A Calligraphic Interface for Managing Agents

Alfredo Ferreira
alfredo.ferreira@inesc-id.pt

Marco Vala
marco.vala@tagus.ist.utl.pt

J. A. Madeiras Pereira
jap@inesc-id.pt

Joaquim A. Jorge
jorgej@acm.org

Ana Paiva
ana.paiva@inesc-id.pt

Department of Information Systems and Computer Engineering
INESC-ID / IST / Technical University of Lisbon
R.Alves Redol, 9, 1000-029 Lisboa, Portugal

ABSTRACT

Despite the considerable work on agent frameworks, user interfaces to manage these are mostly script based. Even though some solutions provide graphical interfaces to build agent worlds these are quite limited and overly dependent on textual input. Recently, calligraphic systems using sketch-based and pen-input have emerged as a viable alternative to conventional direct-manipulation interfaces in a wide range of areas, such as user interface design or mechanical systems simulation. In this paper, we present a preliminary approach to a calligraphic interface for managing agents. It recognizes gestures drawn by users allowing them to create and manage agent worlds flexibly and efficiently using a concise language.

Keywords: Calligraphic Interfaces, Sketch Based Modeling, Agent Modeling Tools, Agent Frameworks

1 INTRODUCTION

Many definitions have been proposed to describe "software agents" or simply "agents". Russell and Norvig [RN03] define agent as "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors". Maes [Mae95] adds that "autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed". And we could add several other agent definitions that focus on particular domains.

Agents are used in a large set of areas with a large set of purposes. They are often seen as a programming methodology and, in that sense, contribute to the appearance of several agent frameworks. But, as these frameworks are becoming increasingly widespread, they lack tools to efficiently create and manage agents.

Recently, some agent frameworks have been including graphical tools to aid in the creation of agent-based applications. These tools are, however, quite limited and dependent on script-based languages and/or textual input. Moreover, they are not really user friendly since they focus more on the agents of the world being created than on the user's task of creating the world.

Thus, we try to take advantage of the way developers sketch agent-based worlds in sheets of paper before creating them, and we propose a novel approach based on a calligraphic interface. The tool we present in this paper, called `editION`, allows users to sketch the agent-based world directly on the computer, reducing scripting and textual input.

After a short discussion of related work, we will present an overview of our approach to sketch-based management of agents, describing the proposed architecture. Then we focus on the creation of agent-based worlds using sketches, presenting our symbol recognition methodology and describing the user interaction with our tool. Finally we present some conclusions and suggest directions for future work.

2 RELATED WORK

Usually agent frameworks do not offer native tools to create agent-based applications. Among positive exceptions are ZEUS Agent Building Toolkit [NNLC99] and AgentFactory [Col01] which have tools to generate starting scripts for creating agents (see Figure 1). Agent Academy [MKSA03] has a tool to parameterize and launch agent-based applications using previously defined agent types. Agent Society Configuration Manager and Launcher (ASCML) [BPB+05] is a tool for the Java Agent Development Framework (JADE) which facilitates the configuration and deployment of agent so-

SHORT COMMUNICATION proceedings
ISBN 80-86943-05-4
WSCG'2006, January 30 – February 3, 2006
Plzen, Czech Republic.

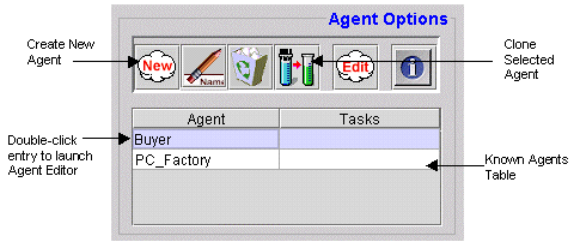

Figure 1: Zeus agent building toolkit

cieties. NetLogo [Wil99] is a programmable modeling environment, featuring hundreds of independent agents, where modelers can give instructions using text-based input (see figure 2).

But all these tools lack some interactivity. Most use a script language and a command interpreter that parses and executes scripts. Even the solutions which provide graphical interfaces are quite limited and dependent on textual input. Moreover, the mouse based interaction in these graphical interfaces mostly relies on drag-and-drop and menu navigation techniques, taking no advantage of the latest[1] user interaction techniques, such as calligraphic interfaces.

Although no calligraphic interfaces had been devised for agent frameworks, several experimental sketch-based systems were developed in recent years for a number of different areas, such as interface design, mechanical systems simulation or control systems analysis. SILK [LM01] is an interactive tool to sketch interfaces using an electronic pad and stylus. Designers can use SILK to quickly sketch the user interface and, when they are satisfied with the early prototype, produce a complete and operational interface. A similar tool, JavaSketchIt, was presented by Caetano *et al.* [CGFJ02] and generates a Java interface based on hand-drawn compositions of simple geometric shapes. The JavaSketchIt evaluation concluded that users consider their sketch-based system more comfortable, natural and intuitive to use than traditional mouse-based tools.

Alvarado and Davies [AD01] developed ASSIST, a program that produces simple 2D mechanical devices from hand-drawn sketches. This system performs real-time interpretation, as the sketch is being created, using

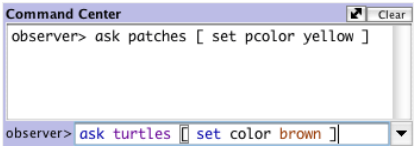

Figure 2: Netlogo command center.

a set of heuristics to construct a recognition graph containing the likely interpretation of the sketch and selects the best one based on both contextual information and user feedback.

Hong and Landay [HL00] developed a Java toolkit to support the creation of pen-based applications. Using this framework, Lin *et al.* [LNHL00] created DENIM, a sketch-based system that helps web site designers in the early stages of the design process. SketchySPICE [HL00] is another program developed using SATIN. It consists in a calligraphic interface for SPICE[2] where users can draw simple circuits gates in two distinct modes. In *immediate* mode recognized sketches are immediately replaced by its formal symbol, while in *deferred* mode recognized objects are left sketchy but the recognized symbols are drawn translucently behind the sketch, in order to give some feedback to users.

More recently, Kara and Stahovich [KS04] presented the SIM-U-SKETCH, an experimental sketch-based interface for Matlab's Simulink software package[3]. With this tool users can sketch functional Simulink models and interact with them, modifying existing objects or add new ones. SIM-U-SKETCH was designed to allow users to draw as they would do on paper, with no constraints imposed by the recognition engine. To that end, this system employs a *recognize on demand* strategy in which the users have to explicitly indicate whenever they want the sketch to be interpreted.

Despite their apparent similarity, distinct approaches and strategies are used in the systems referred above. We studied the advantages and drawbacks of these methodologies and used them to devise a novel calligraphic interface to create agent-based worlds in the context of an agent framework.

3 OVERVIEW

This paper introduces `editION`, a tool to create agent-based worlds on top of the `ION` agent framework. Users sketch world elements or commands in order to create and control the agent-based world. Since `editION` works closely together with the `ION` framework, it will be able to provide immediate visual feedback to users of what is happening in the world.

The architecture of the proposed solution can be divided in two distinct parts: the `ION` framework, which handles the agent world, and the `editION` tool, that provides management capabilities to users. In figure 3 we depict a block diagram of such architecture.

[1] Some of these techniques are not quite novel, but only recently have been adopted by the mainstream manufacturers. An example of this is the pen-based interaction, proposed by Ivan Sutherland [Sut64] almost forty years before being available to the general public.

[2] SPICE is a circuit CAD tool developed at University of California at Berkeley

[3] Simulink is an add-on package for analyzing feedback control system and other similar dynamic systems.

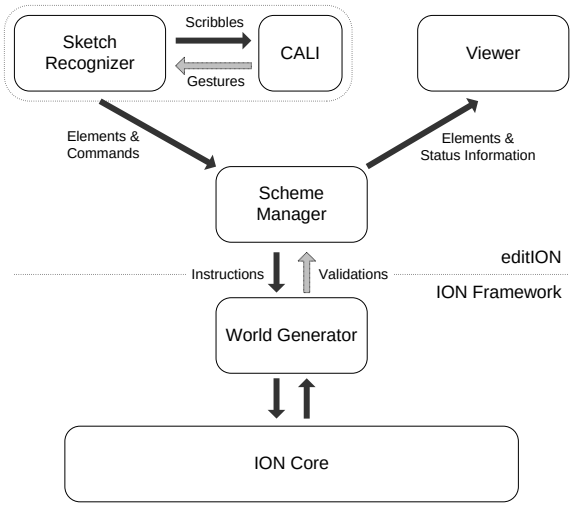

Figure 3: Architecture

3.1 The editION tool

The editION tool includes a calligraphic interface where users sketch the agent world and visual feedback is given. Unlike SketchySPICE, where users can select in which mode they draw, only the *immediate* mode makes sense in editION. Therefore, the sketch is interpreted and validated as it is being drawn, allowing on-the-fly creation instead of having to draw the entire world prior to its creation.

Scribbles drawn by users are processed by the Sketch Recognizer module, which uses CALI [FPJ02] library to recognize them as shapes or commands. Sketch Recognizer then identifies if they are an element of the world or a command, sending the corresponding information to the Scheme Manager module.

The Scheme Manager module can be considered as the core of editION. It handles the diagram representing the agent world, performing syntactic and basic semantic validation. To that end, it applies a set of predefined grammatical rules to each change to guarantee the correctness of the scheme. The scheme is stored as a directed graph, in which the nodes represent the elements and the edges represent the connectors. This way, common graph manipulation techniques could be used to manage the scheme and navigate through it.

The Viewer encapsulates the output details of our approach. It is responsible for providing visual feedback to the user, by selecting the graphical shapes displayed for each element and its properties according to the current state. Based on information received from the scheme manager, the viewer determines which shape must be drawn, its position and color. Moreover, it manages the way messages are shown to users and how long they remain in the screen.

3.2 The ION framework

The ION framework [AV05] is yet another agent framework. However, unlike most agent frameworks, which mainly look at agents that will enrich pre-existent virtual or real environments, the ION framework is also concerned with the creation and the simulation of the environment itself[4]. For the purpose of this paper we will only briefly describe the World Generator and the representation model within the ION core.

The World Generator is a bridging layer. It receives instructions from editION and tries to execute these instructions in the ION core. Depending on the outcome, it also sends feedback which might be useful for further semantic validation.

The ION core manages the world model which is populated by several entities. Entities have properties that store relevant information about the entity, actions to access and modify the environment, and relations with other entities. These relations also have properties that keep information about the role played by the entity in the relation.

The previous representation model allows us to create and simulate different worlds. Imagine, as an example, a small world with a blue object, a red object and a dog that likes red objects and grabs them all the time. Using the ION core, the objects are represented by entities with a single property, its color. The dog is an agent, represented by an entity, which has two actions: look for objects, and grab objects. These actions would be the agent's sensors and actuators. The dog will sense the environment for red objects (using the "look for objects" action) and will act in the environment grabbing the red object (using the "grab objects" action). The dog could even remember the objects that were grabbed before, if we create a relation between the dog and those objects.

4 SKETCHING AGENT-BASED WORLDS

Developers often start by drawing agent-based worlds on paper. Then, they generally use script-based tools to specify the world in the framework. Even when these tools have graphical interfaces with mouse interaction, they are greatly dependent on textual commands. Following the recent developments in pen and sketch-based interfaces, we propose an alternative to standard mouse-based tools to create agent-based worlds.

The interaction with editION is usually made through pen-based hardware such as a TabletPC or a digitizing tablet. After capturing the scribbles drawn by the user, these must be recognized, interpreted and

[4] An example application of the ION framework can be found in [Pra05].

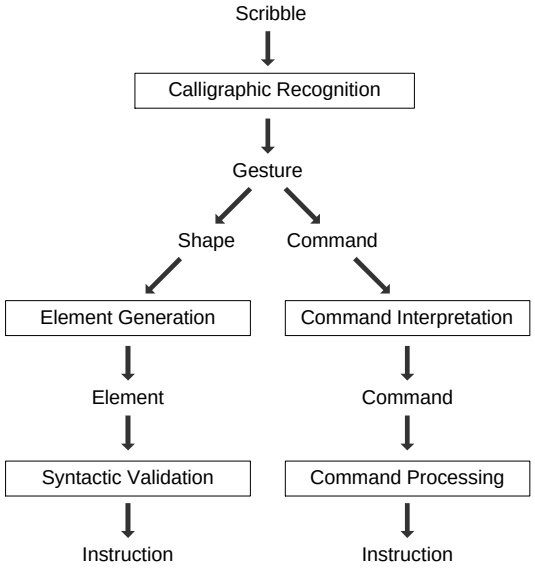

Figure 4: Recognition strategy

validated to become useful. Thus, the whole process of handling the user input and producing corresponding output both to the user and to the framework, plays a major role in our calligraphic tool.

4.1 Symbol Recognition

In order to provide on-line recognition of sketches, our approach processes the scribbles individually and not the entire sketch on demand, as performed by SILK and SIM-U-SKETCH. These scribbles are clusters of strokes drawn by the user which are submitted to a recognition process when the user's pauses are longer than a given time between strokes.

We use a multi-level recognition and parsing strategy, outlined in Figure 4, in order to convert scribbles drawn by users into instructions for `ION` World Generator. This strategy is divided in several steps, detailed below.

We start by performing the calligraphic recognition of submitted scribbles. To that end, we use CALI, a fast, simple and compact scribble recognizer used in JavaSketchIt. CALI identifies shapes of different sizes and rotated at arbitrary angles, drawn with dashed, continuous strokes or overlapping lines. It detects not only the most common shapes in drawing such as triangles, lines, rectangles, circles, diamonds and ellipses, using multiple strokes, but also other useful shapes such as arrows, crossing lines or wavy lines, as depicted in figure 5.

For this work we only need to detect a subset of the CALI gestures to specify elements and commands. These gestures are depicted in Figure 6. The scribbles to represent the elements were selected based on its visual similarity with the hand-drawn elements, usually sketched by developers when schematically representing their agent worlds.

However, since CALI is size and rotation independent, we need to carry out additional computation to determine scribble orientation and size. This calligraphic recognition step yields two categories of gestures: shape gestures and command gestures. Therefore, after identifying the category to which the detected gesture belongs, we apply distinct parsing paths for shapes and commands.

To perform gesture identification we apply the grammar presented in Figure 7. This set of simple rules provides an efficient manner not only to determine if the scribble is a command or a shape, but also to identify the command or element corresponding to a given scribble.

We consider the application of the grammar mentioned above to be the gesture identification process. This process implements the first levels of the recognition strategy, which is the transformation of scribbles into elements or commands. In this process, rectangles are transformed into entities or actions, depending on their geometric properties, triangles into relationships, circles into properties and lines into connectors. On the other hand, pre-specified gestures are transformed into "delete", "select" and "copy" commands.

When a scribble is identified as an element, it goes through validation. To that end, context information is used to verify if such element makes sense in the current scheme. In the case of a connector, such information is also valuable to determine if it is a simple connector or a role. If the generated element passes the syntactic validation, the corresponding instruction is created and sent to the World Generator. Semantic validation is

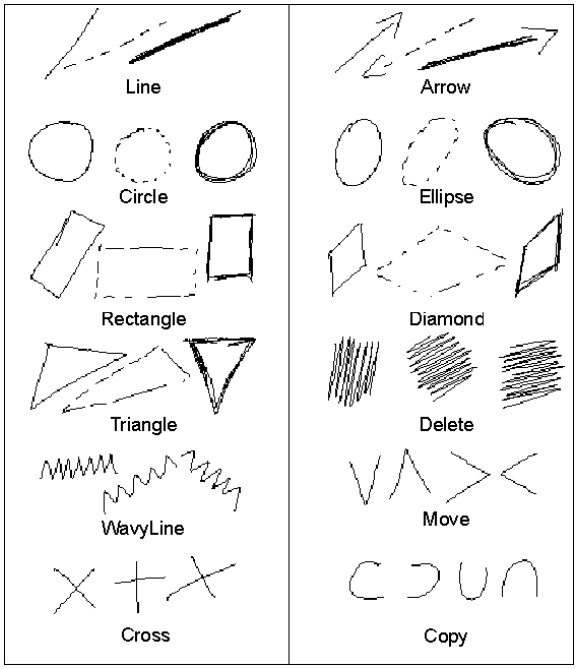

Figure 5: Gestures detected by CALI

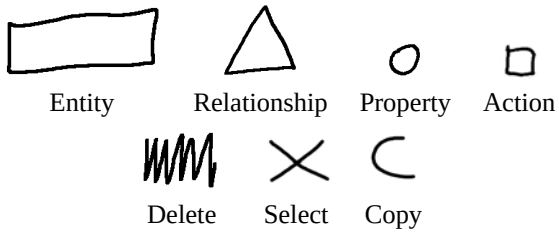

Figure 6: Gestures for elements and commands.

performed by the World Generator and, if the instruction is valid, the ION framework is updated. In any case, the World Generator gives proper feedback to the Scheme Manager. Finally, this information is used to provide visual feedback to the user, replacing the sketch by the corresponding element on the screen or showing an meaningful error message.

If a scribble is identified as a command, the context is analyzed to verify its validity. It uses information extracted from context to produce an instruction to send to the World Generator. As for the elements, the world generator processes the instruction and provides information that will be used to give visual feedback to the user.

4.2 Interaction with editION

Currently, users sketch their agent world in sheets of paper before coding it into the framework. In editION we take advantage of the users' ability to draw agent worlds with a pen to automate the boring and time consuming task of writing unnecessary lines of code. Therefore, users can sketch the world in the editION calligraphic interface using a pen-based digitizer and it will be automatically created in the agent framework.

Since we perform on-the-fly gesture recognition, the sketch is interpreted and validated as it is being drawn.

GESTURE-IDENTIFICATION-GRAMMAR(S)::=
valid_gesture $\rightarrow$ shape | command
shape $\rightarrow$ entity | action | relationship | property | connector
command $\rightarrow$ delete | select | copy
entity $\rightarrow$ $Gesture(S, RECTANGLE)$ &
$SizeWithin(S, \tau_{max^E}, \tau_{min^E})$ & $AspectRatio(S, 4, 3)$
action $\rightarrow$ $Gesture(S, RECTANGLE)$ &
$SizeUnder(S, \tau_A)$ & $AspectRatio(S, 1, 1)$
relationship $\rightarrow$ $Gesture(S, TRIANGLE)$ &
$SizeWithin(S, \tau_{max^R}, \tau_{min^R})$
property $\rightarrow$ $Gesture(S, CIRCLE)$ & $SizeUnder(S, \tau_P)$
connector $\rightarrow$ $Gesture(S, LINE)$
delete $\rightarrow$ $Gesture(S, DELETE)$
select $\rightarrow$ $Gesture(S, CROSS)$
copy $\rightarrow$ $Gesture(S, COPY)$
$Gesture(sc, t) \rightarrow$ Scribble sc recognized by CALI as t
$SizeWithin(sc, t_u, t_l) \rightarrow$ Size of sc is within t_u and t_l
$SizeUnder(sc, t) \rightarrow$ Size of sc is below t
$AspectRatio(sc, w, h) \rightarrow$ Aspect ratio of $sc \simeq w{:}h$

Figure 7: Grammar for gesture identification.

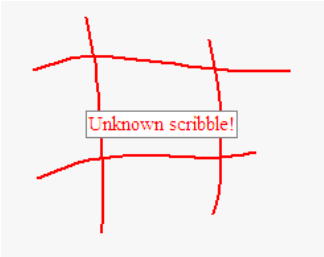

Figure 8: Example of an unrecognized scribble.

Moreover, the on-line connection with the framework allows editION to provide immediate feedback to the user from the agent framework. To that end, syntactic and semantic verifications of the sketches are performed while the world is being constructed. Thus, it is no longer necessary to design the complete world to check if any errors exist, as it usually happens in other tools.

Thus, to create an agent world with editION the user sketches each element at a time using single or multi-stroke scribbles. The scribble is immediately interpreted by the sketch recognizer. When it is interpreted as a valid element, the corresponding formal symbol replaces the sketch.

Besides elements, the user can also sketch commands. These are also interpreted by the recognizer and, if they are valid, the corresponding action immediately takes place and the drawing area is updated accordingly.

If the scribble is not recognized, it is marked in a different color and a text message informs the user of such situation. Figure 8 depicts an example of an unrecognized scribble, showing the feedback given to the user. This information remains on the screen for a couple of minutes or until the user restarts sketching. Then, both the message and the unknown scribble are deleted.

Similarly, if the user sketches a line, recognized as a connector, but one or both of its endpoints are not over an element, it is drawn in a distinct color with the corresponding message. The same happens if the sketched connection is invalid. Invalid connections occur when the connector extremities are over entities that cannot be connected, for instance if the user is trying to connect two entities or an action to a property, as illustrated in Figure 9.

Besides the unrecognized and invalid scribbles, some elements have no meaning unless they are connected with others. The property and action elements depend

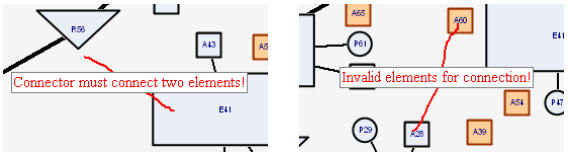

Figure 9: Two examples of invalid strokes.

on an entity and the relationship must be connected to, at least, two entities. In these cases, we consider that the recognized element is incomplete. The editION keeps incomplete elements in the drawing area, but they are represented in a different color.

Figure 10 depicts an agent world being created using the editION while a new entity is being sketched and its recognition is underway. In this example, some elements have already been recognized and validated. However, the relationship is incomplete, since it needs to be connected to, at least, two entities. Likewise, one action and one property remain unconnected, thus incomplete.

Incomplete elements are displayed in a distinct color until the user corrects this situation. While these elements are incomplete they are not considered besides Scheme Manager. This means that no information about them is sent to the World Manager. Thus, their existence is ignored by the framework.

To create agent-based worlds much more information is needed in order to make it fully functional. Specification of the behavior and the state of each element is an important part of the definition of agents. Thus, editION provides an efficient way to edit all the details of each element of the world. A simple click over a recognized entity allows the user to access such details through a pop-up window.

An example of a detail pop-up window is depicted in Figure 11. In this case, the user is changing the details of an action, more specifically, changing the code associated with an event of that action. This kind of changing is submitted to the World Generator, which performs syntactic and, when possible, semantic validation and provides proper feedback to the user.

Many other details can be changed in all elements using the pop-up window, but any change made here is

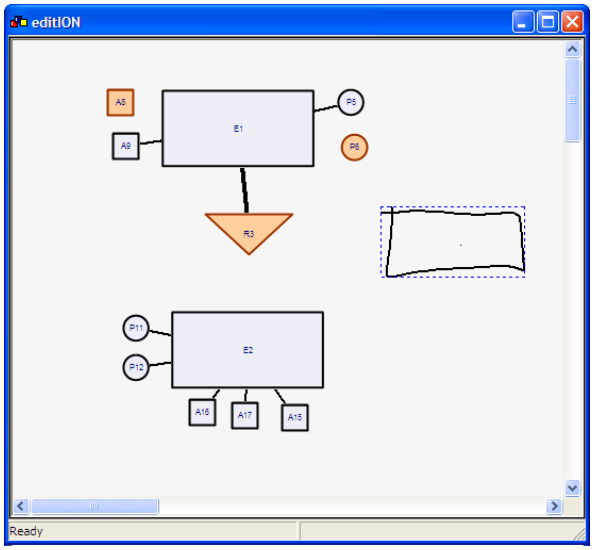

Figure 10: Creating an agent world with editION

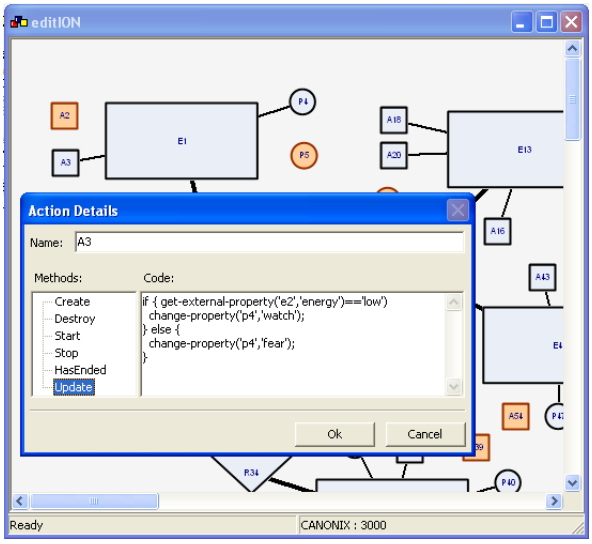

Figure 11: Changing details of an action

always submitted to validation by the Scheme Manager and the World Manager, depending on the type of modification. For instance, when changing an element's name, the name is validated by the Scheme Manager to make sure it does not conflict with other names and then it is validated by the World Manager to check if it is acceptable in the current world.

5 CONCLUSIONS AND FUTURE WORK

The proposed calligraphic interface represents an alternative to current agent management tools. Instead of writing numerous lines of code or dragging and dropping elements from toolbars and menus, with editION we bring agent developers closer to traditional paper-and-pencil methods when creating agent worlds.

Since the sketch is interpreted and validated as it is being drawn, the user receives immediate feedback from the agent framework. Therefore, it is no longer necessary to create the complete world to check for errors or incoherence. Most importantly, with this tool, the user avoids writing sometimes long and complex scripts to describe the agent world. It can be simply done by sketching it.

Despite the fact that the proposed tool was devised for management of the ION framework, we intend to make it as general as possible in order to allow it to work with other agent frameworks in the future without major changes. To that end, we do not embed the manager in the framework. Instead, we consider editION as an independent module that communicates with the framework using a small set of pre-defined instructions. The adopted visual language and identification grammar are, however, best suited for ION framework.

The presented version of the editION is still under development and offers limited functionality. Basically

it allows the user to create, change and delete world elements. But we feel it has potential to grow into a complete and powerful management tool for agent frameworks, while retaining most of its simplicity. To that end, in a near future we plan to add, among other functionalities, debugging capabilities to `editION`, offering total control over the agent world and providing continuous visual feedback on the world status.

When both `editION` and `ION` framework are fully functional we intend to perform users' evaluation involving developers and researchers from the agents area. In these tests we expect not only to validate the proposed methodology for agent world design, but also to collect information in order to refine our approach, according to the users' needs.

ACKNOWLEDGEMENTS

Alfredo Ferreira was supported in part by the Portuguese Foundation for Science and Technology, grant reference SFRH/BD/17705/2004.

REFERENCES

[AD01] Christine Alvarado and Randall Davis. Resolving ambiguities to create a natural sketch based interface. In *Proceedings of IJCAI-2001*, August 2001.

[AV05] Ruth Aylett and Marco Vala. Victec deliverable 3.5.1: Toolkit final version, 2005.

[BPB+05] Lars Braubach, Alexander Pokahr, Dirk Bade, Karl-Heinz Krempels, and Winfried Lamersdorf. Deployment of distributed multi-agent systems. In Franco Zambonelli Marie-Pierre Gleizes, Andrea Omicini, editor, *5th International Workshop on Engineering Societies in the Agents World*, pages 261–276. Springer-Verlag, Berlin Heidelberg, 8 2005.

[CGFJ02] Anabela Caetano, Neri Goulart, Manuel Fonseca, and Joaquim Jorge. Javasketchit: Issues in sketching the look of user interfaces. In *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, pages 9–14, Palo Alto, USA, March 2002.

[Col01] R. W. Collier. *"Agent Factory: A Framework for the Engineering of Agent-Oriented Applications"*. PhD thesis, "University College Dublin", 2001.

[FPJ02] Manuel J. Fonseca, César Pimentel, and Joaquim A. Jorge. CALI: An Online Scribble Recognizer for Calligraphic Interfaces. In *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, pages 51–58, Palo Alto, USA, March 2002.

[HL00] Jason I. Hong and James A. Landay. Satin: a toolkit for informal ink-based applications. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 63–72, New York, NY, USA, 2000. ACM Press.

[KS04] Levent Burak Kara and Thomas F. Stahovich. Sim-u-sketch: a sketch-based interface for simulink. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 354–357, New York, NY, USA, 2004. ACM Press.

[LM01] James A. Landay and Brad A. Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer*, 34(2):56–64, 2001.

[LNHL00] James Lin, Mark W. Newman, Jason I. Hong, and James A. Landay. DENIM: finding a tighter fit between tools and practice for web site design. In *CHI*, pages 510–517, 2000.

[Mae95] Pattie Maes. Artificial life meets entertainment: lifelike autonomous agents. *Commun. ACM*, 38(11):108–114, 1995.

[MKSA03] P. A. Mitkas, D. Kehagias, A. L. Symeonidis, and I. N. Athanasiadis. "a framework for constructing multi-agent applications and training intelligent agents". In *Proceedings of the 4th Int. Workshop on Agent-Oriented Software Engineering (AOSE-2003)*, pages 96–109, 2003.

[NNLC99] H. Nwana, D. Ndumu, L. Lee, and J. Collis. Zeus: a toolkit and approach for building distributed multi-agent systems. In *Proceedings of the 3rd conference on Autonomous Agents*, pages 360–361. ACM Press, 1999.

[Pra05] Rui Prada. *Teaming Up Human and Synthetic Characters*. PhD thesis, Instituto Superior Técnico, Technical University of Lisbon, 2005.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[Sut64] Ivan E. Sutherland. Sketch pad a man-machine graphical communication system. In *DAC '64: Proceedings of the SHARE design automation workshop*, pages 6.329–6.346, New York, NY, USA, 1964. ACM Press.

[Wil99] U. Wilensky. Netlogo. Center for Connected Learning and Computer-Based Modeling. Northwestern University, Evanston, IL., <http://ccl.northwestern.edu/netlogo>, 1999.