

Visualizing Graphical and Textual Formalisms

R. Castello, R. Mili, H. Madabushi
Department of Computer Science
The University of Texas at Dallas
Box 830688, Richardson, TX 75083-0688, USA
email: {castello, rmili, hawi}@utdallas.edu

1. Introduction

Several requirements specification styles for the specification of reactive systems have been proposed in the literature. Informal specifications are the most widely used in the software industry nowadays, mainly because they are “understandable” by domain experts and software engineers, and therefore can be included as part of a software contract. Unfortunately, because of the inherent ambiguity of natural language, these documents are difficult to interpret and maintain. Formal specifications on the other hand, make use of mathematical notations that offer precise syntax and semantics. Unfortunately, because of the complexity of the formal description these documents may not be acceptable to many stakeholders. Visual formalisms bridge the gap between informal and formal specifications by offering graphical notations with semantics. Unfortunately, hand made diagrams become easily unreadable when the requirements complexity increases. In addition, visual formalisms do not enjoy the powerful verification, code optimization, or model-checking techniques that are common to formal notations. The purpose of this work is to combine the advantages of using visual formalisms for the specification of reactive systems with that of using formal verification and program transformation tools developed for textual formalisms. We have developed a tool suite called *ViSta* [1, 3, 2] that automatically produces *statechart* layouts based on information extracted from an informal specification. In this paper, we discuss how *ViSta* is augmented with a tool that automatically translates *statecharts* [4] to *Z* specifications. The informal, *statechart* and *Z* specifications are inter-related. This ensures consistency between the different representations, and therefore facilitates the verification and validation effort.

2. System Overview

ViSta [1, 3, 2] is a tool suite designed to support the requirements specification phase of reactive systems. In this

section we give an overview of version 2.3 that includes the *template wizard*, the *statechart visualization tool* and the *database*. The *Z visualization tool* is discussed in section 3.

The tool’s graphical user interface offers five frames. The *text* frame is used to input a textual description of requirements. The *template wizard* frame is used to extract and store information into a set of templates. The *structured text* frame displays the structured textual description that is derived from the information stored in the database. The *statechart graphical editor* displays the *statechart* drawings. Finally, the *Z editor* displays the *Z* specification.

The data stored in the database is summarized in a structure that we call *decomposition tree*. This tree reflects the underlying structure of *statecharts* (i.e., *AND/OR* tree). The root of a decomposition tree corresponds to the system superstate; leaves correspond to atomic states. Each node in the tree can be decomposed through the *AND* or *OR* decomposition. A node in the decomposition tree includes the following information: its name; its width and height; the coordinates of its origin point; a pointer to its parent; the list of its children; its decomposition type (e.g., *AND*, *OR* or *leaf*); the list of incoming arcs; the list of outgoing arcs; a list of attributes; and finally its aliases. The information is entered in the database either through the *template wizard*, the *statechart graphical editor* or the *Z editor*. *ViSta* uses this information to automatically update the graphical, formal, textual and template representations.

The Template Wizard

The *template wizard* guides the user through the steps necessary for the construction of *statechart* drawings. It is used when an informal textual description exists, and the user wants *ViSta* to generate the drawings. The *template wizard* forces the user to provide specific behavioral information based on the original requirements document; it stores this information in the database and generates a fixed-format

structured document. The user selects information from the textual document and dynamically introduces it into a set of templates. Selected parts turn into a distinct color to inform the user that they were successfully accepted by the templates.

A template is a form-based component that has a predefined structure. It consists of structured propositions with text fields to be filled in with information specific to the requirements.

The Statechart Graphical Editor

The user may decide to directly draw statecharts using the graphical editor. In the graphical editor, statechart elements are described using *item-forms* (e.g., *state-form*, *transition-form*). When the user selects a graphical icon (e.g., state, transition, or default state) and clicks in the drawing area, an item-form pops up. The information captured in the item-forms is immediately stored in the database, and is used by ViSta to generate (or update) the graphical, formal, template and structured descriptions.

The Statechart Visualization Tool

This tool automatically produces statechart drawings. It offers a framework that improves the readability of the drawings by focusing on some of the most important aesthetic criteria for graph drawing [5], namely, *edge-crossing reduction*, *edge-bend reduction*, *labeling*, and *minimization of the drawing area*. It is based on several techniques that include hierarchical drawing, labeling, and floorplanning. Hence, the resulting drawings enjoy several properties: they have a low number of edge crossings; they emphasize the natural hierarchical decomposition of states into substates; and they cover an optimal drawing area.

3. The Z Visualization Tool

The Z visualization tool consists of two components: the *Z translation module* and the *Z editor*. If a statechart drawing exists, the Z Visualization tool is used to a) visualize the corresponding Z specification as a Latex file. This file can be passed on to the *ZEves* tool for type checking and theorem proving; b) to display/modify the specification on the Z editor. In case a statechart drawing does not exist, the user may use the Z editor to enter a specification from scratch. The information is automatically stored in the central database, and is used by *ViSta* to generate the statechart and template representations.

The Z translation module

This module translates statecharts to Z specifications. This is achieved by reading the information stored in the database and applying a set of translation rules. Traditionally, Z specifications describe systems as abstract data types. Hence operations are specified using state before and state after. Our algorithm takes a decomposition tree as an input, and generates a Z specification. It includes the following steps:

1. Declare the set type *STATES*.
2. Declare *STATE_TYPES*.
3. Declare *ACTIVE_STATES_TYPE*.
4. Generate the *DecompositionTree* schema.
5. Generate the *System_Active_States* schema.
6. Declare the set type *EVENTS_ACTIONS*.
7. Generate the *System_Events* schema.
8. Generate the *System_Actions* schema.
9. For each transition *t*, generate the operation schema *originState_to_destinationState*.
10. Generate the *Initial_System_Status* schema.

The static Z specification has to be augmented with a specification that describes the system's allowable concurrent behavior.

The Z Editor

The specification generated by the *Z translation module* is a *Latex* file that cannot be modified by the user. To circumvent this problem, the user is provided with the option of editing the specification using the *Z editor*. This editor offers 13 schema panels. Some of the schemas are static. Others can be modified. When these panels are modified, the tool automatically updates the central database.

References

- [1] R. Castelló. *From Informal Specification to Formalization: An Automated Visualization Approach*. PhD thesis, The University of Texas at Dallas., 2000.
- [2] R. Castelló, R. Mili, and I. G. Tollis. An algorithmic framework for visualizing statecharts. In *Lecture Notes in computer Science: Graph Drawing Conference 2000, September 20-23, 2000*. Springer-Verlag, September 2000.
- [3] R. Castelló, R. Mili, I. G. Tollis, and V. Benson. On the automatic visualization of statecharts: The vista tool. In *Proceedings of the Formal Methods Tools 2000 Conference, held at Ulm, Germany, July 10-13, 2000*. Springer-Verlag, July 2000.
- [4] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
- [5] H. Purchase. Which aesthetic has the greatest effect on human understanding. In G. D. Battista, editor, *Graph Drawing (Proceedings GD'97)*, pages 248–261. Springer-Verlag, 1997. Lecture Notes in Computer Science 1353.