Agent Stories for Java

6.199 Advanced Undergraduate Project Report

Anthony Young-Garner

May 21, 1999

Project Advisor: Professor Glorianna Davenport

Director of Interactive Cinema

Program in Media Arts and Sciences

1 Objective

The goal of this project was to design and deliver a baseline (proof-of-concept) version of Kevin Brooks' Agent Stories system (**see Section 1.1**) on a Java/Web platform. This document assumes familiarity with the current Agent Stories system and software. The deliverables were to provide the functionality seen in the Agent Stories system running on the Metropolis platform in February of this year. The two motivations for this work were: **1)** the marginal nature of the now-discontinued Metropolis platform amongst both developers and users, which was likely to limit further development and use of the system after Brooks' graduation and **2)** that the current version of Agent Stories is hindered by limitations of the Metropolis platform, which is a powerful presentation environment rather than a true programming language.

1.1 Overview

Agent Stories for Java is a metalinear cinematic narrative tool. The theory and process of metalinear cinematic narrative is not the subject of this document, but is described completely in "Multilinear Cinematic Narrative: Theory, Process, and Tool," by Kevin Brooks.

1.1.1 Functionality

The Agent Stories for Java software allows users to create *Story Clips*, describe a *Story Framework* and define *Agent Behaviors*. The user is then able to see the result of the behaviors operating over (constraining the set) clips to produce a narrative that follows the outline of the specified story framework.

To be clear, I will explain each of these terms (note that the teminology used in the software development of Agent Stories for Java differs slightly from that used in theoretical descriptions of Agent Stories):

- A *Story Clip*, or *Narrative Fragment*, is a piece of narration, as opposed to a fully conceived narrative. Story Clips can be linked together in semantically meaningful ways. Links can be used to show that one Story Clip opposes or supports another, or that a certain clip must always be accompanied by another. Generic narrative descriptors ("speaker introduction," "conflict," "resolution") may also be used to classify Story Clips. A clip may belong to more than one *narrative class*.

- A *Story Framework* is a *narrative outline*, an ordering of narrative classes describing the overall structure and flow of a narrative.
- An *Agent Behavior* describes the criteria an Agent uses in choosing clip(s) for a particular narrative class. For example, an Agent behavior may only allow selection of clips that are in conflict with one or more clips belonging to a narrative's main point-of-view character.

Characters are implicit in Agent Stories for Java. The user does not create Characters, but the user must specify a Character for each Story Clip. In this way, statistical information (i.e., the character with the most clips) and demographic assessments (i.e., the most opposed character) can be gathered from Characters.

- The *Narrative* produced by an Agent is simply the linking together of selected Story Clips in a sensible manner. Currently, the narratives are all text-based, however the eventual goal is for Agent Stories for Java to create cinematic narratives based on the underlying textual representation.

Again, these conceptual ideas are explained fully in the Brooks thesis.

1.1.2 Usage

The Agent Stories for Java software may be run as an application on a Java Virtual Machine or as an applet on an appletviewer or web browser.

To invoke Agent Stories for Java as an applet, type the following on the command-line:

```
appletviewer as.html
```

To invoke Agent Stories for Java as an application, type the following on the command-line:

```
java AgentStories
```

To invoke Agent Stories for Java in a web browser, go to the following URL (this location may change or cease to exist without notice):

```
http://web.mit.edu/ajyoung/www/as/as.html
```

1.1.3 Graphical User Interface

Below is the Agent Stories for Java start-up screen as it appears in an application window. This functional graphical user interface provides baseline functionality, however it does not support all of the features of the underlying software. Developing a solid base of underlying software code was given greater time and attention than developing the user interface because the latter is much more naturally amenable to significant modification than the former.

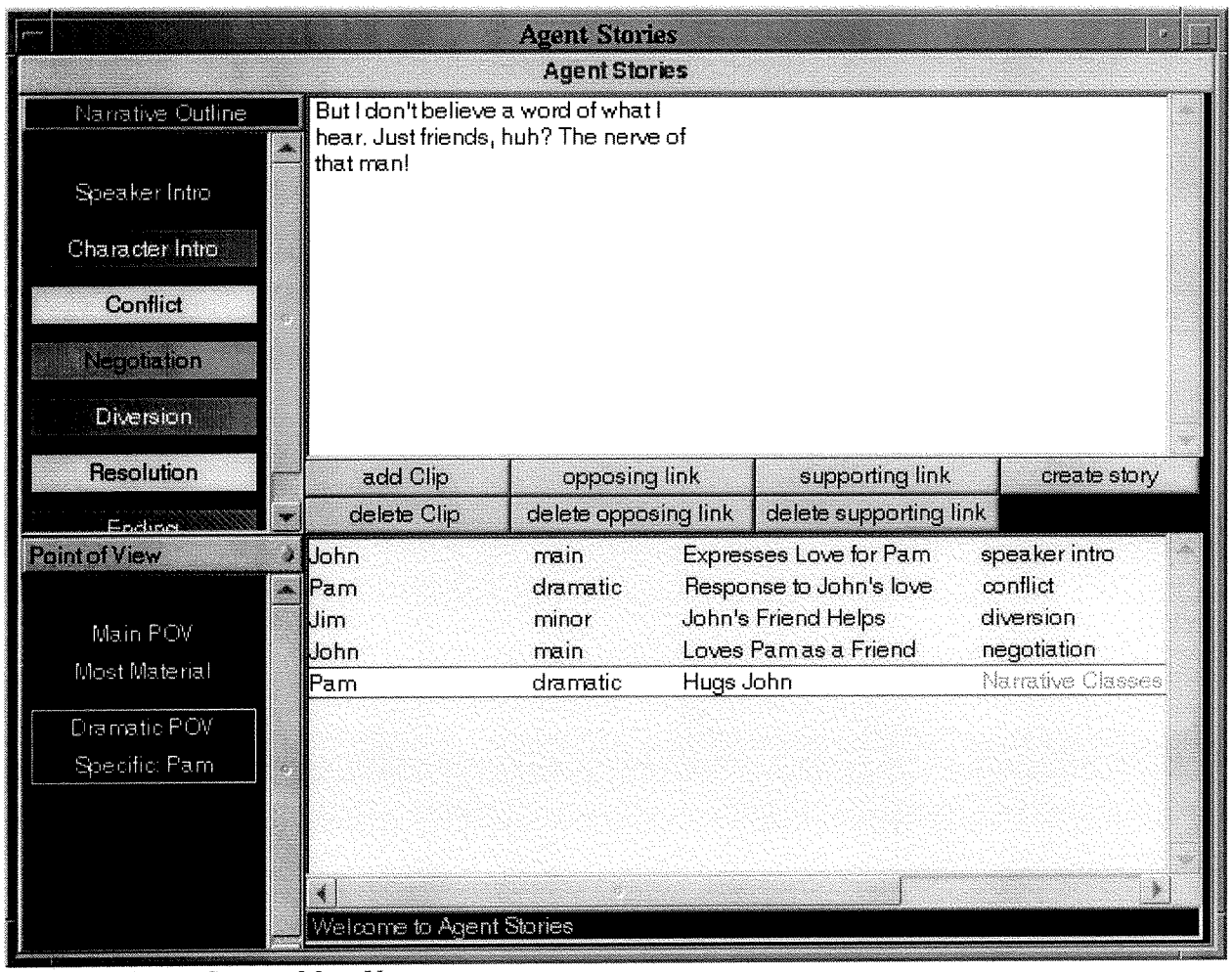

FIGURE 1 – AGENT STORIES MAIN VIEW.

- A user specifies Narrative Classes in the Narrative Outline using the Narrative Class palette (Figure 2) in the following manner:

1. The palette is made visible by clicking the Narrative Outline button.
2. Narrative Classes are dragged from the palette and dropped onto the Narrative Outline window.

3. A Narrative Class may be removed from the Narrative Outline by single-clicking the item – at which point the chosen item is highlighted – and then pressing the delete key.

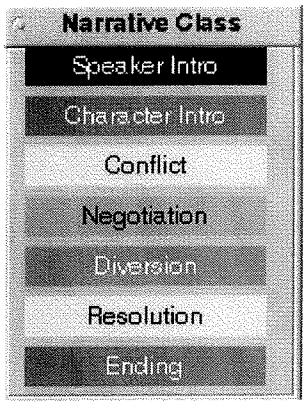

FIGURE 2 – NARRATIVE CLASS PALETTE

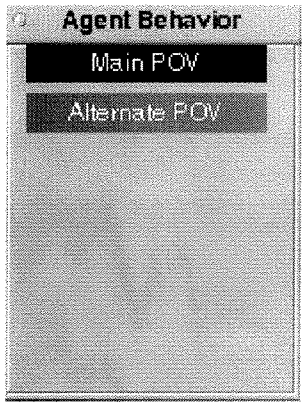

FIGURE 3 – AGENT BEHAVIOR PALETTE

- Each of the eight Agent Behaviors may be specified in the following manner:

1. The desired Agent Behavior is selected from the Agent Behaviors pull-down menu.
2. An Agent Behavior palette is made visible. Behaviors are dragged from the palette and dropped onto the Agent Behavior window.
3. Some Agent Behaviors present the user with additional dialogs to further define the behavior.

- Clips are created by clicking the "add Clip" button.
- Clips are deleted by selecting a clip and then clicking the "delete Clip" button.
- Clips are specified by double-clicking on the desired clip and editing the desired fields.
- The editable text for the selected clip is displayed in the text box just above the listing of clips.
- Clips are linked by selecting (shift-Clicking) two or more clips and then clicking the desired link ("supporting link" or "opposing link") button.
- Links are removed by selecting two or more clips and then clicking the desired delete ("delete supporting link" or "delete opposing link") link button.
- Links are represented on the screen as follows:

  When the currently selected item has links, those links appear highlighted on-screen.

Supporting links are cyan while opposing links are orange.

1.1.4 Modifications

To keep this example user interface simple, *precedes* and *must include* links (see Brooks' thesis) are not supported. In addition, only two-way links are supported. Implementing these features in the underlying software (see next section) should not prove difficult, however designing a graphical user interface which supports this functionality was beyond my ability given the time constraints of this project.

Also, the reasoning for Agent choices is not explicitly displayed as it is the in Metropolis version of the software. This choice was also made for the sake of time and simplicity. However, support for user output of Agent Reasoning is fully implemented in the underlying software so adding this feature to the user interface would no underlying code modification. Also, since Agent Behaviors are always visible, unlike in the Metropolis version of Agent Stories, the user more easily understands the context of Agent reasoning.

2 Design

The major goal in developing and implementing Agent Stories for Java was to develop a software codebase that is easy to maintain and extend. A secondary goal was developing a prototype in a very short timeframe, because my time remained committed to the existing Agent Stories project -- which was of greater importance -- through mid-April. The way in which these goals conflicted with each other was apparent in the evolution of my initial design. I can only hope that I have made the right trade-offs between extensibility, generality, robustness and simplicity, as I am still a bit too close to the project to view it objectively.

2.1 Design Overview and Rationale

The desire to develop a simple, functional and on-time product informed most of the decision-making process during this project. Performance tuning on either the algorithmic or programmatic level was not seriously considered. I leave such considerations to later developers.

2.1.1 Overall Structure of the Program

My design decomposes the Agent Stories problem into three major code blocks: the Narrative Engine, the User Interface and the Agent Engine. The Narrative Engine provides the functionality necessary for clips, links, and narrative classes to be created, specified, and parsed into narratives. The User Interface allows a user to interact with the Narrative Engine. Keeping the User Interface separate from the Narrative Engine allowed them to be tested separately and allows unrestricted development of new user interfaces. The Agent Engine package provides allows behaviors to be defined for use by the Narrative Engine. The Agent Engine package was kept separate from the Narrative Engine to reduce complexity and to allow it to be easily swapped with better performing abstractions and algorithms. Due to some problems with my development environment early on in the project, the Agent Engine and Narrative Engine are not actually in separate packages, however they are designed and implemented as if they were: interfaces were used to provide loose dependencies and abstraction barriers were strictly observed.

The Narrative Engine is composed of the following major abstractions: NarrativeClass, Character, NarrativeFragment, MediaDescriptor, MediaDescriptorInterface, NarrativeEngine, Narrative, SetConstrainerInterface, AgentStories.

The User Interface consists of the following major abstractions: AgentView, FrameworkView, ClipView, ClipItem, Controller.

The Agent Engine consists of the following major abstractions: Behavior, POVBehavior, NarrativeClassBehavior, Agent.

2.1.2 Abstractions

Narrative Engine

- NarrativeClass is an abstraction that represents a narrative class. The abstraction is hard-coded to support these narrative classes: "speaker intro", "character intro", "conflict", "diversion", "negotiation", "resolution", and "ending". A NarrativeClass can store the NarrativeFragments that belong to it. Also, the abstraction supports two-way links between NarrativeClasses (so that, for example, particular Conflicts and Resolutions are linked), however this functionality is not used by the current user interface.

- Character is an abstraction representing the speaker or point-of-view character of a narrative fragment. Characters keep track of various demographic data about themselves, such as how many clips they currently own, how many of other characters' clips oppose their own, etc.
- NarrativeFragment is a data structure that stores the title, narrative class, writer feedback text, presentational media, links, and point-of-view character for a narrative fragment. It also knows whether or not it has been selected for use in a Narrative and, if so, stores the reason why it was chosen.
- MediaDescriptor is an abstraction that stores media content. The current Agent Stories systems only supports Strings, but in the future, sound and video clips may also be supported. This abstraction is preparation for that future.
- MediaDescriptorInterface is an interface that exists so that the user interface is not dependent on the specific MediaDescriptor used by the Narrative Engine.
- NarrativeEngine is an abstraction which provides the high level functionality necessary to manage NarrativeFragments, NarrativeClasses, and Characters; for the creation of and interaction with Agents; and to generate complete Narratives based on its current state.
- Narrative is an abstraction that operates on a NarrativeOutline's NarrativeClasses to return an enumeration of NarrativeFragmentSets for each NarrativeClass
- SetConstrainerInterface is an interface that exists so that the NarrativeEngine is not dependent on any particular Agent representation and so that Agent representations are not unduly constrained.
- AgentStories is the top-level application class.

User Interface

- AgentView is a Netscape IFC component (see http://developer.netscape.com) that allows the user to create and destroy agent behaviors.
- FrameworkView is a Netscape IFC component that allows the user to create and destroy narrative classes.
- ClipView is a Netscape IFC component (based on AddressView from the example AddressBook applet at the ifc section of developer.netscape.com) that allows the user to create, modify, link and destroy narrative fragments.

- ClipItem is a Netscape IFC component (based on AddressItem from the example AddressBook applet at the ifc section of developer.netscape.com) that represents a narrative fragment in the user interface.
- Controller provides the functionality necessary for the user interface to interact with the Narrative Engine.

Agent Engine

- Behavior is an abstraction that constrains a set of Objects, that is homogeneous in at least one dimension, in accordance with a composite rule formed from a finite set of rule primitives and boolean operators.
- POVBehavior is a Behavior that constrains a set of Characters in accordance with a finite set of point-of-view rules and boolean operators.
- CharacterBehavior is a Behavior that constrains a set of NarrativeFragments to include only those that are/aren't owned by specified character(s)
- NarrativeBehavior is a Behavior that constrains a set of NarrativeFragments to include only those which are in a specified NarrativeClass.
- NarrativeClassBehavior is a Behavior that constrains a set of NarrativeFragments in accordance with a finite set of Narrative Class rules and boolean operators.
- PassThroughBehavior is a Behavior that minimally constrains a set of NarrativeFragments.
- Agent is an abstraction representing a collection of Behaviors that constrain sets.

3 Schedule

On-time completion of the Agent Stories for Java project is essential to the goal of developing a solid codebase since I am graduating this June and will not be around to continue or maintain the project. This is why time played an unusually strong role in my design and implementation choices throughout the project.

3.1 Task Split

Both project management and coding tasks were carefully considered before beginning the project. Since this was a single-person project with a very short timeframe, I had to make sure that I wouldn't find myself in any time sinks (with the expected exception of continuing work on the current Agent Stories project).

3.1.1 Project Management

I allowed myself a great deal of time for the code model (especially initial design) and documentation of the Narrative Engine and Agent Engine at the expense of validation and user interface development. This trade-off is the most preferable in developing a solid codebase that others can work on even though it make demonstrations and presentations more difficult in the short-term.

3.1.2 Coding

Though the design splits the code three ways, there were only two major coding tasks: the Engines (Agent and Narrative) and the User Interface. As mentioned in the previous section, I gave the Engines foremost importance. More than sixty percent of my time was devoted to coding the Engines.

3.2 Milestone Misses

My original completion date for the new Agent Stories for Java project was May 1. However, I was unexpectedly required to double the number of hours I was working on the current Agent Stories project during two weeks in April when I had expected to be fully concentrating on the new project. Therefore, this milestone was thrown out in early April and replaced with a less comforting "get it done before graduation" milestone.

3.3 Schedule

Completion Design, implementation, and documentation were complete by May 21, 1999 and a demo is tentatively scheduled for May 27, 1999.

4 Implementation Notes

4.1 Development Environment

4.1.1 Programming Language

I developed using the standard Java programming language in order to meet the goal of developing a web-enabled application. The CodeWarrior Integrated Development Environment was used since it is available on both PCs

(which I use at home) and Macs (which I use in the lab). For the user interface package, I chose to use Netscape's Internet Foundation Classes (IFC) rather than the AWT because the IFC toolkit provide a much more powerful range of tools, including built-in drag and drop support. The increase in deliverable size due to having to include the IFC with the project must be weighed against the possibility that completing the user interface in the given time with the AWT may have been beyond my abilities.

5 Future Work

There are several areas open for continued work on the Agent Stories for Java project. Support for one-way links in the user interface, as well as the previously mentioned *precedes* and *must include* links, would allow for a more flexible and robust product. Perhaps an even more important feature that is currently missing from Agent Stories for Java is the ability to load and save NarrativeFragments, NarrativeOutlines and Narratives. The parsing and connectivity (local disk and/or database) code required to implement these input/output features could probably be written and debugged by a UROP (Undergraduate Researcher) familiar with the project in less than 40 hours. Finally, development of a full-featured and robust user interface would allow Agent Stories for Java to become a continually operating research project operating on the Interactive Cinema web site.