive Pittsburgh*, in a single project course in the Fall of 1991. A comparison between the bulletin board communication patterns of the Spring 1991 and Fall 1991 classes indicates that the use of OMT led to a substantially smaller proportion of intergroup communication during system integration than did the use of a mix of SA/SD and object-based design. This in turn seems to imply that the object-oriented developers encountered fewer of the classic —and costly— surprises which so often arise during integration due to poor intergroup communication in the early phases. Thus we have some evidence that OMT enhances communication even among novice users of the methodology, with a resulting increase in productivity.

We therefore argue that a single project course using object-oriented system modeling can comfortably support a relatively large group of students, and provides a good means for educating students about the issues that arise in large software system development. Because of the size of the project and the use of real clients and real data, we believe that even though our experience is derived from an academic environment it is valuable for a wider audience of system designers currently contemplating a change to object-oriented design in general and/or OMT in particular.

The paper is organized as follows. Section 2 describes the problem statement issued to the students at the beginning of the project. Section 3 describes the project structure, in particular our decision to include a "tools" group to support the formulation of abstractions needed by other groups. Section 4 explains the system model produced by the students and characterizes our experiences with OMT in the context of communication, both among developers and between developers and clients. Section 5 compares the project communication bandwidth of *Interactive Maps*, developed with SA/SD and object-based design, with that of *Interactive Pittsburgh* developed with OMT. Finally, Section 6 summarizes our experience and contains ideas for further improvements.

2 Problem Statement

Formulating a problem statement for an object-oriented software engineering course is an interesting task in its own right. Several factors come into play when choosing a project for a semester-long software engineering course.

First, and certainly most important for an object-oriented software engineering class, a potential task should readily lend itself to an object-oriented design methodology; the task should have many entities with multiple relationships, so that any object-oriented solution will be a challenge. The hope is that students will observe firsthand the relative ease with which an object-oriented solution can be constructed as compared with more standard structured programming techniques. On the other hand, the task should be sufficiently complex that students can appreciate the difficulties involved in building a large system, even with the benefits of an object-oriented approach.

Second, a potential task should have real-world aspects, if at all possible. Software engineering, like many other disciplines, can easily become a dry and sterile topic in the classroom; links to real-world problems and situations provide students with an opportunity to ground software engineering techniques in problems of practical interest to users, and they add a level of interest to motivate students. Students become more enthusiastic about the course when real clients with real problems are involved.

Third, the problem should be ambitious in size and scope, for two reasons. Students who have not worked in the software development industry will benefit from working on a problem sufficiently large that no single person could solve it alone in a semester. Large problems force the students to work together in teams (which is likely a new experience for many) and to work across teams to negotiate system and subsystem designs and functions. Further, an ambitious problem forces students to confront reality: certain portions of a project may not be feasible under time and budgetary constraints. A highly ambitious task can be used to place students in situations where they must evaluate the trade-offs that are part and parcel of successful requirements analysis, design, implementation, and system integration.

The *Interactive Pittsburgh* project was devised to satisfy these goals for a one-semester object-oriented software engineering project course administered in the fall of 1991 at Carnegie Mellon University. A description of the problem and its utility as an educational experience follow.

2.1 Interactive Pittsburgh

As we noted, a useful problem statement for an object-oriented software engineering project course must have at least the following attributes: easy translation to the object-oriented paradigm, real-world connections, and an ambitious goal. We found such a problem in the domain of city planning and cartography.

There is an increasing use of cartographic data for city and county planning, operations, and historical records. For example, land ownership records have traditionally contained detailed map survey information showing lot boundaries, sizes, and right-of-ways. Such information is useful for zoning, establishment of property taxes, and for maintenance and repair of water and sewage lines by municipalities and of electric and gas lines by utility companies.

Traditionally, much of this data was represented by paper maps which were manually annotated and revised. Currently, there is great interest in digital representations of this information to allow for data sharing between various city and county departments to allow for more timely update and easier maintenance, and to provide services based on spatial data that can not be obtained using paper map products. Some examples of these municipal services include routing of fire, police, and emergency medical units to accident sites; city planning for road maintenance; and land-use planning, particularly for large-scale construction and development.

Interactive Pittsburgh, the project we devised for the students, involves the construction of a geographic information system which combines digital road network data and bus route data to provide an interactive browser for the city of Pittsburgh. In addition, *Interactive Pittsburgh* was to be implemented as an extension of *Interactive Maps*, a browser for Carnegie Mellon University created by the Spring 1991 software engineering class. The major goal of the *Interactive Pittsburgh* project was to provide a set of interactive tools for answering questions regarding building and street locations, bus routes, and bus stops, and to provide access to the tools in *Interactive Maps* for querying campus map information.

Interactive Pittsburgh utilizes real-world digital road network and bus data. In general, such data comes from many different sources, often in incompatible formats, and is typically not structured in a way that accommodates easy manipulation or query. The data we used for the project was graciously provided by two sources: the Allegheny County Planning Department, which provided the road network data based on Tiger files maintained by the United States Census Department; and the Planning Department of the Port Authority of Allegheny County, which provided the bus data.

2.2 Requirements

Our project goal was the production of a system that would be usable by the campus community. First, we expected it to function as a stand-alone service that the campus community would want to use. Second, we would ultimately want to incorporate the project into a new campus-wide information system managed by the library automation group at Carnegie Mellon University, which utilizes commercial databases and a window-based workstation environment. In support of this requirement, excellent documentation for users of the system was mandatory.

As indicated above, the students were not asked to design *Interactive Pittsburgh* from scratch; they were asked to modify and extend the design of the *Interactive Maps* system. The *Interactive Maps* system allowed users to navigate around a multi-level map of Carnegie Mellon University and display information about various locations and rooms, such as phone numbers and digitized photographs of the occupants. The *Interactive Maps* system had several deficiencies which the *Interactive Pittsburgh* project was expected to address. Among the deficiencies that were actually addressed in *Interactive Pittsburgh*:

- *Interactive Maps* only dealt with one floor of one building on campus. The *Interactive Pittsburgh* system can handle potentially any floor of any building in the entire city.

- *Interactive Maps* did not handle change over time, i.e., temporal information. For example, class schedules for lecture halls are typically only valid for the duration of the semester, but *Interactive Maps* permanently attaches the information to the room. *Interactive Pittsburgh* handles temporal data in a flexible manner.

Interactive Pittsburgh was expected to support access to three basic kinds of information: geographical, object, and temporal. Geographical information consisted of the map data, such as street data, bus schedules, building floor plans, room locations, the layout of buildings on campus, and so forth. Object information describes the properties of entities in the system, such as the contents of buildings and rooms and information about room occupants. Temporal information is composed of data such as university holidays, class schedules and lecture schedules. In addition to supporting access to these three kinds of data, *Interactive Pittsburgh* was expected to support the connection of geographical and object data with temporal data for relatively complex queries.

The functional requirements for *Interactive Pittsburgh* were stated simply: a user should be able to easily browse around the streets of downtown Pittsburgh and surrounding neighborhoods, obtain information about streets, buildings, and bus stops, and display it on a graphical display in a meaningful way. The following system queries were given to the students:

- Which bus lines go along Fifth Avenue?
- Which bus lines go downtown from Oakland?
- Show me the 61 bus schedule.
- Where does the 61 bus stop near Carnegie Mellon University?
- Where is Heinz Hall?
- Which bus lines go by Heinz Hall?
- I want to get to Horne's Department Store by 10am. What options do I have for taking buses from the corner of Murray and Forward Avenues to Horne's so that the overall trip time is minimized?
- I'm at the corner of Plainfield and Woodmont. Where is the nearest bus stop?
- Where is Forbes Avenue?
- Where is the intersection of Forbes and Murray?
- Display all streets that intersect Forbes.
- Display all streets that fall within a half-mile radius of campus.
- Where is Dave McKeown's office, what is his phone number, and what does he look like?
- Who are the occupants of the offices on the 8100 corridor of Wean Hall?

This problem lent itself to object-oriented design quite well. There are many observable entities in the problem: bus stops, bus routes, bus schedules, streets, street segments, intersections, buildings, floors, offices, and occupants. These entities also have operations that can readily be defined on them, for example the operation of finding the street containing a certain street segment (a street segment being roughly equivalent to a single block).

Our problem had several real-world aspects, the most immediate of which was the data we supplied to the students. We did take the initial step of pre-compiling the raw data into a simpler format to minimize the drudgery of dealing with the data, but we intentionally left certain problems in the data. The bus data and the road network data were in completely different formats (this is not unusual, as there exist a plethora of formats for geographic and spatial data). Further, there were inaccuracies in the bus data in terms of misspellings of streets, as well as inaccuracies in the road network data, which had not been updated in several years while many streets had been renumbered, new streets had been paved, and some streets had even been eliminated. Finally, incompatibilities existed: the bus data referenced certain landmarks and homes adjacent to bus stops, which of course did not appear in the road network data. For *Interactive Pittsburgh* to succeed, the students had to address the real-world issues of data incompatibility.

Another real-world aspect was the use of faculty members and city and county planners as clients, to provide perspective on the project and evaluation in terms of client expectations. The nature of the project (a digital geographic browser) comprised another real-world facet, since such systems are in great demand by the municipal planning and fire, police, and emergency medical communities. These aspects helped to serve as motivational factors for the students by adding some excitement to the project.

3 Object-Oriented Project Management

The specification of the *Interactive Pittsburgh* problem was intentionally made complex enough that it required the combined efforts of several teams. However, the students were not yet familiar enough with project management and object-oriented design issues to tackle such a problem from the beginning. To make it possible to finish the system within the given deadline of a semester, several decisions had to be made by the instructors. For example, we provided the students with a rough decomposition of the problem, consisting of five modules: Bus, Place, Street, User Interface and Tools. A group of students were assigned to each of these modules and were responsible for designing and implementing it as part of the *Interactive Pittsburgh* system.

Such a predefined decomposition of the problem is a departure from real world problem solving: Decisions that should be made during the system design phase are preempted. We believe that in a teaching environment the decomposition has to be made in advance to give the project a good chance of success.

The *Bus group* was responsible for providing a rationalized internal description and a set of data structures for the Port Authority bus data. Further, they were responsible for providing access packages for the spatial aspects of the data (bus stop locations, bus routes) as well as the temporal aspects (route-stop-time data). The bus group was also asked to provide methods for reasoning about bus schedules.

The *Place group* was responsible for compiling and maintaining a place-name-location database containing unique building locations with respect to the map coordinate system used by the road data. They were also charged with the responsibilities of extending the *Interactive Maps* system to allow more flexible building/hall/office access, as well as incorporating the School of Computer Science office database and digitized photographs.

The *Street group* was responsible for providing a rationalized internal description and a set of data structures for the road network data. Furthermore, they were given the task of providing access packages for road segment, road description and road intersection information as well as methods for navigation and searching.

The *User Interface group* was given the responsibility for providing an X/Motif-based display for spatial and temporal data, including scaling of map and bus data as appropriate.

The task of the *Tools group* was to develop and extract reusable objects. The Tools group was the most interesting group from an object-oriented point of view. Booch [2] recommends a *toolsmith* as a new specific position in a object-oriented software development team, and Gibson [8] advocates the *framework team* to focus on abstractions and mechanisms. We were eager to apply these concepts in the classroom setting. Aside from being convinced that these roles were necessary, however, we had no prior experience suggesting how such a tools team should interact with the other students. We described the Tools group as the "glue" between the other groups and gave them the task of providing tools for database update and maintenance, including the street and bus databases.

The roles and tasks of this group were frequently redefined during the project. For example, to support the information flow between the other groups and the tools group we introduced the role of the floating liaison described in section 5. During the implementation phase the group investigated various C++ class libraries such as the NIH library [9] to look for reusable classes. The first *Interactive Pittsburgh* prototype was tested with container classes from another class library. When it turned out that the implementation of the set class contained a bug, the tools group reimplemented these classes while the other groups continued their development without changing their code. Our experience during the *Interactive Pittsburgh* project suggests that the tools group was a useful concept.

For the schedule we followed an object-oriented software life cycle[24] with a large proportion spent on the "front end", namely 9 weeks for requirements analysis and design, 3 weeks for coding and unit testing and 3 weeks for system integration. To convince the students of the feasibility of the system, a prototype system was scheduled after about 2/3 of the total project time. The final deliverable of the project was the client acceptance test, in which the clients were presented with the software product and asked to evaluate it against the requirements laid out at the onset of the project. The software system was evaluated as a success if two criteria were met: first, that a set of documents describing each phase of the project was completed and delivered, and second, that the system was able to provide a certain core functionality, defined as the ability to successfully respond to a major subset of the queries described earlier.

We managed the project at several levels. The instructors served as overall project managers and motivators for each of the groups. Another faculty member and three city and county planning officials served as external clients. Within each group we defined six roles: Group leader, planner, liaison (interface to the other groups), technical writer, record keeper and developer (analyst, designer and programmer). To provide each of the students with the opportunity to experience the issues of leadership, we required the group leader role to be rotated among the group members. The other role assignments were left to the students, allowing them to assume multiple roles if necessary.

4 System Modeling with OMT

We now describe the modeling process and the system model produced by the students using OMT. Figure 1 sketches the OMT modeling process taught to the class. During the requirements analysis the students use the problem statement to formulate the three models. The key is the formulation of the object model; that is, the description of the classes, attributes and associations identified in the problem statement. The functional and dynamic model are especially useful as "sources of methods"; that is, they help to identify the methods for classes of the object model. The object

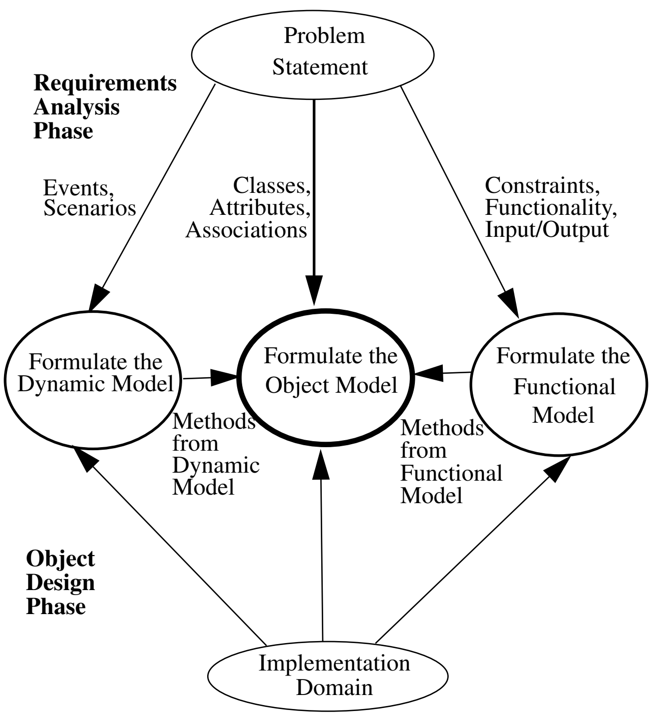

Figure 1. Modeling process used in the Interactive Pittsburgh project

design phase pays attention to implementation-specific issues and generally results in the addition of new classes, attributes, and methods as well as the merging of some existing classes (for example, collapsing classes for optimization purposes). System design was taught between requirements analysis and object design. The students were encouraged to go through several iterations of the system model. Moving between the requirements analysis and object design was quite easy due to OMT's uniform notation.

We also told the students to strike a balance between formality and informality during the modeling process. In particular, we advised them to be ready to deviate from the above approach in case difficulties arose. As Booch points out, "There is no tool and no methodology that magically replaces the basic human creativity that is necessary to successfully design, implement and deploy a complex software system" [4]. We found that such a pragmatic attitude towards the system modeling process worked very well with the students.

In the remainder of this section we describe portions of the *Interactive Pittsburgh* system to illustrate our experience with the use of OMT; a more complete description of the system can be found in [13]. Section 4.1 describes the system design. Section 4.2 gives a short introduction to the OMT notation and describes portions of the *Interactive Pittsburgh* object model. Section 4.3 sketches the dynamic model. Section 4.4 and Section 4.5 relate our experiences with OMT in the project.

4.1 System Design

The *Interactive Pittsburgh* system consists of three major components: a floor and area editor, a data compiler and a database browser called IP. The object and dynamic model presented in the next sections apply to the browser.

Data for enterable buildings came from floor plans created with a floor editor. The floor editor was designed and implemented by the *Place group* to permit the interactive creation of floor plans and room descriptions for enterable buildings. Personal data of people such as phone number and office locations were obtained for Wean Hall, the location of CMU's School of Computer Science.

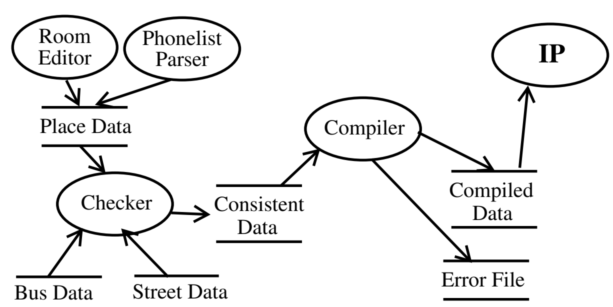

Figure 2. Data flow diagram of the data compiler

Figure 2 shows the functional model of the data compiler, which is needed for the maintenance and consistency checks of the input data. For example, the raw bus data obtained from PAT in the form of text files describing the various bus routes and bus schedules in the Pittsburgh area contained spelling errors. The raw street data obtained from the City of Pittsburgh Planning Department in the form of Tiger Census files contained nonexisting streets. The data compiler also obtains additional place data as the by-product of a consistency check between bus stop locations and street segment names. After the data are made consistent, they are compiled into a format that can be quickly loaded into the browser.

IP, the data base browser, has a closed architecture with 3 layers, as shown in Figure 3. After a user has

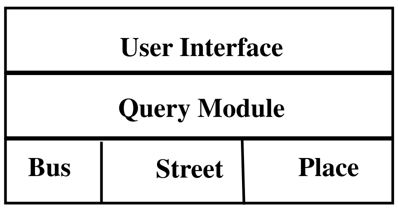

Figure 3. System layers of the data base browser

entered a request into a dialog box, the request is converted by the top-level user interface into a query. The middle layer, the query module, invokes methods of one or more of the bottom layer modules, bus, street and place. The system is event driven with the query module as controller.

At initialization time, IP reads the data from the compiled data base. The street data is read before bus and place data, because the bus stops and areas are attached to street segments. One of the global requirements was an average response time of less than 2-3 seconds for typical queries. To deliver this response time, IP reads the complete compiled data base into memory. This clearly does not scale for large distributed data sets, but this decision was made to get better response times.

During the organization of IP into subsystems, we would have liked to define classes in a way that reflected the hierarchical structure of the system layering. However, OMT does not provide such a facility. One of the problems is the flatness of the class model and the lack of abstraction facilities for the denotation of layers.

4.2 The Object Model and its Notation

Figure 4 through Figure 7 are diagrams of the object model from the requirements analysis document produced by the students. The diagrams use the Object Modeling Technique notation, in which classes are represented by boxes, each of which has a class name and, optionally, sets of attributes and operations [19]. For example, the box labeled `Area` in the upper left corner of Figure 6 represents a class named "Area" which has attributes `Name` and `Address` and an operation `What phones`. Relations between classes — called associations by OMT —are represented by lines joining their boxes in the diagram. For example, `Areas` can be related to `Bus stops` with the association `Contains`. We will not give a complete description of the figures shown here, partly for brevity and partly since they are fairly self-explanatory once some elements of the OMT notation are understood. We will go through them briefly in order to familiarize the reader with this notation; in the process we use some of the descriptive text with which the objects are annotated in the data dictionary of the requirements analysis document [13].

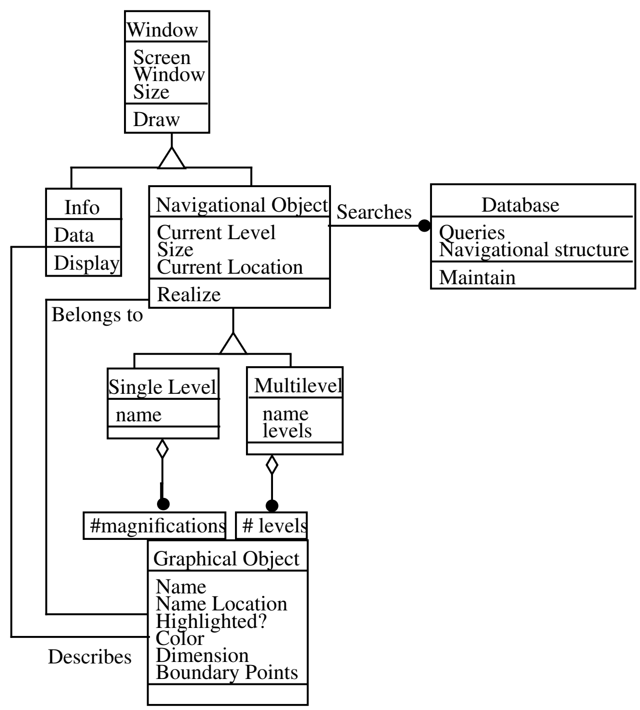

Figure 4. Object model for the user interface

Figure 4 shows the classes that represent the user interface for *Interactive Pittsburgh.* The triangle near the top of the figure indicates that the relationship between `Window` and both `Navigational Object` and `Info` is generalization. For example, the `Naviga-`

tional Object is a kind of Window used to provide a consistent interface to the different Databases that the user can access. Each Navigational Object uses the association Searches to find the Database that it has access to.

The black ball on the Searches relation in Figure 4 is a *multiplicity ball* meaning “zero or more.” Thus while each Database is searched by exactly one Navigational Object, a Navigational Object can search zero or more Databases.The diamond on the association between the Single Level and Graphical Object classes represents aggregation, so a Single Level, which is a kind of Navigational Object, is made up of zero or more Graphical Objects. Classes can be denoted more precisely using qualified association, represented by a small box between the association and the class. This is used in the aggregation in Figure 4, so that a particular Graphical Object, which is part of a Single Level, can be retrieved by specifying the correct value for #magnifications.

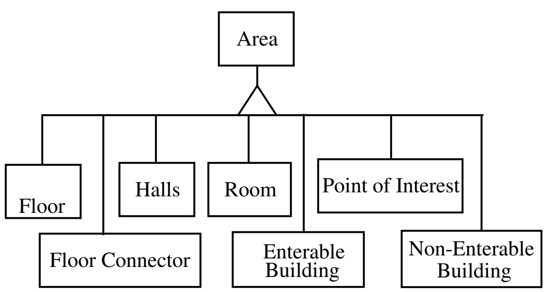

Figure 5. Object model for Area showing only inheritance

Figure 5 introduces the class Area, used in modeling the geographical data, and its subtypes. An area is defined in the data dictionary as “a bounded geographic region that may or may not contain bus stops, street segments, building structures and other areas” [13]. The distinction between Enterable and Non-Enterable Buildings is that, for the former, *Interactive Pittsburgh* has data about the Floors of the buildings and so the user can “enter” them by issuing commands to the user interface.While OMT is helpful in identifying design deficiencies, it of course does not guarantee a good design. Notice the different descriptive levels in Figure 5 and Figure 6. Figure 5 shows that Enterable Building is a subtype of Area, while in Figure 6 the aggregation relation is used to express that an Area comprises Points of Interest, Non-Enterable and Enterable Buildings, and that an Enterable Building is in turn made up of Floors. This idea of combining both inheritance and aggregation is used a number of times in the design of *Interactive Pittsburgh*.

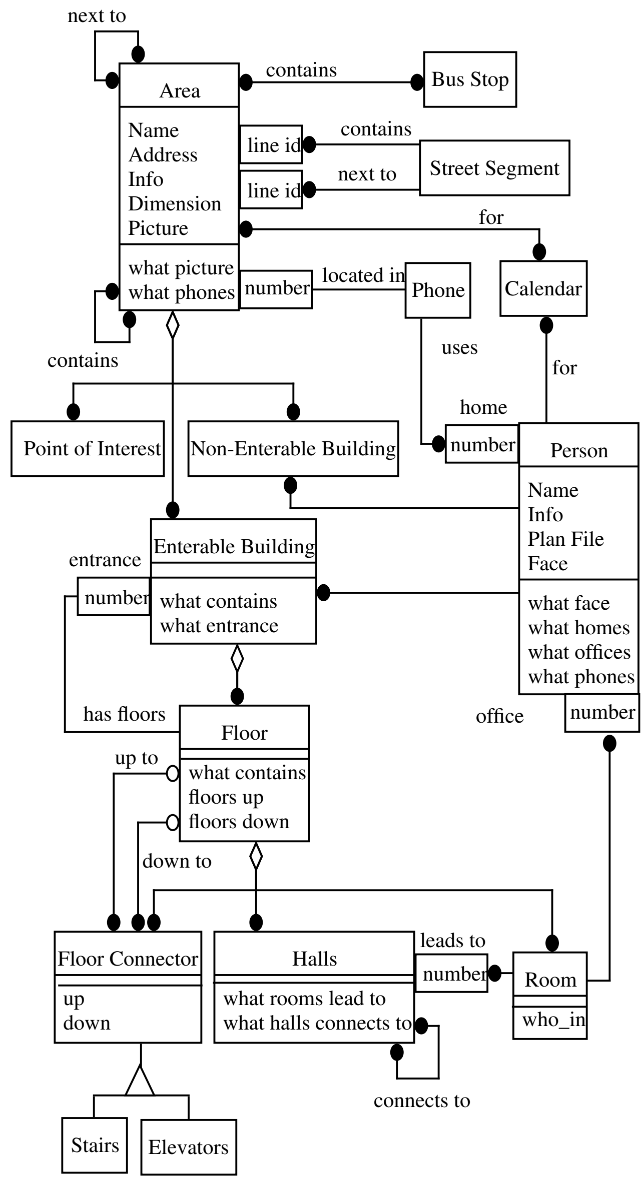

Figure 6. Object model for Area showing aggregation

Unfortunately, in this case it is not consistent, since by inheritance from Area we could expect an Enterable Building to be composed of other Enterable Buildings, which is not correct. This inconsistency in the representation does not mean that the resulting implementation produces incorrect results; however, with more experience the students probably would have created a new class, say Region, from which Area as well as the other classes could

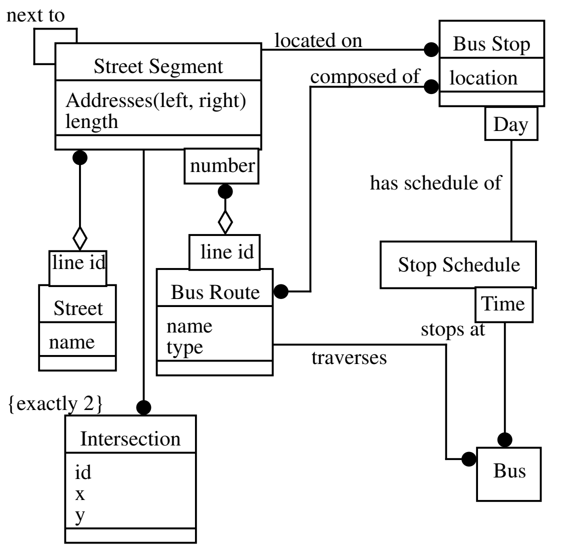

Figure 7. Object model for `Street` and `Bus Stop`

inherit. Notice also that some classes, for example `Street Segment`, appear on more than one sheet (Figure 6 and Figure 7). When this happens, the different graphical objects refer to the same OMT class, and updates to any one will affect all of the graphical representations.

4.3 Dynamic Model

The dynamic model consists of state diagrams for all objects in the object model that display significant dynamic behavior. We use the `Navigational Object` as an example, although we only show a part of its behavior, omitting menu selection for brevity. This class is introduced as part of the object model in Figure 4, and its dynamic model in Figure 8 is taken from the student-produced Object Design Document [13]. In this figure the ovals represent states of the system, and the arrows are labeled with events that cause transitions between the states.

We found that the teams did not maintain the dynamic model as well as the object model. The diagrams were produced by FrameMaker [7] and never fully represented the behavior of the user interface that we wanted to capture. For example, during object design the `Navigational Object` was replaced by other objects, yet the dynamic model was unchanged. This may be because the state-diagram representation is too low-level, both being tedious and not providing significant insight beyond that provided by actual code, at least in the minds of the code writers.

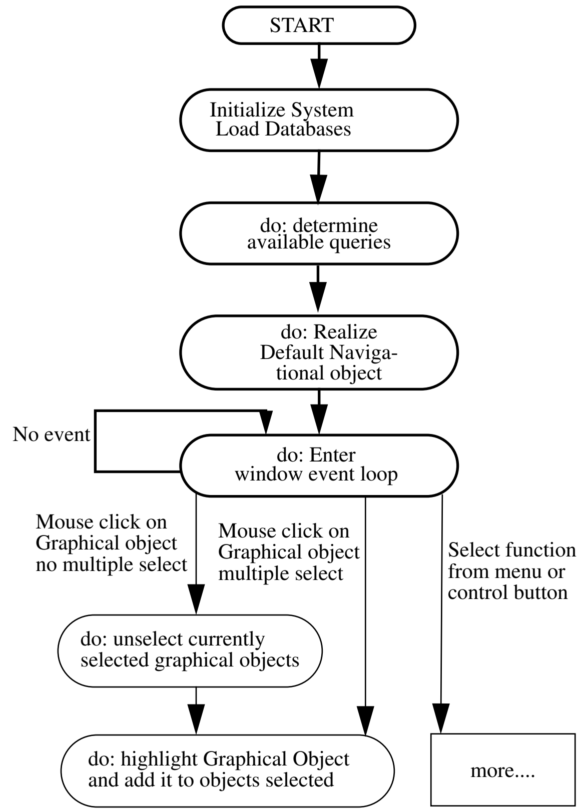

Figure 8. Part of the dynamic model for `Navigational Object`

In addition, there was not a complete coupling between the objects mentioned in the dynamic and object models. For example, the `Dialog Box` was incorporated in the dynamic model but not in the object model "for the sake of implementation details yet to be determined" [13]. In fact, it is never mentioned in the object model at all. Such omissions should be caught and at least queried by the object modeling software, but they cannot be in OMTool because it does not address the dynamic model. We see this as a significant weakness in the current implementation of OMTool.

4.4 Gaining Experience with OMT

The students were essentially learning OMT at the same time as they were using it for the design of the system (although we did assign small homework problems at the beginning of the semester to familiarize them with OMT). It was therefore very important that the modeling technique and its associated editor be easy to learn. We found this to be the case. OMT, while not as complex as Booch's object modeling methodology [2] and lacking expressive power in some circumstances, was ideal for the students, who were new to object-oriented programming and modeling.

The use of OMT as a design tool was helpful for discussions about the design. Many of the groups resorted to object model diagrams in discussions about the evolving design, rather than submit to the vagaries of natural language. This in turn helped the students to learn from each other about the notation and how to use it. In the bus group, for instance, object diagrams were frequently posted to the bulletin board for discussion or entered into the minutes of meetings. As these design models evolve, one can notice both an increased understanding of the domain and an increased sophistication in the use of OMT. This reflects the way the students were learning OMT at the same time as they were using it.

The following three fragments of object diagrams illustrate this. Figure 9, from minutes taken at the outset of the requirements analysis phase, does not label associations and ignores the problem that bus stops have different schedules on weekends and weekdays. It also uses concrete objects rather than classes for all but one box. This diagram was not generated with OMTool.

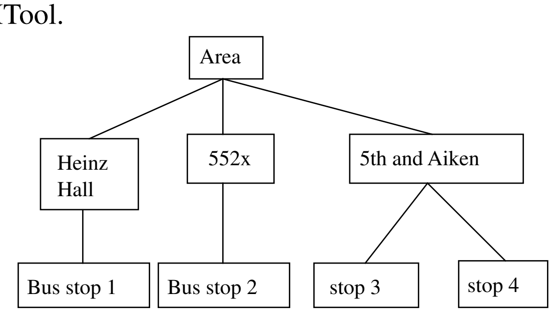

Figure 9. First attempt with OMT

Figure 10, recorded at a meeting two weeks later, uses classes, labeled associations, multiplicity and inheritance. Inheritance is used incorrectly in place of

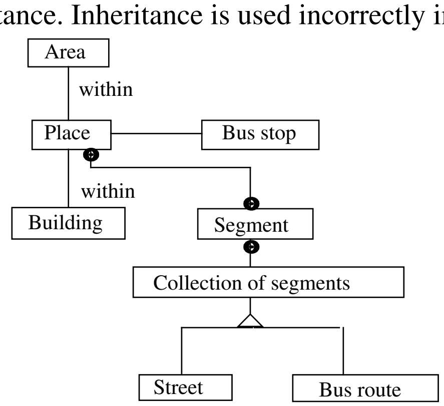

Figure 10. Incorrect use of inheritance

aggregation, however.The diagram shown in Figure 11 comes from minutes taken one week later. It uses qualified associations, and although no associations are labeled it is essentially part of the final object structure that appears in Figure 7.

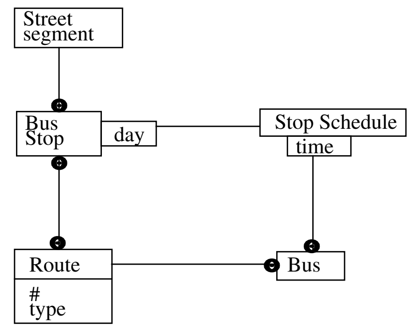

Figure 11. Using qualified associations

4.5 The Use of OMT within the Project

At the beginning of the requirements analysis each of the teams specified an object model for their module separately. While working on the individual models, the group liaisons had the task of ensuring that the groups were making correct assumptions about the models being developed by the other groups. We found that letting each group specify certain “external” objects and relations for the other groups while working on their own “internal” model at the same time provided a useful level of abstraction for this purpose. At the end of the requirements analysis the object models were combined into a single object model, placing the individual modules into separate sheets.

As noted above, the object diagrams were a good medium for communication between and within the groups. In group meetings, alternative designs were discussed using OMT-style drawings on a white-board. Liaisons often took hard copies of their group’s current object document to other group meetings, and object diagrams were included in text form on bulletin boards and in the minutes of meetings. During the client review of the design, one client who was not familiar with object-oriented modeling was able to spot a bug in the design from inspection of the diagrams: several intersections in Pittsburgh involve more than one street segments, which was not allowed by the model being reviewed (The association between `Street Segment` and `Intersection` shown in Figure 7 is *one-to-exactly 2*).

During the object design phase, each group worked on a separate sheet of the object model. At the beginning of the implementation phase, C++ signatures were automatically generated from the model and formed the basis of the code to be written by each group. OMTool generates C++ code without regard to sheets. Thus to assign code modules to the individual teams, the OMTool-generated files had to be split up manually. Once the code was divided there was no easy way to move between the OMT model and the generated code. Changes made either in the code or in the model were not automatically reflected in the other representation. To minimize the resulting "model-code drift", we started to use version control (RCS) for the object model as well as for the code and an *object designer* appointed by each group had to ensure that the current code was consistent with the object model. Using this method, the object model was kept generally in agreement with the generated code from all the groups, except during the final phase of the system integration. In this way the object model performed a useful role as a coordinating blackboard for the groups during the implementation.

Another implementation issue was the programming language. With OMTool's ability to generate C++ skeleton code from the object model, C++ was an obvious choice. While most of the Interactive Pittsburgh teams were able to learn and use C++ productively, much of the graphical user interface was coded in C instead. To provide backward compatibility with the look and feel of Interactive Maps, the user interface was required to use Motif. The students looked at Interviews[16], a C++ toolkit for user interface development, but it did not support Motif at that time. Several students participated in a search for an existing public domain library which would facilitate the use of Motif and X graphics procedures within a C++ program. A set of C++ wrappers developed by the University of Lowell, Massachusetts, was selected due to its ready availability and reasonable size (disk space was a critical resource through much of the implementation phase). Unfortunately, several bugs were later discovered in this early version of the Lowell library. After additional technical difficulties with extending the library's Motif coverage and using the time-saving variable-length argument feature of X from C++, the user interface team reverted to providing an object-oriented interface to the rest of the software system and coding the bulk of the actual graphics procedures in C.

Because of these implementation difficulties, about 30% of the code was written in a non-object-oriented language. With Interviews 3.0 supporting Motif's look and feel, the publication of Young's book [25] and the availability of improved libraries from Lowell and other sources, we expect that future classes will code exclusively in C++.

4.6 System Integration and Delivery

During system integration the students experienced several difficulties. For example, many functions initially did not check for a return result of `null`, even though it was agreed that this value was the indicator of a failed search. Also, the signatures of some functions did not behave as announced due to changes in the code that were not properly updated in the object model.

Other problems were due to inexperience with the cost of object-oriented implementations. During object design the students had decided to make many classes in the object model subclasses of `Graphical Object`. This inheritance scheme looked elegant, but it interfered with modular development and testing of the system. Every time the user interface group changed the `Graphical Object`, the whole system had to be recompiled due to the ripple effect of the inheritance, which slowed down the system testing. The scheme also led to storage allocations that surprised the students. For example, simple classes such as street segment, in addition to the storage for its own attributes, inherited more than 100 bytes of attributes from `Graphical Object`. With 22,000 street segments in the `Street database` alone, this turned into a major storage allocation problem: The bus and street data require approximately 16Mbytes of storage. Support from OMTool to estimate this cost would have been of great help during the object design and implementation phase. A result of this experience for the students was the insight that inheritance has to be used wisely.

The students also experienced response time problems for some of the bus requests which caused a delay in the successful integration of the system. During the acceptance test, the system answered all the required queries except for the bus queries. A run-time profile showed that some of the bus methods spent 60% of their time in set operations which were implemented as linked lists. Some of the students reimplemented the set more efficiently using arrays (a great opportunity to experience the advantage of data abstraction!) and at the beginning of the next semester a second acceptance test was executed which the program passed successfully.

The delivered system consists of 125 classes (about 27,000 lines of C++ and C code). It has a response time of less than 2 seconds for simple queries (such as finding an intersection of two streets) to a few minutes for more complicated ones such as finding the shortest bus route from an intersection to a landmark or building. A snapshot of the user interface for the database browser is shown in Figure 12.

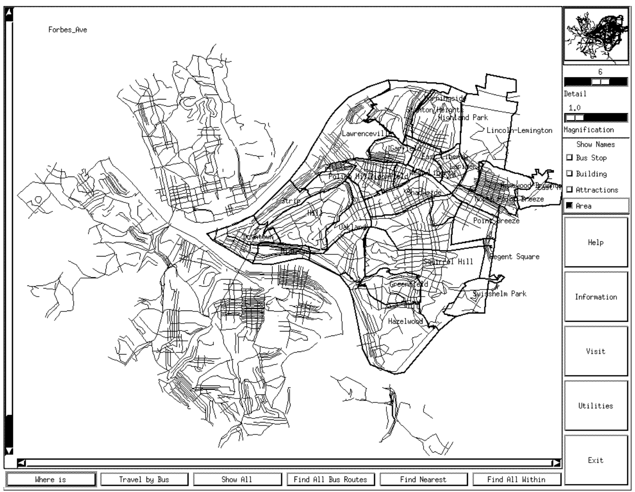

Figure 12. IP User Interface

5 Communication

As outlined earlier, the *Interactive Pittsburgh* developers were separated into five teams from the outset of the project. While the teams were each given a broad outline of their responsibilities, they were expected to negotiate specifics of their interfaces among themselves. With so much to communicate about, so little experience, and no predefined intergroup decision-making mechanism, it might seem that intergroup communication could easily have rivaled or exceeded intragroup communication. This might be expected particularly during system integration, since poor communication early in the project often goes unnoticed until integration is underway, at which time a great deal of intergroup communication can be required to resolve differing understandings. While a pattern of disproportionate intergroup communication during integration *was* observed for *Interactive Maps*, which used more traditional design techniques, the proportion of intergroup communication was greatly reduced for *Interactive Pittsburgh*. As we will explain, the major difference between the projects that might account for this observed difference in communication patterns was the use of different design methodologies. This in turn suggests that the use of OMT enhances communication and thus should improve productivity compared with traditional techniques.

In the remainder of this section we discuss in more detail communication aspects of the *Interactive Pittsburgh* and *Interactive Maps* development projects. In particular, we present quantitative data supporting the notion that the use of OMT enhances communication.

5.1 Comparison of Projects

Interactive Maps and *Interactive Pittsburgh* were quite similar in many ways, facilitating a comparison of communication aspects of each project. Both projects can be described as systems which provide an interactive graphical interface to a database which includes information about people and places. In fact, as shown in Figure 13, the overall system architectures for the two systems are very similar.

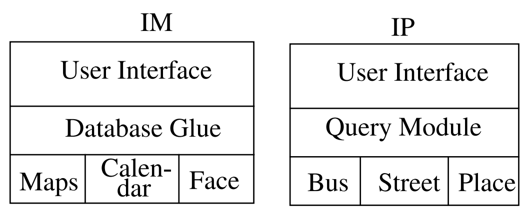

Figure 13.Comparison of system layers for Interactive Maps and Interactive Pittsburgh

The communication and management structures for the two projects were also strikingly similar. First, for intragroup communication, we established two primary mechanisms: a group-specific bulletin board and a project meeting. The group-specific bulletin boards (called `ui, glue, maps, calendar` and `ui, tools, bus, street, place`, respectively) were set up for discussing module-specific problems. Weekly project meetings where held for each group to discuss progress, ideas, and plans. The students were divided into four groups for *Interactive Maps* and five for *Interactive Pittsburgh*, so the teams were somewhat larger on average for the latter (four vs. six students per team).

For intergroup communication, a separate bulletin board called `discuss` was created for each project for the "discussion of project-related issues." Administrative announcements were for the most part placed on a separate bulletin board, so `discuss` was almost exclusively for use by the students. The primary intergroup communication was to be carried out by the liaisons for each group. Their main responsibilities, as outlined by the software project management plan, were to define/negotiate group boundaries with other liaisons and to coordinate tasks with the other teams and project management. Three project reviews, each involving at least one speaker from each of the groups, also contributed to intergroup understanding.

It is interesting to note how these communication mechanisms evolved in practice. All of the groups supplemented the weekly meeting with management with a working meeting of their own for at least a portion of the semester. All of the groups used their bulletin board for internal communication, although, as we will show shortly, some used it much more than others.

Perhaps more interesting was the evolution of the liaison role during the development of *Interactive Pittsburgh*. Early in the semester the liaisons began meeting on a regular basis, and as time went on this group began to assume responsibility for system design decisions. This was instrumental in allowing the class to produce a unified set of documents describing the work done during each phase of the project, satisfying the first of our requirements for a successful system**.** The tools group, which had a particularly nebulous task definition, decided early on to take the liaison idea one step further and sent a team member (a "floating liaison") to each of the other groups' weekly meetings. The `discuss` bulletin board became more-or-less the liaisons' bulletin board, although others (who, for example, might need inputs from several groups for a presentation) often posted there as well. In fact, 26 of the 30 students made at least one post to `discuss`, and 24 made two or more posts.

The management structures for the two projects were also very similar. Groups in both projects were told to assign students to the roles of project leader, liaison, programmer, etc. One difference between the projects was the period of time a student could remain in a role. In *Interactive Maps* we had encouraged the students to switch all of the roles among the group members. The idea was to make it possible for every student to learn about the problems associated with each role, but the concept turned out to be unmanageable [6]. Some intergroup interface problems persisted across a personnel change, but students who had switched roles — for example, from liaison to document editor — often did not feel responsible for the liaison problems after their switch.

In *Interactive Pittsburgh* we required only that the role of project leader be switched among the group members. The remaining roles, in particular the liaison role, were assigned by the groups to students at the beginning of the semester, and students generally retained these roles for the duration of the semester.

The main differences between the projects involved size and complexity: for example, the data about places in *Interactive Pittsburgh* is a very large superset of the data for *Interactive Maps*, and the bus schedule/street data processing in *Interactive Pittsburgh* was much more complex than any data processing performed by *Interactive Maps*. Moreover, twice as many students worked on *Interactive Pittsburgh*. The other main difference, as mentioned in the introduction, was the use of object-oriented methods in *Interactive Pittsburgh*.

In summary, the tasks, communication mechanisms, and project management for *Interactive Maps* and *Interactive Pittsburgh* were very similar in many respects. Of the differences between the two projects in these areas, the much larger size and complexity of *Interactive Pittsburgh* and the larger number of students involved appear to be the factors which would be most responsible for differences in communication complexity between the projects, and these factors would of course lead to increased complexity for *Interactive Pittsburgh*. In spite of this, the data presented below indicates apparently improved communication for *Interactive Pittsburgh*. The remaining difference between the projects that seems most likely responsible for this improvement is the use of the OMT design methodology and OMTool. Some of the smaller differences which may have helped to improve the communication picture for *Interactive Pittsburgh*, such as the floating liaison position, are very possibly also outgrowths of our use of object-oriented development ideas.

5.2 Communication Patterns

Here we present data gathered from the electronic bulletin board posts generated throughout each semester to provide some insight into the effects of object-oriented design practice on project communication requirements. Although bulletin board posts constitute only one slice of the total communications pie, we feel that with hundreds of distinct posts for each semester it is a large enough slice to be at least somewhat representative of the overall communications picture. This data may also allow for relatively objective comparison with similar data derived in other settings. Finally, it should be noted that the students were not aware we would be using the bulletin board posts as data for this study; if they had been, the data could of course have been skewed by that knowledge.

Table 1 and Table 2 present a summary of the overall bulletin board traffic during each development project. There were a total of 933 posts consisting of a total of 36,396 lines of text during *Interactive Pittsburgh* development, 339 posts and 14,726 lines during *Interactive Maps*. If posts which were copies of others are removed from consideration, there were 849 and 300 posts, respectively. Thus an immediate observation is that the *Interactive Pittsburgh* class, while only twice as large as the *Interactive Maps* class, generated almost three times as many posts. Of course, doubling the number of developers roughly quadruples the number of communication paths between pairs of developers, so at least part of this increase may be due to this effect.

Bboard	Posts	Lines
bus	273	10675
place	130	5008
street	110	3247
tools	123	5959
ui	93	4935
discuss	204	6572
discuss%	22%	18%

Table 1. Interactive Pittsburgh bulletin board posts

Bboard	Posts	Lines
calendar	59	3570
glue	38	1222
maps	47	3340
ui	80	3479
discuss	115	3115
discuss%	34%	21%

Table 2. Interactive Maps bulletin board posts

As a first pass at quantifying the intragroup versus intergroup communication, we first consider the numbers of posts to the individual group versus project-wide bulletin boards. Although the *Interactive Pittsburgh* project-wide `discuss` bulletin board had slightly above-average activity, it accounted for only about 20% of all bulletin board communications. On the other hand, *Interactive Maps* `discuss` posts accounted for 34% of all posts made. While *Interactive Maps* `discuss` accounted for only about 20% of the line totals, this is somewhat deceptive: three posts which were cross-posted between `calendar`, `maps`, and `ui` (and thus might better have been placed on `discuss`) accounted for 1482 of the lines for each of these bulletin boards. If these three posts had been made to `discuss` instead, `discuss` would have accounted for 39% of the line totals. By contrast, no post of over 100 lines was posted to more than two bulletin boards during the object-oriented design semester, so the *Interactive Pittsburgh* `discuss` percentage more accurately represents the proportion of multi-group communication.

A more accurate picture of inter- vs. intragroup communication is obtained by considering the authorship of each post and the number of bulletin boards on which it appears. That is, a post on, say, the `ui` bulletin board by a member of the Bus group is actually an intergroup communication. Similarly, a post which appears on both the `bus` and `ui` boards should be considered an intergroup communication regardless of authorship.

As Table 3 and Table 4 show, cross-posting was much heavier during *Interactive Maps* development than during *Interactive Pittsburgh*. Overall, 555 of the 849, or 65%, of the distinct *Interactive Pittsburgh* posts represented intragroup communication; that is, intragroup posts outnumbered intergroup posts almost two to one. Only 120 of 300, or 40%, of the distinct *Interactive Maps* posts represented pure intragroup communication. The percentages are virtually identical (66% and 41%, respectively) when lines are considered rather than number of posts. This data suggests that object-oriented design noticeably reduces the proportion of intergroup communication.

Bboard	Intragroup	Intergroup
bus	237	36
place	99	31
street	87	23
tools	68	55
ui	64	29

Table 3. Interactive Pittsburgh intragroup vs. intergroup posts

Bboard	Intragroup	Intergroup
calendar	36	23
glue	21	17
maps	26	21
ui	37	43

Table 4. Interactive Maps intragroup vs. intergroup posts

Perhaps the most useful way to view the data is across phases of the development project (Table 5 and Table 6).[1] This data suggests that intergroup communications were relatively light for *Interactive Pittsburgh* during each phase, even though a number of interface issues were being addressed throughout development. It should be noted that the intergroup activity during the requirements phase appears relatively low in large part due to the initial requirement that each group submit a separate object model. If the students had begun the semester working on a more integrated model, as they did subsequently, it is likely that the requirements analysis phase would have shown greater intergroup communication.

Phase	Intragroup	Intergroup
Analysis	132	41
Design	83	60
Unit Coding	227	117
Integration	113	76

Table 5. Interactive Pittsburgh posts per phase

Phase	Intragroup	Intergroup
Analysis	46	10
Design	39	23
Unit Coding	17	73
Integration	18	74

Table 6. Interactive Maps posts per phase

The *Interactive Maps* semester, on the other hand, showed a very different communication pattern as the semester progressed, a pattern which is perhaps all-too-familiar to those who have experience with large development projects using traditional development techniques. The semester began with relatively little cross-communication between groups but ended with a four to one ratio in favor of intergroup communication (the ratio for the line counts is even greater!). The similarity between the total number of intergroup posts made in the last two phases for both *Interactive Maps* and *Interactive Pittsburgh* is particularly striking given that *Interactive Pittsburgh* involved twice as many students.

1. Phase boundaries for these projects were somewhat fuzzy, so the data in these tables should be considered illustrative.

5.3 Communication Summary

Our experience in developing a medium-sized project with novice developers using object-oriented design, and specifically the OMT methodology, was quite positive. Our data suggest that object-oriented design enhanced communication, with benefits most noticeable during the latter stages of development. Other than the use of different design methodologies, the main differences between the *Interactive Maps* and *Interactive Pittsburgh* developments were the scope of the projects and the size of the development teams. With a more ambitious project and twice the number of developers we might have expected that the communications picture would have been much worse for *Interactive Pittsburgh* than it was for *Interactive Maps*, particularly during system integration. That the opposite was true seems to indicate that integration was much smoother for *Interactive Pittsburgh*, even with twice the number of developers. Thus OMT may be quite useful for medium-sized development projects, for which system integration is often a major bottleneck.

6 Summary and Conclusions

Learning, teaching, and applying a new software development methodology at the same time is difficult. Recently there has been a profusion of object-oriented analysis and design methods [1], and with so many methodologies emerging, we need to know which one of the methodologies to choose, how much it will cost us, and whether it will eventually improve our effectiveness in producing complex software systems.

We have chosen the OMT methodology for a single project course in a classroom setting for a medium-sized problem with real data and real clients. One goal of this paper was to demonstrate that it is possible to learn OMT and apply it at the same time to a fairly complex problem. OMT strikes a good balance between an easy-to-learn design tool and a highly expressive tool. Even though the instructors and the students were inexperienced with OMT at the beginning of the course, we were able to build a moderately large system. We believe that with the use of the object-oriented modeling technique we have demonstrated that it is now possible to attack problems of such a complexity in a practical industry-oriented software engineering course on a regular basis.

Another objective of the paper was to demonstrate that OMT makes relatively large single project courses manageable. OMT helps to manage such courses better because it reduces the communication overhead between different class designers at the object design and implementation phases. OMT is simple enough to be absorbed by the students while using it for analysis and design, but is powerful enough to be a valuable tool. These features allowed us to tackle a larger problem — giving the students a learning experience in a much more realistic setting with real data and real clients — and still finish most of the project within a semester. We therefore think OMT is a useful methodology for developing moderately complex projects such as *Workstation Fax*, *Interactive Maps* and *Interactive Pittsburgh*.

We note three features of OMT that were important in its success in this project. First, it is a notation that allows the communication of the design to others, both within the project and between the clients and developers, without going down to the coding level. This raises the level at which students can compare and discuss designs. It also makes the evaluation and choice of the final design easier and faster. It proved to be a valuable means of communication within groups, and the diagrams found their way onto the bulletin boards and into the minutes of groups. During the design reviews the clients were suggesting useful changes to the design, presented as an OMT diagram, even though they were not instructed in the semantics of the notation.

OMT was used through the requirements analysis, design and implementation phases of the project, supporting the project life cycle within a project using object-oriented design and programming. This was done without major changes to the model from one phase to another, also allowing for iteration between these phases.

Finally, OMT has an associated tool, OMTool, with which the design can be entered in machine readable form, stored and modified by the developers. This design then formed the basis of the implementation of Interactive Pittsburgh in C++.

Our experience suggests two potential areas for improvements to OMTool. First, OMTool does not allow mapping of the system decomposition onto a team structure. It has a sheet concept, which supports the decomposition of the design into modules; it does not, however, support the mapping of sheets to teams. In particular, when generating code for the design, the code is written into only two files. Because of this problem, we used OMTool only during the requirements analysis, system and object design phases and did a design freeze before entering the implementation phase. Once the initial code was generated by OMTool, the OMT model was used only as a document but not for further code generation. This led to discrepancies between the final code and the OMT model.

Second, OMTool supports only object modeling. For the specification of the dynamic and functional model, we had to resort to drawing tools, which provided no support for maintaining a formal correspondence between the three models. OMTool would be a stronger tool if it supported the formulation of the object, functional, and dynamic models simultaneously, and checked for completeness and consistency among the models

7 Acknowledgments

This paper would have not been possible without the contributions of many people. First and foremost, we would like to acknowledge the contributions of the 30 students of the software engineering course at Carnegie Mellon University in the fall of 1991, who designed and implemented the system in such a short time. Without their enthusiasm and eagerness to learn about object-oriented design, the project course would have not been successful. Dave McKeown from Carnegie Mellon University wrote major parts of the problem statement and served as an internal client. He also provided us with the contacts to Ed Wells and Charles Baclawski from the Pittsburgh City Planning Department and Dave Wilde from Pittsburgh Port Authority Transit, who served as our external clients. They provided valuable feedback to the students during the various project reviews and accepted the delivered system after two attempts, giving the project a completely realistic feel. We would also like to thank Jon Bennett, Karen Fabrizius, Bruce Horn and Robert Coyne from Carnegie Mellon University for their thoughtful comments on earlier versions of the paper.

References

[1] Arnold, S. Bodoff, D. Coleman, H. Gilchrist and F. Hayes. *An Evaluation of Five Object-Oriented Development Methods*, Journal of Object-Oriented Programming, Special issue on Analysis & Design, 107-121, 1991.

[2] G. Booch. *Object-Oriented Design with Applications,* Benjamin Cummings, 1991.

[3] G. Booch. *The Booch Method: Notation*, Rational, Inc., Santa Clara, CA, 1992.

[4] G. Booch. *The Booch Method: Process and Pragmatics*, Rational,Inc.,Santa Clara, CA,1992.

[5] R. Buhr. *Practical Visual Techniques in System Design: with Applications to Ada*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[6] B. Bruegge, J. Cheng and M. Shaw. *A Software Engineering Course with a Real Client*, Carnegie Mellon University, Technical Report CMU-SEI-91-EM-4, July 1991.

[7] Frame Technology Corporation. *FrameMaker Documentation*, September 1990.

[8] E. Gibson. *Flattening the Learning Curve: Educating Object-Oriented Developers*, Journal of Object-Oriented Programming, February 1991.

[9] K. Gorlen, S. Orlow, P. Plexico. *Data Abstraction and Object-Oriented Programming in C++*, John Wiley & Sons Ltd., 1990.

[10] J. Guttag, J. Horning and J. Wing. *The Larch Family of Specification Languages*, IEEE Software, 24-36, September 1985.

[11] HOOD Technical Group. Hood Reference Manual, October 1990.

[12] Interactive Development Environments. *Software through Pictures (StP) User and Reference Manual*, March 1990.

[13] Interactive Pittsburgh. *System Documentation*, 15-413 course documentation submitted by the students, Carnegie Mellon University, December 1991.

[14] J.P. Jacquot, J. Guyard, L. Boidot. *Modeling Teamwork in an Academic Environment*, in Software Engineering Education, J.E. Tomayko (ed.), Lecture Notes in Computer Science, Springer Verlag,110-122, 1990.

[15] P. Knoke. *Medium Size Project Model: Variations on a Theme*,in Software Engineering Education, J.E. Tomayko (ed.), Lecture Notes in Computer Science, Springer Verlag, 7-24, 1991.

[16] M. Linton, J. Vlissides, P. Calder. *Composing User Interfaces with Interviews*, Computer, Vol 22, 8-22, February 1989.

[17] B. Liskov and J. Guttag. *Abstraction and Specification in Program Development,* MIT Press, 1986.

[18] V. Margo, P. Feibig and D. Etter. *Avoiding the Tarpits - Software's Challenge,* IEEE Potentials, 9 - 12, February 1990.

[19] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorenson. *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[20] J. Rumbaugh. *Object Modeling tool (OMTool) User's Manual*, GE Corporate Research and Development, Schenectady, NY, August 1991.

[21] M. Shaw and J. Tomayko. *Models for Undergraduate Project Courses in Software Engineering*, in Software Engineering Education, J.E. Tomayko (ed.), Lecture Notes in Computer Science, Springer Verlag, 33-71, 1991.

[22] J. Tomayko. *Teaching a Project-Intensive Introduction to Software Engineering*, Software Engineering Institute, Carnegie Mellon University, Technical Report CMU-SEI-91-EM-6, July 1991.

[23] L. Werth and J. Werth. *Teaching Object-Oriented Programming Using the Macintosh MPW/ MacApp Environment* in Software Engineering Education, N. Gibbs (ed.), Lecture Notes in Computer Science, Springer Verlag, 141-155, 1989.

[24] R. Wirfs-Brock, B. Wilkerson and L. Wiener. *Designing Object-Oriented Software*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[25]D. A. Young. *Object-Oriented Programming with C++ and OSF/Motif*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[26] E. Yourdon and L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Prentice Hall, Englewood Cliffs, NJ, 1979.