Harmony: An Environment for the Combined Use of UML and Z++ in Software Specification

Sergiu Dascalu and Peter Hitchcock
Faculty of Computer Science, Dalhousie University
Halifax, Nova Scotia, Canada
{dascalus, hitchcp}@borg.cs.dal.ca

Abstract: Graphical notations have traditionally played an important role in the process of software development, particularly during the analysis and design phases. However, in applications in which the reliability of the software is a primary concern a graphical notation such as UML needs to be complemented by a formal language in order to provide enhanced support for pragmatic and rigorous development of specifications. Although many authors have envisaged the advantages of combining informality (or semi-formality) with formality in software construction, there are few reports that address the issue in the context of object-orientation and project its solution over the canvas of time-constrained systems. In this paper we present an integrated environment, entitled Harmony, which supports the combined utilization of two notations, the visual UML and the formal Z++, and thus assists the rigorous object-oriented development of systems on which temporal constraints are placed.

Keywords: software engineering, object-oriented modeling, formal specification, integration of notations, UML, Z++.

1 Introduction

The proposal for a specification environment described in this paper is part of a larger research work that addresses the integration of semi-formal graphical representations with formal, textual notations within a modeling approach aimed at the construction of time-constrained systems. We believe that the two types of notations, graphical (in this paper we refer specifically to semi-formal graphical notations) and, respectively, formal, can efficiently complement each other and provide the basis for a software specification approach that can be both rigorous and practical. The former notations, relying on graphical symbols and diagrams, bring the "power of pictures," which manifests itself through better representation of abstractions and higher expressiveness [11]. The latter notations, formal, precise, based on sound mathematics, increase the product's reliability, the user's assurance and intellectual control [10], and make possible automated synthesis and verification [2]. Although many authors have envisaged the advantages of combining informality (or semi-formality) with formality in software construction (e.g., [3], [12], [20]), there are few reports that address the problem within the context of object-orientation *and* project its solution over the canvas of time-constrained systems. Providing a sound and pragmatic solution for object-oriented modeling of time-constrained systems via synergetic use of formality and visualization in software specification is the goal of our approach.

In this paper, we briefly highlight the main characteristics of our software specification approach and then focus on the integrated environment that supports it. First of all, as a matter of terminology, we use the term *time-constrained systems* (TCS) as an alternative to *real-time systems* in order to emphasize the temporal restrictions imposed on such systems and to shift the focus from specialized products confined to rather restricted domains (e.g., military, nuclear energy generation, plant automation, etc.) to more accessible products such as transaction processing systems, traffic lights controllers, and cellular phones. Also, we rely on Fraser et al. to distinguish between formal, semi-formal, and informal specification techniques. Specifically, *informal techniques* "do not have complete sets of rules to constrain the models that can be created," *semi-formal techniques* have well-defined syntax and their "typical instances are diagrammatic techniques with precise rules that specify conditions under which constructs are allowed and textual and graphical descriptions with limited checking facilities," while *formal techniques* have

precise syntax and semantics and "there is an underlying model against which a description expressed in a mathematical notation can be verified" [9].

The main characteristics of our approach are the following: the combination of formal and semi-formal notations for modeling purposes, the integration into an object-oriented approach of capturing capabilities that target properties of TCS, the adapted utilization, based on subsets of the notations, of the formal specification language Z++ [18] and of the object-oriented modeling standard UML [4], and the proposal of a methodological frame for the reliable development of TCS. Translation algorithms between UML to Z++ are also integral parts of our approach, which is fully supported by the specification environment that constitutes the main subject of this paper.

Because we attempt to combine the benefits of "both worlds" ("formal" and, respectively, "informal") [2] and to "harmonize" the use of the UML and Z++ notations we have assigned the name Harmony to the environment that supports our approach. In Harmony, there is a link between the formal and the graphical representations of the model, ensured by the bi-directional translation mechanisms that we have designed. This allows changes in the graphical representation to be reflected into the formal model, as well as modifications of the formal model to be fed back into the diagrammatic description of the system.

The remaining of this paper is organized as follows: section 2 proposes a classification of semi-formal/formal integrated specification approaches and briefly reviews work related to ours, section 3 succinctly presents the two languages, UML and Z++, that provide the notational foundation for our approach, section 4 describes the overall organization of Harmony and provides details about its operational capabilities and its GUI components, and section 5 analyzes the current status of our work and points to a series of research and development topics that we intend to pursue further.

2 Classification of Integrated Approaches and Overview of Related Work

Integrating formal with informal notations in software development is not a new idea, some forms of combinations being present in a fair number of approaches. After all, formal languages like Z [23] include provisions for textual, natural language annotations, intended to facilitate the interpretation of complex mathematical expressions and to relate abstract descriptions with real-world entities. However, as pointed out by several authors, one of the main reasons which, in addition to lack of tools support, has prevented the wider application of formal methods is that not sufficient attention has been paid to the integration of formal techniques with traditional, semi-formal methods [5, 10]. In the following, we propose a classification of semi-formal/formal combination strategies and then briefly survey approaches related to ours.

In general, the relationship between the formal and the semi-formal (or informal) components of a specification can be one of the following:

- If the graphical (semi-formal or informal) part is built initially and then a translation process is applied to obtain its formal counterpart, we can speak about *derivation* of the formal model from the informal model or simply of *formalization* (e.g., [17], where the translation is from UML to B). Conversely, when the visualization of the formal part is attempted (e.g., [15] where graphical representations of Z constructs are proposed), the process can be denoted *deformalization*;
- If in addition to diagrammatic representations some related formal specifications are produced independently (e.g., [14], where Z specifications supplement UML models), the approach can be characterized as *complementary formalization*. Typically, this approach also involves derivation from informal to formal, a subset of the diagrammatic descriptions of the system being translated into formal specifications (this is the case as well in [14]);
- If changes in any of the specification's parts are continuously propagated in the other, we can speak of a *tight integration of notations* (e.g., [22], where UML models are connected to corresponding ZEST descriptions). In our approach we attempt to achieve a tight integration of notations, as it serves best the iterative development of specifications.

While there are numerous approaches that integrate in various levels graphical, semi-formal notations with elements of formal techniques, much fewer employ a Z-based formalism in conjunction with a

graphical notation within the larger frame of the object-oriented paradigm. Furthermore, among the latter, only a limited number attempt at the same time to deal with TCS. To the best of our knowledge the approaches that are the closest to the direction of work that we have pursued are France et al.'s translation of Octopus models into Z specifications [8], Jia's pragmatic approach for combining the strengths of object-oriented and formal notations, specifically UML and Z [14], Noe and Hartrum's extension of Rational Rose for the inclusion of formal specifications [19] and Kim and Carrington's combination of UML and Object-Z [16]. The only tool developed commercially to support an object-oriented modeling approach that combines the advantages of graphical, semi-formal notations with those of formal notations is RoZeLink, produced by Headway Software Inc. [22] (RoZeLink provides a bridge between the UML notation supported by Rational Software Corporation's Rose environment [21] and the ZEST object-oriented formal specification language supported by Logica's Formaliser [7]). Our alternative, however, is distinct from all these approaches in at least one major aspect. For instance, as opposed to [8], [14], and [19] we use an object-oriented version of Z and in contrast to [14], [19], and [22] we include in our approach provisions for dealing with time-related properties of the systems. Also, our Harmony environment is distinct from all of these approaches through its monolithic construction, meaning that a single CASE tool is used for developing both semi-formal and formal models. In addition, a tight kind of integration of notations similar to ours can be found only in [22].

3 A Brief Look at the Notations

The Unified Modeling Language (UML) has been incorporated in our approach because 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. Nevertheless, in our opinion the incorporation in UML of concepts from numerous sources, although justified by the goal of ending "the methods war," requires some fine-tuning in practical terms (the generality of the notation and its higher level of diffuseness [11] typically necessitate decisions on what to be used, what customizations to be made, and what to be left out from UML in a particular methodology and/or application). For this reason, we use in Harmony only a subset of the UML notation, and try to minimize the "notational overhead" that the full UML may present. UML does include support for specifying real-time systems [4], but for precise specifications supplementary modeling power is necessary, and new solutions such as Real-Time UML [1] have already been proposed.

Such additional modeling power can also be provided by a formal language such as Z++, an object-oriented extension of Z that distinguishes itself through its support for temporal logic specifications. In particular, within the general form of Z++ class specification (presented in Fig. 1, based on [18]) the

```
ZPP_Class ::= CLASS Identifier [TypeParameters]
[EXTENDS Ancestors]
[TYPES TypeDefs]
[FUNCTIONS AxiomaticDefs]
[OWNS Locals]
[RETURNS OpTypes]
[OPERATIONS OpTypes]
[INVARIANT Predicate]
[ACTIONS Actions]
[HISTORY History]
END CLASS
```

Fig. 1 General Form of Z++ Class Specification

HISTORY clause is used to specify the admissible sequences of execution, in the form of a predicate that includes temporal-logic operators such as □ (*henceforth*), ○ (*next*), ✧ (*eventually*), *before* and *until*. The predicates can also be written in Real-Time Logic [13], a solution that we employ in our approach. In essence, it is here, in the HISTORY clause, where Z++'s capability of dealing with temporal aspects of the systems resides.

4 A Prototype for Harmony

The Harmony environment, shown in Fig. 2, is intended to fully support our proposed object-oriented modeling process. This process consists of the following steps: definition of use cases, elaboration of scenarios, construction of class diagrams, specification of sequence diagrams, elaboration of class compounds (described later in this section), elaboration of Z++ specifications, formalization (translation from UML to Z++) and deformalization (reverse translation, Z++ to UML). The environment operates on *specification projects*, which are sets of specifications represented in diagrammatical (UML) and mathematical (Z++) forms. The combined result of the specification activities supported by Harmony constitutes the integrated model of the system. This model consists of a collection of use cases (UC), a collection of scenarios (SC), a collection of sequence diagrams (SQ), a set of class diagrams (CD) -- each with a number of class compounds (COMP) --, and a Z++ specification (ZSPEC) made up primarily of a set of Z++ classes (ZPPC). Since a total formalization of the system is typically not required, the environment can be used for a partial formalization within a complete specification of the system. In practical terms, this means that it is not necessary that a one-to-one correspondence between the UML and the Z++ components is achieved -- some parts of the system can be described both in UML and Z++, while other only in UML or only in Z++.

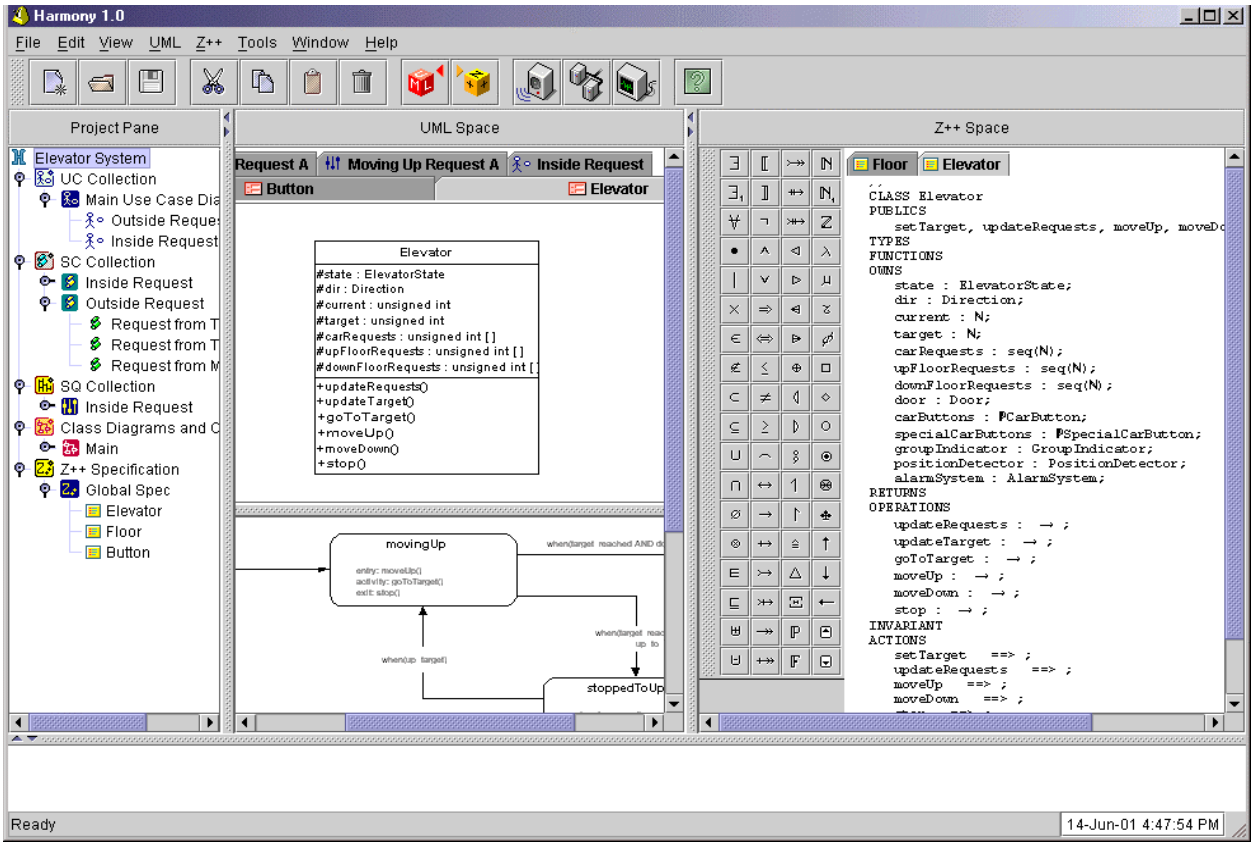

Fig. 2 The Harmony Integrated Specification Environment

One of the distinguishing characteristics of Harmony is that it is monolithic in the sense defined in section 2. By itself, it is sufficient to sustain a complete specification of the system and through its provisions for interfacing with external tools it is also capable to support further development (in particular, we envisage interfacing it with external tools for formal proof and formal refinement). As shown in Fig. 2, it also monolithically integrates two "worlds" (or "spaces"), the visual world of UML and the textual, intensively symbolical world of Z++.

On a more detailed level, support for the modeling element that we denote *class compound* is available, the idea being that the key classes of TCS need be detailed not only in respect with their attributes and their operations, but also with the sequencing of their operations as captured in state diagrams. In essence, a class compound results by juxtaposing a UML class and the UML state diagram that corresponds to that class. From a GUI point of view, a splitter bar is introduced between the area in which the UML class is represented and the one in which the class' state diagram is shown.

Also, 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 is the case shown in Fig. 2, where the Elevator COMP and the Elevator ZPPC are the top tabbed-panes of the UML, and, respectively, Z++ spaces). This mechanism defines a mode of operation that can be viewed as a manifestation, in our terminology, of *the tandem principle*, a principle which allows the "simultaneous" development or the simple inspection of a class in both its UML and Z++ forms.

Fig. 2 presents Harmony in a typical situation, in which a project is loaded and work is undergoing in the UML and Z++ spaces on several modeling elements (specifically, a use case, a scenario, a sequence diagrams, two UML class compounds, and two Z++ classes). As seen in the figure, the environment consists of a main window (browser) which is divided into several panes (the Project Pane, the UML Space, and the Z++ Space) and which contains other GUI elements such as a menu bar and toolbars. Also included in the Harmony browser are a message console and a status bar, as well as several environment specific toolboxes (shown in Fig. 2 is only the Z++ Symbol Box of the Z++ space).

The entire organization of the project can be viewed in the Project Pane, which shows the collections of artefacts that make up the integrated model of the system, work on the semi-formal model is performed in the UML Space, and formal specifications are written in the Z++ Space. It is important to note that in this organization the three panes can all coexist on the screen at any given moment, but they can also be individually turned off. Thus, the specification work can either proceed in parallel in the semi-formal and the formal spaces or can be done only in one of the modeling spaces.

Due to space limitations numerous details of Harmony, including both GUI aspects and operational capabilities are omitted here. We note however that the entire GUI-centred design of Harmony has been completed and that formalization algorithms, dealing with both structural and behavior components of the system (specifically, class diagrams and state diagrams) have been designed, implemented, and exercised on several case studies. Details on the formalization algorithms, including rules for the well-formedness of class diagrams, principles for UML to Z++ translations, and pseudo-code descriptions of the algorithms can be found in [6]. The same reference also presents a set of principles for deformalization, gives details on the proposed specification approach, and illustrates this approach through an Elevator System case study (an excerpt from this case study is shown in Fig. 2).

5 Conclusions

Our proposal for an integrated specification environment is part of an ongoing project in which further work is needed in several directions. In particular, Harmony needs further refinement, and the first step for improving it is to finalize its implementation, which will allow us to concretely experiment with our ideas, validate useful features and discard the unnecessary. Other near future plans include the addition to Harmony of a syntax checker for Z++, the integration of links to tools for formal analysis and refinement, and the use of the environment for the specification of larger systems. We believe that the

premises of our work are valid, and that there is great potential in the combined approach "visualization-and-formalization" for modeling time-constrained systems.

References

[1] Alagar, V.S., and Muthiayen. D., "Towards a Mechanical Verification of Real-Time Reactive Systems Modeled in UML," *Proc. of the 7th Intl. Conf. on Real-Time Computing Systems and Applications*, Dec. 2000, pp. 245 –254.

[2] Alexander, P., "Best of Both Worlds: Combining Formal and Semi-formal Methods in Software Engineering," *IEEE Potentials*, 11(5), Dec. 1995/Jan. 1996, pp. 29-32.

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

[4] Booch, G., Rumbaugh, J., and Jacobson I., *The Unified Modeling Language: User Guide*, Addison-Wesley, 1998.

[5] Clarke, E., and Wing, J.M., "Formal Methods: State of the Art and Future Directions," *ACM Comp. Surveys*, 28(4), Dec. 1996, pp. 626-643.

[6] 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, Sep. 2001.

[7] Formaliser, accessed June 2001 at Logica's web site, http://public.logica.com/~formaliser

[8] France, R.B., Bruel, J.-M., and Raghavan, G., "Taming the Octopus: Using Formal Models to Integrate the Octopus Object-Oriented Analysis Models," *Proc. of the High-Assurance Engineering Workshop*, Aug. 1997, pp. 8-13.

[9] Fraser, M.D., Kumar, K., and Vaishnavi, V.K., "Strategies for Incorporating Formal Specifications in Software Development," *Communications of the ACM*, 37 (10), Oct. 1994, pp. 74-85.

[10] Gerhart, S., Craigen, D., and Ralston, T., "Experience with Formal Methods in Critical Systems," *IEEE Software*, 11(1), Jan. 94, pp. 21-28.

[11] Green, T., and Petre, M., "Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework," *Journal of Visual Lang. and Computing*, 7(2), June 1996, pp. 131-174.

[12] Howerton, W.G., and Hinchey, M.G., "Using the Right Tool for the Job," *Proc. of the 6th IEEE Intl. Conf. on Engineering of Complex Computer Systems*, Sep. 2000, pp. 105 –115.

[13] Jahanian, F., and Mok, A., "Modechart: A Specification Language for Real-Time Systems," *IEEE Transactions on Software Engineering*, 12 (9), Sep. 1986, p. 933-944.

[14] Jia, X., "A Pragmatic Approach to Formalizing Object-Oriented Modeling and Development," *Proc. of the 21st Intl. Conf. on Computer Software and Applications*, Aug. 1997, pp. 240-245.

[15] Kim, S.-K., and Carrington, D., "Visualization of Formal Specifications," *Proc. of the 6th Asia Pacific Software Engineering Conf.*, Dec. 1999, pp. 102-109.

[16] Kim, S.-K., and Carrington, D., "An Integrated Framework with UML and Object-Z for Developing a Precise and Understandable Specification: The Light Control Case Study," *Proc. of the 7th Asia-Pacific Software Engineering Conf.*, Dec. 2000, pp. 240-248.

[17] Laleau, R., and Mammar, A., "An Overview of a Method and its Support Tool for Generating B Specifications from UML Notations," *Proc. of the 15th IEEE Intl. Conf. on Automated Software Engineering*, Sep. 2000, pp. 269-272.

[18] Lano, K., *Formal Object-Oriented Development*, Springer-Verlag, 1995.

[19] Noe, P.A., and Hartrum, T.C., "Extending the Notation of Rational Rose 98 for Use with Formal Methods," *Proc. of the IEEE National Aerospace and Electronics Conf.*, Oct. 2000, pp. 43-50.

[20] Polack, F., "Integrating Formal Notations and Systems Analysis: Using Entity Relationship Diagrams," *Software Engineering Journal*, 7(5), Sep. 1992, pp. 363-371.

[21] Rational Rose, accessed Aug. 2001 at Rational Software's web site, http://www.rational.com

[22] RoZeLink, accessed March 1999 at Headway Software Inc.'s web site, http://indigo.ie/~chrisc

[23] Spivey, J.M., *The Z Notation: A Reference Manual*, Prentice Hall International, 1992.