An Approach To Integrating Semi-formal and Formal Notations in Software Specification

Sergiu Dascalu Peter Hitchcock
Faculty of Computer Science, Dalhousie University
6050 University Avenue, Halifax, NS, B3H 1W5, Canada
1 (902) 494 6449
{dascalus, hitchcp}@cs.dal.ca

ABSTRACT

In this paper the integration of graphical, semi-formal modeling languages with formal notations for software specification purposes is discussed and a proposal for a procedural frame based on the combined use of UML and Z++ is presented. This procedural frame, organized in stages and steps, provides the methodological basis for a pragmatic and rigorous object-oriented modeling approach aimed at the construction of larger software systems, including real-time systems. Within the proposed frame a regular flow of modeling activities is suggested and alternative modeling scenarios are considered. A brief presentation of the Harmony integrated specification environment, a tool designed to support the proposed approach, is also included in the paper.

Keywords

object-oriented modeling, formal methods, integration of notations, UML, Z++.

1. INTRODUCTION

Numerous authors have indicated the benefits of integrating formal techniques with conventional, informal or semi-formal approaches in software development. For instance, Aujla et al. point out that formal techniques are portable and extendable and can be used in various phases of the development as complementary techniques or as alternatives to traditional, non-formal approaches [2], Alexander sees the combination of formality and informality as a way to obtain "the best of both worlds" [1], and Bruel et al. point out that "the main objectives of integrated formal/informal approaches is to make formal methods easier to apply and to make informal methods more rigorous" [8, pp. 52]. Other authors who consider beneficial the integration of notations include Fiona Pollack [27], Moreira and Clark [23], Berry and Weber [5], and Chen et al. [9].

Integration, which in general covers combination of notations, models, and methods [2], has nevertheless its own issues, most notably the fact that interpretations underlying the translation rules from informal to formal are seldom explicitly stated, the focus of formalization is usually on basic constructs, and not structures, and little attention is paid to relating the results of analyzing the generated formal models to the corresponding components of the informal counterparts (Bernhard Rumpe, in [8]). However, using complementary, concerted techniques for developing software systems generally provides greater modeling power, supports the early detection of errors, and increases the developers' confidence in the correctness of the software product being developed. Evidently, these advantages did not pass unnoticed by the researchers and practitioners of the software engineering field and various combination strategies have been proposed. In broad terms, the relationship between the semi-formal (or informal) and the formal components of a specification can be one of the following (note that we refer in particular to semi-formal graphical notations such as UML or data flow diagrams and do not cover formal graphical notations such as Petri Nets or Statecharts):

- If the semi-formal (or informal) part is built initially and then the formal counterpart is obtained through a translation process, we can speak about *derivation* of the formal model from the semi-formal model, or of *formalization* of the semi-formal model (e.g., [22], where diagrammatic and text elements of Bailin's object-oriented requirements specification method OOS are translated into Z counterparts). The reverse process is also possible, albeit rarely attempted (e.g., [19], where Z constructs are mapped to graphical representations). In this case the translation between the formal and semi-formal (informal) parts of the model can be referred to as *deformalization*;
- If in addition to diagrammatic representations some related formal specifications are obtained independently (e.g., [17], where Z specifications supplement UML models), the approach can be described as *complementary formalization*. Generally, this approach also involves the derivation from semi-formal to formal specifications of a subset of the diagrammatic descriptions of the system;
- If changes in any of the specification's parts are propagated to the other, a *tight integration of notations* is achieved (e.g., [28], where continuous, bi-directional links exist between UML models and their corresponding ZEST specifications).

In the above classification the terms semi-formal part and formal part of a specification are used but we should point out that, due to the costs involved, formalizing the entire specification of a software product is generally impractical, if not impossible, and the typical approach is to apply formal techniques only to the critical sections of the software being developed [14]. As such, the correspondence between the diagrammatic (semi-formal) and textual (formal) parts of a specification is typically limited to a subset of the specification's components.

Depending on the number of notations involved, a combination of notations can take the form of either a *dual-notation integration* (e.g., [6], where UML is combined with SDL) or of a *multiple-notation integration* (e.g., Zave and Jackson's multi-paradigm specification technique [34], Paige's pure formal method integration strategy [25], and Day and Joyce's framework solution for integrating multiple notations [11]). Generally speaking, the integration does not necessarily involve a *formal/semi-formal* (or *informal*) combination; it can be of the *formal/formal* type (e.g., [31], where the dynamic aspects of RTS are modeled using Statecharts and FNLOG) or *semi-formal/semi-formal* (e.g., [29], where an integration UML/Lean Cuisine+ is proposed for supporting the early stages of interactive system design).

We believe that the integration of notations can provide a viable solution for modeling complex systems especially because various aspects of the systems need be described in various ways (for a classification and examination of forms of method complementarity, primarily in terms of notations and processes, we suggest [26]). In particular, in the case of formal/semi-formal integrations, it is always possible to "fine tune" the formality level and adjust the balance between the less rigorous diagrammatic representations and the formal specifications to best handle the requirements of a specific application.

Within this context, we propose an object-oriented specification approach based on the combined used of UML [32] and Z++ [21] and aimed at the construction of real-time systems. UML has been incorporated in our approach for obvious reasons: it has remarkable modeling power, is extensible, enjoys a large acceptance in the software development community, and has become the standard notation for object-oriented development. On the other hand, Z++ also brings significant advantages: it is formal, object-oriented, and provides support for specifying real-time systems. In particular, in order to capture the temporal properties of the systems we employ within the frame of Z++ Jahanian and Mok's Real-Time Logic (RTL) [16] and make use of the extensions to RTL proposed by Kevin Lano [21]. A key issue of the approach is the harmonized use of the two notations, which can be greatly helped by a bi-directional link between the diagrammatic and formal parts of the system's model. For this purpose, translation algorithms between a subset of the UML part of the model and its Z++ counterpart have been designed. These algorithms, which support the formalization and deformalization processes and address both structural and behavioral aspects of the system, are described in detail in [10].

The present paper, in its remaining part, is structured as follows. Section 2 describes the proposed procedural frame in terms of modeling activities performed and artifacts produced, suggests a regular flow of activities, and considers alternative modeling scenarios. Section 3 introduces Harmony, the environment designed to support the proposed specification approach. The concluding section 4 analyzes the status of our work, compares it with related integration strategies, and indicates several aspects of our work that we intend to improve in the near future.

2. THE PROCEDURAL FRAME

The specification approach presented in this paper is focused on the structural and behavioral aspects of systems and is aimed at developing object-oriented models in a rigorous, pragmatic, and efficient way. For practical reasons, a number of modeling activities supported by UML are not included in the procedural frame described below and their corresponding artifacts (specifically, diagrams) are not incorporated in the integrated UML/Z++ model. This simplification is possible because the above diagrams are either parallel to some already incorporated (specifically, collaboration diagrams are essentially re-writings of sequence diagrams), can be ignored without losing significant insight into the system (activity diagrams), or can be deferred to later development stages that are beyond the scope of the proposed approach (component diagrams and deployment diagrams).

By considering the usual 4+1 architectural views approach advocated by many (e.g., [7], [30]), only the user view, the structural view, and the behavioral view are dealt with in our approach, and from the diagrams that support the architectural views only the use case diagrams, the class diagrams, the sequence diagrams, and the state diagrams are employed. In short, we model the architecture of the system using 2+1 views, a reduction of the generic 4+1 views approach that nevertheless allows a reliable description of the system. It is worth noting that many of the UML applications described in the recent literature focus typically on use cases, scenarios, class diagrams, and statecharts diagrams (e.g., [3], [15], [18], and [33]) and less frequently other types of diagrams are also presented (e.g., [4] and [12]). In fact, having the class organization completed in terms of both attributes and operations allows the further development of the system possibly up to and including implementation. This is not to say, however, that the omitted diagrams cannot be drawn if necessary, but certain simplifications are needed in order to allow the combined use of techniques in a manner which is not only precise but also practical, lightweight, and rapid.

2.1 Artifacts

Starting from a set of requirements that describe the desired properties of the system, the following five categories of artifacts are developed, making up the combined semi-formal/formal model of the system:

- *Use case diagrams* (UCD), describing the intended high-level behavior of the system as seen from outside the system. These are typical UML use case diagrams, each capturing a portion of the system's externally visible behavior and each containing a number of use cases (UC) that detail this behavior;
- *Scenarios* (SC), specific sequences of actions involving the system and the actors that interact with it. Scenarios, as pointed out by Booch, "are to use cases what instances are to classes, meaning that a scenario is basically one instance of a use case" [7, pp. 225]. UML provides sequence and collaboration diagrams for representing scenarios; however, these diagrams involve a high level of details so we make a distinction between scenarios and sequence diagrams. Specifically, we see a scenario as an informal, analysis-level description of a particular sequence of actions encompassed by a use case, while a sequence diagram is a detailed, design-level description of the same thing in which

responsibilities for carrying on actions are assigned to individual classes and objects (as opposed to the system as a whole);

- *Sequence diagrams* (SQD), developed using the UML notation and providing a design-level representation of scenarios, as described above;
- *Class diagrams* (CD), defining the high level architecture of the system and consisting essentially of classes and relationships among classes;
- *Class compounds* (COMP), each class compound containing a regular class description (CLS) and the state diagram (CLSTD) associated with the class. The notion of class compound is introduced primarily for supporting the formalization process, but it represents in general a simple yet useful extension of the concept of class ("a class with enhanced description of behavior"). The idea of a class compound comes primarily from Z++, but it has also been inspired by Howerton and Hinchey, who propose the combination of UML descriptions and Z specifications in an approach that advocates different notations for modeling different aspects of the system [15]. (However, Howerton and Hinchey do not propose the syntactical concatenation of the UML class and state diagram constructs, and do not suggest a name for their solution);
- *Z++ specification* (ZSPEC), consisting of a set of Z++ classes (ZPPC), each Z++ class corresponding to a class from the UML space. The Z++ specification as a whole is the formal counterpart of the combined contents of the class diagrams that make up the UML component of the system's integrated model.

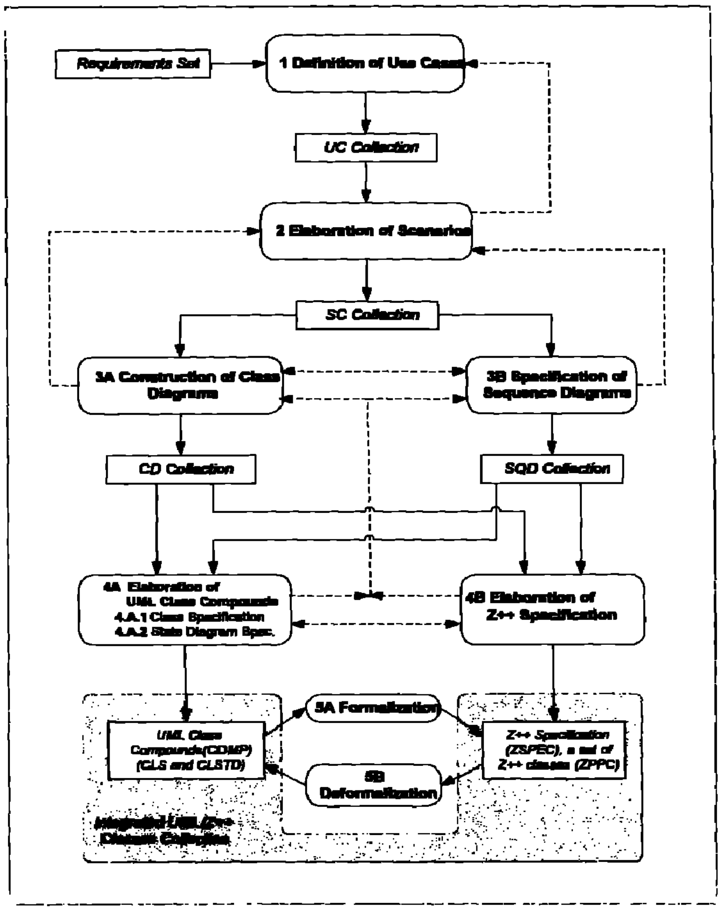
2.2 Activities

Figure 1 gives a diagrammatic description of the proposed procedural frame in terms of modeling *activities (steps)* performed and *artifacts (products)* obtained. Several conven-tions are used in this figure:

- Activities are represented by rounded rectangles;
- Artifacts are represented by regular rectangles;
- Continuous, arrow-ended lines connect activities with their output artifacts and artifacts with activities that use them as input;
- Dashed, arrow-ended lines represent a transfer from an activity to another which does not necessarily require that artifacts are obtained in the originating activity (the decision to move to another activity can be based on the inspection of the already existing artifacts associated with the current activity). These dashed lines are typically used as feedback links in the iterative development of the system's model. For simplicity, forward dashed links are not depicted in Figure 1 and feedback links from activities 5A and 5B to activities 4A and 4B are also omitted;
- The steps are numbered and organised in five stages (or levels), their ordering suggesting the typical flow of activities within the modeling process;
- The set of diagrams obtained as a result of a specific activity in stages 1 to 3 are generically denoted *collection*.

The diagram presented in Figure 1 is flexible enough to accommodate various specification strategies and encompass diverse modeling paths, as discussed more in the next subsection. The activities included in our modeling approach are the following:

- At stage 1, starting from the requirements set that describes the desired system, a number of use cases that capture segments of externally visible system functionality are identified, making up the UC collection of the integrated model;



Requirements Set
1 Definition of Use Cases
UC Collection
2 Elaboration of Scenarios
SC Collection
3A Construction of Class Diagrams
3B Specification of Sequence Diagrams
CD Collection
SQD Collection
4A Elaboration of UML Class Compounds
4.A.1 Class Specification
4.A.2 State Diagram Spec.
4B Elaboration of Z++ Specification
UML Class Compounds(COMP) (CLS and CLSTD)
5A Formalization
5B Deformalization
Z++ Specification (ZSPEC), a set of Z++ classes (ZPPC)
Integrated UML/Z++ Element Collection

Figure 1. The Procedural Frame

- At stage 2 use cases are employed to instantiate a number of scenarios that will serve later for the identification of classes. Normal scenarios (most likely to occur), as well as abnormal scenarios (describing situations that diverge from the normal case) are developed and included in the SC collection of the system's model;
- At stage 3, using the available scenarios two possibly intertwined activities can take place: construction of class diagrams (3A) and specification of sequence diagrams (3B). In practice, the specification of sequence diagrams can be deferred after 3A since in general it is easier to construct the class diagrams by exploiting the information contained in scenarios (the class diagrams may contain only the names of the classes, without other details, while sequence diagrams necessarily include class and object names as well as class operations). In fact, step 3B can be skipped in certain situations, as discussed in section 2.3. The best thing, however, is not to omit it, and to use it at least as a "revision checkpoint," with input from all subsequent levels;
- At stage 4 the CD collection and the SQD collection (if available) provide the basis for the detailed specification of classes. An argument can be raised about the development of classes represented separately from the development of class diagrams and, indeed, there is a blurred line between these two activities. We separate them for systematization purposes and view the specification of class diagrams as an activity in which the rough sketch of the system's class structure is drawn (in terms or

classes, relationships, and cardinality constraints) while the subsequent activities of UML and Z++ class elaboration are concerned with the specification of class details (attributes, operations, and constraints). Regarding the "parallel" steps 4A (elaboration of UML class compounds) and 4B (elaboration of Z++ specification) we note that they can be started and performed simultaneously (this is the reason for placing them on the same level) but the typical way is to perform step 4A first or to perform only the step 4A and rely on the subsequent formalization of class compounds (step 5A) to obtain the Z++ specification of classes;

• At stage 5 the formalization of selected UML class compounds takes place in step 5A by initially applying the algorithms for automated translation and then by manually adding the necessary details to the formal specification. This activity has the role of producing rigorous descriptions of the system, captured in the Z++ specification, and provides the strongest basis for refining the model –many ambiguities, omissions, and inconsistencies are detected here. At the same level of modeling, deformalization of classes initially written in Z++ (step 5B) can be performed following the set of guidelines suggested in [10].

2.3 "Regular" and "Irregular" Sequences of Modeling Activities

A graph-like representation of the regular sequence of modeling activities, which for simplicity omits the products of each activity, is represented in Figure 2. (In UML terms, this can be seen as the normal scenario of the use case represented by the procedural frame described in Figure 1). The modeling stages are highlighted, the direct flow of activities is emphasized by a continuous line, and the iterative revisions of specifications are indicated by a dashed line. This scenario, which in its "forward segment" (that is, not including feedback links) does not encompass the deformalization activity (reserved for "irregular" modeling scenarios), can be succinctly described by the sequence <1, 2, 3A, 3B, 4A, 5A, 4B>, where numbers are associated with activities as indicated in Figure 1.

The procedural frame presented in Figure 1 encompasses different orderings of activities and we do not claim that the "regular" flow suggested above is the unique or the most effective way of developing the integrated UML/Z++ model of the system. There are other alternatives possible, and depending on the particular application, on the experience of the development team, as well as on a series of other factors, including project priorities and deadlines, one of these alternatives may be better suited for the particular needs of a given application.

Among other possible alternatives of sequencing the modeling activities, the "irregular" scenario shown in Figure 3 and described in its forward segment by the sequence <1, 2, 3A, 4A||4B, 5A||5B, 3B> (where the symbol || indicates parallel activities) deserves a brief examination. (Notice that in order to show that 3B comes after 5A and 5B a compromise regarding the notation has been made in Figure 3, where thick dashed lines are used as part of the forward segment; they are nevertheless different from the regular feedback connections, which continue to be represented by thin dashed lines). Two elements are worth noting in this scenario: first, the fact that the description of classes proceeds in parallel in UML and Z++ and, second, that step 3B

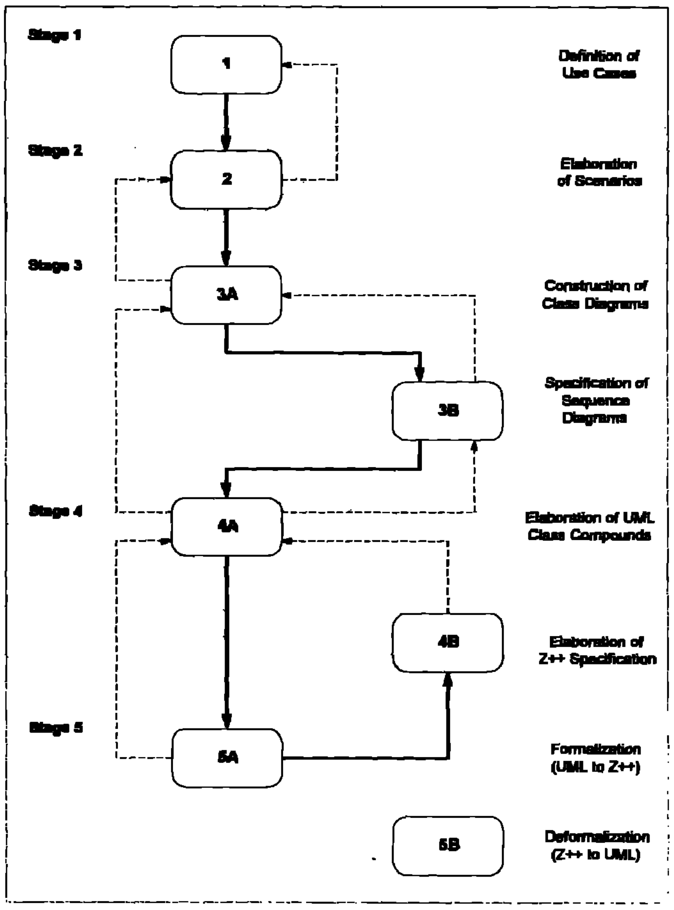
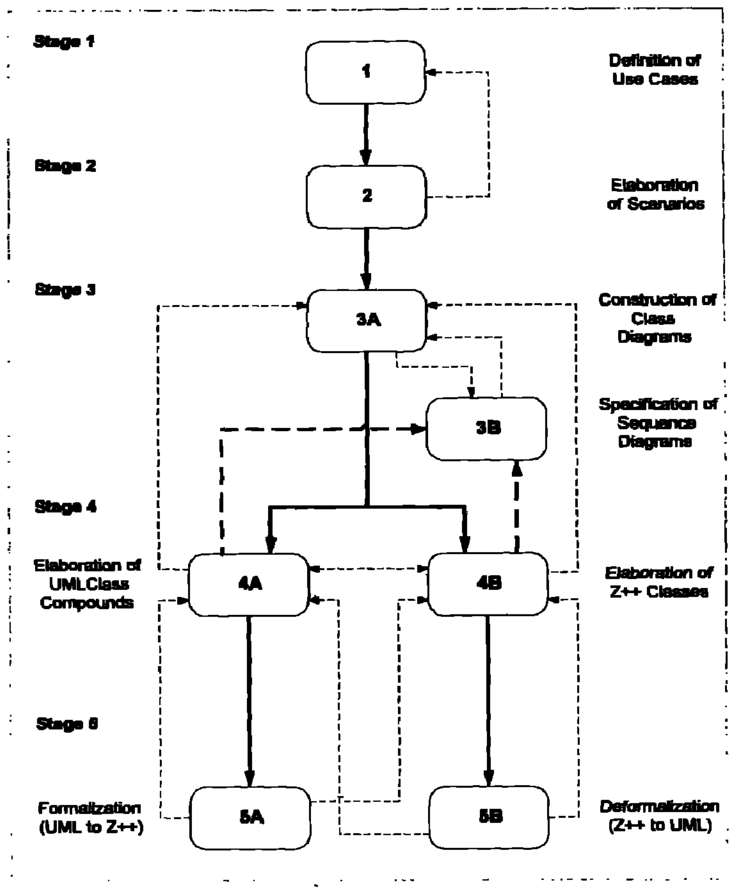

Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
1
2
3A
3B
4A
4B
5A
5B
Definition of Use Cases
Elaboration of Scenarios
Construction of Class Diagrams
Specification of Sequence Diagrams
Elaboration of UML Class Compounds
Elaboration of Z++ Specification
Formalization (UML to Z++)
Deformalization (Z++ to UML)

Figure 2. Regular Sequence of Modeling Activities

comes last in the forward part of the scenario. The first element points to the fact that various teams of specifiers may have various backgrounds and while some would favor the use of UML, some may prefer employing Z++ as their main specification notation. Thus, it is possible to proceed first with the specification of classes in Z++, followed by descriptions in UML and, in fact, it is theoretically possible to have all classes specified in Z++ and not at all in UML. The second element illustrates the idea that sequence diagrams can be used as means for fine-tuning the specification, and thus can be the last set of artifacts developed in the modeling process. Of course, additional refinements for improving the accuracy of the model follow in any case.

Another example or irregular modeling scenario, which stresses rapid development is, in its forward segment, <2, 3A, 4A, 5A, 4B>, meaning that the definition of use cases (step 1) and the specification of sequence diagrams (step 3B) are omitted. In short, this modeling alternative takes a "fast-track route" and, after the elaboration of scenarios, class diagrams are developed, UML classes are detailed, the formalization process takes place, and the detailed specification of Z++ classes is completed. In fact, this modeling scenario represents a shorter version of the regular flow of modeling activities suggested previously. While we recommend the regular alternative described in conjunction with Figure 2, the above shorter chain of activities may be used in certain application contexts.



Stage 1
Definition of Use Cases
1
Stage 2
Elaboration of Scenarios
2
Stage 3
Construction of Class Diagrams
3A
Specification of Sequence Diagrams
3B
Stage 4
Elaboration of UMLClass Compounds
4A
4B
Elaboration of Z++ Classes
Stage 5
Formalization (UML to Z++)
5A
5B
Deformalization (Z++ to UML)

Figure 3. Example of "Irregular" Flow of Modeling Activities

3. THE SUPPORTING ENVIRONMENT

The Harmony environment, shown in Figure 4, is designed to support the object-oriented modeling process presented in section 2 (the name of the environment reflects our goal of smoothly integrating, or "harmonizing," distinct specification languages). The environment allows the development of *specification projects,* which are sets of specifications represented in diagrammatical (UML) and mathematical (Z++) forms. These projects consist of the artifacts described previously in the paper. One of the distinguishing characteristics of Harmony is its "monolithicity," meaning that the environment is able to sustain the complete UML and Z++ combined specification of the system. Through its capabilities for interfacing with external tools such as programs for formal proof and formal refinement the environment is also capable in principle to support the further development of the system's model. As shown in Figure 4, it also monolithically integrates the two "spaces," the graphical space of UML and the textual, formal space of Z++. On a detailed level, support for class compounds is available in the form of a splitter bar that divides the screen area of each class compound in two sections, one for the CLS and the other CLSTD components of the compound.

Figure 4 presents Harmony with a project loaded and several of the project's components visible in the UML and Z++ spaces. More precisely, the Elevator class compound is shown (partially) in the UML Space and the corresponding Elevator Z++ class is displayed on the top tabbed-pane of the Z++ Space. As seen in the figure, the environment consists of a main window, divided into three principal panes –the Project Pane, the UML Space, and the Z++ Space– and includes several other GUI elements such as a message console and a status bar. Environment specific toolboxes, such as the Z++ Symbol Box shown in Figure 4 on the left-hand side of the Z++ Space, are also included in Harmony.

Regarding the three main panes of the environment it is useful to note that the project's organization is displayed in the Project Pane, work on the semi-formal model is performed in the UML Space, and the development of the formal specifications takes place in the Z++ Space. All these three panes can be individually turned on or off, thus allowing the development of specifications either in only one of the spaces or in "parallel" in both the UML and Z++ spaces.

Since an intense work on UML class compounds and on their corresponding Z++ classes is expected, a synchronization mechanism of on-screen presentation of the corresponding COMP and ZPPC constructs is provided. This mechanism defines a mode of operation that can be viewed as a manifestation –in our terminology– of *the tandem principle,* meaning that two entities (in Harmony's case, a UML class compound and its corresponding Z++ class) are collaborating to accomplish a common goal (this is the case in Figure 4, where the Elevator COMP and the Elevator ZPPC are the top tabbed-panes of the UML and, respectively, Z++ spaces). On practical terms, this mechanism allows the simultaneous development or the simple inspection of a class in both its UML and Z++ representations.

Automated translations between UML and Z++ models, applied to subsets of the models and involving inherent simplifications, can be invoked through the menu bar or from the main toolbar (cube-shaped icons "Propagate to UML" and, respectively, "Propagate to Z++ are placed on the main tool bar for this purpose).

4. CONCLUSIONS

Although there are numerous approaches that integrate in various levels graphical, semi-formal notations with elements of formal techniques only few of them employ an object-oriented, Z-based formalism in conjunction with a graphical notation. Moreover, among the latter, only very few contain provisions for dealing with real-time systems. To the best of our knowledge the closest approaches to ours are France et al.'s formalization in Z of Octopus models [13], Jia's AML-based solution for exploiting thethe strengths of both object-oriented and formal notations [17], Kim and Carrington's combined use of UML and Object-Z [20], and Noe and Hartrum's inclusion of formal specifications in Rational Rose [24]. The only tool developed commercially to support an object-oriented modeling approach and combine the advantages of graphical, semi-formal notations with those of formal notations is RoZeLink [28]. Nevertheless, our alternative is distinct from all these approaches in at least one significant way. Specifically, as opposed to [13], [17], and [24] we use an object-oriented version of Z and in contrast to [13], [20], and [28] we include in our approach provisions for dealing with time-

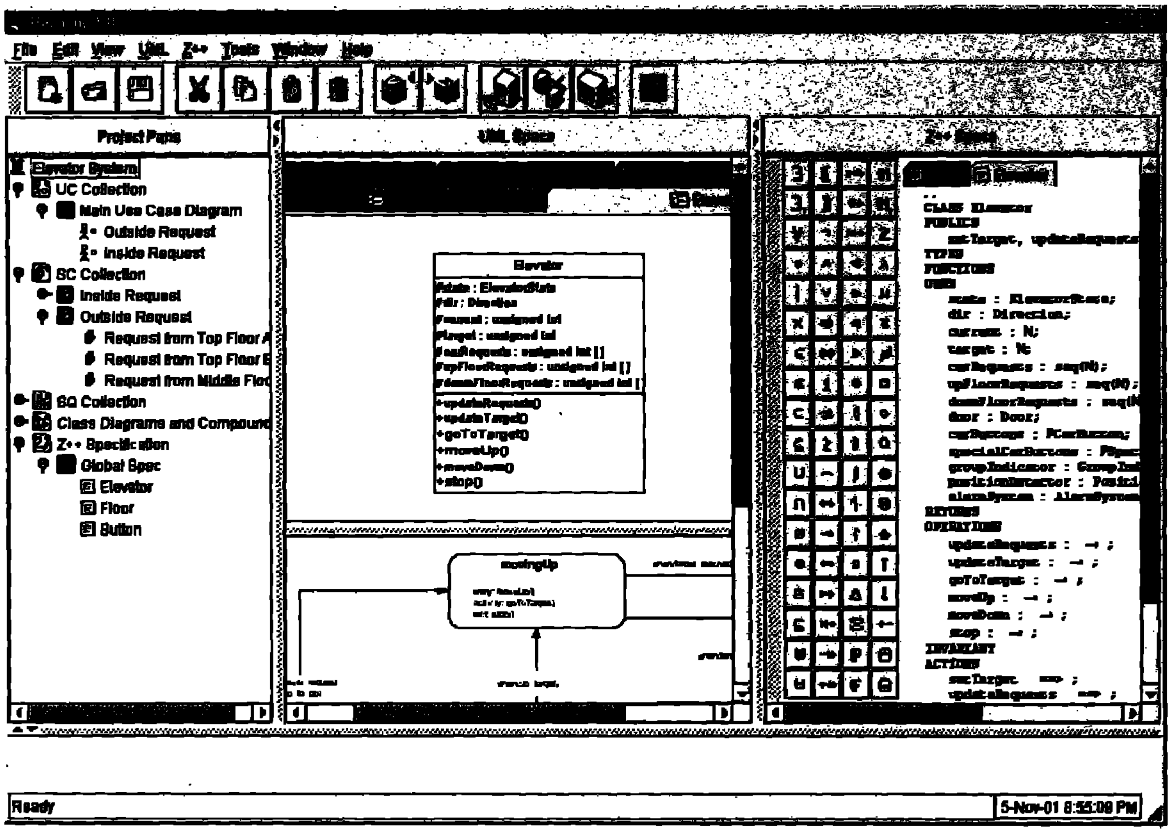


Figure 4. The Harmony Environment

related properties of the systems (we rely on Z++'s capability of dealing with real-time systems via Jahanian and Mok's Real Time Logic [16] and Lano's extensions to RTL [21]). Also, the Harmony environment distinguishes itself through its monolithic construction, as described in Section 3 of this paper, and is characterized by a tight kind of integration of notations that can be found only in [28].

The design of Harmony has been completed, including its GUI appearance, and detailed formalization algorithms have been designed, implemented, and tested. The input, the output and pseudo-code description of these algorithms, together with a Java implementation of the algorithm for formalizing state diagrams (AFSD) are presented in [10]. The proposed modeling approach has been exercised on several medium-size case studies, including the Elevator case study from which the screenshot presented in Figure 4 has been extracted. Details on the application of our approach to the Elevator system are also provided in [10].

Further work is planned in several directions. In particular, Harmony needs the refinement of its design and the completion of its implementation, which will allow us to validate the useful features of the environment and improve its functionality. Other plans include the inclusion in Harmony of a syntax checker for Z++, the integration of links to tools for formal refinement and verification, the enhancement of the existing UML to Z++ translation algorithms, and the use of the proposed approach and its accompanying environment for the specification of more complex systems.

REFERENCES

[1] Alexander, P. Best of Both Worlds: Combining Formal and Semi-Formal Methods in Software Engineering. IEEE Potentials, 14 (5) (Dec. 1995/Jan. 1996), 29-32.

[2] Aujla, S., Bryant, T., and Semmens, L. Applying Formal Methods Within Structured Development. IEEE Journal On Selected Areas in Communications, 12 (2) (Feb. 1994), 258-264.

[3] Barrios, S.D., and Lopez J.C. Heterogeneous Systems Design: a UML-based Approach. In Proc. of the 25th EUROMICRO Conf. (Milan, Italy, Sep. 1999), vol. 1, 386-389.

[4] Bell, A.E., and Schmidt, R.W. UMLoquent Expression of AWACS Software Design, Comm. of the ACM, 42 (10) (Oct. 1999), 55-61.

[5] Berry, D.M., and Weber, M. A Pragmatic, Rigorous Integration of Structural and Behavioral Modeling Notations. In Proc. of ICFEM'97 (Hiroshima, Japan, Nov. 1997), 38-48.

[6 Bjőrklander, M. Graphical Programming Using UML and SDL. IEEE Computer, 33 (12) (Dec. 2000), 30-35.

[7] Booch, G. , Rumbaugh, J., and Jacobson, I. The Unified Modeling Language User Guide, Addison–Wesley, 1999.

[8] Bruel, J.M., Cheng, B., Easterbrook, S., France, R., and Rumpe, B. Integrating Formal and Informal Specification Techniques. Why? How? In Proc. of WIFT'98 (Boca Raton FL, Oct. 1998), 50-57.

[9] Chen, Z., Cau, A., Zedan, H., Liu, X., and Yang, H. A Refinement Calculus for the Development of Real-Time Systems. In Proc. of APSEC'98 (Taipei, Taiwan, Dec. 1998), 61-68.
[10] Dascalu, S.M. Combining Semi-Formal and Formal Notations in Software Specification: An Approach to Modelling Time-Constrained Systems. PhD thesis, Dalhousie University, Halifax, NS, Canada, 2001.
[11] Day, N. A Framework for Multi-Notation Requirements Specification and Analysis. In Proc. of ICRE'00 (Schaumburg IL, June 2000), 39-48.
[12] Fernandes, J.M., Machado, R.J., and Santos, H.D. Modeling Industrial Embedded Systems with UML. In Proc. of CODES 2000 (San Diego CA, May 2000), 18-22.
[13] France, R.B., Bruel, J.-M., and Raghavan, G. Taming the Octopus: Using Formal Models to Integrate the Octopus Object-Oriented Analysis Models. In Proc. of HASE'97 (Bethesda MD, Aug. 1997), 8-13.
[14] Gerhart, S., Craigen, D., and Ralston, T. Experience with Formal Methods in Critical Systems. IEEE Software, 11 (1) (Jan. 1994), 21-28.
[15] Howerton, W.G., and Hinchey, M.G. Using the Right Tool for the Job. In Proc. of ICECCS 2000 (Tokyo, Japan, Sep. 2000), 105-115.
[16] Jahanian, F., and Mok, A. Modechart: A Specification Language for Real-Time Systems. IEEE Trans. on Software Engineering, 12 (9) (Sep. 1986), 933-944.
[17] Jia, X. A Pragmatic Approach to Formalizing Object-Oriented Modeling and Development. In Proc. of COMPSAC'97 (Washington DC, Aug. 1997), 240-245.
[18] Jigorea, R., Manolache, S., Eles, P., and Peng, Z. Modelling of Real-Time Embedded Systems in an Object-Oriented Design Environment with UML. In Proc. of ISORC 2000 (Newport Beach CA, March 2000), 210-213.
[19] Kim, S.-K., and Carrington, D. Visualization of Formal Specifications. In Proc. of APSEC ' 99 (Takamatsu, Japan, Dec. 1999), 102-109.
[20] Kim, S.-K., and Carrington, D. A Formal Mapping between UML Models and Object-Z Specifications. In Bowen, J.P., Dunne, S., Galloway, A., and King, S. (eds.), Intl. Conf. of B and Z Users ZB2000, LNCS 1878 (Feb. 2000), Springer-Verlag, 2-21.
[21] Lano, K. Formal Object-Oriented Development. Springer-Verlag, 1995.
[22] Lee, J., Pan, J.I., and Huang, W.T. Integrating Object-Oriented Requirements Specifications with Formal Notations. In Proc. of ICTAI'95 (Washington DC, Nov. 1995), 34-41.
[23] Moreira, A.M.D., and Clark, R.G.Adding Rigour to Object-Oriented Analysis. Software Engineering Journal, 11(5) (Sep. 1996), 270-280.
[24] Noe, P.A., and Hartrum, T.C. Extending the Notation of Rational Rose 98 for Use with Formal Methods. In Proc. of NAECON 2000 (Dayton OH, Oct. 2000), pp. 43-50.
[25] Paige, R. F. Heterogeneous Notations for Pure Formal Method Integration. Formal Aspects of Computing, 10 (3) (June 1998), 233-242.
[26] Paige, R.F., "When Are Methods Complementary?," Information and Software Technology, 41 (3) (Feb. 1999), 157-162.
[27] Polack, F. Integrating Formal Notations and Systems Analysis: Using Entity Relationship Diagrams. Software Engineering Journal, 7 (5) (Sep. 1992), 363-371.
[28] RoZeLink (product description). Headway Software Inc.'s web site, accessed May 1999 at http://indigo.ie/~chrisc
[29] Scogings, C., and Phillips, C. A Method for the Early Stages of Interactive System Design Using UML and Lean Cuisine+. In Proc. of AUIC 2001(Queensland, Australia, Jan./Feb. 2001), 69-76.
[30] Si Alhir, S. UML In A Nutshell: A Desktop Quick Reference. O'Reilly & Associates, 1998.
[31] Sowmya, A., and Ramesh, S. Extending Statecharts with Temporal Logic. IEEE Trans. on Software Engineering, 24 (3) (March 1998), 216-231.
[32] OMG Unified Modeling Language Specification, v.1.3. (March 2000), ftp://ftp.omg.org/pub/docs/formal/00-03-01.pdf
[33] Xu, R., Masaru, Z., and Zhang, H-Q. Object-Oriented AGVS Modeling with UML. In Proc. of the 39th SICE Annual Conf. (Iizuka, Japan, July 2000), 261-264.
[34] Zave, P., and Jackson, M. Where Do Operations Come From?A Multiparadigm Specification Technique. IEEE Trans. on Software Engineering, 22(7) (July 1996),508-528.