

Swimmy: A Framework of Multi-Agent Instruction System for Children

Koji Yokokawa
Mamezou Co., Ltd.
koji.yokokawa@riverside.skr.jp

Abstract

I developed a framework for an interactive programmed instruction system, named Swimmy, in the Squeak eToys. This system is based on a multi-agent architecture for adaptability and open-endedness. Therefore, Swimmy is flexible for complex situations in classrooms. Moreover, Swimmy aims to be fun and easy to use for children. Children interact with software agents just like interacting with pets. Children can program the agents with the aid of a visual tool just like painting.

1. Background

One of the most difficult things in the classrooms is for the teachers to give sufficient attention to all children. Therefore, a good instruction system is necessary.

Many programmed instruction systems are like slide shows. They have no extended instructions for curious children. Some systems have linear concept. They force to advance on only a fixed order. There is no way to amplify children's imaginative and investigative impulses. Others are branching at programmed timing, but the pre-ordered branches could not handle a large number of patterns of combination users and complex environment like eToys [1] world of Squeak [2].

A good programmed instruction system should provide advices suitable for user's level. The children's problems in the learning process differ from each other. Each user needs appropriate advices according to his/her level.

Usually, the user interfaces of the programmed instruction systems are very limited. Many systems force children to select one of choices or branches for next steps that sets limits to their learning. Moreover, this defeats the purpose of fun-learning programs.

2. Basic Concept

Swimmy is a framework of interactive programmed instruction system. Swimmy is based on a multi-agent

architecture for flexible instruction and adaptation for learner's level.

2.1. Flexible Architecture

A multi-agent architecture is flexible and open-ended [3]. Therefore, applications upon this framework will be adaptable for complex situations in the classrooms. An application that is built on this framework consists of software agents. In addition, an agent communicates with users, objects, and other agents in this framework by sending messages asynchronously.

An agent in Swimmy is like a building block. A block is small and simple. Nevertheless, when many blocks are put together, they can form big complex shape of anything that children imagine. This agent has very simple functions. It handles only a small part of instructions. It continuously watches behaviors of users and objects in the environment. It gives users relevant and timely suggestions. Finally many agents cooperate with each other for accomplishment of learning process of users.

2.2. Programming Easily

Many children disintegrate a mechanical-clock to figure out how it works. In doing so, it satisfies their curiosity and learn through actual experience. In the same manner, Swimmy users can directly view the state-transition diagrams of agents' program. Visualized logics of the agents will have many educational effects for children. Users can learn how to construct logical programs by states and transitions.

An agent has its own state-transition model as its own program. User can edit the model by state-transition editor. This state-transition editor is a visual programming tool for agent. This tool can display and modify an agent's state-transition model dynamically. Using state-transition editor, users can add/remove/modify states and transitions of a model of an agent.

In addition, users can assign scripted morphs as activities of agents. Therefore, users will make more

complex program by integration of simple scripted morphs.

2.3. Friendly Interface

In this framework, users can interact with agents through Morphic UI [4] by using a mouse. An agent which is usually an animal character like a small fish, a frog, or a little dog, can communicate with users.

An agent has a SwimmyMorph as a display character. SwimmyMorph is a subclass of SketchMorph. A SketchMorph is a subclass of Morph which can be handled as an ordinary Morph. Therefore, users can change the sketch of a SwimmyMorph to any character and users can interact with agents through mouse.

3. Comparison with other systems

Swimmy is a framework system which builds complex cooperation between agents and users. This helps the users to easily access the agents' complex roles. With a visual editor of Swimmy, users can build an agent which changes its behavior dynamically or based on past states.

AgentSheets [5] and StarLogo [6] are another type of multi-agent system for children. Differed from Swimmy, these systems are made especially for simulations to analyze group-behavior of large number of agents. Their languages are simple rule-based. Therefore, it is hard to make an agent to change its behavior dynamically or based on its past state changes.

4. What can be done with this framework

This framework helps users to make software-agents. Users can build a set of software-agents to help the users in their tasks or to play with interactive games etc.

Assuming that the users are not expert on Smalltalk (the computer language in Squeak), they will create educational applications on this framework (for sample application please see Figure 1).

With this framework, users can:

- create agents as a Morph
- construct programs of agents by state-transition diagrams
- construct collaborations of agents by message sending system
- assign scripted morphs as an agent's activities
- agent has multiple parallel state-transition models as roles to play
- interact with agents by mouse

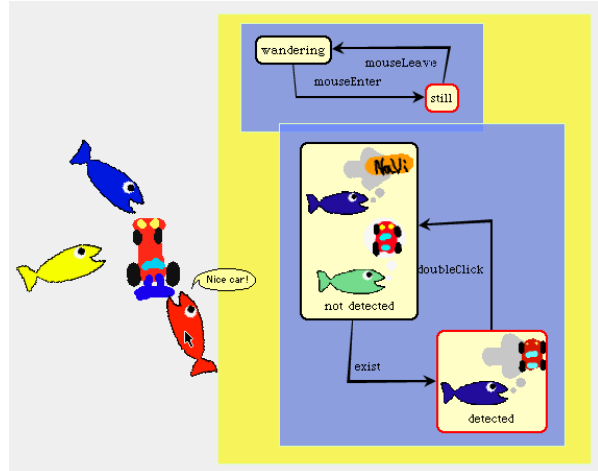


Figure 1: A sample application on Swimmy. The software agents, the fish, interact with users through mouse. Programs of these fish can be modified with a visual tool.

4.1. Design and Implementation

The base architecture of this framework is a combination of three main parts: the active object, asynchronous communication, and state-machine (please refer to Figure 2).

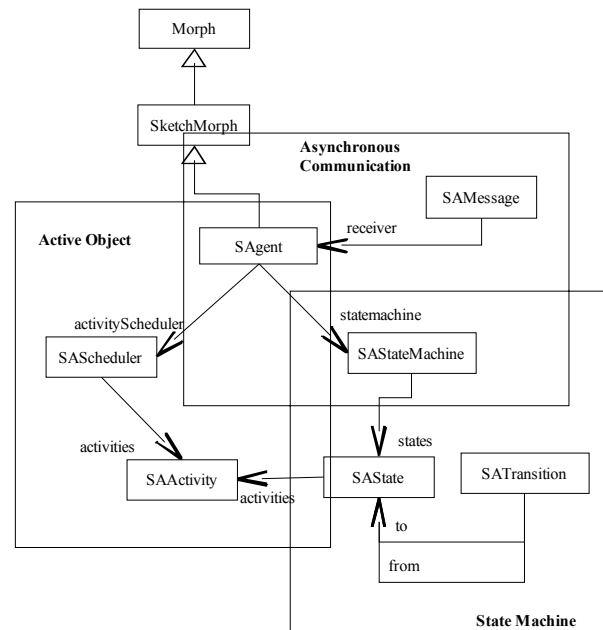


Figure 2: Class diagram of multi-agent framework

4.2. Active Object

An agent is an active object that keeps its own state. The agent's activities are executed in *act* method. In Morphic world, *act* method of an agent is executed by *step* method. The activities are scheduled by a SAScheduler of the agent. An activity can access the agent at the execution as shown in Figure 3.

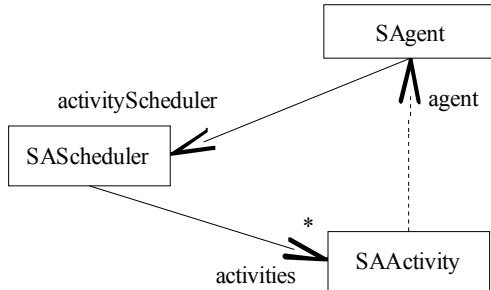


Figure 3: Class diagram of agent as an active-object

An activity is a component of a process. It has its own logic and variables and its own life cycle. An activity will be executed by an agent intermittently. However, the activity can hold its states and use own variables continuously for the next execution.

An agent schedules its activities. It will execute an activity at a call of its *act* method. An activity has its own timing to execute. The agent checks the pre-conditions of the activity. A typical pre-condition is interval-time condition. This condition checks elapsed time from the last execution.

4.3. Asynchronous Communication

Agents can interact with each other by sending messages. A user can create a message and send it to an agent through SwimmyMorphs or his own code.

In Figure 4, messages are sent and received asynchronously. When an agent receives a message, it puts the message into its own message-queue. The agent will read the message when its own *act* method is called.

Message in this framework is extendable for semantics. An agent has interpreters to read the received messages. A message has *performative*. A *performative* consists of an ontology and a word. Ontology is a scheme to interpret word. Users can define their own ontology. One ontology will only be for temporary ad-hoc communications, and another will be compatible with a standard specification like FIPA's [7]. An agent can have multiple interpreters

corresponding with ontology that will be used in the received messages.

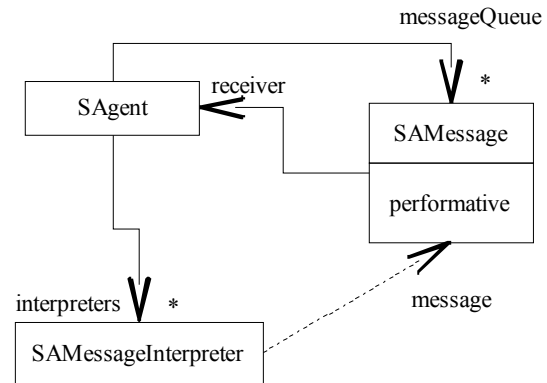


Figure 4: Class diagram of message handling

4.4. State Machine

An agent has its own state machines to execute its state-transition models. A state-transition model is a program of an agent. An agent changes its activities according to its state-transition model as specified in Figure 5.

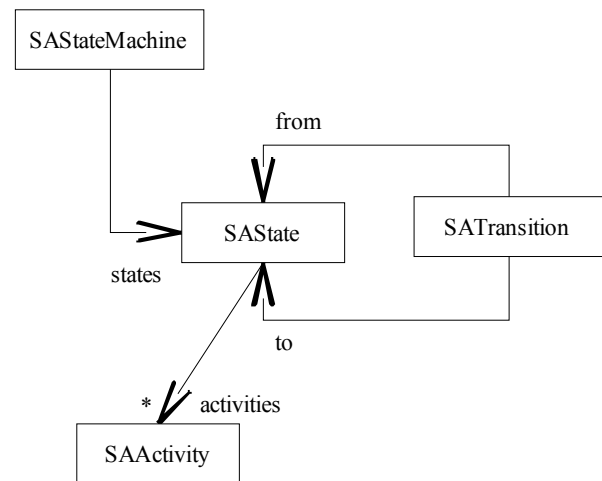


Figure 5: Class diagram of state machine

A state has a set of activities. An activity is a behavior of an agent. When the current state of an agent is changed, the agent removes all activities that is held by the previous state from its activity scheduler, and adds the set of activities of the new state into the scheduler.

Figure 6 shows that a message received by an agent will trigger a transition of the agent's current state. When a message (message1) is received by the agent in

state1, a transition takes place and changes the state to state2.

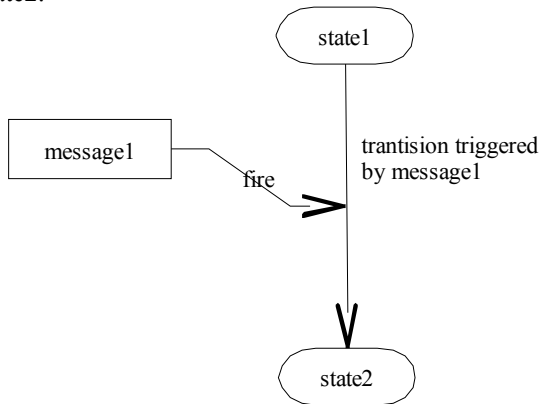


Figure 6: State-transition diagram of a typical transition

4.5. Agent Editor

Users open an agent editor morph to edit its program. An agent editor holds multiple state-transition diagrams of an agent (see Figure 7). The modification on the editor will affect the agent immediately.

A state-transition diagram displays a state-transition model for an agent. This diagram uses Connectors [8] to display and operate graphical objects.

A state-transition diagram can accept a Morph as an activity of agent. When a scripted morph is dropped on a state, the agent will perform the scripts of the dropped morph. Multiple morphs are added in a state and mixed into an agent's behavior.

A state-transition diagram is copied as ordinary Morph. Therefore, users can use a ready-made diagram or share diagrams with each other.

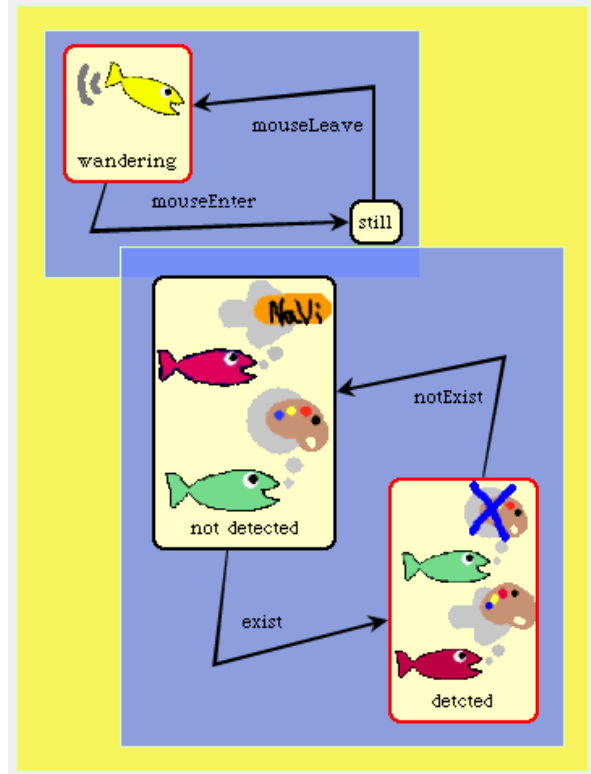


Figure 7: An agent editor which holds two state-transition diagrams

5. Demo: Tutorial supporters

This construction of a simple supporter agent for the eToys tutorial demonstrates the use of Swimmy (Figure 8, Figure 9).

The demonstration shows:

- how to make simple agents
- construction of state-transition models
- construction of custom activities

This tutorial supporter helps how to draw a picture in eToys system.

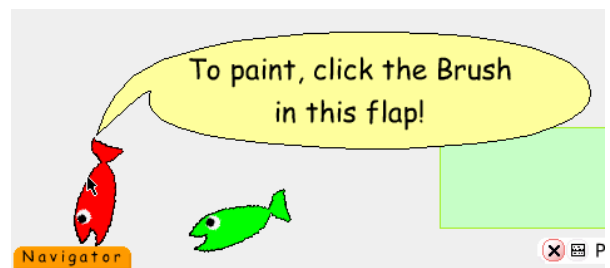


Figure 8: Scene of giving a suggestion to a learner

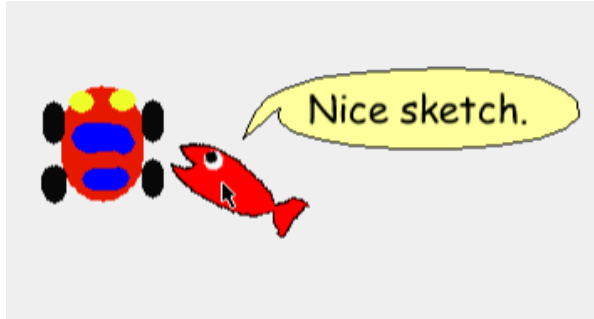


Figure 9: Scene of approving the learner

5.1. Create agents

The user creates new agents for his/her task. An agent can be created by program or by copying an existing agent.

An agent's picture can be drawn with a paint tool by the user. The agent's class is SwimmyMorph. The SwimmyMorph is a subclass of SketchMorph. This means that the SwimmyMorph is draw using the SketchMorph's paint tool. It also loads a picture form a gif file, jpeg file or from other files.

5.2. Construction of state-transition models

After creating an agent, it is necessary for the user to edit the agent's state transition mode. This model is a program of an agent which can be visually edited with a state transition diagram (Figure 10).

A user can construct state-transition model the eToys way. A trigger of a transition is assigned with a tile of eToys (Figure 11). In addition, the action of the transition is scripted as ordinary eToys script for mouseDown or mouseUp.

A user makes an ordinary scripted morph and test the script then put it on a state on the diagram (Figure 12). The scripts of the dropped morph are executed when the agent is on the state.

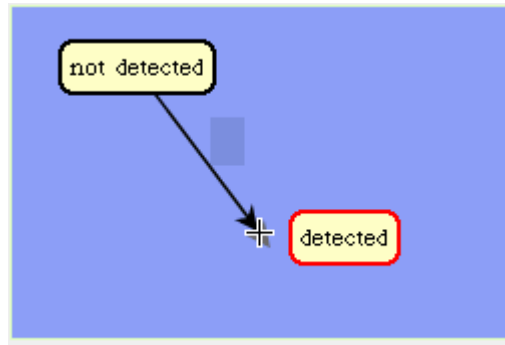


Figure 10: Construct state-transition diagram with mouse

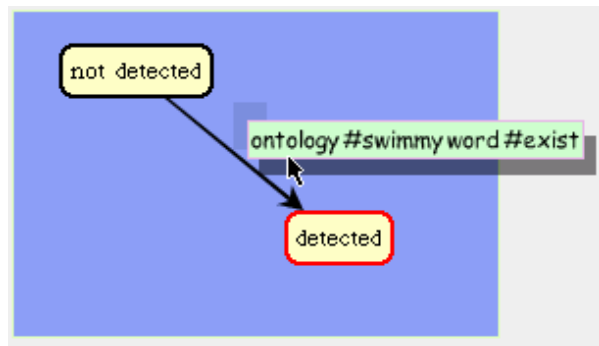


Figure 11: Assign a trigger of a transition

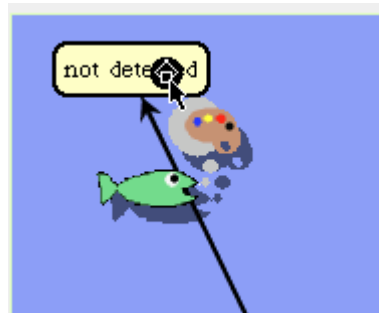


Figure 12: Dropping a scripted morph on a state

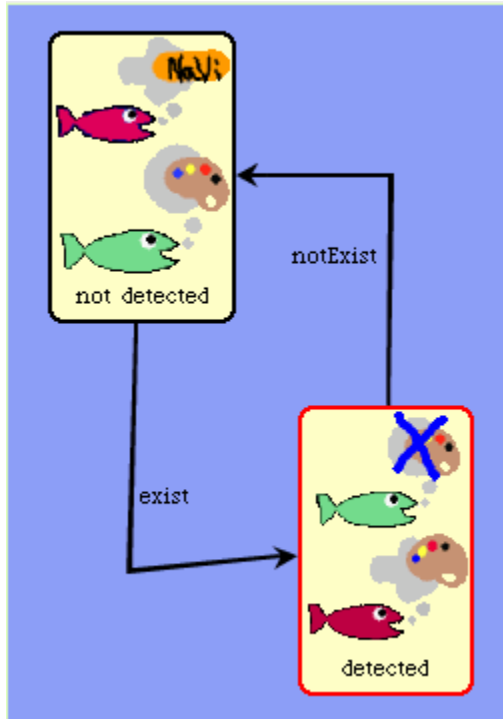


Figure 13: A diagram of painting instructor role. According to this diagram, an agent suggests user how to paint when there is no sketch in the morphic world.

When the state-transition diagram is dropped or moved to an agent editor, the agent will act according to the diagram. An agent can have multiple state-transition models as parallel state-transitions.

User can watch the current states of agents through the diagram. Triggered transitions and current state are indicated on the diagram dynamically.

5.3. Construction of custom activity

User can create agent's activity by the normal way of eToys scripting. Any scripted morph can be used as an activity. Ticked scripts will be used as activities.

User can also create a new activity class to extend the framework. To create a new activity, the user needs to make a new class as a subclass of the SAAActivity. SAAActivity is the abstract class of all activity classes.

6. Conclusion

In the project, I constructed a base framework for multi-agents system. Basic functions are implemented in this framework which enable the users to build a simple instruction system.

In building a sample application, I noticed that instructions in the balloon-help of agents would be too long when the operations are complex. To avoid this complication, activities that provide a more intuitive instructions should be added.

7. Future work

This system is open for the public under the Squeak License. I hope that many people will build instructions or variety of agent based programs on this system which will eventually be refined depending on the users' feedback.

Another programming interface into agents will be added in the future. Swimmy agents will have simple and intuitive user-interface like training a pet which makes programming an agent interesting. This means that inexperienced users like children can assign tasks to agents.

Acknowledgements

I would like to thank Alan Kay and Kim Rose for the support they have given me through out the development of this project. I am also grateful to the generosity of the Information-technology Promotion Agency Japan.

This paper is partially supported by Exploratory Software Project grant of Information-technology Promotion Agency Japan.

8. References

- [1] eToys, <http://www.squeakland.org/author/etoys.html>
- [2] Squeak, <http://www.squeak.org>
- [3] Gerhard Weiss (Editor). *Multiagent Systems*. MIT Press, 1999.
- [4] Morhic, <http://minnow.cc.gatech.edu/squeak/30>
- [5] AgentSheets, <http://agentsheets.com/>
- [6] StarLogo, <http://education.mit.edu/starlogo/>
- [7] FIPA, <http://www.fipa.org>
- [8] Connectors, <http://minnow.cc.gatech.edu/squeak/1773>