Vision Document

For agentTool III (Dynamic)

Version 1.0

Submitted in partial fulfillment of the requirements of the degree of MSE

Binti Sepaha
CIS 895 – MSE Project
Kansas State University
# TABLE OF CONTENTS

1. Introduction ........................................................................................................................................... 3
   1.1. Motivation ........................................................................................................................................... 3
   1.2. Agenttool ........................................................................................................................................... 3
2. Project Overview ....................................................................................................................................... 4
   2.1. Introduction ........................................................................................................................................ 4
   2.2. Goal .................................................................................................................................................... 4
   2.3. Purpose .............................................................................................................................................. 5
3. Requirements Specification .................................................................................................................. 5
   3.1. Use Case 1 – Draw Diagrams ........................................................................................................... 6
       3.1.1. Description ................................................................................................................................. 6
       3.1.2. Stimulus/Response Sequence .................................................................................................... 13
       3.1.3. Associated Functional Requirements ...................................................................................... 16
   3.2. Use Case 2 – Import Xml Models ................................................................................................... 16
       3.2.1. Description ................................................................................................................................. 16
       3.2.2. Stimulus/Response Sequence .................................................................................................... 16
       3.2.3. Associated Functional Requirements ...................................................................................... 16
   3.3. Use Case 3 – Exporting Xml Models ............................................................................................... 17
       3.3.1. Description ................................................................................................................................. 17
       3.3.2. Stimulus/Response Sequence .................................................................................................... 17
       3.3.3. Associated Functional Requirements ...................................................................................... 17
   3.4. Use Case 4 – View System ............................................................................................................... 17
       3.4.1. Description ................................................................................................................................. 17
       3.4.2. Stimulus Response Sequence .................................................................................................... 17
       3.4.3. Associated Functional Requirements ...................................................................................... 17
   3.5. Use Case 5 – Printing Diagrams ....................................................................................................... 17
       3.5.1. Description ................................................................................................................................. 17
       3.5.2. Stimulus/Response Sequence .................................................................................................... 17
       3.5.3. Associated Functional Requirements ...................................................................................... 18
4. Assumptions ........................................................................................................................................... 18
5. Constraints ............................................................................................................................................ 18
6. Environment ........................................................................................................................................... 18
1. Introduction

1.1. Motivation

The motivation for this project comes from the need to develop a new version of the agentTool that is able to draw Interaction Diagrams such as Sequence, Activity and State Chart diagrams. The existing tool does not have the feature to draw activity diagrams. Also it couples the controller very tightly with the model and the view. In this version we plan to remove this tight coupling and provide the feature to be able to draw these two mentioned dynamic UML diagrams. agentTool III (Dynamic) will be an independent tool, which will be lately integrated with a similar tool being built for static diagrams.

1.2. agentTool

The agentTool allows agent system designers to formally specify the required structure and behavior of a multiagent system and semi-automatically synthesize multiagent systems that meet those requirements. The system designer formally defines high-level system behavior graphically using our Multiagent Systems Engineering methodology. The system design defines the types of agents in the system as well as the possible communications that may take place between agents. This system-level specification is then refined for each type of agent in the system. To refine an agent, the designer either selects or creates agent architecture and then provides detailed behavioral specification for each component in the agent architecture.
2. Project Overview

2.1. Introduction
The MSE Project involves the development of agentTool III (Dynamic) software that will help the users to draw dynamic diagrams. These diagrams include the sequence and activity diagrams.

Figure 1 shows at the highest level, how the agentTool will work. It is designed to generate the sequence and activity diagrams using the object model. It is supposed to save the models when the user wants to. This action generated the XML code for the model. Also, when a user imports a particular model, it should display the corresponding diagrams by parsing that XML file and loading its contents in the tool.

2.2. Goal
To develop a tool that enables the users to draw the sequence, activity and state chart diagrams. These behavioral diagrams help the user understand the flow of data and actions in an efficient way.
2.3. Purpose

To improve on the existing tool by adding more features to it and making the controller loosely coupled with the model and the view.

3. Requirements Specification

![Use-Case Model Diagram]

**Figure 2. Use-Case Model**
3.1. Use Case 1 – Draw Diagrams
3.1.1. Description
This use case describes the capability of drawing the diagrams. It includes three other use cases as follows:

3.1.1.1. SR1 (Critical Requirement) - Use Case 1.1 - Draw Sequence Diagrams
This use case describes the capability of drawing sequence diagrams. Sequence diagrams describe interactions among entities in terms of exchange of messages over time. The user of the tool should be able to draw such diagrams to represent the interactions taking place among different entities of an organization. It shows the message sequence that is shared among these entities in the order it is supposed to occur.

The ability to draw all these entities mentioned below used in a sequence diagram should be provided by the tool.

**Frame:** A frame is a rectangular box that delimits the sequence diagram. The purpose of this frame is to encapsulate all the elements used in the interaction protocol as a unit.

- **SR1.1** – Each sequence diagram will automatically have exactly one frame when created. The frame will have a default protocol name.
- **SR1.2** – The user cannot delete this frame. He can only resize the frame.

**Protocol Name:** Each frame has a notation to it that follows the `sd` keyword by a unique name. `sd` stands for sequence diagram. This notation is placed in a “snipped-corner” pentagon in the upper-left corner of the frame. It represents the protocol name. These protocols may have parameters.

- **SR1.3** – Edit the protocol name.
- **SR1.4** – add/edit parameters

![sd ProtocolName](image)

Figure 3. Frame and Protocol Name
**Alternative Actions:** This frame depicts a set of alternative actions that can be taken by roles. This looks similar to the frame but always comes inside a frame. Dashed horizontal lines will separate the alternate actions.

![Alternative Actions Diagram]

*Figure 4. Alternative Action*

**SR1.5** – Add a frame for alternative action  
**SR1.6** – Delete the frame  
**SR1.7** – Move it in the main frame  
**SR1.8** – Edit the name

**Action in loop:** This is denoted by a rectangular box similar to an alternative action box. It has a guard in the upper left corner, which determines when the loop exists.

**SR1.9** – Add a frame for loop  
**SR1.10** – Delete the frame  
**SR1.11** – Move it in the main frame  
**SR1.12** – Edit the guard

**Reference to another Protocol:** The call to another protocol is shown by a solid-outlined rectangle with a snipped-corner pentagon with the keyword ref (for reference). The content of the rectangle is the name of another interaction protocol.
SR1.13 – Add a frame for reference to another protocol
SR1.14 – Delete the frame
SR1.15 – Move it in the main frame
SR1.16 – Edit the name of the reference

**Class Roles:** Class roles describe the way an object (subsystem/actor, agent and external system roles) will behave in context. These are drawn across the top of the diagram. Notation is a rectangle containing Role Name:Class/Object Name.

SR1.17 – Add class roles to the frame
SR1.18 – Delete/edit Class Roles

**Lifelines:** Lifelines are dashed vertical lines that indicate the object's presence over time.

SR1.19 – Lifelines will be added with the class roles automatically with the class roles.

**Messages:** Messages handle the communication between objects. These are denoted by labeled horizontal arrows. It shows the direction of the flow of information. There are two types of messages.

**Synchronous messages:** Synchronous messages are messages with a filled arrowhead.

**Asynchronous messages:** Asynchronous messages are denoted by open arrowheads.
**Messages to Self:** A message can be self looped too. It can start at a lifeline and finish at the same lifeline. In this case the sender may or may not receive the message. These two conditions will be denoted separately as shown below.

```
|   |
|   |
|   |
|   |
```

A message that the sender will receive

```
|   |
|   |
|   |
|   |
```

A message that the sender will not receive

**SR1.20** – Add messages between any two lifelines  
**SR1.21** – Delete messages  
**SR1.22** – Add labels to the messages  
**SR1.23** - Message overlapping will be represented by making a bridge between the messages.

**Stop:** The stop operator is depicted by a cross in the form of an X at the bottom of a lifeline as shown below. It denotes that the specific role no longer interacts in the diagram.

```
X
```

**SR1.24** – Add Stop sign to a lifeline  
**SR1.25** – Remove stop sign from a lifeline

**Timing Constraints:** Time on sequence diagrams allows designers to represent that some messages have to be received before a certain delay. The message to which the timing constraint is applied has to be received between the lower bound and the upper bound. Lower
and upper bounds can be either natural numbers such as \{0..3\} (the message has to be received between now and 3 units of time). Relative time constraints are written as an interval as follows: \{ initial time..final time\}. The interval is rendered as a horizontal bar on the first message in the interval, a horizontal bar on the last message in the interval, a vertical line directed in both ways between the two bars and the timing constraints near the vertical directed line. Absolute time constraints respect this notation except that the interval can be a date or a time or any combination of these two elements.

**SR1.26 (Future Requirement)** – Add timing constraints between two messages

**SR1.27 (Future Requirement)** – Remove timing constraints

3.1.1.2. **SR2 (Critical Requirement) - Use Case 1.2 - Draw Activity Diagrams**

This use case describes the capability of drawing activity diagrams. An activity diagram illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. The user should be able to draw these diagrams to model the activity flow. The ability to draw all these entities mentioned below used in a sequence diagram should be provided by the tool.

**Frame:** A rectangular box inside which the activity diagram will be drawn.

**SR2.1** – Each activity diagram will automatically have exactly one frame when created.

**Swim Lanes:** These are vertical lines in the frame to depict different roles participating in the activity diagram.

**SR2.2** – Add swim lanes for different roles in an activity diagram

**SR2.3** – A swim lane should determine one role

**Initial and Final nodes:** The point of starting of an activity is called the initial state and similarly, where it stops is called its final state. The tool should provide the capability to draw these states.

**Initial Node:** A filled circle followed by an arrow represents the initial action node.
SR2.4 – Add initial node to the activity diagram.
SR2.5 – There will be no incoming action flow to the initial node.
SR2.6 – There will be only one initial node per diagram.

Final Node: An arrow pointing to a filled circle nested inside another circle represents the final action node. This is where all the processing the activity is stopped.

SR2.7 – Add final node to the activity diagram.
SR2.8 – There will be no outgoing action flow from the final node.
SR2.9 – There will be only one final node per diagram.

Action States: Action states represent the non-interruptible actions of objects. You can draw an action state using a rectangle with rounded corners.

SR2.10 – Add action states
SR2.11 – Remove action states
SR2.12 – Add names to action states
SR2.13 – Edit names of the action states.

Action Flow: Action flow arrows illustrate the relationships among action states. Activities will be connected through activity flow arrows. These arrows may have labels on them.

SR2.14 – Add action flows between 2 activities
SR2.15 – Add action flows from an activity to a synchronization node, outgoing event or to a final state.
SR2.16 – Add action flows from an incoming event to an activity or synchronization node.
SR2.17 - Add action flows from a synchronization node to an activity, final node or outgoing event.

**Synchronization Points:** These are solid bars that are used for synchronization among activities.

**SR2.18** – Add synchronization points
**SR2.19** – Delete synchronization points

**Outgoing Events:** These are outgoing events from an activity or a synchronization point from an activity diagram. They are denoted by a pentagon as shown below.

![Outgoing Event Diagram]

**SR2.20** – Add outgoing events
**SR2.21** – Delete outgoing events
**SR2.22** – Action flows cannot flow out of the outgoing events

**Incoming Events:** These are incoming events denoted by the pentagon as shown below.

![Incoming Event Diagram]

**SR2.23** – Add incoming events
**SR2.24** – Delete incoming events
**SR2.25** – Action flows cannot flow in the incoming events

**Decision Node:** A decision node is a control node that chooses between outgoing action flows. One and only one action flow is chosen. These have guards on them that let only one action flow to be chosen. The following diamond denotes it.

![Decision Node Diagram]

**SR2.26** – Add decision nodes
SR2.27 – Remove decision nodes
SR2.28 – Add guards to outgoing action flows
SR2.29 – Edit these guards

Flow Final Node: This denotes that a specific flow has stopped but all the other processing does not stop here. A cross in a circle as shown below represents it.

SR2.30 – Add flow final node
SR2.31 – Remove flow final node
SR2.32 – There will be no outgoing action flows from the flow final node.

3.1.1.3. SR3 (Future Requirement) - Use Case 1.3 - Draw State Charts
A statechart diagram shows the behavior of classes in response to external stimuli. This diagram models the dynamic flow of control from state to state within a system. The ability to be able to draw state charts will be a future requirement.

State: States represent situations during the life of an object. States are represented by rectangular boxes.

SR3.1 – Add states to the statechart diagram.
SR3.2 – Remove states
SR3.3 – Add/edit names on the states

State with Compartments: A state may be subdivided into multiple compartments separated from each other by a horizontal line.
The compartments of a state are:

**Name Compartment**: This compartment holds the (optional) name of the state, as a string.

**Internal Activities Compartment**: This compartment holds a list of internal activities or state (do) activities that are performed while the element is in the state. The activity label identifies the circumstances under which the activity specified by the activity expression will be invoked. A number of activity labels are reserved for various special purposes and, therefore, cannot be used as event names. The following are the reserved activity labels and their meaning:

- **entry**: This label identifies an activity, specified by the corresponding activity expression, which is performed upon entry to the state (entry activity).
- **exit**: This label identifies an activity, specified by the corresponding activity expression, that is performed upon exit from the state (exit activity)
- **do**: This label identifies an ongoing activity (“do activity”) that is performed as long as the modeled element is in the state or until the computation specified by the activity expression is completed (the latter may result in a completion event being generated).

**Internal Transition Compartment**: This compartment contains a list of internal transitions, where each item has the form as described for Trigger. Each event name may appear more than once per state if the guard conditions are different. The event parameters and the guard conditions are optional. If the event has parameters, they can be used in the activity expression through the current event variable.

**SR3.4** - Add compartments to states
**SR3.5** - Delete compartments from states
**SR3.6** - Add activity labels
**SR3.7** - Edit/delete activity labels

**Initial and Final nodes**: The tool should provide the capability to draw object’s initial and final states.

**Initial Node**: A filled circle followed by an arrow represents the initial action node.

**SR3.8** - Add initial node to the statechart diagram.
**SR3.9** – There will be no incoming transitions to the initial node.

**SR3.10** – There will be only one initial node per diagram.

**Final Node:** An arrow pointing to a filled circle nested inside another circle represents the final action node.

![Final Node Image]

**SR3.11** – Add final node to the statechart diagram.

**SR3.12** – There will be no outgoing transitions from the final node.

**SR3.13** – There will be only one final node per diagram.

**Transition:** A solid arrow represents the path between different states of an object. Label the transition with the event that triggered it and the action that results from it.

![Transition Image]

**SR3.14** – Add transitions from one state to another or to a synchronization point.

**SR3.15** – Remove transitions

**SR3.16** – Add labels to the transition

**SR3.17** – Edit/remove the labels

**SR3.18** – Change start and end-points of a transition.

**Synchronization Points:** These are solid bars that are used for synchronization of states.

![Synchronization Points Image]

**SR3.19** – Add synchronization points

**SR3.20** – Delete synchronization points
**Splitting of Control:** This represents a splitting of control that creates multiple states.

SR3.21 – Add splitting of control points  
SR3.22 – Delete splitting of control points

3.1.2. **Stimulus/response sequence**  
The user selects the option of being able to draw sequence/activity/state diagram. The tool displays the drawing tools necessary for the same. The user can select icons and drag and drop them in the drawing pane as per his requirements and annotate them. A description will be attached with every entity.

3.1.3. **Associated Functional Requirements**

**SR3.23 (Critical Requirement) - System Object Model Generation:** Corresponding to each user action for every diagram, the system object model will be modified by the tool in the background.

3.2.1. **SR4 (Critical Requirement) - Use Case 2 – Import XML Models**

3.2.2. **Description**  
This use case describes importing XML representations of the environment model into the tool.

3.2.3. **Stimulus/response sequence**  
There must be an existing environment model that the user needs to import. A menu item will be provided to import files. This file will be in XML format describing the organizations sequence and activity diagrams. When the user selects to import a file, he will be provided with a dialog box to help him select the file he needs. When loading, the tool will create the appropriate object model from the XML file format.

3.2.4. **Associated Functional Requirements**

3.2.4.1. **SR4.1 (Critical Requirement) - Loading Diagrams from an XML file**  
After importing the file, when the user selects to view the sequence/activity/state chart diagrams, the selected diagram is shown in its respective diagram pane.
3.3. **SR5 (Critical Requirement) - Use Case 3 – Exporting XML Models**

3.3.1. **Description**
This use case describes the feature of exporting the environment models in XML file format.

3.3.2. **Stimulus/response sequence**
A menu item will be provided that enables the user to export/save the models. On saving a model, XML code will be generated corresponding to the object model generated by the tool and saved in a file. The user will be provided with a dialog box that will let the user mention the name of the XML file and the location where he wants to save it.

3.3.3. **Associated Functional Requirements**

**SR5.1 (Critical Requirement) -** The system model will be saved to a file.

**SR5.2 (Critical Requirement) -** The file will be in an XML compatible format.

3.4. **SR6 (Critical Requirement) - Use Case 4 – View System**

3.4.1. **Description**
This use case describes the capability of the user to view a listing of the entire system diagrams and browse through each system diagram.

3.4.2. **Stimulus Response Sequence**
The user loads the system model from a previously saved model file. The tool then displays a textual hierarchical structure of each diagram saved in the system.

3.4.3. **Associated functional requirements**

**SR6.1 (Critical Requirement) -** The user will be able to click on any diagram in the hierarchy to view or edit it.

**SR6.2 (Critical Requirement) -** The user can delete diagrams or create new ones as part of the loaded system.

3.5. **SR7 (Regular Requirement) - Use Case 5 – Printing Diagrams**

3.5.1. **Description**
This use case describes the functionality of being able to print the two diagrams using the printer.

3.5.2. **Stimulus/response sequence**
An icon will be provided to the user to be able to print the diagram it has selected using the appropriate diagram tab. When the user has select sequence diagram, a corresponding print icon will be displayed for it. Similarly, it will also show for the activity diagram. When the user hits
the icon, a window will open up displaying the print options such as number of copies, etc. The other options will be determined in the future versions of the requirements specification. On selecting ‘OK’ on this window, printout will appear at the printer terminal.

3.5.3.  Associated Functional Requirements

3.5.3.1.  SR7.1 (Regular Requirement) - Printing Diagrams on paper
As mentioned above, a separate print icon will appear with every diagram that appears in its respective pane.

3.5.3.2.  SR7.2 (Regular Requirement) - Select the printer
The user should be able to select the type of the printer he wants to use for printing. A dialog box will appear for this purpose.

3.5.3.3.  SR7.3 (Regular Requirement) - Scale the diagrams to fit the paper
The diagrams will be made to fit on the paper. The user will have the option to select this in the dialog box mentioned above. By default, the diagram will be printed as it appears on the panel.

4.  Assumptions

- For the tool to work the user should have JVM 1.3.1 or later installed on his system.
- The implemented version of the tool can be easily converted into an Eclipse plug-in.

5.  Constraints

- Since the language used for coding will be Java, speed will be an issue. But this is required to make the tool platform independent.
- The tool only deals with dynamic diagrams. The static diagrams are will be developed as another project. These two tools will be integrated later on which is out of the scope of this project.
- The underlying object model for the project will be common to the static tool i.e. agentTool III (Static) and the dynamic tool i.e. agentTool III (Dynamic). The object model will have to be agreed upon by the designers of both tools.

6.  Environment

- The tool will be written in SWT and compiled in the Eclipse environment with the Jigloo plug-in as the IDE.