The Role of Integrated Specification Techniques in Teaching Complex System Modeling and Analysis

Robert B. France
CS&E Department
Florida Atlantic University
Boca Raton, FL 33431, USA
robert@cse.fau.edu

Jean-Michel Bruel*
Laboratoire IRIT/SIERA
Université Paul Sabatier
31062 Toulouse Cedex
bruel@irit.irit.fr

Abstract

The development of complex systems often requires the use of a variety of software modeling and analysis techniques. For example, in a single project one may use a graphically-based object-oriented modeling technique to model the structure of a system and a formal method to specify and analyze the critical parts of the system. Unfortunately, multiparadigm approaches to development are often overlooked in academic software courses and texts.

In this paper we describe a software engineering course we developed that emphasizes a multiparadigm approach to software modeling. In the course the students are introduced to an object-oriented and a formal method, and are shown how the two methods can be effectively integrated through examples in lectures and group projects involving the use of the methods.

1. Introduction

Many academic software engineering texts and courses present real-time system development techniques through a variety of formal and less-formal software development techniques. The problem is that most academic materials treat the development methods as alternatives, and present their application in singular terms, that is, specification examples and case studies are often presented in terms of a single method. This leaves the students with a collection of 'single-method' approaches to specification and analysis. This is not necessarily a 'bad' thing. If students are made to understand the strengths and limitations of the various specification techniques then they are in a position to choose the most appropriate techniques for particular types of problems. A deficiency of the 'single-method' approach is that it does not exploit another approach to specification problems: the integrated-method approach (or the multi-paradigm approach). The integrated-method approach takes the view that a single project may require the application of specification techniques from a variety of methods, for example, one may use a graphical object-oriented technique to model structure and a formal specification technique to specify the more critical aspects of behavior within a single project. We feel that students should be prepared to apply a variety of specification and analysis techniques, possibly from different methods, that are appropriately integrated.

*This work was partially funded by NSF grant CCR-9410396.

The particular problem that we attempt to address in our formal methods and software engineering courses relates to the use of formal specification techniques (FSTs) and graphical, structured object- and function-based techniques, such as Fusion and Structured Analysis/Real-Time, in the development of complex systems. The 'single-method' view taken in most academic materials often leads students to the the view that FSTs and the more graphical, but less formal structured techniques are alternative approaches to system development. Students are often mistakenly led to believe that FSTs cannot be applied in the context of more 'traditional' development methods. Our research indicates that integrated FSTs and structured methods can be used in the context of practical development processes and that they are benefits to their application over use of single FSTs, and structured methods.

We have used integrated methods and supporting processes in two undergraduate software engineering project courses to date. In both classes, students were asked to develop requirements models of a Students Advising System using an integrated Structured Analysis (SA) and Z method we developed as part of our research. At the end of the projects students presented their results and discussed their experiences with the use of the integrated methods. The students experiences indicated that they observed the complementary roles that the informal and formal techniques can play in system development and realized the benefits in terms of a deeper understanding of the problem they were modeling [6].

One of the major complaints made by students in these two projects was the lack of tool support for the integrated method. We have since then developed a tool environment, called FuZE, that supports the application of an integrated Fusion and Z method. FuZE is intended to facilitate practical application of FSTs in the development of complex systems. The environment supports the building and rigorous analysis of models that are precisely expressed and rigorously analyzable.

The conceptual integration model on which FuZE is based is given in Figure 1.

In the model a graphical object-oriented model is 'formalized', the result being a more precise model of the behavior and/or structure captured by the graphical OO model. The formalization activity can uncover problems with the graphical OO model related to incomplete, imprecise, ambiguous, and inconsistent requirements and design information. The insights gained from formalization can be used to improve the graphical OO models, that is, make them more precise and informative. The effect of formalization on the graphical OO model is shown as feedback from the formalization activity to the OO model. Once a formal model is produced, rigorous analysis of behavior and structure is possible. Such analysis can yield further insights into requirements and designs that can be used to further improve both the formal and OO models.

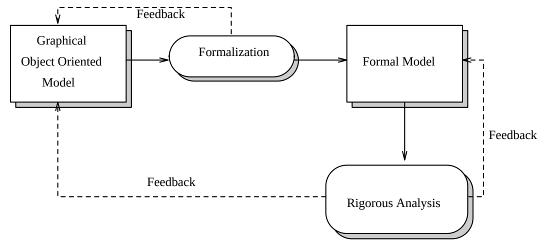

Figure 1. The integration process

In the software engineering graduate project course that will be offered in the Fall 1996 term, students will use FuZE with the Octopus [11] OO real-time system development method to build and analyze requirements and design models for a traffic management system. A major objective is to make students aware of how formal specification techniques can be used in conjunction with more graphical, but less formal, object-oriented analysis methods. In the following sections we describe the structure, and materials that will be used in the project part of the graduate course, CEN 6040 Computer-Aided Software Engineering.

2. Course Overview and Objectives

The Computer-Aided Software Engineering course will have two components: a lecture component and a laboratory component. The lecture component will focus on presenting to students materials on the principles, methods, and processes underlying the tools they will be using in the laboratory. In the early part of the course the laboratory sessions will reinforce the lecture components by providing students with case studies and exercises that complement materials on the development methods and notations covered in the lectures. Later, laboratory sessions will be used mainly for project related activities.

For the project, students will be divided into groups of 5 persons. They will be charged with developing requirements models, an architectural design, and implementable subsystem designs for a traffic controller.

The students will use the FuZE tool, developed by the Integrated Methods Group at FAU, in their project with the Octopus real-time system design method. FuZE supports the building of Fusion object-oriented (OO) models and their translation to Z specifications. For analysis, the translation to Z is mostly mechanical. For design, the translation requires significant human interaction, but the tool provides good support for the more mundane tasks of formal specification, as well as support for rigorous analysis of creative results.

At the end of the course students should:

- understand the complementary roles that formal and less formal, highly-structured, graphical techniques can play in modeling and analyzing application behavior and structure,
- be familiar with, and understand the rationale behind processes required to make effective use of integrated formal and informal techniques in the object-oriented modeling and analysis of complex systems,
- have created and analyzed a problem with safety-critical and real-time properties, and
- have working knowledge of the FuZE CASE environment, and its underlying modeling and specification techniques.

3. Support materials for course

A World Wide Web (WWW) page on FuZE is currently being developed to support the use of FuZE on projects. Included in the page will be a tutorial for first time users, case studies on the application of FuZE (for a simple traffic control system, a student advising system, and a gas dispensing system), and a selection of papers on the tool and the underlying rules.

Students will also have one day of scheduled classes in the Software Engineering Laboratory (consists of Sun workstations). The scheduled Lab times will not be sufficient to complete the project; students are expected to use the open lab outside of class hours.

The following will be the text books for the course:

Required Texts:

- Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion; by M. Awad, J. Kuusela, and J. Ziegler; Prentice-Hall.
- An Introduction to Formal Specification and Z; by B. Potter, J. Sinclair, and D. Till; Prentice-Hall.

Recommended Texts:

- Object-Oriented Development: The Fusion Method; by D. Coleman et. al.; Prentice-Hall.
- Object-Oriented Modeling and Design; by J. Rumbaugh et. al.; Prentice-Hall.
- The Z Notation; by J. M. Spivey; Prentice-Hall.

The text books will be supplemented by lecture notes and a set of papers on FuZE and real-time system modeling.

Students are assumed to have the following skills and knowledge:

- Skills related to programming in the small (e.g., structured programming).
- Knowledge of key software engineering principles (e.g., separation of concerns, rigor, and incrementality), in particular, basic design principles (e.g., design for change).
- Knowledge of software development processes.
- Familiarity with object-oriented modeling methods.
- Have at least undergraduate level Discrete Maths.

4. Course Contents

Each lecture is one hour and twenty minutes. Lectures are held twice a week.

1. Introduction (Lecture C1; week 1)
   (a) Review of object-oriented software engineering principles and development processes.
   (b) Real-time system development: an overview.
2. Integrating formal and informal techniques (Lecture C2; week 1)
   (a) Introduction to formal specification techniques.
   (b) Rationalizing integrated techniques.
   (c) A development model for integrated techniques.
3. Integrated OO techniques and Z (Lectures C3-C16; weeks 2-14)
   (a) The development process.
   (b) A small example.

(c) The Fusion analysis notation.

(d) The Z notation.

(e) Integrating Fusion analysis and Z.

(f) Real-time extensions of Fusion design: The Octopus approach.

(g) Integrating Z and real-time extensions of Fusion.

4. Using FuZE (Lab L1-L3; weeks 3-5)

    (a) Modeling in FuZE.

    (b) Rigorous analysis in FuZE.

    (c) Small case study.

5. The Problem (Lab L4; week 6)

    (a) Presentation of the Traffic Control System problem.

    (b) Refining the problem statement.

6. Project planning and execution. (Lab L5-L12; weeks 7-14)

7. Project reviews. (Lab L13; week 15)

8. Project analysis. (Lab L14; week 16)

Sections 1 to 3 will be covered in the lecture component of the course, and sections 4 to 8 will be covered in the lab component of the course. The lab component will commence after the Fusion notation is introduced in the lecture component (i.e., after 3(a) lectures). Early lab assignments will focus on reinforcing Fusion and Z modeling concepts introduced in the lectures, through the use of FuZE.

5. The Lecture Component

5.1. Introduction

The introductory part of the course gives students an overview of the principles they will be applying in the project. The materials presented in this section should provide the students with the context needed to understand and justify the use of the processes, methods, and tools used later in the course. The intent is to make clear to students how the development mechanisms they use support (and/or hinder!) real-time system development goals.

The nature of software engineering will be discussed, a review of significant software engineering principles (as discussed in [4]: Rigor and formality, separation of concerns (e.g., views, abstraction, modularity), generality, anticipation of changes, and incrementality), and a review of how object-oriented techniques support software engineering principles and concerns will be given.

Students will also be given an overview of the special concerns of real-time system development here.

5.2. Integrating formal and informal techniques

The objective of this part of the course is to motivate the use of integrated formal and informal techniques in software development, and to instill in the students an appreciation for the complementary roles formal and graphical informal techniques can play in software development.

During the early eighties research on systematic approaches to software development resulted in a number of graphically-based specification methods for real-time system, which we will collectively refer to as *structured specification methods* (SSMs) (e.g., see [8]). In most cases, these methods were extensions to existing *Structured Analysis/Design* (SA/SD) methods. The specifications produced by SSMs were often more concise and sometimes more precise than purely natural language specifications. The evolution of SA/SD-related research and practice has resulted more recently in the creation of *object-oriented methods* (OOMs) (e.g., see [5, 13]). Most popular graphical object-oriented modeling techniques support the creation of models that are concise, visually-appealing, and abstract. On the other hand, the lack of firm semantic bases for the modeling constructs used makes it difficult to rigorously reason about the structures and behavior modeled, and can make understanding the models more apparent than real. Most SSMs and OOMs are *informal* in the sense that their applications are likely to produce ambiguous specifications that are not amenable to rigorous semantic analyses. In this course, we refer to such techniques as *informal specification techniques* (ISTs).

The critical nature of some real-time software systems often necessitates the use of FSTs. Here, a FST consists of a precise language and mechanisms for deriving consequences from statements in the language. Despite the benefits FSTs can bring to development, there are some problems with their application to the development of non-trivial applications. Among the problems are the lack of adequate structuring mechanisms, making specification development a mostly monolithic task and the resulting specifications difficult to read, and the scarcity of tools that ease the creation and analysis of specifications.

The application of *Integrated FSTs and ISTs* (FISTs) can be beneficial in the following respects:

- *FISTs enable an evolutionary approach to the use of FSTs in industry.* An FIST enables the use of FSTs in the context of the ISTs. This allows specifiers to become familiar with FSTs while still taking advantage of their IST skills. FISTs allows an organization to preserve, and even enhance, its investment in ISTs while taking advantage of FST-related benefits.

- *FSTs and ISTs can complement each other.* The highly-structured and graphical nature of IST specifications makes them more presentable than the more detailed, often textual, formal specifications. Furthermore, the flexibility resulting from the use of intuitively-defined concepts and visual constructs makes ISTs suitable for the early, probing phases of software development, since they are more likely to result in abstract models that can be used to gain a high-level understanding of the problem or solution. The abstract models produced through the use of FSTs are often more difficult to understand. On the other hand, the lack of a firm semantic basis for ISTs inhibits their use in rigorous specification and analysis of behavior. FSTs are needed for such activities. FSTs can also enhance the applications of ISTs. Formalizing an informal specification often reveals ambiguities, inconsistencies, and gaps in the informal specification.

- *An FIST can be used as a basis for systematic transformation of abstract notions of behavior and structure to more precise expressions.* Requirements engineering is concerned with transforming vague statements of requirements into more precise statements of the problem. During this transformation a number of intermediate specifications, each varying in degree of rigor and detail, may be produced. FISTs allows one to impose structure on the transformation of vague requirements to formal specifications. FISTs can play a similar role in design, where design is viewed as the creation of a series of decreasingly abstract solution specifications.

In this course the students will use a formalization process called the *P-E-V* model to guide the application of the integrated techniques to the project problem. In the P-E-V model, development within a lifecycle phase is viewed as an activity consisting of a *probing (exploratory) phase*, an *elaboration phase*, and a *validation phase*.

In the probing (exploratory) phase, the modeler attempts to impose a structure on the problem or create a solution structure. Given the exploratory nature of this phase, the tool support should be flexible, that is, allow for the easy incorporation of changes, and should allow the modeler to view the problem or solution in an abstract, but insightful, manner. Graphical object-oriented methods (OOMs) such as Fusion and Octopus are well-suited to this phase since they mostly employ graphical constructs supported by rich structuring mechanisms, and provide visual abstract views of structure and behavior from a variety of perspectives. Formal specification techniques, on the other hand, are not well-suited to this phase because they may produce models of structure and behavior that are not as easy to understand and modify.

When the modeler is satisfied that an adequate structure has been developed, then he proceeds to the elaboration phase where details pertaining to elements in the structure and their relationships are more rigorously specified. In this phase, formality should be favored over informality because the objective is to create a specification that is analyzable and precise. During the elaboration phase, ambiguities, inconsistencies and incomplete information may be identified, and attempts at resolving them may lead to modifications of the structure initially defined for the problem or solution. Development in this model is thus an iteration of probing and elaboration phases activities, as indicated by the double headed arrow between the two phases in Figure 2.

The validation activity is concerned with establishing the validity of the models. Animation and proof techniques can be used to support formal reviews here.

5.3. Integrated OO techniques and Z

In this section of the course the students will cover the integrated techniques and processes that will be used in the project. The integrated technique is based on Fusion, Z, and the Octopus [11] real-time extensions to Fusion.

The system development process that will be introduced (and used in the project) is based on the Octopus development method. The model, shown in Figure 2, consists of three major phases: Requirements Specification, Architectural Design, and Subsystem Development.

The Requirements Specification phase is concerned with modeling the desired behavior of the system in terms of the behaviors that are observable at the

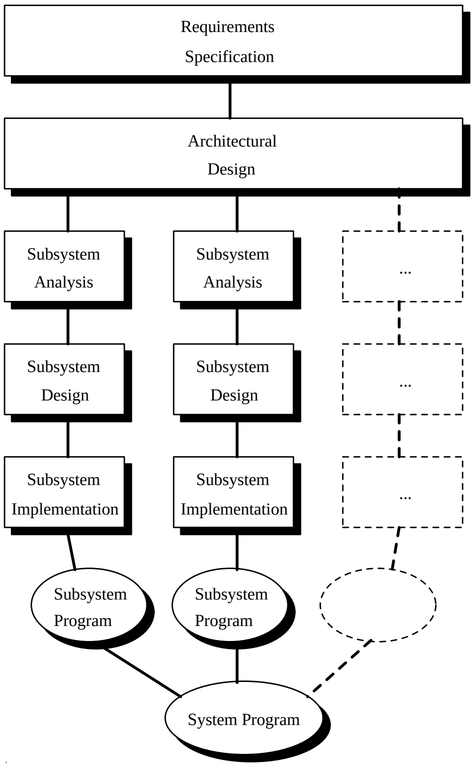

Figure 2. A Development Process for Real-Time Systems

boundaries of the system. Concurrency is implicit at the requirements level, that is, each object is assumed to have its own processing resources. Fusion analysis models and Z specification will be used to to express the static and dynamic aspects of the required behavior. Constraints (i.e., non-functional requirements) will be expressed in English text, and, when appropriate, used to annotate the models.

In the Architectural Design phase subsystems and their interfaces will be identified and defined. Here, a subsystem consists of objects and their relationships. The Fusion object model produced in the requirements phase will be used as a base for identifying subsystems; classes in the model will be grouped together into subsystems, where a subsystem is intended to capture a particular domain (i.e., an area that has a relatively independent body of knowledge, terminology, and solution strategies). The structure of the architecture is modeled by Fusion analysis models, and the interfaces are defined using Fusion operation schemas.

Once the subsystems are identified each subsystem is analyzed, designed, and implemented. In the analysis phase the object model is extended with design objects, a high-level description of state dependent behavior for the subsystem is created, and the significance of events is determined. The state-dependent behavior of a subsystem is defined in terms of Harel's statecharts [7], and is produced using the guidelines provided by the Octopus method. The analysis of the significance of events is also guided by the Octopus method.

The design phase of subsystem development introduces concurrency, priorities and time. The extensions to the Fusion Object Interaction Graphs made by Octopus will be used to model interactions between object instances in a model. The Octopus also provides guidelines for grouping objects into operating system level processes (instances that communicate synchronously are grouped into processes) and for handling shared objects (objects belonging to more than one group).

In each of the phases the P-E-V model will be used, that is, students will first express their ideas using the graphical notations and then attempt to formalize those parts that are deemed critical and/or requires some analysis (using Z). The students will not be required to formalize all aspects of their design.

In the lectures students will be introduced to Fusion, Z, and the particular Octopus notations that will be used in the project. In the Fusion part, analysis and design models will be covered. For analysis the Fusion object (static) and interface (dynamic) models will be emphasized, and for design the Fusion Object Interaction Graphs, statecharts (used in Octopus), and scenarios will be introduced. The material in this part will be covered in 4 lectures. In the laboratory section, the students will develop Fusion analysis and design models for a small library problem.

The Z part of this section will include a review of predicate logic and set theory, the Z type system, coverage of Z functions, relations, sequences, schemas and schema operations (inclusion, decoration, logical connectives, composition, piping), and an introduction to formal reasoning with Z. The material will be covered in 4 lectures. Students will develop Z specifications for the library problem in the lab component of this course using the Formalizer tool.

The materials in this part of the course will be covered in 12 lectures.

6. The Laboratory Component

6.1. Using FuZE

FuZE (Fusion/Z Environment) is an automated support environment for creating Fusion object-oriented analysis models and for transforming them into Z specifications [14]. FuZE is built on top of the Paradigm Plus object-oriented modeling CASE tool [12]. The extensions to the tool we have made thus far allow modelers to generate Z specifications from Fusion object models and from state transition diagrams. From Paradigm Plus, Z analysis tools, e.g., typecheckers and animators, can be called to analyze the generated specifications.

Paradigm Plus (Release 2.01) provides support for modeling based on OMT [13], Fusion [5], Booch [2] and others, and automatically generates C, C++, Smalltalk or Ada code from the models. A powerful script language allows custom reporting, checking, and code generation. We used the script language to build FuZE on top of Paradigm Plus. Figure 3 gives an overview of the process that allows one to formally analyze a Fusion model in FuZE. Beginning with an

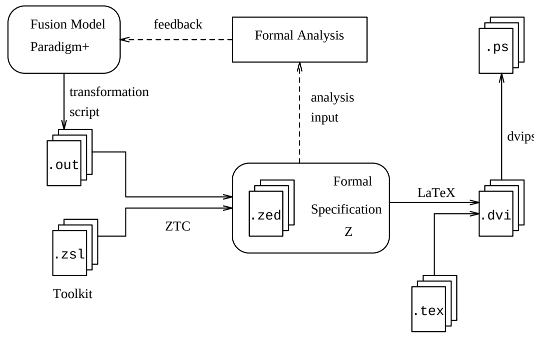

Figure 3. From a Fusion Object Model to a type-checked Z specification

Object Diagram in Fusion, the formalization process generates a `.out` file of the corresponding Z specification. A Z tool, ZTC [10] is then launched by running a script, to type-check the specification and produce a printable file. A `.tex` file can also be generated by a script to allow the use of LaTeX to generate a Postscript version of the specification. The `.zed` file can also be analyzed by a Z animator such as ZANS (this tool has not been integrated into FuZE as yet).

It is not our aim to detail the rules in this paper (for more details see [1]).

Paradigm Plus can be customized by its powerful, full-featured script language. The script language supports methods checking, reports and code generation. The script language syntax is similar to the syntax of Basic-like languages. Familiarity with higher level languages makes writing scripts in Paradigm Plus easier.

To translate a Fusion model into a Z specification, one selects from a menu the option to run a script (see Figure 4). More details about the results of automatic

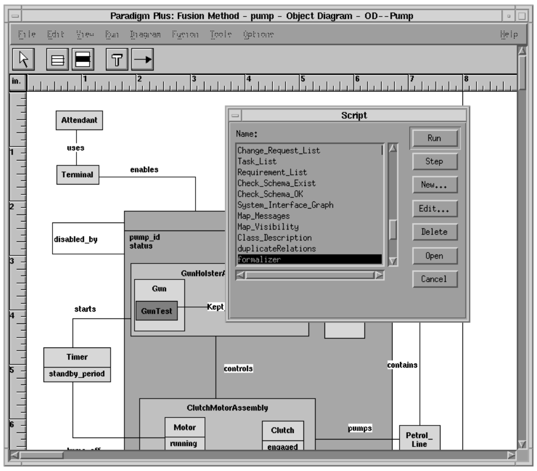

Figure 4. Selection of a transformation rule

translations in FuZE can be found in the draft FuZE technical report [3].

As illustrated by Figure 3, we use ZTC to type-check the specification. The Paradigm Plus script language allows one to access the operating system shell. We use this feature to invoke ZTC with the script output and the mathematical toolkit as its inputs. After the output result of the translation rules, ZTC is launched to type check the generated Z specification and to generate (if the specification is correct) the corresponding LaTeX output.

We also include in the output specification some comments used by the ZANS [9] animation tool that allows one to animate the specification by defining very simple operations that manipulate the objects in the model.

6.2. The project

The project problem is stated as follows:

> On the island of St. May there is an intersection at the top of a hill that is notorious for its rush hour traffic snarls and pedestrian accidents. The junction, known as Casson Hill Point, occurs where a major road linking the east side to the west side (this is the only link) intersects with a road linking two residential areas. The Works department, after numerous complaints from residents in the area, have decided to install traffic and pedestrian lights at the junction. Each approach to the junction will have a pedestrian crossing (with pedestrian lights on each side), a sensor to detect the presence of vehicles on the approach, and turning lanes. The traffic system is required to be safe, reliable, maintainable, easy to extend, and effective.

One of the first tasks of the requirements analysis is to analyze the problem statement and make it more precise. This involves first identifying and distinguishing the functional and non-functional requirements. There is a single functional requirement: the system should control the flow of traffic. The non-functional requirements concern safety, reliability, maintainability, extensibility, and effectiveness. The functional requirement can be refined using the models supported by FuZE and Octopus. Refinement of the non-functional requirements is also needed so that their impact on the required behavior can be precisely determined. For example, the non-functional safety requirement can be refined as follows:

> A safe system will not cause traffic to flow across a designated crossing point at the same time, and will allow a safe period to elapse between stopping a flow and allowing a crossing flow to proceed. More precisely, no two crossing flow roads should allow traffic flow within 2 seconds of each other.

The effectiveness requirement can be refined as follows:

> Deadlock of the system should not occur, specifically, the system should not allow the four approaches to the intersection to be closed for more than 5 minutes when traffic has arrived during the first 4 minutes of the period. Also, no motorist or pedestrian should have to wait more than one minute to proceed. During rush hours (between 7 and 8am and between 4 and 5pm) the traffic on the east/west road is heavier; in these cases, motorists on the residential road may have to wait more than one minute, but no more than 2 minutes, before proceeding.

The result of the requirements refinement is a precise model of the functional requirements, expressed in terms of Fusion analysis models and Z, and statements detailing how the non-functional requirements constrain the functionality of the system. When these are produced, a review meeting is held. Before the meeting, the students will exchange their requirements documents. They will be given 7 days to carry out the review. In the next scheduled lab class (at the end of the 7 days) a review meeting will be held where the students discuss problems they uncovered. The lecturer will play the role of the customer in the review. After the review, the students address any problems uncovered and submit the changed requirements document to the lecturer. The students will be given 3 weeks to complete a first pass at requirements specification.

The design will be done in an incremental manner, where the students first focus on defining the core subsystem(s) and then (as time permits) extend the design to cover other aspects of the design. For example, the students may decide to ignore pedestrian control and turning signals in the first increment. It is expected that students will spend approximately 5 to 6 weeks on the design. At the end of each increment a review will be held. The review will be carried out as described previously. It is expected that students complete at least one increment during this course.

7. Conclusion

We realize that the course outlined as above is intensive. Some of the problems we expect to encounter are as follows:

- Students will be overwhelmed by the number of notations used.
- Students may spend more time on learning to use the tool than on developing solutions to the problem.
- For class sizes greater than ten, course management could be a nightmare.

To alleviate these problems we plan to keep class size below ten students (i.e., two groups), and provide facilities such as an online tutorial for the tool, a WWW page on using the tool, and an electronic bulletin board where students can post their questions and complaints to the lecturer and the research assistants that developed the tool. Furthermore, we plan to have a research assistant hold tutorials, one hour per week on the tool. During these tutorials, students will delve deeper into features of the tools introduced in the classroom, and the research assistant will address any project-related problems that the students may have.

Acknowledgments

This work was supported in part by NSF grant CCR-9410396. The authors wish to thank Maria Petrie for her participation in this project. We are also grateful to Bharat Chintapally and Gopal Raghavan for implementing the rules into FuZE. The authors would also like to thank the referees for their valuable observations.

References

[1] B. W. Bates, J.-M. Bruel, R. B. France, and M. M. Larrondo-Petrie. Formalizing Fusion Object-Oriented Analysis Models. In E. Najm and J.-B. Stephani, editors, *Proceedings of the First IFIP International Workshop on Formal Methods for Open Object-based Distributed Systems, Paris, France.* Chapman & Hall, London, U.K., 4–6 March 1996.

[2] G. Booch. *Object-Oriented Analysis and Design with Applications.* Second edition. Benjamin/Cummings, Menlo Park, CA, 1994.

[3] J.-M. Bruel, B. Chintapally, R. B. France, and G. K. Raghavan. FuZE–Draft of the User's Guide. FAU Technical Report TR-CSE-96-9, Department of Computer Science & Engineering, Florida Atlantic University, Boca Raton, USA, February 1996.

[4] M. J. C. Ghezzi and D. Mandrioli. *Fundamentals of Software Engineering.* Prentice Hall, Englewood Cliffs, NJ, 1991.

[5] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development: the Fusion Method.* Prentice Hall, Englewood Cliffs, NJ, 1994.

[6] R. B. France and M. M. Larrondo-Petrie. Understanding the role of formal specification techniques in requirements engineering. In *Proceedings of 8th SEI Conference on Software Engineering Education*, pp. 207–222. Springer-Verlag, New York, 1995.

[7] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3), 1987.

[8] D. Hatley and I. Pirbhai. *Strategies for Real-Time System Specification.* Dorset House, New York, 1987.

[9] X. Jia. *An Approach to Animating Z Specifications.* Division of Software Engineering, School of Computer Science, Telecommunication, and Information Systems, DePaul University, Chicago, IL, USA, 1995.

[10] X. Jia. *ZTC: A Z Type Checker, User's Guide, version 2.01.* Division of Software Engineering, School of Computer Science, Telecommunication, and Information Systems, DePaul University, Chicago, IL, USA, May 1995. Available via anonymous ftp at `ise.cs.depaul.edu`.

[11] J. Z. M. Awad, J. Kuusela. *Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion.* Prentice Hall, Upper Saddle River, NJ, 1996.

[12] Protosoft, Inc. *Paradigm Plus – Release 2.0.* Protosoft Technical Support, Houston, Texas, USA, January 1994.

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

[14] J. M. Spivey. *The Z Notation: A Reference Manual.* Second edition. Prentice Hall, Englewood Cliffs, NJ, 1992.