Presented at the 1995 Monterey Workshop on Formal Methods and Architecture

Formal Methods in Describing Architectures

Dr. Paul C. Clements
Software Engineering Institute[1]
Carnegie Mellon University
Pittsburgh, PA 15213

Introduction

Formal methods are gaining prominence in software engineering as a way to insure that a specification is consistent with its intended meaning, and that two formally-rendered artifacts (e.g., a specification and an implementation) are consistent with each other in some precise way. Formal methods in the arena of software architecture tend to manifest themselves in representation technology, principally in *architecture description languages* (ADLs). Rapide, UniCon, Wright, ACME, ArTek, RESOLVE, Gestalt, and other ADLs are populating the software architecture literature, each offering a formal way to represent the architecture of a software system.

But to what end? Formal methods are useful to help a human organize thought patterns into a more disciplined form, thus heading off conceptual errors. However, formal methods are most valuable when they precipitate automated checking of an artifact, or automated translation of an artifact from one form to a more useful form. Where do ADLs stand on these capabilities?

Modechart: A language on the edge of ADLs

This paper will present a language that stands at the edge of the world of ADLs. Modechart is a specification language for hard-real-time embedded computer systems developed at the University of Texas at Austin. For a complete overview, see [5]. Modechart was not invented to be an ADL; however, by imposing restrictions on its usage, it more than adequately represents software structures and the interactions among them. Why bother? Because Modechart features a sophisticated analysis environment that provides the user with a wealth of information about the system being specified, and the technology is based squarely on classical formal methods such as model-checking verification. Given a specification of the system, Modechart allows powerful analysis about the system itself (as opposed to checking the specification itself). We present the Modechart paradigm as an example of a fruitful trend for ADL research.

Introduction to Modechart

The Modechart user provides the following information in a specification:

- Modes: The user provides a set of modes and the contained in relation for those modes. A mode is either primitive, meaning it contains no other modes, serial, meaning that its immediate children may only be active one at a time in any nonzero time interval, or parallel, meaning that all of its immediate children are active when it is. The contained-in relation is transitive, antisymmetric, and antireflexive; it is modeled graphically by drawing boxes anywhere as long as no edge of one box intersects an edge of another.

1. This work is sponsored by the U. S. Department of Defense.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1995		**00-00-1995 to 00-00-1995**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Formal Methods in Describing Architectures	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,Software Engineering Institute,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**7**	

A mode represents a program state; its children represent logical subsets of the state. A mode is active if the state it represents is true. For instance, a mode might represent all states in which an aircraft's landing gear is down; it might have two children that would each be active according to whether or not the aircraft was on the ground.

However, the mode structure can clearly be used to represent software components. In this case, a mode being active corresponds to the software component being active and making progress. Parallel modes and serial modes have the obvious meaning with respect to software components.

- Actions: The user defines a set of actions. An action definition consists of a host mode assignment, a program that computes a value, a state variable (defined elsewhere) to which the computed value is assigned whenever the action completes, and a designation of the action as periodic or sporadic. A sporadic action is executed exactly once when its host mode is entered. A periodic action is executed when its host mode is entered, and then again each time the period interval passes. The user supplies the relevant timing information.

  Actions are the way to specify behavior, both in terms of what behavior is precipitated (by stating the predicate that the action changes) and when the behavior is precipitated (by associated it with a mode whose activation triggers the action). In software architecture terms, actions define the behavior of the modes, which for our purposes are now models of software components.

- Transitions: The user provides a set of mode transition expressions; each mode transition expression is associated with an ordered pair of modes. The mode transition defines a condition that causes the first mode of the pair to be exited and the second mode of the pair to be entered. The pair may consist of the same mode twice. The mode pairs are subject to syntactic constraints of Modechart; e.g., a transition between two sibling modes in parallel with each other is forbidden. Mode transitions are defined to take zero time in Modechart. A transition takes place either as a result of timing (the exited mode has been active for a specified length of time) or as the result of events (other modes becoming active or inactive, predicates changing value, external events occurring, or actions starting or completing).

  In architectural terms, transitions are the representation of the interconnection mechanisms between components (modes). Invocation is the mechanism most easily imagined, but others are possible.

- External events: The set of external events relevant to the system. For each one, a separation parameter s that promises the minimum separation time between two consecutive occurrences of the event. External events are those over which the system being modelled has no control; their occurrence appears completely nondeterministic, subject to the minimum separation constraint.

- State variables: An action assigns a (computed) value to a state variable. The user provides the set of state variables and their correspondence to an action. The user may provide an initial value for each variable.

The state space of the system at any instant of time consists of, the set of active modes, the values of all of the state variables, the external events that occur at that time, the status of each of the actions (started, in progress, completed, aborted) at that time; and the value of the system clock.

Time is modelled by the nonnegative integers in Modechart. Timing constraints are applied to actions' execution times, mode transition times, and the minimum separation between consecutive occurrences of each external event.

The Modechart environment allows other information to be provided by the user, but these things are all that the semantic model requires. Modechart is primarily a graphical language, although a textual form is also available; Figure 1 illustrates a trivial example.

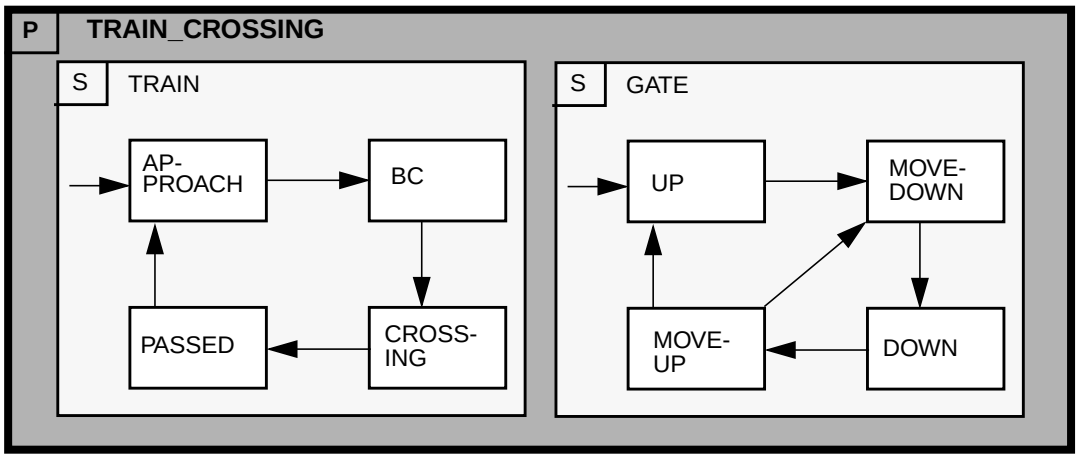

FIGURE 1. Graphical representation of a modechart. Non-atomic modes are tagged as being either serial (S) or parallel (P). Arrows may be labeled with the transition expressions that cause the transitions, but these are omitted here for readability.

The Modechart development and analysis environment

A suite of software engineering environment tools to facilitate use of the Modechart language is under development at the University of Texas at Austin [7], as well as at the U. S. Naval Research Laboratory in Washington [1] [2] [3]. A software engineer will have at his or her disposal the following major facilities available under the Modechart environment:

- a sophisticated graphical user interface for entering a modechart, displaying it, searching it, navigating through it, modifying it, and analyzing it [4];
- a simulator for observing an execution of a system modelled by the Modechart specification;
- a verifier for checking the consistency of the modechart against a logical assertion that is desired to be invariant [6] [8];
- auxiliary tools for converting modecharts to and from textual form, a database for storing modecharts, and a suite of consistency and completeness checkers to provide user feedback about the properties of the specification; and
- tools for entering a real-time systems specification in a language other than Modechart and having it translated automatically into a semantically consistent modechart.

Future work will encompass synthesis, the automatic generation of executable programs from

a Modechart specification.

The Modechart simulator

To run the simulator, the user first assembles a file of options (or uses a default set) that tells the simulation tool how nondeterministic decisions are to be made. These decisions include choosing a transition to take when more than one is enabled, scheduling the next time a randomly-occurring external event is to occur, and fixing the execution time of an action between given bounds. For example, although external events may occur any time after a specified delay (also referred to as a minimum separation), the user may instruct the simulator to model these events at specific times, at random times with given statistical distributions, or never. The user also specifies how long he or she wishes the simulation to run, the set of objects (e.g., modes) to be displayed, and a set of breakpoints. The simulator uses breakpoints much like a symbolic debugger. When a breakpoint occurs, the simulator halts to allow the user to inspect or to alter the computation. Breakpoints may occur at specified times or time multiples or they may coincide with events, e.g., whenever a particular mode is entered.

The simulator produces a bar graph. Each bar represents the behavior of a single Modechart object over time, where time begins at zero and is displayed horizontally from left to right. For modes, a thick line indicates that the mode is active at a particular time; a thin line indicates that it is not active. The simulator can also display the temporal behavior of external events, actions, transitions, and boolean variables.

The simulator is a powerful tool during specification creation and refinement, because it lets the user think in terms of behaviors of the system under design. However, it has the drawback that it can only exhibit one behavior of the system at a time, although the user has a great deal of influence about the choice of behavior illustrated.

The Modechart verifier

A more powerful tool for analysis is the Modechart verifier. The verifier allows a user to pose an assertion in a first-order predicate logic (enhanced in a minor way with the addition of an event occurrence function). The verifier then compares that assertion with a computation graph for the specification. A computation graph is a tree of all possible legal states of the system. The verifier checks to see if the assertion is true in all states, in some states, or in no states.

The verifier comes with a set of “pre-packaged” assertions that the user can inquire about. These assertions are suitably parameterized; they include:

- asking the maximum or minimum duration of a particular mode;
- asking whether two modes are ever active simultaneously;
- asking whether an activation of a particular mode is always surrounded in time by the activation of another particular mode; or
- asking whether a particular state is reachable (i.e., ever occurs in any computation);

The last query is particularly powerful. By phrasing an undesirable property in terms of a Modechart state and then inquiring as to that state’s reachability, a user can quickly perform a safety analysis on the specification. If the undesirable state is not reachable, then the user is

guaranteed that no computation of a system that correctly implements the architecture will produce the undesirable state.

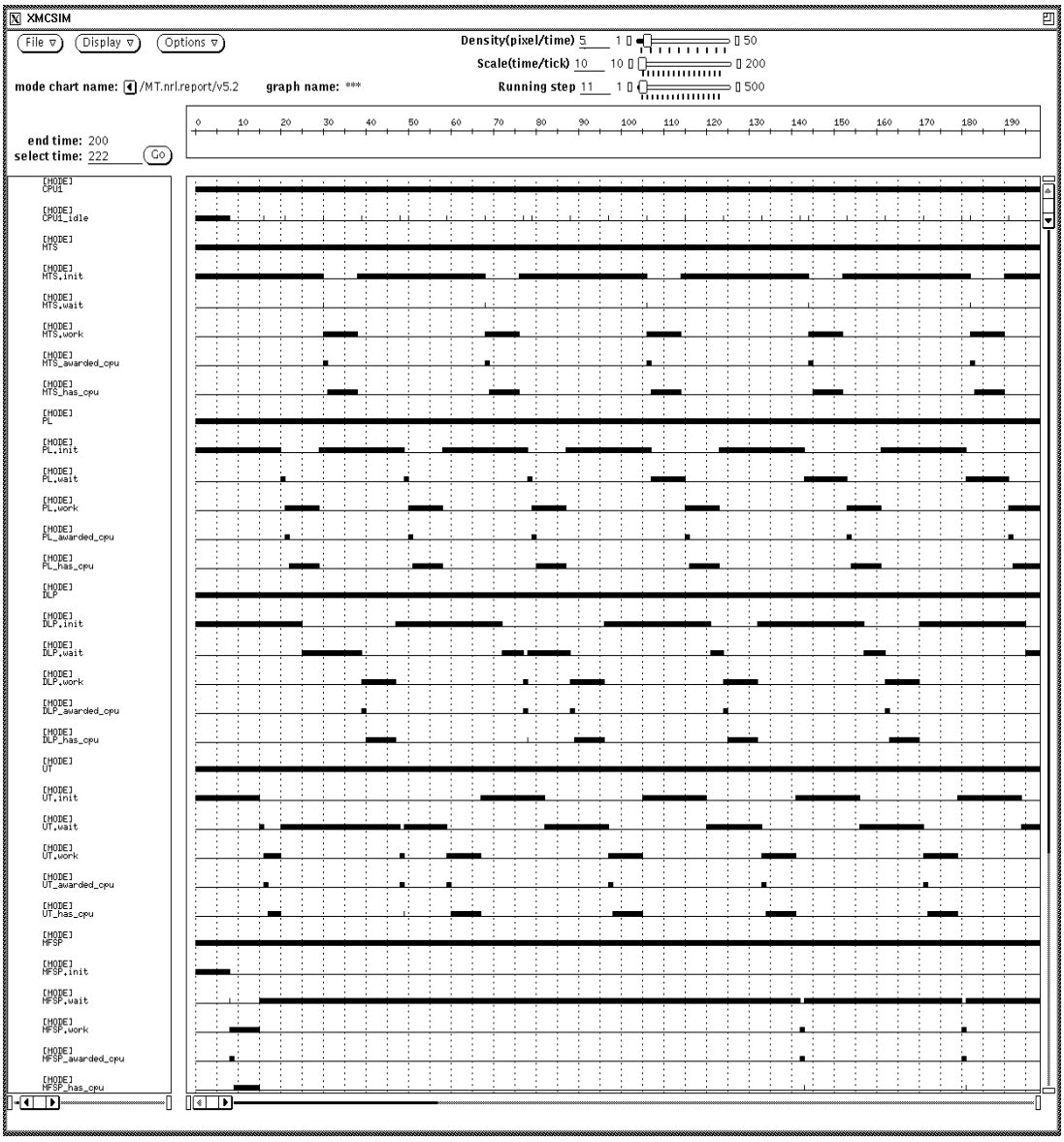

FIGURE 2. The Modechart Simulator.

The verifier is based on model-checking algorithms that compute the specifications entire computation graph and then traverse it checking for the states implied by each query. As such, it suffers from a state-explosion problem that renders it impractical for most real-world problems. However, state-compression techniques are being applied to ameliorate this problem, as is an approach that combines the verifier with the simulator, so that the combined analysis tool can simulate a single computation path up to a point, and then verify a sub-tree of the computation graph from then on. One can also imagine incorporating theorem provers into this framework.

Summary

As indicated previously, we have presented Modechart not because it is an ideal ADL, but because it is a language that can be used as an ADL and has an exemplary analysis environment. Modechart has been used to model the software of an air-to-air radar-guided missile, and the specification was primarily architectural in nature (i.e., the modes represented software components more than computation states) as described here. The analysis performed with the simulator and verifier indicated their usefulness in a development environment, and suggest ways in which formal methods may be incorporated into the software architecture framework.

References

1. Clements, P., C. Heitmeyer, B. Labaw, and A.K. Mok, "Engineering CASE Tools to Support Formal Methods for Real-Time Software Development," *Proceedings, 5th International Workshop on Computer-Aided Software Engineering*, Montreal, 7-9 July 1992. Also available as U. S. Naval Research Laboratory Report 202 (HCI-92-042), February 1992.

2. Clements, P., C. Heitmeyer, B. Labaw, and A. Rose, *A Toolset for Specifying and Analyzing Real-Time Systems: Overview and Example*, U. S. Naval Research Laboratory Report 7405, October 1993.

3. Clements, P., C. Heitmeyer, B. Labaw, and A. Rose, "MT: A Toolset for Specifying and Analyzing Real-Time Systems", *Proceedings, 1993 Real-Time Systems Symposium,* Durham, NC, December 1993.

4. Heitmeyer, C. and B. Labaw, "Software Development for Hard Real-Time Systems," *Proceedings, 7th IEEE Workshop on Real-Time Operating Systems and Software*, Charlottesville, VA, 10-11 May 1990.

5. Jahanian, F. and A.K. Mok, "Modechart: A Specification Language for Real-Time Systems," *IEEE Transactions on Software Engineering*, vol. 20, no. 12, December 1994, pp. 933-947.

6. Jahanian, F. and D. Stuart, "A Method For Verifying Properties of Modechart Specifications," *Proceedings, 9th Real-Time Systems Symposium*, Huntsville, Alabama, 1988.

7. Mok, A.K., "SARTOR-a Design Environment for Real-Time Systems," *Proceedings, 9th IEEE COMPSAC*, pp. 174-181, Chicago, Ill., October 1985.

8. Stuart, D.A., "Implementing a Verifier for Real-Time Systems," *Proceedings, 1990 Real-Time Systems Symposium*, pp. 62-71, Orlando, FL, 5-7 December 1990.