A Testbed Environment for Interactive Storytellers

Federico Peinado
Depto. Ingeniería del Software e Inteligencia Artificial
Facultad de Informática UCM, Spain
+34 913947599
email@federicopeinado.com

Álvaro Navarro
Depto. Ingeniería del Software e Inteligencia Artificial
Facultad de Informática UCM, Spain
+34 913947599
alvaro.nav@gmail.com

Pablo Gervás
Instituto de Tecnologías del Conocimiento
Facultad de Informática UCM, Spain
+34 913947639
pgervas@sip.ucm.es

ABSTRACT

Today there is a number of automatic systems for developing interactive digital storytelling applications. Each one uses its own architecture, data structure and user interface which make practically impossible to create a single universal quantitative metric to compare them. While these differences are intrinsic to the artistic nature of narrative applications, developers of underlying technology could be benefited from some "evaluation standards" for these systems' functionality, interoperability and performance. This paper describes a testbed environment that has been designed as an example scenario for testing how different interactive storytelling systems confront a set of "common challenges" of this kind of applications. In order to avoid additional programming efforts an adapter that allows the connection of this environment with other systems has been implemented and released as open source.

Categories and Subject Descriptors

D.3.0. [**Programming Languages**]: Languages Java, C++ and Neverwinter Nights script.

I.2.1 [**Computing Methodologies**]: Applications and Expert Systems – Games.

I.3.4 [**Computing Methodologies**]: Graphics Utilities – Application packages, Virtual device interfaces.

I.3.7 [**Computing Methodologies**]: Three-Dimensional Graphics and Realism – Animation, Virtual Reality.

General Terms

Algorithms, Design, Experimentation, Human Factors, Standardization, Languages.

Keywords

Interactive Digital Storytelling, Narrative Environments and Game Based Interfaces.

1. INTRODUCTION

Today, members of the Interactive Digital Storytelling (IDS) research community are building software systems that automatically or semi-automatically control what happens inside an IDS application. These systems are mainly designed to deal with a characteristic conflict that can be found in any kind of interactive narrative phenomenon, known as the Interactive Storytelling Dilemma: "What should happen in a good interactive narrative experience when the interactors decide to do something different from what was initially planned by the authors?". Assuming the interactors' interface offers enough freedom of action and the authors have some ideas about the structure and content of the experience, it is normal to find clashes between both groups of participants.

These narrative controllers, usually conceived as artificial intelligence artefacts, can take many different forms. Some of them are impartial mediators between interactors and authors; others are artificial directors with their own criteria about how to guide the narrative experience. Some controllers are implemented using a centralized approach, while others are built on top of a distributed architecture of software agents.

IDS applications also differ a lot in the way the narrative stream is presented to the user. Most of them use some kind of virtual environment with narrative or dramatic qualities, such as visual capabilities for showing text dialogs, representing 2D or 3D characters, objects and locations, giving feedback to the user through game-like HUDs, etc. These virtual environments are, of course, interactive, so their states are continuously changing, frequently in real time, depending on the interactors' actions and the behaviours implemented in the environment (action-reaction rules, logical implications, physics, autonomous agents, hard-coded events, etc.).

The current trend towards expanding the range of platforms and media for interaction – to include mobile devices, multimodal displays, affective interfaces... – suggests that any move that made existing narrative controllers capable of interoperation across a range of interfaces or platforms would broaden the spectrum of possible interactions by providing both particular media with powerful narrative control and narrative applications with various enhanced means of presenting their output.

This paper describes a testing scenario, including the proposal of a communication protocol and language designed for connecting bidirectionally a testbed environment and a remote controller system in a generic way, abstracting low-level details relative to the composition of the multimedia effects of the virtual

environment and the internal representation of the whole model that the controller uses privately.

The structure of the paper is as follows: Section 2 gives an overview of the related work on this topic. Section 3 presents the communication protocol of our proposal. Section 4 is dedicated to the syntax and semantics of the new communication language. The implementation of the testbed environment is described in Section 5 and finally Section 6 and 7 are the discussions and conclusions of this research.

2. RELATED WORK

The need of communication between narrative environments and controllers has been there since the very first projects on IDS. Paradoxically there are few identifiable literature references on this subject, probably because much of that communication happens internally in applications that integrate a particular environment and its controller in a single application. In most cases communication is just performed using a proprietary format, with no special intention of establishing a general or reusable language.

Three projects with communication mechanisms comparable to our proposal have been chosen for consideration in this section. The first one is the Interactive Drama Architecture [1] proposed by Brian Magerko, the second one is Zocalo service-oriented architecture [2], part of Thomas M. Vernieri M.Sc. thesis, and finally the third one is the Shadow Door agent interface [3] by Robert Zubek, which was indeed the starting point of the development of our connection toolkit.

2.1 Interactive Drama Architecture

The Interactive Drama Architecture (IDA) is an IDS proposal that includes an automatic director of the interactive experience implemented as a rule-based system. This architecture is interesting because it has features of a mixed approach, in which both a centralized director (or drama manager) and semi-autonomous (directed) characters collaborate in order to develop an expressive, variable and interesting plot. The authors claim that a completely distributed controller is more complex than this semi-distributed structure in which only partial information (relevant parts of the plot) is given to a concrete character in a concrete situation of the story.

The system is integrated with an environment created with the Unreal engine using something similar to a “software adapter” for sending and receiving messages from/to the environment. There are few technical details documented about the implementation of this adapter, but authors explained that atomic events that happen in the internal world model of Unreal are generalized to simple predicates as MOVE-TO-ROOM(x, y), more easily interpretable by the director.

The controller of the story sends messages that are basic commands for a Non-Player Character (NPC) available at the environment for the player to interact with, messages such as “explore the environment”, “get this item”, “go to that room”, “say something to the player”, etc.

2.2 Zocalo

Zocalo is a service-oriented architecture for creating interactive narratives within game-based environments in which direction by planning is performed by Web services running on different machines. This project take serious considerations about scalability, security and interoperability issues of IDS applications’ software components. Unfortunately the .NET implementation of this promising open architecture for educational applications has not been released, but it is expected to work with different game engines (as Unreal or Half-Life 2).

Extensions to different components distributed in different operating systems are needed in order to communicate efficiently each component of the network. These extensions can be implemented in the form of a dynamically linked library (DLL) or a TCP socket connection. XML Schemas have been documented [4] for any XML message format that is sent and received between services, across a communication layer called Execution Manager - Socket Shell (EM-SS).

These formats are well documented, so they can be reused when developing extensions to the architecture. The content of the messages (mainly operators) has been found to be relatively coupled with the semantic specification of Longbow’s input, a main planner that this research group uses, but specially coupled with the planning paradigm.

2.3 Shadow Door

Shadow Door is an agent control interface for Neverwinter Nights (NWN) [5], a Computer Role-Playing Game (CRPG). It allows external applications to control one single character in the game. Using this interface developers can implement external controllers, coding them in arbitrary languages (as Lisp or Java), for the game play.

This interface is distributed as a DLL extension to the server application of NWN that communicates with external processes through a TCP socket. An extension to the game module called Neverwinter Nights Extender (NWNX) [6] is necessary, and also special scripts for receiving commands and passing them to the character, and sending facts (observations) that happen inside the world to the external controller. An Eliza-like example written in Lisp is included to illustrate how the complete system works.

3. COMMUNICATION PROTOCOL

RCEI (Remote-Controlled Environments Interface) is a communication protocol and language for IDS systems and narrative environments. Thanks to this connection, any IDS system can perform an interactive dramatic representation in the testbed environment we have created with a reasonable level of detail, with the sole additional requirement of adding RCEI adapters at both ends of the connection.

Assuming the goal of connecting an interactive narrative environment showing some realistic features with an AI-based remote controller, the RCEI protocol is presented as a solution dedicated to synchronize the sent/received messages between the system and the environment (see Figure 1). This protocol has some requirements which describe the characteristics of the communication.

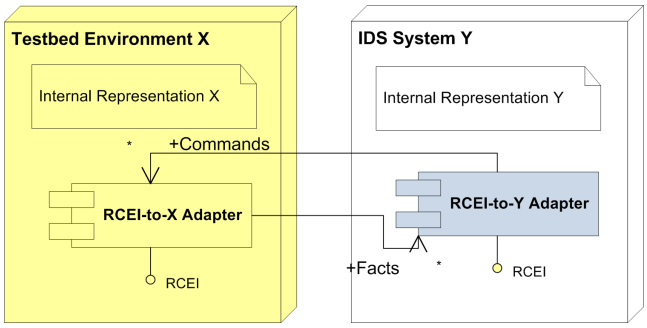

Figure 1. The RCEI communication protocol.

Firstly, the communication is blocking and synchronous, at the same time the sending of messages must be strictly alternate (e.g. starting with a first message of the system, then the answer of the environment, another message from the system, etc.).

The remote controller must be the one to start the interchange between both ends. This is because, generally, the answers of the environment will be *facts* that will confirm the system's *commands*. However, the environment could send additional unexpected facts such as consequences of long-term actions produced by the execution of previous commands, or actions performed by the interactors and other autonomous entities of the fictional world.

Secondly, the first message of the virtual environment must be an *identification message*, listing every domain-specific language extension supported by the current module that is running and also listing every condition referred to the initial state that the environment must know.

Thirdly, the remote-controlled environment must be the one to finish the communication by sending either an ending message to the environment, or an "unexpected error" message when something goes wrong at that side of the connection. Additionally, if the controller has been waiting for too long in a blocking state, expecting to receive messages, it will automatically produce an "unexpected ending due to request timeout" error.

4. SYNTAX AND SEMANTICS

The communication language is an extensible tool that allows communicating commands and facts between RCEI-aware systems and environments, with a syntax based on the tree-form of the XML language.

Each RCEI message has a *subject* and a *predicate*. The subject indicates the element affected by the action described, and it is either an agent of the environment or the environment itself, which means the action is probably location-based or a question of general presentation of the world (e. g. the change between day and night). The predicate contains the description of the action, and it must contain a *process* entry, which identifies the particular kind of action involved. This action can be relative to the environment or any object or agent, as indicated by the subject. Additionally, a command (or fact) may have other fields which specify the parameters needed for each type of message. These fields vary from one type of message to another.

Messages sent from the controller to the virtual environment act as imperative statements, indicating actions that must be performed in the environment. They are referred to as *commands*. Messages sent from the environment to the controller act as declarative statements, informing of actions that have taken place in the environment. They are referred to as *facts*.

The commands and facts must have at least two arguments and a maximum of four. At the present time we have distinguish some different categories between the components of the narrative environment; these are agents, objects, links and locations. The objective of the establishment of this simple hierarchy is to understand easily the structure and the functionality of the different commands and facts.

There are three special pairs of messages with a different structure: *begin*, *end* and *synchronize*. These messages allow communication of the current configuration of the world in terms of concepts currently existing in the world and the relations between them. The functionality of these instructions -when they are commands- is starting a new game, ending the current game, or forcefully reconfiguring the current game by performing changes (creating, deleting, moving…) in the available elements of the environment. When the instructions are facts, they are useful to report the state of the environment at the beginning, at the end or at any significant moment –determined by the AI system- during the game session.

4.1 Content of Messages

While the messages sent by the environment are interpreted as facts that have been produced recently in the environment, the messages sent by the remote controller are interpreted as "events that should happen in the environment as soon as possible". Due to the real time and non-deterministic constraints there are no guarantees that the actions expressed in those facts will happen. This may depend on whether the virtual environment manages commands by execution them on reception or by queuing them up, and whether some mechanism is available for cancelling the execution of commands unduly delayed in the queue to avoid out of synch behaviour of characters.

The contents of the facts that the controller can receive from the environment and the commands that it can send are slightly different. The basic repertory of RCEI contains instructions relative to the simulation domain of the environment, but the language can be extended to support narrative or interactive domain commands for the controller in order to manipulate the HUD shown in the game engine and not only the things represented in the virtual world.

The expressiveness of the instructions relative to the simulation is inspired on Conceptual Dependency Theory [7]. A list of the main instructions available is given in Table 1. Firstly, there is an instruction called *speak* to perform conversations between two agents. If we want to perform more advanced conversations, we also can use the instruction *change* to modify the state of characters involved in the dialog (emotion, attitude, mood, etc.).

The instruction *move* describes the execution of small animations of a given character, intended to convey visually some activity or emotional change in that character. They can range

from physical activities such as drinking or reading, to modulated verbal expression such as imploring or talking in anger or in laughter, including conventional moves of social interaction such as greeting or courtseying. In general terms, they allow communication of events that are not easy to represent using the visual capabilities of the environment.

There are *create* and *destroy* instructions for the creation of a location in a determined situation (linked to other locations), creation of a new agent or object in a determined locations, and creation of links between locations, which describe basically transitions from an origin location to a destination location. Characters' inventories or container objects may also be considered as locations.

The movement of an agent towards a given location is represented by the instruction *go.* Using the go command the system can specify the place towards which an agent must move. If no parameters are specified, a "random movement" is assumed.

The *change* instruction is used for changing the weather or ambient conditions of the virtual environment. It is also used to change the objects' state of things like doors, chests, weapons, lights sources, etc. This instruction has a great versatility for the simulation.

Table 1. RCEI Instructions

Process	Arguments	Description
speak	*subject*: <agent> *dirComp*: string	To speak with other agents.
move	*subject*: <agent> *dirComp*: "none", "drink", "reed", "greeting", "listen", "talk", "implore", "furious", "laugh", "victory", "adore", "harass", "reverence", "steal"	The agent will make a simple animation during a few seconds. The kind of animation is determined by the direct complement.
attack	*subject*: <agent> *dirComp*: <agent>, <object>	The agent will attack and fight with other agent or object.
go	*subject*: <agent> *dirComp(optional)*: <agent>, <object>, <link>	The agent translates himself to another position in the location. If there isn't direct complement the agent realizes a random movement translation.
take	*subject*: <agent> *dirComp*: <object>	An agent takes an object of the location.
drop	*subject*: <agent> *dirComp*: <object>	An agent drops an object of the location.
give	*subject*: <agent> *dirComp*: <object> *indComp*: <agent>	An agent gives an object to another agent. It is necessary that the object must be in the agent's inventory.
equip	*subject*: <agent> *dirComp*: <object>	An agent takes an object of his inventory and put it on.
unequip	*subject*: <agent> *dirComp*: <object>	An agent takes an object of his inventory and takes it off.
create	*subject*: <environment> *dirComp*: <agent> *placeComp*: <location>	The environment will create a new agent in the specific location determined by the place complement.
destroy	*subject*: <environment> *dirComp*: <agent>, <object>	The environment will destroy a new agent in the specific location determined by the place complement.
change	*subject*: <environment>, <agent> *dirComp*: "weather", "sky", "fog", "light", "camera", <agent>, <object> *placeComp(optional)*: <location> *modeComp(optional)*: "open", "close", "state".	The environment or an agent will change some properties. This command can be applied to change the weather, the state of a door, the environment properties, the state in a conversation, etc.
begin	*subject*: <environment> *dirComp(optional)*: string	To restart a game. If we put the name of an existing animation the environment will reproduces it.
end	*subject*: <environment> *dirComp (optional)*: string	To finish a game. We can visualize a final animation.
sync	*subject*: <environment>	The synchronize command produces a synchronize fact by the environment.

In a more specific domain of knowledge, there are some instructions to fulfil active processes between different agents or between agents and objects. These instructions have been included due to the game-like orientation of the scenarios that the testbed scenario is trying to generalize. They represent values of actions to, for instance, unsheathe a sword, cast spells, deliver an object to another agent, drop or take an object. These instructions are: *attack*, *take*, *drop*, *give*, *equip* and *unequip*.

The specific instructions of the RCEI metadomain are *begin*, *end* and *synchronize*. The purpose of these instructions is to identify some basic virtual environment requirements. Its functionality has been explained in the previous section.

5. THE TESTBED ENVIRONMENT

Using Neverwinter Nights as implementation tool, we have developed a testbed environment for the evaluation of interactive storytellers. This environment takes the form of a minigame called "Capture the Pig!". It is based on a fantastic medieval background, presenting a big farmyard with four gates as the main scenario. At those gates there are four different NPCs representing farmers who have the goal of catching and killing a restless and dangerous pig that is located inside the farmyard. The player controls another character that act as a spectator of the crazy hunting and, optionally, another participant. Finally there are some weapons and other interactive objects distributed across the scenario.

The set of possible stories generated within this environment is constrained to the different strategies that the farmers may follow in order to achieve the goal. There are different personalities, with their corresponding NWN behaviors, that are randomly assigned to the NPCs when the module starts. It is possible to see farmers fighting their competitors before concentrating on their final goal, looking for the strongest weapon before entering the farmyard or running directly toward the pig with the only help of their naked hands. Results of each pursuit and combat are not absolutely predictable because of the random factor inherent to NWN complex game mechanics, and characters are respawned shortly after their deaths, so all this make easy to repeat the experiment many times in order to obtain statistical data for measuring the results. Each farmer add one point to his "score" when the pig die by one of his attacks, while the pig add another point when it go out of the farmyard. An screenshot of this environment is shown in Figure 3.

The metrics we propose for the evaluation of different interactive storytelling systems are based on the definition of a set of "common challenges" that these systems must be able to solve and then, counting how many times each system is successful in solving the challenge, how much time it takes for solving the challenge and how many interventions (in the form of RCEI messages) it uses. Challenges are usually related to make the story unfolds toward a specific situation or ending (also called *author goals* in opposition to the characters' goals) at the same time system's interventions are coherent, believable and not too much intrusive, so other subjective or domain-specific criteria can be added to our proposal (e.g. "divine" solutions as making the pig or the farmers die by themselves are not allowed, etc.).

Capture the Pig! is presented as an environment that can support concrete implementations of some of those challenges. Table 2 describes a list that cover a wide spectrum of possible drama management situations in the context of this testbed environment. More challenges can be defined and many of them can be combined in complex ways, so the list does not pretend to be a canonical one but a first proposal for a complete pack of experiments. Due to the complexity of techniques that the, virtually omnipotent, systems have, it is a good idea to force several system to compete, assigning them contradictory goals. This idea can be complemented by the use of "sparrings", specially hard-coded algorithms that gets an optimum result solving a concrete challenge, performing illegal interventions if necessary, stablishing a hard competition against the real systems that are being tested.

Table 2. Capture the pig! challenges

Challenge	Description
A farmer prevails	The system must make one concrete farmer to have the biggest score at the end of the experiment.
The pig prevails	The system must make the pig to have the biggest score at the end of the experiment.
A farmer fails	The system must avoid one concrete farmer to have a good score at the end of the experiment.
The pig fails	The system must avoid the pig to have a good score at the end of the experiment.
A team of farmers prevail	The system must make some farmers to create a team, collaborating in having the biggest collective score at the end of the experiment.
No farmer prevails	The system must avoid any concrete farmer to have a bigger score than the others at the end of the experiment.
High scores as possible	The system must make the farmers and the pig to achieve their goals the most as possible until the end of the experiment.
Low scores as possible	The system must avoid the farmers and the pig to achieve their goals the most as possible until the end of the experiment.
Balanced scores	The system must make the farmers and the pig to have balanced scores (collective score of farmers = pig's score) at the end of the experiment.

While this connection toolkit have been conceived to be used by a knowledge-base interactive storyteller, this environment can be also controlled by a storyteller that distributes the knowledge between a group of of autonomous agents, each one dedicated to the control of one NPC.

The environment can also be tested manually using RCEI Wizard. This application is basically a GUI (see Figure 4) for users to test RCEI functionalities over their environments before finally incorporating them into their projects. A default environment, shown in Figure 5, is distributed with the toolkit.

Figure 3. An overview of the testbed environment: the group of farmers surrounding the pig.

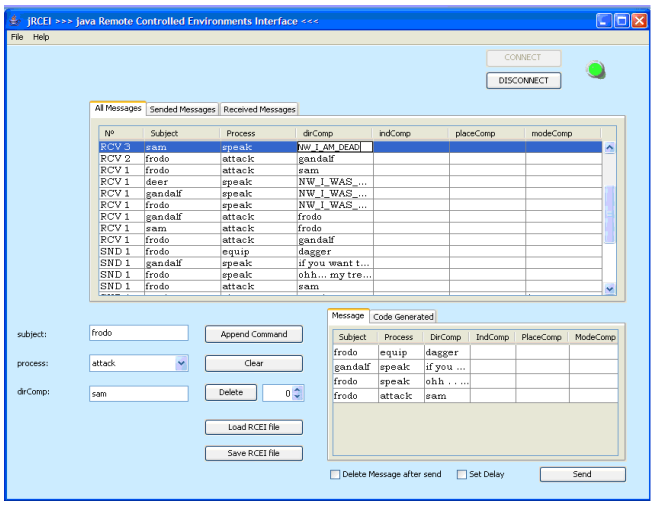

Figure 4. The user can send/receive commands to/from the environment using this interface.

The RCEI language has been implemented using Java and XML technologies in an API we called jRCEI [8]. The jRCEI distribution is available on the Internet as an open source project composed by different packages such as the parser, the serializer, the API itself and some additional testing code. The distribution also includes the RCEI adapter developed for NWN-based environments, developed on top of the NWNX extension mentioned before.

The RCEI protocol has been implemented using sockets with serialized 8-bit ASCII strings (maximum length of 1024 bytes) waiting 6 seconds and using TCP or UDP by the port 1890.

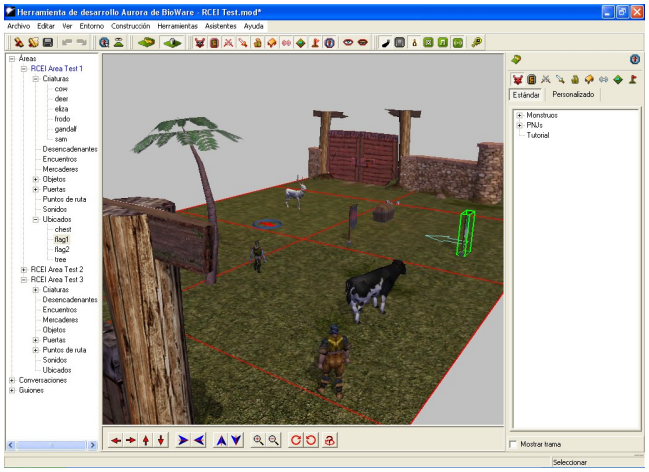

Figure 5. Editing the default environment: different characters and objects in an open scenario.

6. DISCUSSION

RCEI assumes that the remote controller works in god-mode and ideally it is omnipotent and independent from the virtual environment. This last consideration is a realist characteristic because the nature of both software systems (AI artifacts and graphic engines) is very different and they do not have many reusable features in common.

This interface is designed for working with knowledge-based narrative controllers, independently of whether they are plot-oriented or character-oriented, but it assume that at least lowest level behaviour of characters is determined by the environment.

Although the C++ open source code of Shadow Door was taken as the starting point of RCEI, the proposal presented in this paper already constitutes a significant improvement on the original interface. Shadow Door was intended to connect individual agents to a virtual environment, so that it supports a single NPC and it does not allow control of multiple agents nor the environment itself. Syntax and semantics of Shadow Door are significantly simpler than RCEI (only four types of commands and two types of facts are allowed in Shadow Door) so expressivity of the connection toolkit has been also improved.

It is known that RCEI has also some drawbacks, apart from the obvious loss of control granularity due to the abstraction required to communicate separate systems and to ignore particular details of each possible IDS system. The first drawback is caused by the lack of support for the possibility of connecting more than one intelligent controllers (e.g. software agents or distributed applications as web services, ec.). Other objection is that the RCEI standard assumes that the system has a *proactive* behaviour, and the environment a *reactive* behaviour, so if the case is not that clear, the integration could be more complex. Another issue is that every single command of the AI system try to be executed in parallel, in a synchronous way. In the future, we will consider the possibility to request that some orders may be executed in order, synchronized and with determinate parameters, but now the AI system is responsible of controlling that low-level synchronization.

Finally, this system has been designed for immediate communication in real time and it is not allowed to send planning-like statements or delayed commands between the controller and the environment: if something goes wrong, it must be captured and solved on-the-fly.

7. CONCLUSIONS

Nowadays there are several researchers in the IDS field who are developing software for controlling virtual environments. In some cases, the testing environments are extremely simple because researchers have no time for developing their own up-to-date 3D game engine or integrating one of those engines with their systems. Latest 3D virtual environments are very powerful tools with enormous possibilities to develop IDS applications. RCEI is trying to bring that potential to the intelligent system research field in an easy way, allowing different results to be presented in a coherent and similar way.

The objective of the RCEI project is to design and develop a communication standard between interactive narrative systems and virtual environments. If RCEI achieves our ambitious goal and it becomes to be widely used for prototyping in the community, adapters from one side and another could be reused in many projects, saving much time for developers and allowing development of studies about how different systems can create stories with the same set of resources and solve challenges in the same testbed environment.

The language proposed offers a basic but powerful vocabulary to communicate with the environment. In the future we expected to include extensions to the basic repertory of RCEI vocabulary, adding new instructions (facts and commands) but without coupling RCEI with any specific environment (creating new adapters for NWN2 or Unreal 3) or IDS system specification.

RCEI is going to be used in the final distribution of the Knowledge-Intensive Interactive Digital Storytelling (KIIDS) system [9] but the project is open to the collaboration of other researchers interested in contributing with their own requirements and ideas.

8. ACKNOWLEDGMENTS

This research is funded by the Spanish Ministry of Education and Science (TIN2006-14433-C02-01 project), Complutense University of Madrid and the G.D. of Universities and Research of the Community of Madrid (UCM-CAM-910494 research group grant). Second author held a Beca-Colaboración 2006-2007 grant from the Spanish Ministry of Education and Science.

9.REFERENCES

[1] Magerko, B., Laird, J.E.: Building an Interactive Drama Architecture. International Conference on Technologies for Interactive Digital Storytelling and Entertainment. Darmstadt, Germany (2003)

[2] Young, M., et al.: Zocalo (2007) http://zocalo.csc.ncsu.edu/

[3] Zubek, R.: Shadow Door: Neverwinter Nights NPC Control Interface (2003). http://www.zubek.net/robert/software/shadow-door/

[4] Cheong, Y.G., Michael R.: A Framework for Summarizing Game Experiences as Narratives. Liquid Narrative Group. Department of Computer Science North Carolina State University, Raleigh, NC 27695. http://liquidnarrative.csc.ncsu.edu/pubs/aiide06summ.pdf

[5] BioWare: Neverwinter Nights: Diamond Compilation Pack (DVD-ROM). Atari (2005)

[6] Stieger, I.: Neverwinter Nights Extender (2004) http://www.nwnx.org/

[7] Lytinen, S.L.: Conceptual Dependency and its Descendants. Computers and Mathematics with Applications, 23(2-5):51-73 (1992)

[8] Peinado, F. and Navarro, A.: RCEI, A Remote-Controlled Enviroments Interface (2007). http://federicopeinado.com/projects/rcei

[9] Peinado, F.: Knowledge-Intensive Interactive Digital Storytelling system (2007) http://federicopeinado.com/projects/kiids/