Authoring the "Intelligence" of an Educational Game

M. Zancanaro, A. Cappelletti, C. Signorini, C. Strapparava

{zancana|cappelle|signori|strappa@irst.itc.it}

ITC-irst

Panté di Povo – 38050 Trento Italy

Abstract

In this paper, we describe a frame-based production rule system that works as the Artificial Intelligence Engine of an educational computer game. We discuss the need of an authoring environment clearly separated by the game in order to allow a technical staff without any skills in either AI or Computer Science to encode the "intelligence" of the game. Finally, we briefly introduce two graphical interface for authoring and testing frame hierarchies and production rules.

The production rule system and the authoring tools have been developed in the context of a project funded by the European Community to develop a prototypical educational computer game.

Introduction

Today, there is a wide acceptance on the role of AI to build more compelling computer games ([Laird and van Lent, 2000]), yet very little concern has been shown on letting content experts rather than programmers design the "intelligence" of the system. The authoring issue gains dramatically importance in the design of educational (and yet engaging!) computer games, where you would like to let content experts or an editorial technical staff to define and test the rules of the game. Indeed, in the near future it might be valuable to hire professional script writers even for non-educational games.

In this paper, we briefly discuss our experience in the design and implementation of a rule-based engine to be used in a 3D on-line educational computer game and its authoring environment. This work is part of a project called RENAISSANCE[1] funded by the European Community in the action line of "access to scientific and cultural heritage". The project was officially started in January 2000 and therefore what is discussed here must be considered a work in progress.

The RENAISSANCE Project

The aim of the RENAISSANCE project is to develop a computer game that makes use of high quality 3D graphics and engaging interaction while still able to deliver scientifically validated contents. The long term goal is to experiment with an innovative pedagogical approach: delivering

[1] The partners of the RENAISSANCE project (IST-1999-12163) other than ITC-irst are Giunti Multimedia, one of the biggest Italian publishing companies, as the main contractor; Blaxxun Interactive a german-based company whose main business is 3D-based virtual environments over the Internet and Iridon Interactive a Swedish company that produces and distributes computer games.

culture in an effective and amusing way at the same time.

The game is conceived as a 3D-based multi-user role-playing virtual community over the Internet. The game environment is the renaissance court of Urbino in central Italy around the first half of the fourteenth century. The term Renaissance describes the period of European history from the early 14th to the late 16th century, the name comes from the French word for rebirth and referred to the revival of the values and artistic styles of classical antiquity during that period, especially in Italy. This scenario was chosen because life in that period was subject to complex and subtle behavioral rules so precisely defined that have been codified in handbooks, in particular the famous "Book of Courtier" by Baldassarre Castiglione, published in 1528.

The players, as courtiers, have to increase their social positions and compete to obtain the Duke and Duchess' favors. The ultimate goal is to enable users to experience, as realistically as possible, the complexity of social life during that fascinating historical period while having the same fun of playing a "state of the art" video game.

The score of each player is expressed in terms of his fame, fortune, faith and force which can vary according to his "opportunistic" behavior in different situations. The "intelligence" of the games resides in a rule-based system (called the Evaluation Engine) that computes the "effect" of the players actions in the virtual world.

In the next section, we briefly introduce the system architecture focusing on the internal structure of the Evaluation Engine. Then, in the last section, we will describe the authoring environment actually used by an editorial staff to encode the rules of life in our virtual renaissance court.

The game architecture

The RENAISSANCE game is a 3D-based multi-user role-playing game over the Internet. The 3D rendering engine is local to each client and a Virtual Community Server (VCS) is in charge of maintaining the synchronization among the different clients. At each user action, the VCS computes the visible effects (in terms of rendering) and communicates the changes to the other clients. The Evaluation Engine, instead, is in charge of maintaining the coherence of the world from a semantic point of view: at each user action, it computes the "pragmatic" effects both for the user that performed the action and for the rest of the world. The Evaluation Engine is updated and queried by the VCS through a message protocol based on KQML [Labrou, 1997].

The Evaluation Engine

The Evaluation Engine is based on a frame system called CLOS-i built on top of CLOS (the Common Lisp Object System) exploiting the meta-object capabilities of this language. In designing CLOS-i our aim was to develop a "light" knowledge representation system yet efficient enough to be used in complex scenarios. The production rules system employs an implementation of the RETE algorithm [Forgy 1982] modified to be used together with a hierarchy of frames.

Rules and frames are two complementary knowledge representation schemes. There are several attempts to integrate these two approaches, but few efforts (in particular, [MacGregor, 1988] [Yen, 1991]) have been made to incorporate the terminological knowledge of frame-base systems into a rule-based paradigm.

We think that this approach improves conventional ruled-based programming from many points of view. In particular, the pattern matching operation is based on terminological definitions, not just on symbols (like in OPS5, for example) and conflict resolution can be based on well-defined specificity relationship among rules. Moreover, this approach encourages the development of a large and coherent knowledge base that is shared among the rules.

Example of a situation

We discuss here an example of a situation modeled in the very first KB of the RENAISSANCE game: every day at 10 a.m. an evening dinner with the Duke is organized. Each courtier with more than 500 points of fame receives an invitation. The dinner starts at 7pm. Courtiers who received an invitation and do not attend the dinner loose 100 points of "fame". In order to model the organization of the dinner, the more general frame of **activity** has been defined so that the starting and finishing of activities can be implemented as general rules. The **dinner** frame is defined as a sub-frame of **activity**, it has no slots because it has no special properties. Indeed, we need this new frame in order to write a more specific rule: every day at 10am the dinner (but not necessarily all the other activities) is scheduled; the rule **dinner_organization** is fired every time an instance of **set_time** is received with 10 as value of the **hour** slot; the action is the creation of a new instance of **dinner**.

The rule **dinner_invitation** is triggered by the creation of an instance of **dinner**, the other condition is that there should exist a courtier with more than 500 points of fame. An action for the creation of an instance of **invitation** is built for any such courtier. The rule **invitation_notify** takes care of communicating the events.

Once the dinner starts (according to the general rule **activity_start**), the rule **dinner_attendance** will fire on each courtier for which an instance of **invitation** exists and it will decrease his/her fame.

The Evaluation Engine Authoring Environment

We decided to employ a frame-based production rule system because our main concern was to allow a staff of technical editors of writing the "intelligence" of the system. Other researchers showed that production rules are a tool powerful enough for describing human cognition (see for example, Newell 1991) and simple and intuitive enough to be understood by naïve users (see for example, Anderson 1993). Yet we realized that we had to provide interactive tools to allow the editors to graphically manipulate the frame-based system and interactively test the rules independently from the game engine in order to let the editorial work proceed parallel to the work of the programmers and to the work of the designers.

We implemented two graphical interfaces: the Knowledge Base Editor and the Knowledge Base Shell.

The Knowledge Base Editor allows to graphically manipulate the frame hierarchies, to define and to edit frames and slots and to write rules. It exports the knowledge-bases as XML files.

Figure 1 depicts a snapshot of the KBE. The main window is divided into two parts, on the left window the user can choose whether to work on the frame hierarchy or on the set of rules; the right window is used to edit the particular frame/instance/rule selected on the left window. In the snapshot, the frame *courtier* is selected on

the left window. Each frame has a number of slots that represent the attributes of the concept. A frame automatically inherits the slots of its parent frame[2].

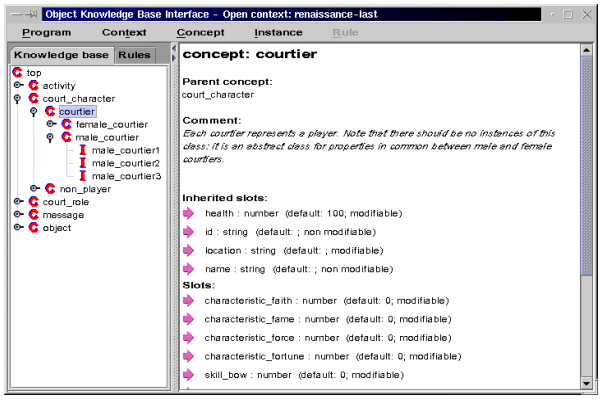

Figure 1.

Editing the frame hierarchy means editing frames and slots (i.e. working on the terminological part) or editing the instances of an already defined frame (usually, instances are created, modified and deleted at run time by the Evaluation Engine, yet it can be useful to have some pre-defined instances, for example non-player characters, furniture, etc.). These two activities can be interleaved, KBE is able to maintain the whole knowledge base consistent (for example, deleting a frame means removing all its instances; more subtly, it sometimes requires removing a slot from another frame and in turn all the corresponding slot values from its instances). Usually, KBE performs silently these operations, yet when the amount of deletions is big it warns the users before continuing. Moreover, the interface has been designed to minimise the likelihood of having inconsistent knowledge bases. For example, the user can never create a dangling frame (that is a frame without a parent): the only way to create a new frame using the interface is to add a child frame to an existing frame[3].

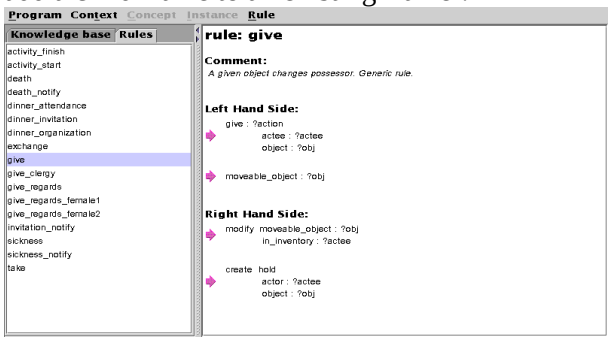

Figure 2.

KBE supports the rules writing task as well (see figure 2). The task of writing rules logically occurs after the creation of the knowledge base (because the left-hand side of a rule is expressed in terms of frames and possibly instances.) In our experience, however, the two tasks are highly interleaved: a first sketch of the frame hierarchy is necessary before any rule can ever be conceived, yet the actual writing of rules usually suggests new frames or even a different organisation of the hierarchy. Therefore, we designed the interface with the goal of making it easy for the user to interleave the two tasks. In order to avoid inconsistencies as much as possible the rules are composed by direct manipulation: before using a concept in a rule, the corresponding frame has to be defined in the hierarchy. As in the task of knowledge base editing, a lot of checks are performed automatically to maintain consistency: for example, if a frame is deleted, all the rules that use the corresponding concept are deleted as well.

The second tool of the authoring environment is the Knowledge Base Shell (or KBS, for short). It communicates with the Evaluation Engine in the very same way as the game does (i.e. KQML messages). The technical staff can therefore

[2] At present, multiple inheritance is not allowed. This feature can be dealt with in the present evaluation of the Evaluation Engine yet it may led to very inefficient and confuse knowledge bases.

[3] The very first frame is automatically created by the system and it's name is always *top*.

perform the operations that the game engine will perform during a game session, namely creating modifying and removing instances or querying the state of the knowledge base. Moreover, the actual rules fired at each interaction can be monitored.

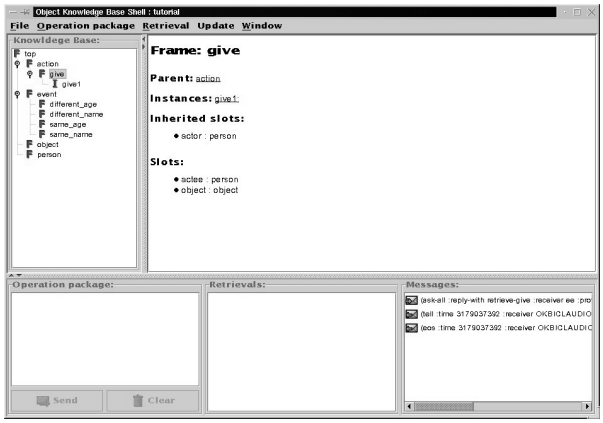

Figure 3.

Figure 3 shows a snapshot of the graphical interface. The application is composed by five windows: (1) the "KB Box" window, above on the left, displays the frame hierarchy and the instances created so far; (2) the "Control Box" window, displays detailed information on the selected element (i.e. either a frame, an instance, a message etc.); (3) the "Operation Packages Box" window, bottom on the left, stores the operations on instances already defined but not yet sent to the Evaluation Engine; (4) the "Retrievals" window, bottom middle, stores the queries to be submitted to the Evaluation Engine; and finally (5) the "Message Box" window, stores all the messagessent to and received from the Evaluation Engine.

KBS actually interprets the KQML messages received by the Evaluation Engine and it maintains the consistency in the windows, in particular in the "KB Box" where the instances created, deleted, and removed by either an user operation or the effect of a rule application are properly displayed. Yet, we decided to maintain visible the message exchanged to help the technical staff in better visualizing what is going on during a game session.

Conclusions

In this paper, we introduced a first attempt to build an authoring environment for the AI of (educational) computer games targeted to a technical editors staff. We think that in providing support of this kind of user testing is as much important as editing, in particular if the editorial works has to be made in parallel with the graphical design and with the programming, as it is usually the case.

This work is still in progress and it has been conduct in the context of a project funded by the European Community to develop a prototypical educational computer game, we would like to acknowledge the support of the other partner of the project for their suggestions and fruitful discussions.

Bibliography

J. R. Anderson. *Rules of the Mind*. Lawrence Erlbaum Associated, Hillsdale NJ, 1993.

Forgy "RETE: a Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", *Artificial Intelligence*, 19, 1982.

Y. Labrou, *Semantics for an Agent Communication* Language, PhD Thesis, University of Baltimore MA, 1997

J. Laird and M. van Lent, "Human-level AI's Killer Application: Interactive Computer Games", AAAI2000 Invited Talk, Austin TX, 2000.

A. Newell. *Unified Theories of Cognition.* Cambridge University Press. Cambridge MA, 1991.

R. M. MacGregor. "A Deductive Pattern Matcher". In Proceedings of the Seventh National Conference on Artificial Intelligence, (AAAI 88), pp. 403-408, 1988.

Yen, "CLASP: Integrating Term Subsumption System and Production Systems" IEEE Transaction on Knowledge and Data Engineering, 3(1) 1991.