M.E. Project Report

Redesigning of CRSM Editor

Submitted By

Arun Joshi
Roll No: 8003101

For the partial fulfilment of the degree
of
Master of Engineering
in
Software Engineering

Internal Guide
R. S. Salaria
Asst. Professor,CSE Department.
T.I.E.T. Patiala
Patiala

External Guide
Dr. S. Ramesh
Professor,CSE Department
I.I.T. Bombay
Mumbai

Department of Computer Science and Engineering
Thapar Institute Of Engineering and Technology
Patiala

December 2001

CERTIFICATE

This is to certify that the thesis entitled **" REDESIGNING OF CRSM EDITOR "** submitted by **Mr. Arun Joshi** in partial fulfillment for the award of degree of **Master of Engineering** in **Software Engineering** of Thapar Institute of Engineering and Technology (Deemed University), Patiala, Punjab, is an authentic record of student's own work carried out by him under my guidance & supervision.

The matter embodied herein has not been submitted to any other University for the award of any other degree.

R. S. Salaria
Assistant Professor,
Computer Science and Engineering Department,
Thapar Institute of Engineering and Technology,
Patiala.

Seema Bawa 4.1.2002

Mrs. Seema Bawa
Head,
Computer Science and Engineering Department,
Thapar Institute of Engineering and Technology,
Patiala.

Dean
Academic Affairs

CERTIFICATE

This is to certify that the thesis entitled **" REDESIGNING OF CRSM EDITOR "** submitted by **Mr. Arun Joshi** in partial fulfillment for the award of degree of **Master of Engineering** in **Software Engineering** of Thapar Institute of Engineering and Technology (Deemed University), Patiala, Punjab, is an authentic record of student's own work carried out by him under my guidance & supervision.

The matter embodied herein has not been submitted to any other University for the award of any other degree.

R. S. Salaria
Assistant Professor,
Computer Science and Engineering Department,
Thapar Institute of Engineering and Technology,
Patiala.

Seema Bawa 4.1.2002
Mrs. Seema Bawa
Head,
Computer Science and Engineering Department,
Thapar Institute of Engineering and Technology,
Patiala.

Dean
Academic Affairs

Acknowledgement

I would like to thank my guide, **Prof. S. Ramesh** for his invaluable guidance and encouragement while every phase of project work.

I also convey my gratitude to my internal guide **Prof. R. S. Salaria** for his persistent encouragement.

Finally I would like to thank all the people of CFDVS and all others who were directly or indirectly involoved in the successful completion of this project.

Arun Joshi
M.E.
TIET, Patiala
December 16, 2001

Contents

1 Introduction 11

2 Reactive Systems 13

2.1 Reactive System 13

2.2 Synchronous Languages 14

2.3 Distributed Control Systems 15

2.4 Communicating Reactive Process (CRP) 16

3 Communicating Reactive State Machines 17

3.1 CRSM: The Language 17

3.1.1 Automata 18

3.1.2 Hierarchical Composition 19

3.1.3 Synchronous Parallel Composition 20

3.1.4 Communicating 21

4 CRSM Editor 25

4.1 Introduction and System Overview 25

4.2 Design Considerations 26

4.2.1 General Considerations 26

4.2.2 Development Methods 26

4.3 Detailed Design 27

4.3.1 UML diagrams and screen shot 27

4.3.2 Data Structure for Storage 27
4.3.3 Class details 31
4.3.4 CRSMMain class 31
4.3.5 Treeclass class 33
4.3.6 class UserCellRenderer 34
4.3.7 Tabs class 34
4.3.8 MyPanel class 37
4.3.9 UpperToolBar class 41
4.3.10 TabChange class 42
4.3.11 CRSM_At_Single_Level class 43
4.3.12 Node class 43
4.3.13 State class 46
4.3.14 Channel class 48
4.3.15 Connection class 50
4.3.16 Coord class 52
4.3.17 DrawObject class 52
4.3.18 DialogBox class 54
4.3.19 GeneralDialog class 54
4.3.20 NewGeneralDialog class 54
4.3.21 NodeDialog class 55
4.3.22 ConnectionDialog class 55
4.3.23 NodeI_ODialog class 56
4.3.24 Signals_Dialog class 56
4.3.25 NodesPanel class 56
4.3.26 ChannelPanel class 57
4.3.27 StatesPanel class 57
4.3.28 TransitionsPanel class 58
4.3.29 OrthoPanel class 58
4.3.30 Signal class 59

4.4 Conclusion 59

5 Conclusion **61**

Abstract

Reactive systems are usually very complex. Various programming languages for reactive systems like Esterel, Argos, Statecharts and Lustre have been proposed for reactive systems. Most of these synchronous languages are useful only for centralized controllers. There are many distributed controller applications which requires more than one local controller controlling their local environment and also communicating with other controllers to maintain the global state of the system. Communicating Reactive state Machines (CRSM) [3] is a pictorial language which provides communication primitives for such kind of distributed application.

This project is a part of a broader project for designing an environment for CRSM, which includes development of verification tool, translator, editor and simulator for CRSM. This project is redesigning the existing editor for verification environment of CRSM programs.

Chapter 1

Introduction

This CRSM Editor Design Document provides information about the design issues related with the latest version of the editor developed. It contains the UML diagram, flow chart, screen shots of the editor interface and detailed explanation of the code for all classes. The contents of this document concentrate more on the editor design than the vast knowledge domain of Communicating Reactive State Machines.

The Software Requirements Specification Document may be reffered to along with this document.

Chapter 2

Reactive Systems

2.1 Reactive System

A system whose main component is a reactive program is called a reactive system. Real-time process controller, digital watches, signal processing unit, video games and operating systems are typical example of reactive system. A reactive program can be considered to be composed of three layers:

- An interface with the environment that is in charge of input reception and output productions. It transforms external physical events into internal logical ones and conversely.
- reactive kernel that contains the logic of the program. It handles the logical inputs and outputs. It decides what computations and what outputs must be generated in reacting to inputs.
- A data handling layer that performs computations requested by the reactive kernel.

Reactive kernel is central and most complex part of the reactive system. Various programming languages for reactive systems like **Esterel, Argos, Statecharts,**

Lustre and **CRSM** are concerned with reactive kerbels. The important features of reactive systems are following

- They involve concurrency : Reactive programs naturally permit hierarchical and modular program development. They also allows more than one sub-process running in parallel.
- They are subjected to strict timing constraints : Reactive programs are generally subjected to strict timing constraint like emit signal within 5 second.
- They are deterministic : Determinism is an important characteristic of reactive programs. A deterministic reactive program produces identical output sequences when fed with identical input sequences.
- There reliability is an important goal : Reactive programs are growingly used in safety critical system, so their reliability is always desirable.

A reactive process is executed at arbitrary points of time (usually periodically at regular intervals of time). Any such execution is called its reaction. A reaction is atomic in the sense that no other process in the system can preempt it during the reaction. A reaction takes a bounded and a priori known amount of time. Reactions are defined for all possible states of the reactive process and for all possible status of input signals; this is called reactivity.

2.2 Synchronous Languages

There are two approaches to parallel programming: the synchronous and nonsynchronous approach. The classical concurrent languages such as Communicating Sequential Process,Ada and Occam belongs to the class of asynchronous languages. These languages allow physical concurrency and communication is done through asynchronous rendezvous mechanisms. These concurrent languages are

nondeterministic in nature and hence are not suitable for designing reactive systems.

The synchronous approach is adopted by languages such as Esterel, Luster, Statecharts and Argos. They are suited for designing reactive systems. Instantaneous reaction, broadcast, logical concurrency and determinism are their main features. The synchronous paradigm is based on the assumptions that the external stimuli are provided at well divided instants of time, and the response is generated in the same instant as the stimulus. These languages adopt *synchrony hypothesis*

synchrony hypothesis: each reaction is assumed to be instantaneous and therefore atomic in any possible sense.

A reaction is instantaneous in the sense that, firstly it takes no time with respect to the external environment. Second, each process takes no times with respect to the other subprocess; subprocesses react instantly to each other. In synchronous languages, inter processes communication is done by instantly broadcasting events. All process therefore share the same of environment and view the same status of each other.

Thus synchrony allows to reconcile concurrency and determinism, to write simple and more rigorous programs and to dissociate the logic of the system from implementation dependent features such as reaction times.

2.3 Distributed Control Systems

A distributed control system consists of a number of autonomous subsystems controlling disjoint physical systems but at the same time having system-wide control laws. They are prevalent due to their inherent advantages like fault tolerance, concurrency and modularity. However, behavior of these systems are quite complex as the subsystems beside controlling their local environments normally involve complex interaction with each other in order to maintain system

wide properties. Complex applications like robot drivers and distributed process controllers require both logical and physical concurrency provided by the two parallel programming approaches: synchronous and asynchronous approaches. Communicating Reactive Process(CRP) is a paradigm that combines both the approaches of parallel programming.

2.4 Communicating Reactive Process (CRP)

Communicating Reactive Process enables the communication between the reactive nodes by asynchronous rendezvous mechanism; asynchronous in the sense that it may take an arbitrary amount of time between the desire of communication and its actual completion. The two unique features of CRP are

- CRP nodes while waiting for rendezvous can compute local reactions.
- A rendezvous that is about to be made can be preempted in these local reactions.

Chapter 3

Communicating Reactive State Machines

This chapter describes the theory proposed by Prof S. Ramesh in [3] and is included in this report, as CRSM forms the core of the project There are many languages which are prevalent for programming reactive systems. Esterel [1], Luster, Signal and Argos [2] are some popular language for programming reactive systems. These languages are suitable only for localized controller not for distributed controller.

3.1 CRSM: The Language

Communicating Reactive State Machine (CRSM) is based on Argos. It is useful for describing the behavior of real-time distributed process controllers. The characteristic features of this language are that it has a pictorial syntax, a precise formal semantics and a efficient implementation. CRSM includes a primitive for communication between CRSM programs. A process, beside controlling its local environment, communicates with other process to maintain system wide control laws. This interaction is described by Communicating Reactive Process(CRP)

Chapter 3

Communicating Reactive State Machines

This chapter describes the theory proposed by Prof S. Ramesh in [3] and is included in this report, as CRSM forms the core of the project There are many languages which are prevalent for programming reactive systems. Esterel [1], Luster, Signal and Argos [2] are some popular language for programming reactive systems. These languages are suitable only for localized controller not for distributed controller.

3.1 CRSM: The Language

Communicating Reactive State Machine (CRSM) is based on Argos. It is useful for describing the behavior of real-time distributed process controllers. The characteristic features of this language are that it has a pictorial syntax, a precise formal semantics and a efficient implementation. CRSM includes a primitive for communication between CRSM programs. A process, beside controlling its local environment, communicates with other process to maintain system wide control laws. This interaction is described by Communicating Reactive Process(CRP)

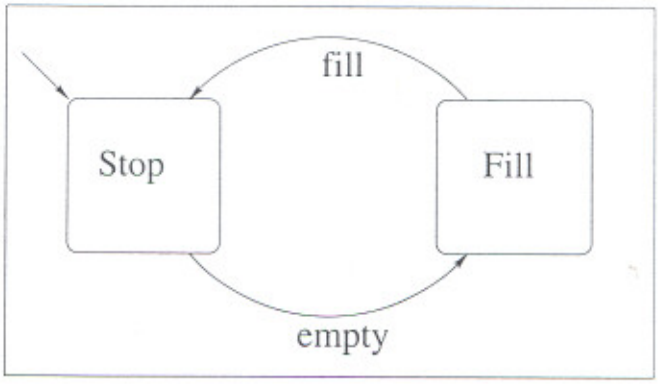

Figure 3.1: A simple Automata

communication primitives for communication while the local behavior is described using Argos construct: mealy automata with inputs and outputs being signal combination, concurrency, hierarchy and signal hiding construct.

A precise formal semantics of CRSM, based upon the boolean automata model of Argos. A CRSM is a network N1//N2...//Nm of independent reactive programs or nodes, Ni, each node having its own reactive interface with separate input/output signals and its own notion of instants. Each node is locally reactive driving a part of a complex process that is handled globally by the network.

3.1.1 Automata

A simple reactive behavior may be described as a labeled transition system as shown in figure 3.1. The transition system has unique initial state. State are represented by boxes. Transitions are made of two parts: the input part I, and the output part O. The complete label is denoted by I/O. The input part is a conjunction or disjunction of signals or negation of signals. The output part gives the signals that reactive kernel outputs to its environment, when reacting to a given input.

The automata shown in the figure 3.1 is an example of automatic water filling system in a tank. Its behaviour will be as follows:

- Initially the controller is in the state STOP.
- Once the water tank is empty, an *empty* signal is generated and the controller. goes into the state FILL and starts filling the water tank.
- Once the water tank is full, the signal *full* is generated and the controller goes into the state STOP.

The graphical representation makes the detection of non-determinism very easy; it is an important notion for reactive systems. In this framework non-determinism of a reactive behavior is simply the existence of two transition sources in the same state, with the same input parts, and different output parts and/or target state.

As the systems get complex, it become difficult to model the system as simple state machine. CRSM support many construct for easier and efficient modeling of reactive system.

3.1.2 Hierarchical Composition

Hierarchical composition or refinement operation is not symmetrical. One of the component, which has to be automaton, is the controller, and the other one is controlled by it. The controller may start and kill them.

A hierarchical structure can be introduced in flat state machine using this construct. Given a machine A with a state, say B, another machine C can be placed inside B. The behavior of state machine shown in figure 3.2 can be given by following rules:

- Initially state A1 is active.
- On signal start, a transition from A1 to state A2 is taken. At the end of the reaction state A2 and state B2 are active.

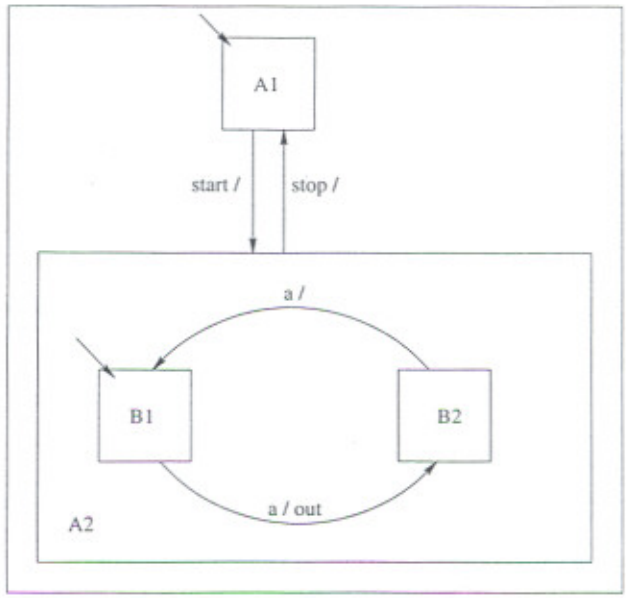

Figure 3.2: A Hierarchical Composition

- As long as the system is in state A2, the state information of the automata inside A2 is kept track of. On signal a transition from B1 to B2 is taken. At the end of the reaction state A2 and B2 are active.
- Once signal stop comes to a transition from state A2 to A1 is taken and all the state and signal information is lost.

3.1.3 Synchronous Parallel Composition

Given two machines A, B, a parallel machine that runs both A and B concurrently can be written using parallel composition. Such a description specifies that the system behaves simultaneously both like A and B. Given a set of input signals representing the state of the environment, appropriate transition in both the automata are taken and the result is the union of the sets of signals generated by both the automata. Moreover, concurrent components can interact. Signal generated by one component can trigger transition in the other components. Such triggering is instantaneous with no delay between the transitions in different

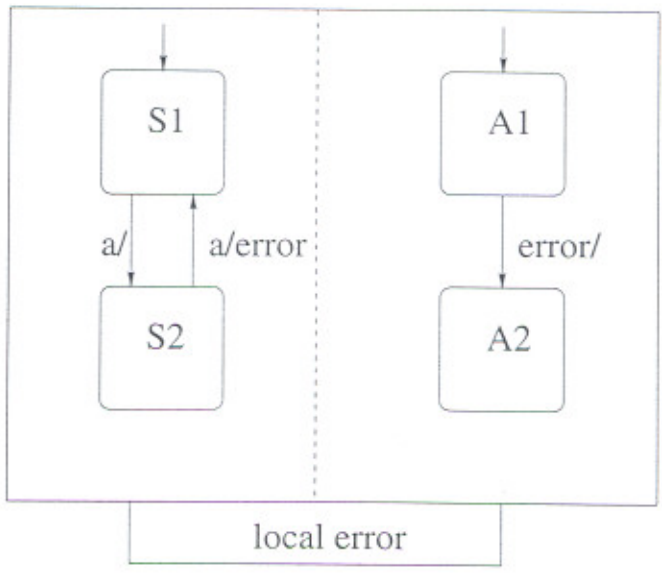

Figure 3.3: Parallel Composition

components.

The above example in figure 3.3 exhibits parallel composition. The behavior of this machine is described as follows:

- Initially both the automata are in their initial state S1 and A1 respectively.
- error is a local signal which is shared between the two automata.
- When the first A signal is seen on a transition, a transition from S1 to S2 is taken. At the end of the reaction S2 and A1 are active.
- On seeing the second A a transition from S2 to S1 is taken generating the signal error. Simultaneously the transition from A1 to A2 is taken on the signal error. At the end of the reaction S1 and S2 are active.

3.1.4 Communicating

Communication between parallel components is done by the signals which are output by one component and input by the another one. The semantics of the

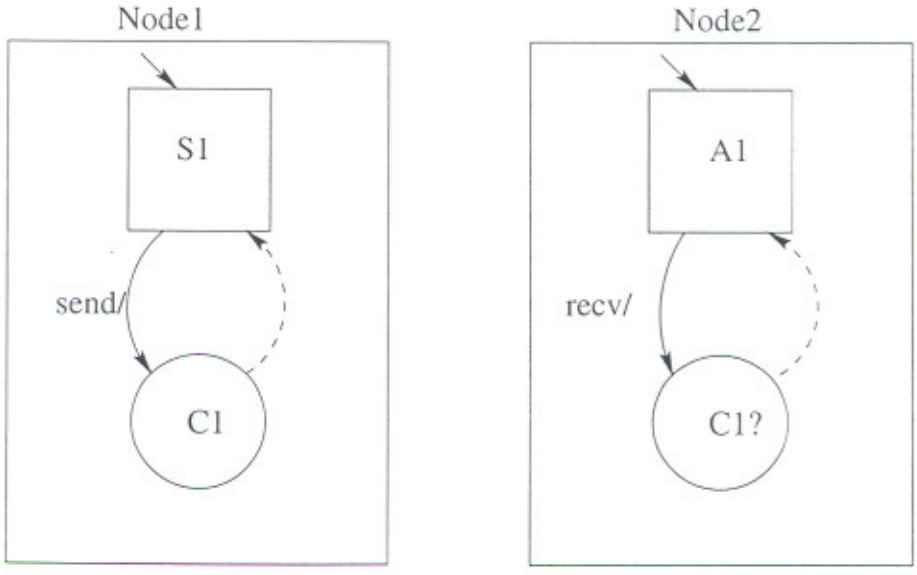

Figure 3.4: Composition

communication mechanism is defined formally as a synchronous broadcast. When a component outputs a signal, it is broadcast it towards its whole environment (the other components and the global environment of the system). Similarly, its input may be from the global environment, or the signal generated by another component.

An example automata is given in figure 3.4.The set of states of a parallel composition is the cartesian product of the sets of states of its component processes. All component runs in an environment made of the global environment and of other components. In each global state, the global reaction is defined by the following rules.

- Both the nodes initially are in the state S1 and A1.
- On seeing the signal send in the node 1 a transition from state S1 to C1 is taken and node 1 is ready to communicate with node 2.
- On seeing signal receive node 2 takes a transition from state A1 to state C1.

- Once both the nodes are present in there respective communicating state communication takes place successfully and the enter the state S1 and A1 respectively.

Chapter 4

CRSM Editor

4.1 Introduction and System Overview

Introduction

This CRSM Editor Design Document provides information about the design issues related with the latest version of the editor developed. It contains the UML diagram, flow chart, screen shots of the editor interface and detailed explanation of the code for all classes. The contents of this document concentrate more on the editor design than the vast knowledge domain of Communicating Reactive State Machines.

The Software Requirements Specification Document may be reffered to along with this document.

System Overview

The CRSM Editor is developed to provide a graphical tool for the description and verification of the properties of Communicating Reactive State Machine. For the design of the editor, analogy is drawn from real state machine/automata and classes are designed to imitate their behaviour closely. The editor must correctly

store and reproduce the properties of the state machines drawn.

4.2 Design Considerations

4.2.1 General Considerations

The editor is designed using the object oriented approach. The language used for writing the programs is Java for its object-oriented support and interoperability on different machines. It also provides a graphics package, Swing for GUI support.

The design of the Editor was assumed to be divided into two type of classes, one part will handle the GUI and the other part will store the state machines drawn. For dynamic storage of objects, Vectors are used. All the information then is written in a particular format in a text file. Storing in a text file was preffered than writing object streams because a text file can read, and interpreted directly. Any changes or editing can be made to it without opening it in the Editor. The properties can be written this way in a text file outside the Editor and it can be opened in the Editor. The editor is supposed to be seamlessly combined with the CRSM simulator developed independently to provide an integrated environment.

4.2.2 Development Methods

The formal development practice was peer reviews and meetings. All the the design issues, scope for further enhancement were discussed and resolved in these meetings.

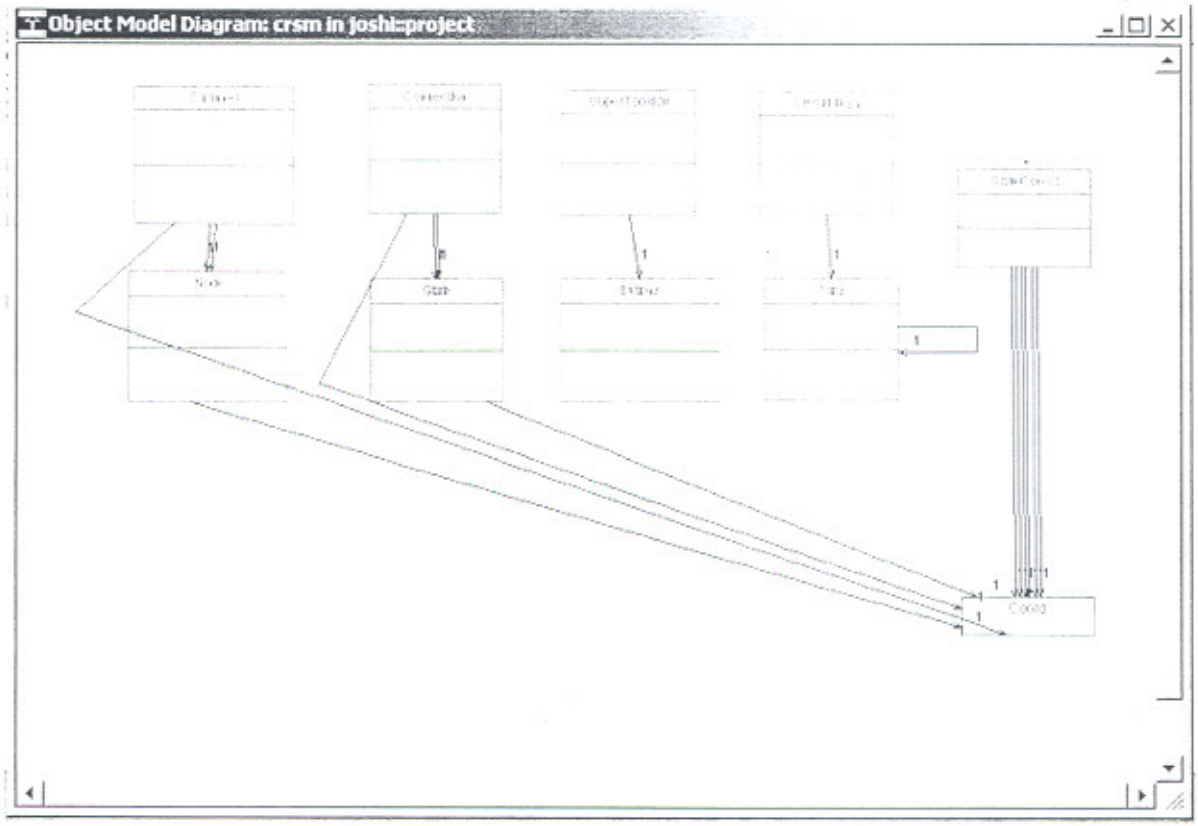

Figure 4.1: UML diagram

4.3 Detailed Design

4.3.1 UML diagrams and screen shot

The hierarchical UML diagram for the classes is in the figure 4.1 The screen shots for top level and other levels are in the figure 4.2

4.3.2 Data Structure for Storage

Storing the CRSM

The data structure used to store the information of the state machines is mainly vectors. Objects added to the vectors are of the classes Node, State, Channel,

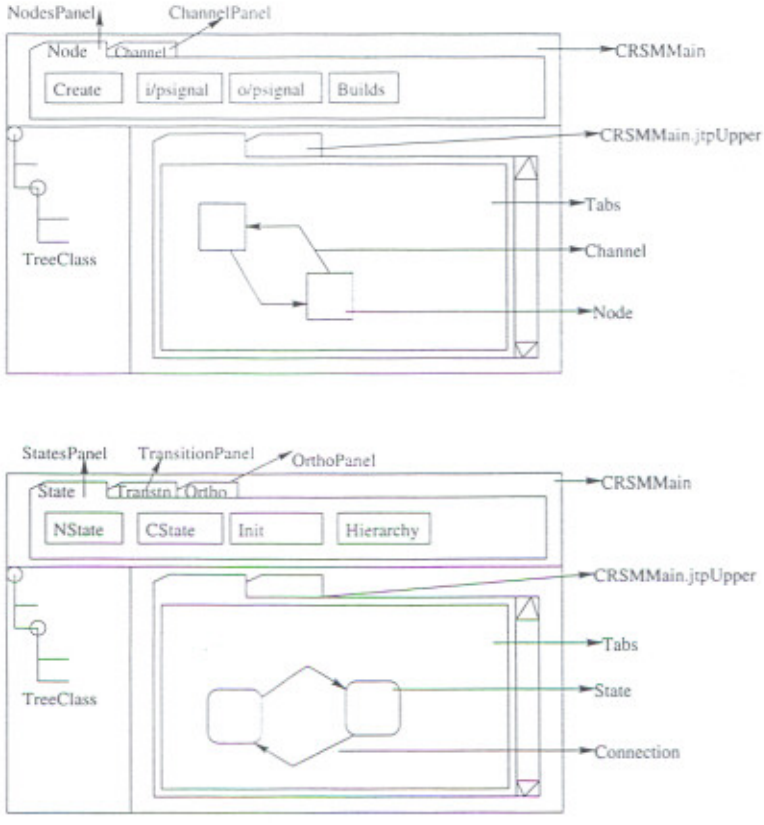

Figure 4.2: Screen shot

Connection type. An example is shown in the figure 4.3 and the corresponding data structure is shown in the figure4.4.

All the state machines drawn at a single level or contained in a single Tab are stored in the Vector statesObj of the CRSMM_At_Single_Level class. Objects of type States class are stored in this vector. An instance of the CRSMM_At_Single_Level class is created whenever a new tab is opened.

A particular machine is stored as a instance of the States class. this class contains three vectors,

stateV

To store nodes at top level, else states. Objects stored are of type Node or State.

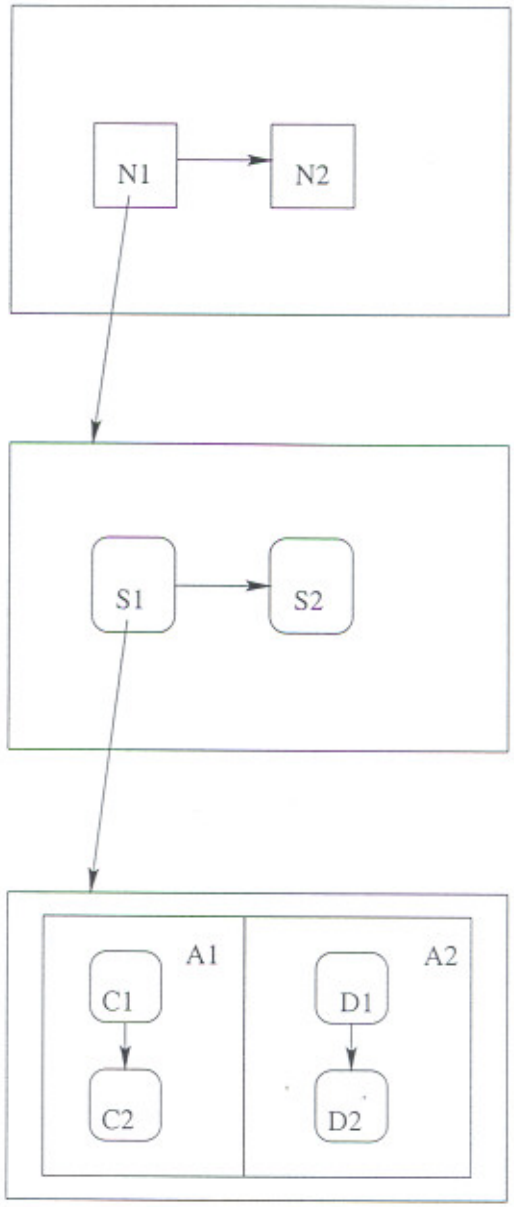

Figure 4.3: Figure 3

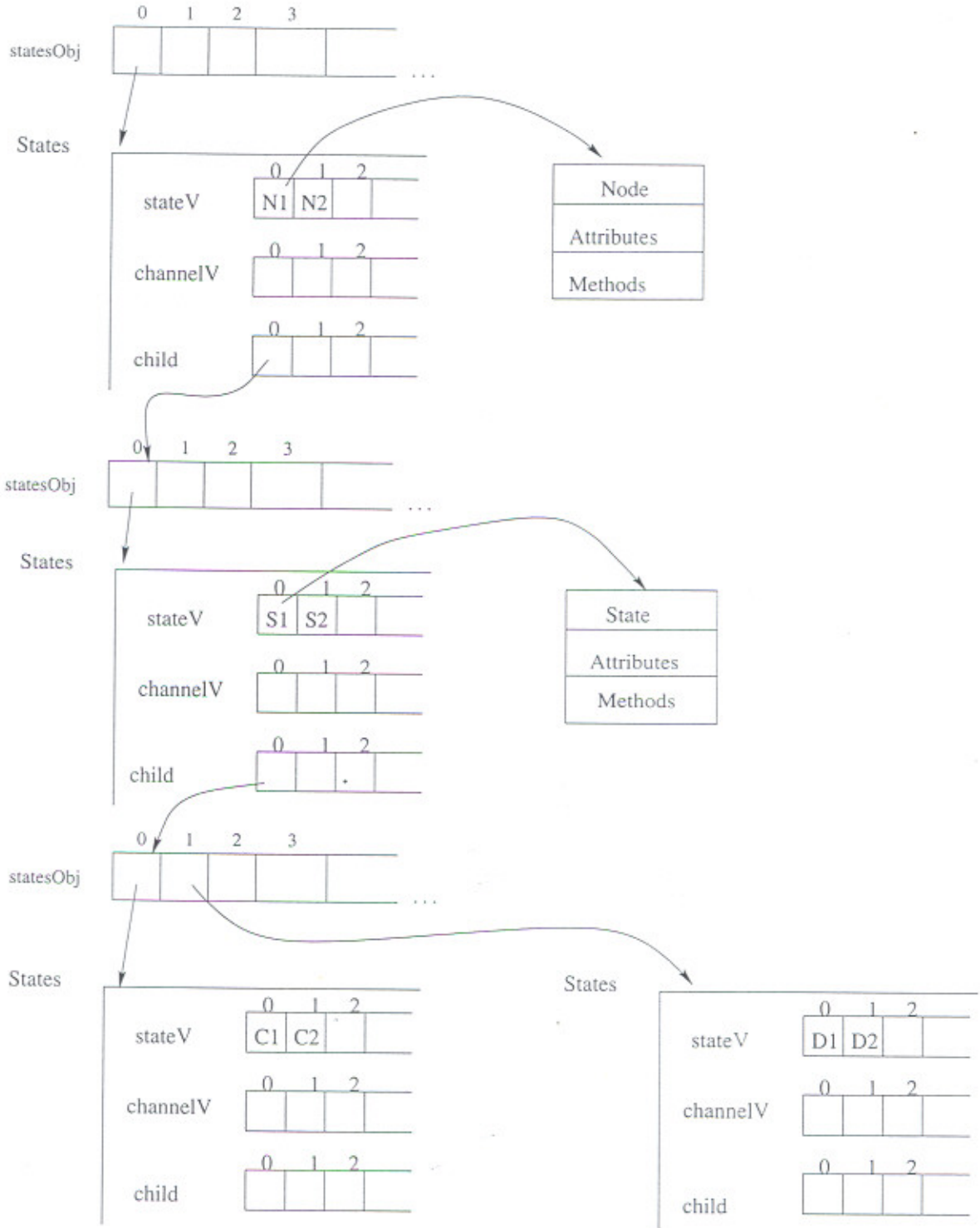

Figure 4.4: Figure 4

connectV

To store the channels at top level, else transition. Object stored of type Channel, Transition.

child

To store all the information of the child of this state machine. a child is created when we build automata inside the Node or when we build hierarchy inside the state. Object stored in the child is an statesObj vector for the child tab.

The whole CRSM is stored in these data structures and is saved in a particular format in a text file that can read again and edited again.

4.3.3 Class details

4.3.4 CRSMMain class

It is the main class of type JFrame which is used to launch the application. All other class objects are controlled through this class.

Member data

1. static CRSMMain mainObj

   It is handle for the object of the CRSMMain class.

2. static Container frameContainer

   frame container for the main application window JFrame. All components on this frame are added to this container.

3. JSplitPane split_pane

   Used to vertically split the frame of the CRSMMain class into two parts. An panel containing object of TreeClass is added to the left and a JTabbedPane class object is added to the right part.

4. static JTabbedPane jtpUpper

   It is a JTabbedPane object that is container for all Tabs class objects.

5. static TreeClass treeclass

   It is used to create a tree correesponding to the state machines drawn on the Tabs Pane.

6. static DefaultMutableTreeNode cur

   It is used to point to the node of the tree corresponding to to the current state machine.

7. static Vector global

   Its a vector of strings to store global signals, currently not used.

8. Tabs jsp

   An object of the Tabs class that will be added to the jtpUpper as a default pane.

9. static UpperToolBar t0

   It extends the ToolBar class and is the tool bar of the the application. Different panels of tool buttons are added to it according to the selected tab. If the current selected tab is of type TOP_LEVEL, then instances of NodesPanel class and ChannelPanel class are added otherwise, instances of StatesPanel, TransitionPanel and OrthoPanel classes are added. It also have fixed buttons for methods that are common to both the levels like label, delete, clear, move.

10. JScrollPane treeJsp

    It is scroll pane to which tree is added. An instance of the TreeClass is added to this JScrollPane and then it is added to the split pane split_pane.

11. Toolkit tk

    Its used to set the default display position of the editor. The frame is

initially displayed in the center of the screen and takes one-fourth area of the screen.

12. boolean firstSave

    Its a boolean flag set to true if this file has never been saved to disk, false after first save.

Member methods:

1. public static void main()

   Its the method which is used to launch the application. The object mainObj is assigned memory in this method. Then show() method is called on it.

2. CRSMMain() Constructor for CRSMMain class. It creates the instances for almost all the data members of the class and then adds them to the frameContainer.

3. public void windowClosing() The method is overridden to react to the event when the user tries to close the main frame. The code to prompt the user and then save or cancel the CRSM is written in this method.

4.3.5 Treeclass class

The Tree drawn corresponding to the state machines drawn at the tab panels, is an instance of this class. It also uses another class named UserCellRenderer (extends DefaultTreeCellRenderer) to add new nodes to the Tree.

Member method:

1. public TreeClass(DefaultTreeModel)

   The object of this class is first created in the CRSMMain class. To create a Tree, a tree model is passed as an argument to the TreeClass constructor. An object of the UserCellRenderer class is also called in this constructor.

4.3.6 class UserCellRenderer

It inherits from the class DefaultCellRenderer and implements the interface TreeCellRenderer. It gives a color convention to the nodes- for a top level node, black . color;for a state machine, green color;for a state. red color.

Member method:

1. public Component getTreeCellRendererComponent(JTree , Object , boolean , boolean , boolean , int , boolean)
   This method adds the Object passed to it in the JTree and at the row given by the int argument and returns a component.

4.3.7 Tabs class

This class represents a tab which is added to the static member jtpUpper of class CRSMMain. Tabs class extends JScrollPane class. It also has a Inner class MyPanel which extends JPanel class. A panel object is created from Mypanel class and is made scrollable by the upper class i.e., Tabs. All the mouse event handlers and Graphics objects are added to the panel and thus to inner class, MyPanel.

Member data:

1. MyPanel jp
   It stores a data type of inner class MyPanel. MyPanel class will be explained later in this section.

2. int indexS
   It contains the index of the state/node that was clicked. This index is of the stateV Vector, a member of States class. This vector stores the objects

of type Node for top level, and type State for other levels. indexS is equal to indexStateMachine at the top level.

3. int indexStateMachine

   It contains the index of the state machine that was clicked as there can be any number of state machines on a single level. At top level indexStateMachine is equal to indexS.

4. int indexC

   It stores the index of the transition that was clicked. Transitions are stored in the Vector channelV of the States class. It stores objects of Channel type for top level and Connection type for other levels.

5. Tabs parentTabs

   This points to the Tabs object of the parent tab of this tab. Parent Tab means the Tab which contains the state machine inside which the current Tab is build or a hierarchy is created. Of course parent tab for the top level is null.

6. Tabs currentTabs

   It points to the Tabs object of the current tab selected.

7. int parentIndex

   It stores the index of the parent node/state which is the parent of this panel. It signifies the place of the parent machine (inside which the current tab is build) in the parent tab.

8. int parentMachineIndex

   The complete information for a single state machine is stored as one element of the statesObj Vector of the CRSM_At_Single_Level class. parentMachineIndex contains the index of this Vector.

9. DefaultMutableTreeNode parent

   The tree node that is the parent of nodes / statemachine / states on this panel

10. DefaultMutableTreeNode cur = null

    It refers to the current node or position at which the machine is to be added.

11. int newNailIndex

    For the AddNail function, the index of the channel on which the mouse was clicked, is needed. This is stored in indexC. The particular nail of the selected channel is also required to execute the drag function correctly. newNailIndex exactly returns the position in the nail list at which the new coord should be added. The nails are stored in an array list data structure in the Channel/Connection classes.

Member methods:

1. public Tabs(int typeVal, Tabs parentTabs, int parentMachineIndex, int parentIndex)

   This is the constructor for the Tabs class.

2. int typeVal

   sets the type of new Tab created. the type can be equal to Constants.ZOOM_LEVEL, Constants.HIERARCHY_LEVEL, or Constants.ORTHO_LEVEL.

3. Tabs parentTabs

   It passes an object of Tabs type which is the parent tab of the current tab.

   int parentMachineIndex

   This argument contains the index of the parent state machine on the parent tab.

4. int parentIndex

   It gives the index of the parent node/state of the current tab. so the parent

of the current tab can be traced as,
(State)(((States)(parentTabs.jp.levelObj.statesObj.
elementAt(parentMachineIndex))).stateV.elementAt(parentIndex))

4.3.8 MyPanel class

It extends a JPanel and provides the panel required in Tabs class to add to the static member jtpUpper of CRSMMain class. This is an inner class of the Tabs class. so it can access all the members of Tabs class directly but vice-versa is not true.

Member data:

1. int tool

   The tool value is referred everytime the user clicks a mouse on panel, selects a menu item or performs any of action. e.g., when the Create button of NodesPanel is pressed, it sets the tool equal to Constants.CNode; now when mouse is clicked on the panel, the tool value implies to draw a node.

2. int type

   It gives the type of the tab to which this object will be added.It can have values Constants.TOP_LEVEL, Constants.
   HIERARCHY_LEVEL. For ZoomIn type is
   Constants.ZOOM_LEVEL. For modify type is
   Constants.ORTHO_LEVEL.

3. int currentIndex

   tells the index of the current channel in the connectV Vector. Its used in functions duplicate(), findConnection() while creating a Channel or adding nails to it.

4. int count = 0

   To count the mouse clicks while drawing transitions and channels

5. Coord coord = new Coord()

   Stores the coordinates of the point where mouse was clicked.

6. int indexNail

   stores the index of the nail on which mouse was clicked. Useful for add nail and remove nail functions.

7. int previndexStateMachine: While adding nails to a state machine on a panel having orthogonal machines, the possibility of dragging a connection to a different state machine should be taken care of. so the initial state machine getting the mouse click should be remebered. previndexStateMachine does that.

8. int previndexC

   Its used for the AddNail functionality in the MyPanel class. While adding nails to the connections of a state machine on a panel having orthogonal machines, the possibility of dragging and clicking on some other connection than the initial one is possible.so the initial connection index is stored in this variable.

9. int previndexS

   To move a state on a orthopanel, the state should not enter in the fence of the other state machine. previndexS stores the state machine index on which mouse was clickef first time.

10. Vector oldStatesObj = null

    this vector is used when a previously saved CRSM file is opened and new information is added to it. The old copy is stored in the oldStatesObj Vector. Now when the user tries to close the application, he is prompted

for saving it. If the choice is NO then the new data structure is ignored and oldStatesObj is added again to its proper place. This vector is used in the implementation of the noSave() method.

11. CRSM_At_Single_Level levelObj: This is the object which stores CRSM drawn on this MyPanel object.

12. NodeDialog dialog1
    Dialog box for the labeling of nodes/states.

13. boolean needSave
    If true, this panel needs to be saved, else not.

Member methods:

1. MyPanel(int)
   Its an constructor for MyPanel. The int type argument passed decides which type of panel to create. It can be.of three types, Constants.TOP_LEVEL, Constants.ZOOM_LEVEL,
   Constants.ORTHO_LEVEL. The constants do not effect the behaviour of the JPanel that MyPanel isThese are used only to assign the data structure accordingly to each level.
   Objects of two classes MouseHandler and MouseMotionhandler are added to this panel. These classes deal with all the mouse events on the panel and are expalined in detail later.

2. public void clear()
   This method is invoked for clearing the panel

3. void findMinMax(Vector vect, int index, Coord min, Coord max)
   Used for finding the minx, maxx values used to fence statemachines

4. void fence(int index, Graphics g, Coord min, Coord max)
   Used to fence a statemachine

5. int[] findState(Coord coord,int typeVal): Used to find on which state and statemachine mouse was clicked on.returns an array of int

6. int findConnection(Coord coord,int typeVal, int index)
   used to find which channel/connection was clicked on

7. Vector findAssociatedConnections(int index, int typeVal.int choice): To find out which are Channels/Transitions associated (to/from) to the Node/State.

8. void setTool(int toolType)
   It sets the tool to some static value of the Constants class.

9. boolean duplicate(int typeVal)
   This function checks for a duplicate channel or transition

10. void saveFunction()
    This function is used to save the part of CRSM drawn on this panel

11. void noSave()
    noSave() method is used to undo any changes done on a panel before closing it when user chooses "No" option in the save dialog.

12. void close()
    This method is invoked as a part of closing a panel or the application. It majorly deals with prompting the user for saving the changes.

13. public void paintComponent(Graphics g): The method is invoked when repaint() is called, repaints the panel.

Besides these there are also two inner class in MyPanel class which contain all the mouseListener methods. These classes contains the methods mouseClicked,

mousePressed, mouseMoved. since these are the inner classes of MyPanel (MyPanel is an inner class of Tabs) so any member of the outer classes can be accessed directly inside their body.

class MouseMotionHandler

It extends MouseMotionAdapter interface. mouse clicked(), mousePressed() methods are overriden in this class.

class MouseHandler

This class contains mouseMoved() method. The method is ovevrriden here to work out the functions like, drawing a channel/ transition drag a nail for AddNail and moving the state/node in the panel.

4.3.9 UpperToolBar class

It contains two components. First one is jtpLower, a JTabbedPane to add diferent tool panels. Second is a panel containing buttons like label, delete, etc to perform functions that are common to both Nodes and States. Two classes Buttons and TabChangeHandler are associated specifically to this class and have not been used anywhere else. So both are expalined in this class.

Member data

1. JTabbedPane jtpLower
   Its an TabbedPane to add the different tool panels.

2. int type
   its set to values Constants.TOP_LEVEL for top level and Constants.HIERARCHY_LE
   for all others except for modify orthogonality. For modify orthogonality,
   type is Constants.ORTHO_LEVEL

3. Buttons btns: Different objects of this Buttons class are created by passing the argument in its constructor telling if current level is top or not.

Member methods

1. UpperToolBar(int typeVal)
   This method is the constructor. It creates an object of JTabbedPane class, assigns it to jtpLower, then checks the type value passed as argument. If the value tells that the current tab is top level then it adds NodesPanel and ChannelPanel objects to the jtpLower otherwise it adds StatesPanel, TransitionPanel, OrhtoPanel class objects to jtpLower. An instance of the Buttons class is also added to jtpLower.

class TabChangeHandler

It implements the interface ChangeListener attached to the jtpLower object of the UpperToolBar class. The method stateChanged() is overriden in this class to set the tool type to some default value corresponding to each tool panel except OrthoPanel.

class Buttons

It extends a panel and implements the ActionListener interface to catch the events of buttons on the panel. The argument passed in the constructor is used to choose the buttons to show on top level and otherwise.

4.3.10 TabChange class

It implements the interface ChangeListener. It is attached to the static JTabbed-Pane object, jtpUpper of the CRSMMain class. It acts as the listener interface for the tabs contained in jtpUpper.

Member method

1. public void stateChanged(ChangeEvent e)
   This is standard method of the ChangeListener interface and is overrided to act on the changes in the tab panes of type Tabs in jtpUpper.
   The purpose here to is to change the tool panel for Node if the selected tab is of zero index otherwise keep the tool panel set for State. This change in tool panel is done in UpperToolBar class.

4.3.11 CRSM_At_Single_Level class

This class is part of the data structure used to store all the information about the machines drawn on the Tabs pane.

Member data

1. Vector statesObj This is the vector which stores CRSM drawn on a particular level (panel). We add objects of type States in this vector.
2. Vector local = new Vector(5,2) This vector stores signals which are local for this level
3. float miny, maxy The miny and maxy coordinate values used to fence all statemachines on this level (panel) with uniform height boxes. Initially set as miny= Float.MAX_VALUE, and maxy=Float.MAX_VALUE

Member methods

1. CRSM_At_Single_Level()
   Default constructor

4.3.12 Node class

This class represents a single node in the top level view of the CRSM.

Data Members

1. private Coord position

   The coordinate of the centre of the node.

2. private int totalChildren

   Number of child state machines for this node.

3. private String name

   Name of the node;

4. Vector input

   Vector storing input signals to this node.

5. Vector output

   Vector storing output signals to this node.

6. public boolean open

   True if another panel is opened out of this node, false otherwise. It is used to decide if this node can be deleted. If it is true, the node can not be deleted. Also if any of the nodes on the current panel has this variable set, the panel can not be closed.

7. public boolean refined

   True if hierarchy is created for this node machine, false otherwise. This variable is primarily used to decide if the node is to be shown highlighted. Also it is used to decide the type of the tab to be opened when **NInside** button is selected. If the node on which the mouse is clicked after clicking on the NInside button has refined == true, ZOOM_LEVEL type panel is opened, otherwise HIERARCHY_LEVEL type of panel is opened.

8. public boolean savedBefore

   If it is true, it means there the part inside the node has been saved at least

once before. This variable is used to decide if the **Save** dialog box is to be invoked when Save menu item is chosen. Also when first time hierarchy is created inside a node, it is used to set variable to false if actually nothing is created inside the node at the end of the operation.

9. public Vector childStates
   Stores number of states in individual state machines on this node's child panel.

10. public Vector childConnections
    Stores number of connections in individual state machines on this node's child panel.

Function Members

Purpose of most of the following functions can easily be known from their names. Hence only those which really require explaination have been elaborated.

1. void inctotalChildren()
   Increments the value of the variable totalChiledren.

2. void dectotalChildren()
   Decrements the value of the variable totalChiledren.

3. void settotalChildren(int count)
   Sets the value of the variable totalChiledren.

4. int gettotalChildren()
   Returns the value of the variable totalChiledren.

5. public void setInput(Vector inputVector)
   Sets the value of the vector **input**.

6. public Vector getInput()
   Returns the value of the vector **input**.

7. public void setOutput(Vector outputVector)
8. public Vector getOutput()
9. public boolean emptyInput()
   Returns true if vector input is empty, false otherwise.
10. public boolean emptyOutput()
11. public void setLabel(String name)
12. public String getLabel()
13. public void setPos(Coord p)
14. public Coord getPos()
15. public boolean isEqualTo(Node node)
    Returns true if this node is equal to the parameter node, false otherwise.

4.3.13 State class

This is a class to represent a single state in a CRSM state machine.

Data Members

1. private Coord position
   The coordinate of the centre of the state.
2. private int totalChildren
   Number of child state machines for this state.
3. private String name
   Name (label) of the state.
4. private int type
   Type of the state, CState or NState.

5. public boolean open

   True if another panel is opened out of this state, false otherwise. It is used to decide if this state can be deleted. If it is true, the state can not be deleted. Also if any of the states on the current panel has this variable set, the panel can not be closed.

6. public boolean refined

   True if hierarchy is created for this state, false otherwise. This variable is primarily used to decide if the state is to be shown highlighted. Also it is used to decide the type of the tab to be opened when **Hierarchy** button is selected. If the state on which the mouse is clicked after clicking on the Hierarchy button has refined == true, ZOOM_LEVEL type panel is opened, otherwise HIERARCHY_LEVEL type of panel is opened.

7. public boolean savedBefore

   If it is true, it means there the part inside the state has been saved at least once before. This variable is used to decide if the **Save** dialog box is to be invoked when Save menu item is chosen. Also when first time hierarchy is created inside a state, it is used to set variable to false if actually nothing is created inside the state at the end of the operation.

8. public Vector childStates

   Stores number of states in individual state machines on this state's child panel.

9. public Vector childConnections

   Stores number of connections in individual state machines on this state's child panel.

Function Members

Purpose of most of the following functions can easily be known from their names. Hence only those which really require explaination have been elaborated.

1. void inctotalChildren()
   Increments the value of the variable totalChiledren.
2. void dectotalChildren()
   Decrements the value of the variable totalChiledren.
3. void settotalChildren(int count)
   Sets the value of the variable totalChiledren.
4. int gettotalChildren()
   Returns the value of the variable totalChiledren.
5. public void setLabel(String name)
6. public String getLabel()
7. public void setPos(Coord p)
8. public Coord getPos()
9. public boolean isEqualTo(State state)
   Returns true if this state is equal to the parameter state, false otherwise.

4.3.14 Channel class

This is a class to represent a single Channel in the top level view of the CRSM.

Member data

1. private Node fromNode
   Source node of the Channel.

2. private Node toNode

   Destinition node of the Channel.

3. private String label

   Label of the channel.

4. ArrayList nails

   A 2-d array with 2 locations to store nails on the channel.

5. private Coord labelCoord

   Coordinate where label of the channel is to be displayed

Member methods

1. public void setLabelCoord(Coord p)

   Sets the coordinate where the label of the channel is to be displayed.

2. public Coord getLabelCoord()

   Returns the coordinate where the label of the channel is to be displayed.

3. public void setNails(ArrayList nailsList)

   Sets the array for nails of the channel to the parameter array.

4. public ArrayList getNails()

   Returns the array for nails of the channel;

5. public void addNail(Coord nail)

   Adds a nail to the channel.

6. public void removeNail(Coord nail) Removes a nail from the channel.

7. public void removeLastNails(int extraMoves)

   Removes last **extraMoves** number of nails from the nails array.

8. public void setLabel(String a)

9. public String getLabel()

10. public Node getFrom()

    Returns the source node of the channel.

11. public void setFrom(Node from)

12. public Node getTo() Returns the destinition node of the channel.

13. public void setTo(Node to)

4.3.15 Connection class

This is a class to represent a single connection in a CRSM state machine.

Member data

1. private State fromState

   Source State of the Connection.

2. private State toState

   Destinition State of the Connection.

3. private String action

   Action label of the connection.

4. private String event

   Event label of the connection.

5. ArrayList nails

   A 2-d array with 2 locations to store nails on the connection.

6. private Coord labelCoord

   Coordinate where label of the connection is to be displayed

7. private int type

   Constants.PTrans for preemptive transition and NTrans for Normal exit.

Member methods

1. public void setLabelCoord(Coord p)

   Sets the coordinate where the label of the connection is to be displayed.

2. public Coord getLabelCoord()

   Returns the coordinate where the label of the connection is to be displayed.

3. public void setNails(ArrayList nailsList)

   Sets the array for nails of the connection to the parameter array.

4. public ArrayList getNails()

   Returns the array for nails of the connection.

   item public int getTotalNails()

   Returns the total number of nails on this connection.

5. public void addNail(Coord nail)

   Adds a nail to the connection.

6. public void removeNail(Coord nail) Removes a nail from the connection.

7. public void removeLastNails(int extraMoves)

   Removes last **extraMoves** number of nails from the nails array.

8. public void setLabel(String a)

9. public String getLabel()

10. public State getFrom()

    Returns the source State of the connection.

11. public void setFrom(State from)

12. public State getTo() Returns the destinition state of the connction.

13. public void setTo(State to)

4.3.16 Coord class

This is a simple class to represent two floating point co-ordinates (x and y) for representing a point on the panel.

Members data

1. public float x

   The X co-ordinate of the point.

2. public float y

   The Y co-ordinate of the point.

Function Members

1. public int dist(float x, float y)

   Returns the distance between the current and the co-ordinate pair given in the parameter list.

2. public int dist(Coord d)

   Returns the distance between the current and the co-ordinate given in the parameter list.

4.3.17 DrawObject class

Member data

1. private int type

   It tells which type of shape is to be drawn. e.g. if type =CChannel then draw channel.

2. private boolean refined

   If the state/node is refined, tells it to color dark.

3. private Coord start,end,arrowtip,arrow2,arrow3,labelCoord

   the coords to draw an arrow shape.

4. private String name,action,event

   Strings to label on the channel/transition.

5. private ArrayList nails

   ArrayList is a class similar to Vector. Used to store the Coord type objects.

6. float xpoints[ ] = new float[3]

7. float ypoints[ ] = new float[3]

8. Shape shape

   Shape is an interface defined in the java2D kit. Used to draw shapes in Graphics2D class.

9. boolean ip,op

   To decide whether a node is having input or output signals.

10. float[ ] dashpattern = $10.0f, 10.0f, 10.0f, 10.0f$

Member methods

1. public DrawObject(int objectType,String name,boolean input,boolean output, Coord start,boolean refineValue)

   It is constructor for the class. Used to draw a node with input/output signals.

2. public DrawObject(int objectType,String name,Coord start,boolean refineValue)

   It is a constructor used to set the shape to draw state

3. public DrawObject(int objectType,Coord start,Coord end,Coord arrowtip
   Used to set the shape to draw a straight linewith an arrrow tip.

4. public DrawObject(int objectType,Coord start,Coord end,Coord arrowtip
   To draw a channel/transition having many edges.

5. public void drawShape(Graphics gr)
   The method used to draw figures of the type Shape class.

4.3.18 DialogBox class

This class is meant for the dialog in which user is prompted for saving the changes. Because of the simplicity of the class, not much to be explained here.

4.3.19 GeneralDialog class

This class is meant for displaying various error, warning and any other general types of messages to the user while he performs operations. The message to be displayed is passed as a parameter to the constructor of the class. Because of the simplicity of the class, not much to be explained here.

4.3.20 NewGeneralDialog class

This is a class for a special type of dialog in which user is prompted to close any child level panels that are open at the time of saving a particular panel. The displayed message is.

"Child panel/panels open, Continuing U will lose changes made to the lower level panels"

4.3.21 NodeDialog class

This is a class for the Label dialog box for nodes. states and channels. It uses the boolean parameter **type** for identifying for which entity it is being used.

Member data

1. boolean type

   It is used to identify for which type of entity is this dialog being used. If type = true, a node or a state is being labelled. Otherwise, i channelis being labelled. If the operation is invoked on toplevel, then a node or a channel is being labelled. Otherwise, a state is being labelled.

Member methods

1. public String getName()

   It returns the name of the entity give in the dialog box.

4.3.22 ConnectionDialog class

This is a class for the Label dialog box for connections.

Member data:

1. boolean type

   It identifies the type of transition being labelled. If it is true, a preemptive type transition is bing labelled. Otherwise a normal exit type transition is being labelled.

Member methods:

1. public String getAction()

   Returns the action string of the dialog.

2. public String getEvent()

   Returns the event string of the dialog.

4.3.23 NodeI_ODialog class

This is a class for the dialog box used for specifying the input and output signals for nodes.

Member data

1. boolean type

   If it is true, the dialog box will be invoked for specifying input signals, else it willbe invoked for specifying output signals.

Member method

1. public Vector getI_OSignals()

   Returns the vector containing the input or output signals depending on the value of the variable type.

4.3.24 Signals_Dialog class

This is a class used for the dialog that is used to manipulate local and the global signals. end classes for all Dialog Boxes

4.3.25 NodesPanel class

It is used to hold the tool buttons for when nodes are drawn at the top level.

Member methods

1. NodesPanel()

   This is the default constructor. Instances of all the buttons are created here

and added to the paanel.

2. actionPerformed(ActionEvent)
   It sets the the value of the tool variable using the method setTool of the Mypanel class inside Tabs.

4.3.26 ChannelPanel class

It is used to hold the tool buttons for when channels are drawn at the top level.

Member methods

1. ChannelPanel()
   This is the default constructor. Instances of all the buttons are created here and added to the paanel.

2. actionPerformed(ActionEvent)
   It sets the the value of the tool variable using the method setTool of the Mypanel class inside Tabs.

4.3.27 StatesPanel class

It is used to hold the tool buttons for when states are drawn at the panel.

Member methods

1. StatesPanel()
   This is the default constructor. Instances of all the buttons are created here and added to the paanel.

2. actionPerformed(ActionEvent)
   It sets the the value of the tool variable using the method setTool of the Mypanel class inside Tabs.

4.3.28 TransitionsPanel class

It is used to hold the tool buttons for when transitions are drawn at the top level.

Member methods

1. TransitionsPanel()
   This is the default constructor. Instances of all the buttons are created here and added to the paanel.

2. actionPerformed(ActionEvent)
   It sets the the value of the tool variable using the method setTool of the Mypanel class inside Tabs.

4.3.29 OrthoPanel class

It is used to hold the tool buttons. The create button fences the already drawn state machine so that the new state machine is drawn parallel to it. The modify button is used when the user wants to modify any of the fenced machine. A new tab is opened with the state machine to be modified on it. The user modifies it and zoom out adding the changes to the stateObj vector also.

Member methods

1. OrthoPanel()
   This is the default constructor. Instances of all the buttons are created here and added to the paanel.

2. actionPerformed(ActionEvent)
   It sets the the value of the tool variable using the method setTool of the Mypanel class inside Tabs.

4.3.30 Signal class

The signals stored in this class will be used for simulation. Presently not in use.

4.4 Conclusion

Amid other synchronous programming languages CRSM provides communicating primitives for distributed controller applications. Its pictorial syntax is useful from user point of view. Since application of reactive system is increasing in safety critical and mass application systems, there is a continuous need for verification of reactive programs.

In this report we have discussed languages for reactive system in general and CRSM in particular. We have also discussed how to use the CRSM editor developed as the end-product of this project.

Chapter 5

Conclusion

Amid other synchronous programming languages CRSM provides communicating primitives for distributed controller applications. Its pictorial syntax is useful from user point of view. Since application of reactive system is increasing in safety critical and mass application systems, there is a continuous need for verification of reactive programs.

In this report we have discussed languages for reactive system in general and CRSM in particular. We have also discussed how to use the CRSM editor developed as the end-product of this project.

References

[1] Gerard Berry and Georges Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[2] F. Maraninchi. The argos language: Graphical representation of automata and description of reactive systems, 1991.

[3] S. Ramesh. Communicating reactive state machine: Design, models, and implementation. In *IFAC Workshop on Distributed Computer Control Systems. Pergamon Press*, September 1998.