

State-of-the-Art  
Survey

Frank Dignum (Ed.)

LNAI 6525

# Agents for and Simu

Trends in Techniques

# Lecture Notes in Artificial Intelligence

6525

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

Frank Dignum (Ed.)

# Agents for Games and Simulations II

Trends in Techniques, Concepts and Design



Springer

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada  
Jörg Siekmann, University of Saarland, Saarbrücken, Germany  
Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editor

Frank Dignum  
Utrecht University  
Institute of Information and Computing Sciences  
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands  
E-mail: [dignum@cs.uu.nl](mailto:dignum@cs.uu.nl)

ISSN 0302-9743 e-ISSN 0302-9743  
ISBN 978-3-642-18180-1 e-ISBN 978-3-642-18181-8  
DOI 10.1007/978-3-642-18181-8  
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2009940795

CR Subject Classification (1998): I.2, D.2, F.3, D.4.6, I.7, K.4.2

LNCS Sublibrary: SL 7 – Artificial Intelligence

© Springer-Verlag Berlin Heidelberg 2011

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

*Typesetting:* Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))

# Preface

Research in Multi-Agent Systems offers promising technologies to implement non-playing characters embodying more realistic cognitive models. However, the technologies used in today's game engines and multi-agent platforms are not readily compatible due to some differences in their major concerns. For example, where game engines focus on real-time aspects that prioritize efficiency and central control, multi-agent platforms privilege agent autonomy instead. And while multi-agent platforms typically offer sophisticated communication capabilities, these may not be usable, or even appropriate, when the agents are coupled to a game. So, although increased autonomy and intelligence may offer benefits for a more compelling game play, and may even be essential for serious games, it is not clear whether current multi-agent platforms offer the means that are needed to accomplish this. Indeed, when current approaches to game design are used to incorporate state-of-the-art Multi-Agent System technology, the autonomy and intelligence of the agents might even be seen as more of a hindrance than an asset. A very similar argument can be given for approaches centered around agent-based (social) simulations.

In the current volume, *Agents for Games and Simulations*, we include papers presented at AGS 2010: the Second International workshop on Agents for Games and Simulations held on May 10 in Toronto. We received 12 submissions of high quality covering many of the aspects mentioned above. Each submission was reviewed by at least three Program Committee members. We accepted 11 papers for presentation, which can be found in this volume. This set of papers is complemented by some extended versions of papers from other workshops and the AAMAS conference in Toronto. Together this collection of papers give some answers to the issues raised above.

We have grouped the papers into three sections. The first section contains papers that are related to architectures combining agents and game engines. Besides new results from the Pogamut platform itself, there is also a paper discussing the integration of GOAL agents to Unreal Engine with the use of the Pogamut framework. It is nice to see this result stemming from last year's AGS workshop. Another paper in this section compares different multi-agent-based systems for crowd simulation. Indeed this is an important topic for many (serious) games incorporating disasters in public spaces where crowds are involved. The other two papers in this section treat issues with individual agent behavior in games. One looks at the combination of human and AI control of virtual characters, such that humans take care of those aspects that they are good at and the AI controls the parts that humans are less good at. This is an interesting point of view that might lead to new types of agent architectures as well. The last paper advocates the use of ontologies during the design of the game environment such that agents can use the ontology in their communication. This

prevents, for example, one agent referring to an object as being a table while another agent calls the object a desk. Avoiding this possible confusion makes it possible to model the communication at a more abstract level, providing more flexible protocols.

In the second section of this volume we included papers that focus on the training aspects of the games. Three of the papers discuss the directing of the game. The first paper uses value-driven characters. The second paper uses some implicit mechanisms in the game and the third paper discusses a planning approach. The last paper in this section is also about keeping the game interesting, but it uses an on-line adaptation mechanism to keep the game interesting for the trainee. All of these papers show that the objective of agents in a gaming environment should not just be to optimize some behavior, but rather to behave in a way that the game is as interesting as possible for the user. Therefore we should keep track of some overall storyline and objectives of the game as a whole.

The last section groups some papers around social and organizational aspects of games and agents. Two of the papers discuss certain approaches from agent institutions and organizations to model and implement the agent-based games. Using these approaches gives agents a degree of individual freedom but also keeps some central control over the game. The third paper discusses formal approaches to model social practices which can be used in gaming. Finally, one paper discusses the semi-automated classification of speech acts in a game. This type of data-mining technique can assist in modeling interesting behavior of agents based on the behavior of human players in a game.

All in all we are very happy with the papers contained in this volume. We are sure they form a valuable overview of the current state of the art for people that want to combine agent technology with (serious) games. Finally, we would like to thank the Program Committee members, without whom the reviewing would not have been possible and who gave valuable comments on all papers.

# Conference Organization

## Program Chairs

Frank Dignum	Utrecht University, The Netherlands
Annerieke Heuvelink	TNO, The Netherlands
Jeff Orkin	MIT, USA
Jeff Bradshaw	IHMC, USA

## Program Committee

Andre Campos	UFRN, Brazil
Bill Clancey	NASA, USA
Rosaria Conte	ISTC-CNR, Italy
Vincent Corruble	LIP6, France
Yves Demazeau	CNRS-LIG, France
Virginia Dignum	Delft University, The Netherlands
Willem van Doesburg	TNO, The Netherlands
Alexis Drogoul	LIP6, France
Corinna Elsenbroich	University of Surrey, UK
Klaus Fischer	DFKI, Germany
Koen Hindriks	Delft University, The Netherlands
Michael Lewis	University of Pittsburg, USA
Stacy Marsella	USC, USA
Hector Munoz-Avila	Lehigh University, USA
Anton Nijholt	UT, The Netherlands
Emma Norling	MMU, UK
Joost van Oijen	VSTEP, The Netherlands
Ana Paiva	IST, Portugal
Michal Pechoucek	CTU, Czech Republic
David Pynadath	USC, USA
Gopal Ramchurn	University of Southampton, UK
Avi Rosenfeld	JCT, Israel
David Sarne	Bar Ilan University, Israel
Barry Silverman	University of Pennsylvania, USA
Pieter Spronck	Tilburg University, The Netherlands
Duane Szafran	University of Alberta, Canada
Joost Westra	Utrecht University, The Netherlands

## External Reviewer

Ivo Swartjes	Twente University, The Netherlands
--------------	------------------------------------

# Table of Contents

## Section 1. Architectures

UNREAL GOAL BOTS: Conceptual Design of a Reusable Interface . . . . .	1
<i>Koen V. Hindriks, Birna van Riemsdijk, Tristan Behrens, Rien Korstanje, Nick Kraayenbrink, Wouter Pasman, and Lennard de Rijk</i>	
A Periphery of Pogamut: From Bots to Agents and Back Again . . . . .	19
<i>Jakub Gemrot, Cyril Brom, and Tomáš Plch</i>	
Goal-Based Communication Using BDI Agents as Virtual Humans in Training: An Ontology Driven Dialogue System . . . . .	38
<i>Joost van Oijen, Willem van Doesburg, and Frank Dignum</i>	
Evaluation and Comparison of Multi-agent Based Crowd Simulation Systems . . . . .	53
<i>Bikramjit Banerjee and Landon Kraemer</i>	
Towards an Architecture for Collaborative Human/AI Control of Interactive Characters . . . . .	67
<i>James Niehaus and Peter Weyhrauch</i>	

## Section 2. Training and Story Lines

An Architecture for Directing Value-Driven Artificial Characters . . . . .	76
<i>Rossana Damiano and Vincenzo Lombardo</i>	
Implicitly and Intelligently Influencing the Interactive Experience . . . . .	91
<i>Michael J. O’Grady, Mauro Dragone, Richard Tynan, Gregory M.P. O’Hare, Jie Wan, and Conor Muldoon</i>	
Creating Customized Game Experiences by Leveraging Human Creative Effort: A Planning Approach . . . . .	99
<i>Boyang Li and Mark O. Riedl</i>	
Guiding User Adaptation in Serious Games . . . . .	117
<i>Joost Westra, Frank Dignum, and Virginia Dignum</i>	
Using Agent Technology to Build a Real-World Training Application . . .	132
<i>Michal Cap, Annerieke Hewelink, Karel van den Bosch, and Willem van Doesburg</i>	



### Section 3. Social Behavior and Organization

Semi-Automated Dialogue Act Classification for Situated Social Agents in Games .....	148
<i>Jeff Orkin and Deb Roy</i>	
Using Exclusion Logic to Model Social Practices .....	163
<i>Richard Evans</i>	
Making Games ALIVE: An Organisational Approach .....	179
<i>Sergio Alvarez-Napagao, Fernando Koch, Ignasi Gómez-Sebastià, and Javier Vázquez-Salceda</i>	
Building Quests for Online Games with Virtual Institutions .....	192
<i>Gustavo Aranda, Tomas Trescak, Marc Esteva, and Carlos Carrascosa</i>	
<b>Author Index</b> .....	207

# UNREAL GOAL Bots

## Conceptual Design of a Reusable Interface

Koen V. Hindriks<sup>1</sup>, Birna van Riemsdijk<sup>1</sup>, Tristan Behrens<sup>2</sup>, Rien Korstanje<sup>1</sup>,  
Nick Kraayenbrink<sup>1</sup>, Wouter Pasman<sup>1</sup>, and Lennard de Rijk<sup>1</sup>

<sup>1</sup> Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands  
{k.v.hindriks,m.b.vanriemsdijk}@tudelft.nl

<sup>2</sup> Clausthal University of Technology, Julius-Albert-Straße 4, 38678 Clausthal,  
Germany  
behrens@in.tu-clausthal.de

**Abstract.** It remains a challenge with current state of the art technology to use BDI agents to control real-time, dynamic and complex environments. We report on our effort to connect the GOAL agent programming language to the real-time game UNREAL TOURNAMENT 2004. BDI agents provide an interesting alternative to control bots in a game such as UNREAL TOURNAMENT to more reactive styles of controlling such bots. Establishing an interface between a language such as GOAL and UNREAL TOURNAMENT, however, poses many challenges. We focus in particular on the design of a suitable and reusable interface to manage agent-bot interaction and argue that the use of a recent toolkit for developing an agent-environment interface provides many advantages. We discuss various issues related to the abstraction level that fits an interface that connects high-level, logic-based BDI agents to a real-time environment, taking into account some of the performance issues.

**Categories and subject descriptors:** I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—*Intelligent Agents*; I.6.7 [Simulation Support Systems]: Environments

**General terms:** Design, Standardization, Languages.

**Keywords:** agent-environment interaction, agent-oriented programming.

## 1 Introduction

Connecting cognitive or rational agents to an interactive, real-time computer game is a far from trivial exercise. This is especially true for logic-based agents that use logic to represent and reason about the environment they act in. There are several issues that need to be addressed ranging from the technical to more conceptual issues. The focus of this paper is on the design of an interface that is suitable for connecting logic-based BDI agents to a real-time game, but we will also touch on some related, more technical issues and discuss some of the challenges and potential applications that have motivated our effort.

The design of an interface for connecting logic-based BDI agents to a real-time game is complicated for at least two reasons. First, such an interface needs to be

designed at the right *abstraction level*. The reasoning typically employed by logic-based BDI agents does not make them suitable for controlling low-level details of a bot. Intuitively, it does not make sense, for example, to require such agents to deliberate about the degrees of rotation a bot should make when it has to make a turn. This type of control is better delegated to a behavioral control layer. At the same time, however, the BDI agent should be able to remain in control and the interface should support sufficiently fine grained control. Second, for reasons related to the required responsiveness in a real-time environment and efficiency of reasoning, the interface should not flood an agent with percepts. Providing a logic-based BDI agent with huge amounts of percepts would overload the agents' processing capabilities. The *cognitive overload* thus produced would slow down the agent and reduce its responsiveness. At the same time, however, the agent needs to have sufficient information to make reasonable choices of action while taking into account that the information to start with is at best incomplete and possibly also uncertain.

We have used and applied a recently introduced toolkit called the Environment Interface Standard to implement an interface for connecting agents to a gaming environment, and we evaluate this interface for designing a high-level interface that supports relatively easy development of agent-controlled bots. We believe that making environments easily accessible will facilitate the evaluation and assessment of performance and the usefulness of features of agent platforms.

Several additional concerns have motivated us to investigate and design an interface to connect logic-based BDI agents to a real-time game. First, we believe more extensive evaluation of the application of logic-based BDI agents to challenging, dynamic, and potentially real-time environments is needed to assess the current state of the art in programming such agents. Such an interface will facilitate *putting agent (programming) platforms to the test*. Although real-life applications have been developed using agent technology including BDI agent technology, the technology developed to support the construction of such agents may be put to more serious tests. As a first step, we then need to facilitate the connection of such agents to a real-time environment, which is the focus of this paper. This may then stimulate progress and development of such platforms into more mature and effectively applicable tools. Second, the development of a high-level agent-game bot-interface may *make the control of game bots more accessible* to a broader range of researchers and students. We believe such an interface will make it possible for programmers with relatively little experience with a particular gaming environment to develop agents that can control game bots reasonably well. This type of interface may be particularly useful to prototype gaming characters which would be ideal for the gaming industry [1]. We believe it will also *facilitate the application of BDI agent technology by students* to challenging environments and thus serve educational purposes. The development of such an interface has been motivated by a project to design and create a new student project to teach students about agent technology and multi-agent systems. Computer games have been recognized to provide a fitting subject [2]. As noted in [1],

Building agents situated in dynamic, potentially antagonistic environments that are capable of pursuing multiple, possibly conflicting goals not only teaches students about the fundamental nature and problems of agency but also encourage them to develop or enhance programming skills.

Finally, an interesting possibility argued for in e.g. [23] is that the *use of BDI agents to control bots* instead of using scripting or finite-state machines *may result in more human-like behavior*. As a result, it may be easier to develop characters that are believable and to provide a more realistic feel to a game. Some work in this direction has been reported in [4], which uses a technique called Applied Cognitive Task Analysis to elicit players' strategies, on incorporating human strategies in BDI agents. [3] also discuss the possibility to use data obtained by observing actual game players to specify the Beliefs, Desires, and Intentions of agents. It seems indeed more feasible to somehow "import" such data expressed in terms of BDI notions into sophisticated BDI agents, rather than translate it to finite-state machines. The development of an interface that supports logic-based BDI agent-control of bots thus may offer a very interesting opportunity for research into human-like characters (see also [15,6,7]).

As a case study we have chosen to connect the agent programming language GOAL to the game UNREAL TOURNAMENT 2004 (UT2004). UT2004 is a first-person shooter game that poses many challenges for human players as well as computer-controlled players because of the fast pace of the game and because players only have incomplete information about the state of the game and other players. It provides a real-time, continuous, dynamic multi-agent environment and offers many challenges for developing agent-controlled bots. It thus is a suitable choice for putting an agent platform to the test. [8] argue that UNREAL TOURNAMENT provides a useful testbed for the evaluation of agent technology and multi-agent research. These challenges also make UT2004 a suitable choice for defining a student project as students will be challenged as well to solve these problems using agent technology. Multi-agent team tasks such as coordination of plans and behavior in a competitive environment thus naturally become available. In addition, the 3D engine, graphics and the experience most students have with the game will motivate students to actively take up these challenges. Moreover, as a competition has been setup around UT2004 for programming human-like bots [5], UT2004 also provides a clear starting point for programming human-like virtual characters. Finally, the UNREAL engine has enjoyed wide interest and has been used by many others to extend and modify the game. As a result, many modifications and additional maps are freely available. It has, for example, also been used in competitions such as the RobocupRescue competition [9] which provides a high fidelity simulation of urban search and rescue robots using the UNREAL engine. Using the UNREAL TOURNAMENT game as a starting point to connect an agent platform to thus does not limit possibilities to one particular game but rather is a first step towards connecting an agent platform to a broad range of real-time environments. Moreover, a behavioral control layer called *Pogamut* extending *Gamebots* is available for UT2004 [10,8] which facilitates bridging the gap that exists when trying to implement an interface

oriented towards high-level cognitive control of a game such as UT2004. Throughout the paper the reader should keep in mind that we use these frameworks. Technically, UT2004 is state of the art technology that runs on Linux, Windows, and Macintosh OS.

Summarizing, the paper’s focus is on the design of a high-level interface for controlling bots in a real-time game and is motivated by various opportunities that are offered by such an interface. Section 2 discusses some related work. Section 3 briefly introduces the GOAL agent programming language. Section 4 discusses the design of an agent-interface to UT2004, including interface requirements, the design of actions and percepts to illustrate our choices, and the technology that has been reused. This section also introduces and discusses a recently introduced technology for constructing agent-environment interfaces, called the Environment Interface Standard [11,12]. Section 5 concludes the paper.

## 2 Related Work

Various projects have connected agents to UT2004. We discuss some of these projects and the differences with our approach.

Most projects that connect agents to UT2004 are built on top of Gamebots [8] or Pogamut [10], an extension of Gamebots: See e.g. [13,14] which use Gamebots and [7] which use Pogamut.<sup>1</sup> Gamebots is a platform that acts as a UT2004 server and thus facilitates the transfer of information from UT2004 to the client (agent platform). The GameBots platform comes with a variety of predefined tasks and environments. It provides an architecture for connecting agents to bots in the UT2004 game while also allowing human players to connect to the UT2004 server to participate in a game. Pogamut is a framework that extends GameBots in various ways, and provides a.o. an IDE for developing agents and a parser that maps Gamebots string output to Java objects. We have built on top of Pogamut because it provides additional functionality related to, for example, obtaining information about navigation points, ray tracing, and commands that allow controlling the UT2004 gaming environment, e.g. to replay recordings.

A behavior-based framework called pyPOSH has been connected to UT2004 using Gamebots [14]. The motivation has been to perform a case study of a methodology called Behavior Oriented Design [1]. The framework provides support for reactive planning and the means to construct agents using Behavior Oriented Design (BOD) as a means for constructing agents. BOD is strongly inspired by Behavior-based AI and is based on “the principle that intelligence is decomposed around expressed capabilities such as walking or eating, rather than around theoretical mental entities such as knowledge and thought.” [14] These agents thus are behavior-based and not BDI-based.

Although we recognize the strengths and advantages of a behavior-based approach to agent-controlled virtual characters, our aim has been to facilitate the use of *cognitive* agents to control such characters. In fact, our approach has been to design and create an interface to a behavior-based layer that provides access

---

<sup>1</sup> [15] is an exception, directly connecting READYLOG agents via TCP/IP to UT2004.

to the actions of a virtual character; the cognitive agent thus has ready access to a set of natural behaviors at the right abstraction level. Moreover, different from [1] the actions and behaviors that can be performed through the interface are clearly separated from the percepts that may be obtained from sensors provided by the virtual environment (although the behaviors have access to low-level details in the environment that is not all made available via the interface). The main difference with [1] thus is the fact that cognitive agent technology provides the means for action selection and this is not all handled by the behavior-layer itself (though e.g. navigation skills have been “automated”, i.e. we reuse the navigation module of Pogamut).

An interface called *UtJackInterface* is briefly discussed in [16]. This interface allows JACK agents [17] to connect to UT2004. The effort has been motivated by the “potential for teaming applications of intelligent agent technologies based on cognitive principles”. The interface itself reuses components developed in the Gamebots and Javabots project to connect to UT2004. As JACK is an agent-oriented extension of Java it is relatively straightforward to connect JACK via the components made available by the Gamebots and Javabots projects. Some game-specific JACK code has been developed to “explore, achieve, and win” [16]. The interface provides a way to interface JACK agents to UT2004 but does not provide a design of an interface for logic-based BDI agents nor facilitates reuse.

The cognitive architecture Soar [18] has also been used to control computer characters. Soar provides so-called *operators* for decision-making. Similar to GOAL - which provides reserved and user-defined actions - these operators allow to perform actions in the bots environment as well as internal actions for e.g. memorizing. The action selection mechanism of Soar is also somewhat similar to that of GOAL in that it continually applies operators by evaluating if-then rules that match against the current state of a Soar agent. Soar has been connected to UT2004 via an interface called the *Soar General Input/Output* which is a domain independent interface [19]. Soar, however, does not provide the flexibility of agent technology as it is based on a fixed cognitive architecture that implements various human psychological functions which, for example, limit flexible access to memory. An additional difference is that Soar is knowledge-based and does not incorporate declarative goals as GOAL does.

Similarly, the cognitive architecture ACT-R has been connected to UNREAL TOURNAMENT [20]. Interestingly, [20] motivate their work by the need for cognitively plausible agents that may be used for training. Gamebots is used to develop an interface from UNREAL TOURNAMENT to ACT-R.

Arguably the work most closely related to ours that connects high-level agents to UNREAL TOURNAMENT is the work reported on connecting the high-level logic-based language READYLOG (a variant of GOLOG) to UT2004 [15]. Agents in READYLOG also extensively use logic (ECLIPSe Prolog) to reason about the environment an agent acts in. Similar issues are faced to provide an interface at the right abstraction level to ensure adequate performance, both in terms of responsiveness as well as in terms of being effective in achieving good game performance. A balance needs to be struck in applying the agent technology

provided by READYLOG and the requirements that the real-time environment poses in which these agents act. The main differences between our approach and that of [15] are that our interface is more detailed and provides a richer action repertoire, and, that, although READYLOG agents are logic-based, READYLOG agents are not BDI agents as they are not modelled as having beliefs and goals.

Summarizing, our approach differs in various ways from that of others. Importantly, the design of the agent interface reported here has quite explicitly taken into account what would provide the right abstraction level for connecting logic-based BDI agents such as GOAL agents to UT2004. As the discussion below will highlight (see in particular Figure 1), a three-tier architecture has been used consisting of the low-level Gamebots server extension of UT2004, a behavioral layer provided by a particular bot run on top of Pogamut, and, finally, a logic-based BDI layer provided by the GOAL agent platform. Maybe just as important is the fact that we have used a generic toolkit [11][12] to build the interface that is supported by other agent platforms as well. This provides a principled approach to reuse of our effort to facilitate control of UNREAL bots by logic-based BDI agents. It also facilitates comparison with other agent platforms that support the toolkit and thus contributes to evaluation of agent platforms.

### 3 Agent Programming in GOAL

GOAL is a high-level agent programming language for programming rational or cognitive agents. GOAL agents are logic-based agents in the sense that they use a knowledge representation language to reason about the environment in which they act. The technology used here is SWI Prolog [21]. Due to space limitations, the presentation of GOAL itself is very limited and we cannot illustrate all features present in the language. For more information, we refer to [22][23], which provides a proper introduction to the constructs introduced below and discusses other features such as modules, communication, macros, composed actions, and more.

The language is part of the family of agent programming languages that includes e.g. 2APL, Jadex, and Jason [24]. One of its distinguishing features is that GOAL agents have a mental state consisting of *knowledge*, *beliefs* and *goals* and GOAL agents are able to use so-called *mental state conditions* to inspect their mental state. Mental state conditions allow to inspect both the beliefs and goals of an agent's mental state which provide GOAL agents with quite expressive reasoning capabilities.

A GOAL agent program consists of various sections. The *knowledge base* is a set of concept definitions or domain rules, which is optional and represents the conceptual or domain knowledge the agent has about its environment. For the purposes of this paper, the **knowledge** section is not important and we do not explain the relation to beliefs and goals here (see for a detailed discussion [23]). The **beliefs** section defines the initial *belief base* of the agent. At runtime a belief base, which is a set of beliefs coded in a knowledge representation language (i.e. Prolog in our case), is used to represent the current state of affairs. The **goals** section defines the initial *goal base*, which is a set of goals also coded in the same

knowledge representation language, used to represent in what state the agent wants to be. The **program** section consists of a set of action rules which together define a strategy or policy for action selection. The **actionspec** section consists of action specifications for each action made available by the environment; an action specification consists of a *precondition* that specifies when an action can be performed and a *postcondition* that specifies the effects of performing an action. Although GOAL provides the means to write pre- and post-conditions it does not force a programmer to specify such conditions, and actions may be introduced with empty pre- and/or postconditions; we will discuss the usefulness of empty conditions later in the paper again. Finally, a set of *the percept rules* specify how percepts received from the environment modify the agent's mental state.

Actions are selected in GOAL by so-called *action rules* of the form

**if <cond> then <action>**

where <cond> is a mental state condition and <action> is either a built-in or an action made available by the environment. These rules provide GOAL agents with the capability to react flexibly and reactively to environment changes but also allow a programmer to define more complicated strategies. Modules in GOAL provide a means to structure action rules into clusters of such rules to define different strategies for different situations [25]. Percept rules are special action rules used to process percepts received from the environment. These rules allow (pre)processing of percepts and allow a programmer to flexibly decide what to do with percepts received (updating by inserting or deleting beliefs, adopting or dropping goals, or send messages to other agents). Additional features of GOAL include a.o. a macro definition construct to associate intuitive labels with mental state conditions which increases the readability of the agent code, options to apply rules in various ways, and communication.

## 4 Agent Interface for Controlling UNREAL Bots

One of the challenges of connecting BDI agents such as GOAL agents to a real-time environment is to provide a well-defined interface that is able to handle events produced by the environment, and that is able to provide sensory information to the agent and provides an interface to send action commands to the environment. Although Gamebots or Pogamut do provide such interfaces they do so at a very low-level. The challenge here is to design an interface at the right abstraction level while providing the agent with enough detail to be able to “do the right thing”. In other words, the “cognitive load” on the agent should not be too big for the agent to be able to efficiently handle sensory information and generate timely responses; it should, however, also be plausible and provide the agent with more or less the same information as a human player. Similarly, actions need to be designed such that the agent is able to control the bot by sending action commands that are not too finegrained but still allow the agent to control the bot in sufficient detail. Finally, the design of such an interface should also pay attention to technical desiderata such as that it provides support for



debugging agent programs and facilitates easy connection of agents to bots. This involves providing additional graphical tools that provide global overviews of the current state of the map and bots on the map as well as event-based mechanisms for launching, killing and responding to UT server events. In the remainder of this section, we describe in more detail some of the design choices made and the advantages of using the Environment Interface toolkit introduced in [11,12]. We begin with briefly discussing UNREAL TOURNAMENT 2004 and then continue with discussing the interface design.

#### 4.1 UNREAL TOURNAMENT

UT2004 is an interactive, multi-player computer game where bots can compete with each other in various arenas. The game provides ten different game types including, for example, *DeathMatch* in which each bot is on its own and competes with all other bots present to win the game where points are scored by disabling bots, and *Team DeathMatch* which is similar to *DeathMatch* but is different in that two teams have to compete with each other. One of the key differences between *DeathMatch* and *Team DeathMatch* is that in the latter bots have to act as a team and cooperate and coordinate. The game type that we have focused on is called *Capture The Flag* (CTF). In this type of game, two teams compete with each other and have as their main goal to conquer the flag located in the home base of the other team. Points are scored by bringing the flag of the opponent's team to one's own home base while making sure the team's own flag remains in its home base.

The CTF game type requires more complicated strategic game play [15] which makes CTF very interesting for using BDI agents that are able to perform high-level reasoning and coordinate their actions to control bots. An interface "at the knowledge level" [26] facilitates the design of strategic agent behavior for controlling bots as the agent designer is not distracted by the many low-level details concerning, for example, movement. That is, the interface discussed below allows an agent to construct a high-level environment representation that can be used to decide on actions and focus more on strategic action selection. Similarly, by facilitating the exchange of high-level representations between agents that are part of the same team, a programmer can focus more on strategic coordination. As one of our motivations for building an agent interface to UT2004 has been to teach students to apply agent technology in a challenging environment, we have chosen to focus on the CTF game type and provide an interface that supports all required actions and percepts related to this scenario (e.g. this game type also requires that agents are provided with status information regarding the flag, and percepts to observe a bot carrying a flag).<sup>2</sup>

<sup>2</sup> Our experience with student projects that require students to develop soccer agents using basically Java is that students spent most of their time programming more abstract behaviors instead of focussing on the (team) strategy. Similar observations related to UT2004 are reported in [13], and have motivated e.g. [10]. We hope that providing students with a BDI programming language such as GOAL will focus their design efforts more towards strategic game play.

## 4.2 Requirements

As has been argued elsewhere [1], in order to make AI accessible to a broad range of people as a tool for research, entertainment and education various requirements must be met. Here, we discuss some of the choices we made related to our objective of making existing agent technology available for programming challenging environments.

The tools that must be made available to achieve such broad goals as making AI, or, more specifically, agent technology accessible need to provide quite different functionality. One of the requirements here is to make it possible to use an (existing) agent platform to connect to various environments. We argue that agent programming languages are very suitable as they provide the basic building blocks for programming cognitive agents. Agent programming languages, moreover, facilitate incremental design of agents, starting with quite simple features (novices) to more advanced features (more experienced programmers).

Additional tools typically need to be available to provide a user-friendly development environment, such as tools to inspect the *global* state of the environment either visually or by means of summary reports. Auxiliary tools that support debugging are also very important. GOAL provides an Integrated Development Environment with various features for editing (e.g. syntax highlighting) and debugging (e.g. break points). Similar requirements are listed in [19], which adds that it is important that the setup is flexible and allows for low-cost development such that easy modifications to scenarios etc are feasible. For example, in the student project, we plan to use at least two maps to avoid student teams to bias their agents too strongly with respect to one map. This presumes easy editing of maps, which is facilitated by the many available UT2004 editors.

## 4.3 Interface Design

The *Environment Interface Standard* (EIS) [11,12] is a proposed standard for interfaces between (agent-)platforms and environments. It has been implemented in Java but its principles are portable. We have chosen to use EIS because it offers several benefits. First of all, it increases the reusability of environments. Although there are a lot of sophisticated platforms, the exchange of environments between them is very rare, and if so it takes some time to adapt the environment. EIS on the other hand makes complex multi-agent environments, for example gaming environments, more accessible. It provides support for event and notification handling and for launching agents and connecting to bots.

EIS is based on several principles. The first one is *portability* which means in this context that the easy exchange of environments is facilitated. Environments are distributed via jar-files that can easily be plugged in by platforms that adhere to EIS. Secondly, it imposes only *minimal restrictions on the platform or environment*. For example, there are no assumptions about scheduling, agent communication and agent control. Also there are no restrictions on the use of different technical options for establishing a connection to the environment, as TCP/IP, RMI, JNI, wrapping of existing Java-code et cetera can be used. Another principle is *the separation of concerns*. Implementation issues related to

the agent platform are separated from those related to the environment. Agents are assumed to be percept-processors and action-generators. Environment entities are only assumed to be *controllable*, i.e. they can be controlled by agents and provide sensory and effectoric capabilities. Otherwise EIS does not assume anything about agents and entities and only stores identifiers for these objects, and as such assures the interface is agnostic about agent and bot specifics.

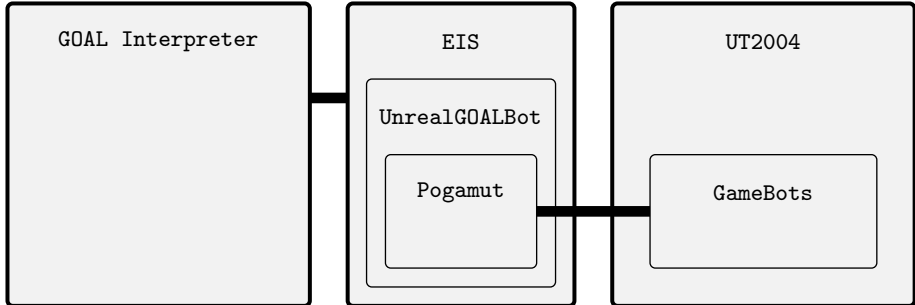
EIS provides various types of implementation support for connecting an agent platform to an environment. It facilitates acting, active sensing (actions that yield percepts), passive sensing (retrieving all percepts), and percepts-as-notifications (percepts sent by the environment). Another principle is a *standard for actions and percepts*. EIS provides a so called *interface intermediate language* that is based on an abstract-syntax-tree-definition. The final principle is the *support for heterogeneity*, that is that EIS provides means for connecting several platforms to a single instance of an environment. EIS is supported by and has been tested with 2APL, Jadex, Jason, and by GOAL.

The connection established using EIS between GOAL-agents, which are executed by the GOAL-interpreter, and UT2004 bots in the environment consists of several distinct components (see Fig. 1). The first component is GOAL's support for EIS. Basically this boils down to a sophisticated MAS-loading-mechanism that instantiates agents and creates the connection between them and entities, together with a mapping between GOAL-percepts/actions and EIS ones. Connecting to EIS is facilitated by Java-reflection. Entities, from the environment-interface-perspective, are instances of *UnrealGOALBot*, which is a heavy extension of the *LoqueBot* developed by Juraj Simovic. LoqueBot on the other hand is built on top of Pogamut [10]. Pogamut itself is connected to *GameBots*, which is a plugin that opens UT2004 for connecting external controllers via TCP/IP.

Entities consist of three components: (1) an instance of *UnrealGOALBot* that allows access to UT, (2) a so called *action performer* which evaluates EIS-actions and executes them through the *UnrealGOALBot*, and (3) a *percept processor* that queries the memory of the *UnrealGOALBot* and yields EIS-percepts.

The instantiation of EIS for connecting GOAL to UT2004 distinguishes three classes of percepts. *Map-percepts* are sent only once to the agent and contain static information about the current map. That is navigation-points (there is a graph overlaying the map topology), positions of all items (weapons, health, armor, power-ups et cetera), and information about the flags (the own and the one of the enemy). *See-percepts* on the other hand consist of what the bot currently sees. That is visible items, flags, and other bots. *Self-percepts* consist of information about the bot itself. That is physical data (position, orientation and speed), status (health, armor, ammo and adrenaline), all carried weapons and the current weapon. Although these types of percepts are implemented specifically for UT2004, the general concepts of percepts that are provided only once, those provided whenever something changes in the visual field of the bot, and percepts that relate to status and can only have a single value at any time (e.g. current weapon) can be reapplied in other EIS instantiations. Here are some examples: `bot(bot1,red)` indicates the bot's name and

its team, `currentWeapon(redeemer)` denotes that the current weapon is the Redeemer, `weapon(redeemer,1)`, indicates that the Redeemer has one piece of ammo left, and `pickup(inventoryspot56,weapon,redeemer)` denotes that a Redeemer can be picked up at the navigation-point `inventoryspot56`.



**Fig. 1.** A schematic overview of the implementation. The GOAL-interpreter connects to the EIS via Java-reflection. EIS wraps UnrealGOALBot, a heavy extension of Loquebot. UnrealGOALBot wraps Pogamut, which connects to GameBots via TCP/IP. GameBots is an Unreal-plugin.

Actions are high-level to fit the BDI abstraction. The primitive behaviors that are used to implement these actions are based on primitive methods provided by the LoqueBot. Design-choices however were not that easy. We have identified several layers of abstraction, ranging from (1) really low level interaction with the environment, that is that the bot sees only neighboring waypoints and can use raytracing to find out details of the environment, over (2) making all waypoints available and allowing the bot to follow paths and avoid for example dodging attacks on its way, to (3) very high-level actions like *win the game*. The low level makes a very small reaction-time a requirement and is very easy to implement, whereas the high level allows for longer reaction times but requires more implementation effort. We have identified the appropriate balance between reaction-time implementation effort to be an abstraction layer in which we provide these actions: `goto` navigates the bot to a specific navigation-point or item, `pursue` pursues a target, `halt` halts the bot, `setTarget` sets the target, `setWeapon` sets the current weapon, `setLookat` makes the bot look at a specific object, `dropweapon` drops the current weapon, `respawn` respawns the bot, `usepowerup` uses a power-up, `getgameinfo` gets the current score, the game-type and the identifier of the bot's team. Due to space limitations we do not provide all the parameters associated with these actions in detail. Note that several but in particular the first two actions take time to complete and are only initiated by sending the action command to UT2004. Durative actions such as `goto` and `pursue` may be interrupted. The agent needs to monitor the actions through percepts received to verify actions were succesful. EIS does support providing percepts as "return values" of actions but this requires blocking of the thread

executing the action and we have chosen not to use this feature except if there is some useful “immediate” information to provide which does not require blocking. Special percepts were implemented to monitor the status of the goto action, including e.g. whether the bot is stuck or has reached the target destination. Moreover, the agent can control the route towards a target destination but may also delegate this to the behavioral control layer.

#### 4.4 An Example: The UNREAL-Pill-Collector

Figure 2 shows the agent-code of a simple GOAL-agent that performs two tasks: (1) collecting pills and (2) setting a target for attack. The agent relies on the reception of percepts that are provided by the environment to update its beliefs during runtime. The beliefs present in the **beliefs** section in the agent program code are used to initialize the belief state of the agent. The first fact listed states that initially the agent has no target. The second fact represents the initial parameters associated with the bot’s position, its rotation, velocity and moving state, together called the *physical-state* of the bot (the moving state of a bot can be *stuck*, *moving*, and *reached*). Similarly, the **goals** section is used to initialize the agent’s goal base and initially will contain the goal of collecting special items, represented simply by the abstract predicate *collect*, and the goal to target all bots (implicitly only bots part of another team will be targeted as it is not possible in UT2004 to shoot your own team mates). The first rule in the **program** section makes the bot go to the specific location of a special-item (a so-called *pickup* location) if the agent knows about such a location and has the goal of collecting special items. The second rule sets the targets from none to all bots.

In the example only two out the total number of actions that were briefly introduced above have been used. We discuss these action more extensively here because they help to clarify how the interface with UT2004 works. Actions defined in the **actionspec** section need to be made available by the environment, in our case UT2004. They need to be specified in GOAL because the *name* and *parameters* of the action need to be specified to be able to use it in action rules, and because preconditions and postconditions of actions may be specified (but need not be; they can be left empty). The *goto* action in the **actionspec** section allows the bot to move in the environment. The *setTarget* action sets the enemy bots that will be targeted if visible. These actions are quite different. The *goto* action takes more or less time to complete depending on the distance to be traveled. The *setTarget* action in contrast is executed instantaneously as it only changes a mode of operation (a parameter). This difference has important consequences related to specifying the pre- and postcondition of these actions. Whereas it is quite easy to specify the pre- and postcondition of the *setTarget* action, this is not the case for the *goto* action. As *goto* is a durative action that may fail (if only because an enemy bot may kill the bot) it is not possible to specify the postcondition uniquely. Moreover, some of the “details” of going somewhere as, for example, the exact route taken may (but need not be) delegated to the behavioral layer; this means that most of the time only

through percepts the exact route can be traced. Therefore, it makes more sense in a dynamic environment that an agent relies on percepts that are made available by the environment to inform it about its state than on the specification of a postcondition. For this reason, when an action is selected, GOAL does not “block” on this action until it completes. Instead, upon selection of an action, GOAL sends the action command to the environment and then simply continues executing its reasoning cycle; this design explicitly allows for *monitoring* the results of executing the action command *while it is being performed* by the bot in the environment. For some actions, among which the `goto` action, the interface has been designed such that specific monitoring percepts are provided related to events that are relevant at the cognitive level. The moving state percepts `stuck`, `moving`, and `reached` are examples that illustrate how an agent may conclude the `goto` action has failed, is ongoing, or has been successful. The setup of sending an action command to the environment while continuing the agent’s reasoning cycle also allows for *interrupting* the action if somehow that seems more opportune to the agent; it can simply select a `goto` action with another target to do so.

This discussion also clarifies that providing an action specification in an agent programming language like GOAL in dynamic environments is more of a (pragmatic) design issue than a task to provide a purely logical analysis and specification of a domain. It would require unreasonably complex specifications to handle all possible effects whereas perception allows for much more effective solutions.<sup>3</sup> The action specification for `goto` has been setup in such a way in the example program, however, that another `goto` action is only selected if the agent believes a position has been successfully reached in order to make sure that the agent does not change its mind continually (something which obviously will need to be changed in a truly multi-agent setting where the bot can get killed; the example program is mainly used here for illustrative purposes).

The previous discussion will have made clear the importance that perception has for controlling bots in a real-time strategy game such as UT2004. Rules to process percepts (as well as possibly messages sent by agents) are part of the **perceptrules** section of a GOAL agent. In our example, the first percept rule stores all `pickup` positions in the belief base whereas the second one stores the movement state.

Though this agent is simple it does show that it is relatively simple to write an agent program using the interface that does something useful like collecting pills. Information needed to control the bot at the knowledge level is provided at start-up such as where pickup locations are on the map. The code also illustrates that some of the “tasks” may be delegated to the behavioral layer. For example, the agent does not compute a route itself but delegates determining a route to

---

<sup>3</sup> To be sure, we do not want to suggest that these remarks provide a satisfactory or definite solution for these issues; on the contrary, there remain many issues for future work. It does make clear, however, that in simulated environments such as games some of these issues can be resolved by the design of a specific perceptual interface, as we have done.

pickup navigation point. One last example to illustrate the coordination between the agent and the bot routines at lower levels concerns the precondition of the goto action. By defining the precondition as in Figure 2 (which is a design choice not enforced by the interface), this action will only be selected if a previously initiated goto behavior has been completed, indicated by the `reached` constant.

```

main: unrealCollector { % simple bot, collecting special items, and setting shooting mode
  beliefs{
    targets([]). % remember which targets bot is pursuing
    moving(triple(0,0,0), triple(0,0,0), triple(0,0,0), stuck). % initial physical state
  }
  goals{
    collect. targets([all]).
  }
  program{
    % main activity: collect special items
    if goal(collect), bel(pickup(UnrealLocID,special,Type)) then goto([UnrealLocID]).
    % but make sure to shoot all enemy bots if possible.
    if bel(targets([])) then setTarget([all]).
  }
  actionspec{
    goto(Args) {
      % The goto action moves to given location and is enabled only if
      % a previous instruction to go somewhere has been achieved.
      pre { moving(Pos, Rot, Vel, reached) }
      post { not(moving(Pos, Rot, Vel, reached)) }
    }
    setTarget(Targets) {
      pre { targets(OldTargets) }
      post { not(targets(OldTargets)), targets(Targets) }
    }
  }
  perceptrules{
    % initialize beliefs with pickup locations when these are received from environment.
    if bel( percept(pickup(X,Y,Z)) ) then insert(pickup(X,Y,Z)).
    % update the state of movement.
    if bel(percept(moving(Pos, Rot, Vel, State)), moving(P, R, V, S))
      then insert(moving(Pos, Rot, Vel, State)) + delete(moving(P, R, V, S)).
  }
}

```

Fig. 2. A very simple UNREAL-GOAL-agent collecting pills and setting targets

## 4.5 Implementation Issues

It is realized more and more that one of the tests we need to put agent programming languages to concerns performance. With the current state of the art it is not possible to control hundreds or even tens of bots in a game such as UT2004<sup>4</sup>. The challenge is to make agent programs run in real-time and to reduce the CPU load they induce. The issue is not particular for agent programming, [2] reports, for example, that Soar executes its cycle 30-50 times per second (on a 400MHz machine), which provides some indication of the responsiveness that can be maximally achieved at the cognitive level. Although we recognize

<sup>4</sup> Part of the reason is UT2004: increasing the number of bots also increases the CPU load induced by UT2004 itself.

this is a real issue, our experience has been that using the GOAL platform it is possible to run teams that consist of less than 10 agents including UT2004 on a single laptop. Of course, a question is how to support a larger number of bots in the game without sacrificing performance. Part of our efforts therefore have been directed at gaining insight in which parts of a BDI agent induce the CPU load. The issues we identified range from the very practical to more interesting issues that require additional research; we thus identify some topics we believe should be given higher priority on the research agenda. Some of the more mundane issues concern the fact that even GUI design for an integrated development environment for an agent programming language may already consume quite some CPU. The reason is quite simple: most APLs continuously print huge amounts of information to output windows for the user to inspect, ranging from updates on the mental states to actions performed by an agent. More interesting issues concern the use and integration of third-party software. For example, various APLs have been built on top of JADE [27]. In various initial experiments, confirmed by some of our colleagues, it turned out that performance may be impacted by the JADE infrastructure and performance improves when agents are run without JADE (although this comes at the price of running a MAS on a single machine the performance seems to justify such choices). Moreover, as is to be expected, CPU is consumed by the internal reasoning performed by BDI agents. Again, careful selection of third-party software makes a difference. Generally speaking, when Prolog is used as reasoning engine, the choice of implementation may have significant impact. Finally, we have built on top of *Pogamut* to create a behavioral controller for UT bots. Measuring the performance impact of this layer in the architecture illustrated in Figure 1 that connects GOAL through various layers to UT2004 is complicated, however; to obtain reasonable results for this layer using e.g. profilers therefore remains for future work.

In retrospective, we have faced several implementation challenges when connecting to UT2004 using EIS. EIS though facilitated design of a clean and well-defined separation of the agent (programmed in GOAL) and the behavioral layer (the UnrealGOALBot) to the UNREAL-AI-engine. The strict separation of EIS between agents as percept-processors and action-generators and entities as sensor- and effector-providers facilitated the design. We also had in mind right from the beginning that we wanted to use the UT2004-interface in order to provide the means for comparing APL platforms in general. Since support for EIS is easily established on other platforms we have solved this problem as well, by making the interface EIS-compliant.

## 4.6 Applications

The developed framework will be used in a student project for first year BSc. students in computer science. Before the start of the project, students will have had a course in agent technology where Prolog and GOAL programming skills are taught. The students are divided into groups of five students each. Every group will have to develop a team of GOAL agents that control UT bots in a CTF scenario where two teams attempt to steal each other's flag. In this



scenario, students have to think about how to implement basic agent skills regarding walking around in the environment and collecting weapons and other relevant materials, communication between agents, fighting against bots of the other team, and the strategy and team work for capturing the flag. The time available for developing the agent team is approximately two months, in which each student has to spend about 1 to 1,5 days a week working on the project. At the end of the project, there will be a tournament in which the developed agent teams compete against one another. The grade is determined based on the students' report and their final presentation. The purpose of the project is to familiarize students with basic aspects of agent technology in general and cognitive agent programming in particular, from a practical perspective.

Designing the interface at the appropriate level of abstraction as discussed above, is critical for making the platform suitable for teaching students agent programming. If the abstraction level is too low, students have to spend most of their time figuring out how to deal with low-level details of controlling UT bots. On the other hand, if the abstraction level is too high (offering actions such as *win the game*), students hardly have to put any effort into programming the GOAL agents. In both cases they will not learn about the aspects of agent technology that were discussed above.

## 5 Conclusion and Future Work

As is well-known, the UNREAL engine is used in many games and various well-known research platforms such as the USARSIM environment for crisis management that is used in a yearly competition [9]. We believe that the high-level Environment Interface that we have made available to connect agent platforms with UT2004 will facilitate the connection to other environments such as USARSIM as well. We believe the availability of this interface makes it possible to connect arbitrary agent platforms with relatively little effort to such environments which opens up many possibilities for agent-based simulated or gaming research. This is beneficial to put agent technology to the test. It will also make it possible to research human-agent mixed systems that control bots in UT2004.

The interface and architecture for connecting GOAL to UT2004 have been used successfully in a large student project at Delft University of Technology with 65 first-year BSc students that were trained to program GOAL agents first in a course on multi-agent systems. The multi-agent systems that were developed by the students competed against each other in a competition at the end of the project. Some lessons learned and an analysis of the agent programs that were written are reported in [28]. The project has resulted in many insights on how to design the agents controlling bots themselves, as well as on how to improve some of the associated tools and methodologies for authoring agent behavior. At the moment of writing, we are in the process of migrating the code of the behavioral layer based on Pogamut 2 to the new, redesigned Pogamut 3 [29].

The connection of an agent programming language for rational or BDI agents to UT2004 poses quite a few challenging research questions. A very interesting

research question is whether we can develop agent-controlled bots that are able to compete with experienced human players using the same information the human players possess. The work reported here provides a starting point for this goal. Even more challenging is the question whether we can develop agent-controlled bots that cannot be distinguished by experienced human players from human game players. At this stage, we have only developed relatively simple bots but we believe that the interface design enables the development of more cognitively plausible bots.

As noted in [30] and discussed in this paper, efficient execution is an issue for BDI agents. By increasing the number of bots and the number of agents needed to control these bots performance degrades. A similar observation is reported in [31]. Although it is possible to run teams of GOAL agents to control multiple bots, our findings at this moment confirm those of [30]. We believe that efficiency and scalability are issues that need to be put higher on the research agenda.

## References

1. Brom, C., Gemrot, J., Bida, M., Burkert, O., Partington, S.J., Bryson, J.: POSH Tools for Game Agent Development by Students and Non-Programmers. In: Proc. of the 9th Computer Games Conference (CGAMES 2006), pp. 126–133 (2006)
2. Laird, J.E.: Using a computer game to develop advanced ai. *Computer* 34(7), 70–75 (2001)
3. Patel, P., Hexmoor, H.: Designing Bots with BDI Agents. In: Proc. of the Symposium on Collaborative Technologies and Systems (CTS 2009), pp. 180–186 (2009)
4. Norling, E., Sonenberg, L.: Creating Interactive Characters with BDI Agents. In: Proc. of the Australian Workshop on Interactive Entertainment (IE 2004) (2004)
5. Botprize competition, <http://www.botprize.org/> (Accessed 30, January 2010)
6. Davies, N., Mehdi, Q.H., Gough, N.E.: Towards Interfacing BDI With 3D Graphics Engines. In: Proceedings of CGAIMS 2005. Sixth International Conference on Computer Games: Artificial Intelligence and Mobile Systems (2005)
7. Wang, D., Subagdja, B., Tan, A.H., Ng, G.W.: Creating Human-like Autonomous Players in Real-time First Person Shooter Computer Games. In: Proc. of the 21st Conference on Innovative Applications of Artificial Intelligence (IAAI 2009) (2009)
8. Kaminka, G., Veloso, M., Schaffer, S., Sollitto, C., Adobbati, R., Marshall, A., Scholer, A., Tejada, S.: Gamebots: A flexible test bed for multiagent team research. *Communications of the ACM* 45(1), 43–45 (2002)
9. RobocupRescue, <http://www.robocuprescue.org/> (Accessed 30, January 2010)
10. Burkert, O., Kadlec, R., Gemrot, J., Bida, M., Havlíček, J., Dörfler, M., Brom, C.: Towards fast prototyping of iVAs behavior: Pogamut 2. In: Pelachaud, C., Martin, J.-C., André, E., Chollet, G., Karpouzis, K., Pelé, D. (eds.) IVA 2007. LNCS (LNAI), vol. 4722, pp. 362–363. Springer, Heidelberg (2007)
11. Behrens, T.M., Dix, J., Hindriks, K.V.: Towards an Environment Interface Standard for Agent-Oriented Programming. Technical report, Clausthal University of Technology, IfI-09-09 (September 2009)
12. Behrens, T., Hindriks, K., Dix, J., Dastani, M., Bordini, R., Hübner, J., Braubach, L., Pokahr, A.: An interface for agent-environment interaction. In: Proceedings of the The Eighth International Workshop on Programming Multi-Agent Systems (2010)

13. Kim, I.C.: UTBot: A Virtual Agent Platform for Teaching Agent System Design. *Journal of Multimedia* 2(1), 48–53 (2007)
14. Partington, S.J., Bryson, J.J.: The behavior oriented design of an unreal tournament character. In: Panayiotopoulos, T., Gratch, J., Aylett, R.S., Ballin, D., Olivier, P., Rist, T. (eds.) *IVA 2005. LNCS (LNAI)*, vol. 3661, pp. 466–477. Springer, Heidelberg (2005)
15. Jacobs, S., Ferrein, A., Ferrein Lakemeyer, G.: Unreal GOLOG Bots. In: *Proceedings of the 2005 IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, pp. 31–36 (2005)
16. Tweedale, J., Ichalkaranje, N., Sioutis, C., Jarvis, B., Consoli, A., Phillips-Wren, G.: Innovations in multi-agent systems. *Journal of Network and Computer Applications* 30(3), 1089–1115 (2007)
17. JACK: Agent Oriented Software Group, <http://www.aosgrp.com/products/jack> (Accessed 30, January 2010)
18. Laird, J.E., Newell, A., Rosenbloom, P.: Soar: An architecture for general intelligence. *Artificial Intelligence* 33(1), 1–64 (1987)
19. Laird, J.E., Assanie, M., Bachelor, B., Benninghoff, N., Enam, S., Jones, B., Kerfoot, A., Lauver, C., Magerko, B., Sheiman, J., Stokes, D., Wallace, S.: A test bed for developing intelligent synthetic characters. In: *Spring Symposium on Artificial Intelligence and Interactive Entertainment (AAAI 2002)* (2002)
20. Best, B.J., Lebiere, C.: Teamwork, Communication, and Planning in ACT-R. In: *Proceedings of the 2003 IJCAI Workshop on Cognitive Modeling of Agents and Multi-Agent Interactions*, pp. 64–72 (2003)
21. SWI Prolog, <http://www.swi-prolog.org/> (Accessed 30, January 2010)
22. Hindriks, K.V.: Programming Rational Agents in GOAL. In: *Multi-Agent Programming Languages, Tools and Applications*, pp. 119–157. Springer, Heidelberg (2009)
23. Hindriks, K.V.: *GOAL Programming Guide* (2010), Can be downloaded from <http://mml.tudelft.nl/~koen/goal>
24. Bordini, R., Dastani, M., Dix, J., Seghrouchni, A.E.F.: *Multi-Agent Programming Languages, Platforms and Applications*. Springer, Heidelberg (2005)
25. Bordini, R., Dastani, M., Dix, J., Seghrouchni, A.E.F.: *Multi-Agent Programming Languages, Tools and Applications*. Springer, Heidelberg (2009)
26. Newell, A.: The Knowledge Level. *Artificial Intelligence* 18(1), 87–127 (1982)
27. Bellifemine, F.L., Caire, G., Greenwood, D.: *Developing Multi-Agent Systems with JADE*. Wiley, Chichester (2007)
28. Hindriks, K.V., van Riemsdijk, M.B., Jonker, C.M.: An empirical study of patterns in agent programs: An UNREAL TOURNAMENT case study in GOAL. In: *Proceedings of the 13th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA 2010)* (2010)
29. Gemrot, J., Brom, C., Plch, T.: A periphery of pogamut: from bots to agents and back again. In: Dignum, F. (ed.) *Agents for Games and Simulations II. LNCS (LNAI)*, vol. 6525, pp. 19–37. Springer, Heidelberg (2011)
30. Bartish, A., Thevathayan, C.: BDI Agents for Game Development. In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, pp. 668–669 (2002)
31. Hirsch, B., Fricke, S., Kroll-Peters, O., Konnerth, T.: Agent programming in practise - experiences with the jiac iv agent framework. In: *Sixth International Workshop AT2AI-6: From Agent Theory to Agent Implementation*, pp. 93–99 (2008)

# A Periphery of Pogamut: From Bots to Agents and Back Again

Jakub Gemrot, Cyril Brom, and Tomáš Plich

Charles University in Prague, Faculty of Mathematics and Physics  
Malostranské nám. 2/25, Prague, Czech Republic

**Abstract.** Despite virtual characters from 3D videogames – also called bots – seem to be close relatives of intelligent software agents, the mechanisms of agent reasoning are only rarely applied in videogames. Why is this? One possible reason is the incompatibility between representations used by agent decision making systems (DMS) and videogame worlds, as well as different handling of these representations. In recent years, we developed Pogamut, which is a toolkit for coupling videogame worlds with DMSs originating within the agent oriented research as well as other disciplines, allowing for controlling in-game characters by these DMSs. To this end, Pogamut features an interface bi-directionally bridging the “representational gap” between a game world and an external DMS. This paper conceptualises functionality of this interface based on our experience with connecting Pogamut to various game worlds, most notably Unreal Tournament 2004. We present a general abstract framework, which verbalises requirements an agent researcher must fulfil in order to employ his/her reasoning mechanism for controlling in-game virtual characters. This paper also reviews Pogamut, which the researcher can utilise.

**Keywords:** videogames, agents, action selection, Unreal Tournament.

## 1 Introduction

After years of stagnation, the field of artificial intelligence (AI) for videogames seems to have caught second wind [1]. Results achieved in disciplines such as planning [2], evolutionary computation [3], or stochastic modelling [4] inspire new solutions and approaches to known problems. Can knowledge accumulated by the multi-agent systems (MAS) community during the last decade be, to some extent, also employed in the context of 3D videogames? This idea stems from the fact that it may seem, superficially, that fields of gaming AI and MAS study entities of the same kind.

Entities studied by the MAS community are often called *intelligent software agents* [5]. Creatures employed in the gaming AI field are typically called *bots*, *non-player characters* (NPCs), *virtual characters* or *virtual agents*. To avoid ambiguity, we will strictly use the term *agents* for the former entities while *bots* for the latter.

The broad field of MAS studies includes many subfields, such as multi-agent communication, cooperation and negotiation, learning, and agent reasoning. It can be contemplated about how the knowledge gained in each of these subfields can be

employed in the context of videogames. This paper is concerned with reasoning of a single or few agents, leaving the other subfields for future work.

The field of videogames is as diverse as the MAS field. In this paper, when speaking about videogames, we restrict ourselves only to 3D games modelling individual persons such as first-person shooters (FPS) or role-playing games (RPG). For the purposes of this paper, we also include 3D virtual reality simulations, such as crowd simulations or virtual storytelling systems, into the definition of a “3D game”. That is, virtual characters inhabiting these simulations and carrying out multiple goals, often called intelligent virtual agents, will be considered as bots here. However we will not use the term bots for conversational characters, unless they are fully embodied within a virtual world and have multiple goals besides chatting (e.g., a sport game commentator is not considered as a bot here unless he can also shoot, run, etc.). Similarly, we exclude from the definition of a “3D game” games without individual persons, e.g., statistical strategies and logical games, even though decision processes in these games can be conceived as agents by some (e.g., [6]). We also exclude games simulating only or predominantly vehicles, such as flight simulators and racing games, even though when these vehicles are controlled by a computer, they may be called bots by some.

Now, we can rephrase and refine the question this paper aims at analysing: *Can knowledge accumulated in the MAS field concerned with agent reasoning be used for reasoning of individual bots or a couple of bots?* That means, can developers of bots employ MAS knowledge representations, reasoning algorithms, or goal-oriented software architectures such as Jason [7], Jack [8] or Jadex [9]? How to achieve this? What are the obstacles, what are the drawbacks and benefits?

This paper does not provide conclusive answers. Instead, it proposes the way towards practical utilisation of MAS knowledge in the videogame domain, which leads through connecting these architectures to game engines, enabling empirical experiments. The paper explains what exactly it means to establish such connection and offers a robust toolkit we have developed to help in this work.

An agent decision making system (DMS) can be connected to a game engine either internally or externally (see also [10] on this point). *Internal* connection means integrating a DMS into a game engine when the engine’s source code is available. However, the source code of games is available only rarely and typically for ten years old games (with some exceptions). Fortunately, several modern games allow for information exchange between their worlds and an external system, enabling a researcher to couple his/her architecture with a game *externally*, in most cases on a client-server basis. However, the researcher should be aware of two stumbling blocks: a) the way how bots are controlled internally (i.e., from the game within) may differ from the way how the bots can be controlled externally, b) bot DMSs use different input/output information than typical agent DMSs, no matter whether an agent DMS is connected internally or externally (mind the distinction between a bot DMS and an agent DMS<sup>1</sup>). While Point (a) is merely technical, Point (b) presents a deep conceptual difference. To understand this difference, Section 2 of this paper will explain the main distinctions between agents and bots in detail.

---

<sup>1</sup> Whereas a bot DMS is a native controlling mechanism hardwired within the game engine, an agent DMS is a stand-alone application its creator is claiming it is MAS principles compliant.

When aiming at connecting an agent DMS to a game, it is an advantage to know about what can be expected from game engines; which information is available and which it is not, and how to acquire the desired information. To enable the reader to understand these points, Section 3 formally conceptualises game engines, providing a model of a bot DMS, game engine and information it holds. Based on this formalisation, Section 4 discusses the issue of connecting an agent DMS.

Sections 3 and 4 did not appear from thin air. On the contrary, we have been working for several years on Pogamut, which is a robust toolkit for coupling videogame worlds with external DMSs [11]. This toolkit has been widely used, both by us [e.g., 12, 13] as well as others for the purpose of prototyping gaming algorithms [14] or for prototyping BDI-based agents acting in real-time, dynamic and complex environments [15]. The formalisations are derived based on our experience we gained during our work on Pogamut. Section 5 reviews Pogamut and presents a particular instantiation of the formal model, a multi-layered AgentSpeak(L)-based DMS [16] coupled to the Unreal Tournament 2004 game (UT 2004) [17]. The most important point is that Pogamut can be utilised for connecting other agent DMSs to videogames in an “out of the box” fashion (see also [15] on this point). Section 6 documents that Pogamut can be used not only in the context of the UT 2004 game: it reviews our work in progress on extending Pogamut to operate with Virtual Battle Space 2 [18], which is a multi-agent military simulator, as well as connecting Pogamut to StarCraft [19] and Defcon [20], strategy games.

## 2 Bots Are Not Agents

The border between bots (as defined above) and agents (as understood by MAS community) is not clear-cut, but there are several traits that help understand the difference. Arguably, the two most important traits are believability and embodiment. Firstly, bots should be *believable*, which is the ability to convey the *illusion* of reality [21] by whatever means necessary. In this aspect, bots and agents differ; while the ideal of agents is strong autonomy, which does not necessarily imply believability in terms of [21], believable bots need not be strongly autonomous. Secondly, bots are embodied; they have virtual bodies subject to constraints of their 3D virtual worlds<sup>2</sup>.

Because of their embodiment, and this is crucial, bots require different information inputs about their surroundings than typical agent-oriented DMSs work with. Because of real-time constraints, bots have to cope with ever-changing virtual world rapidly, which requires acquiring and processing of external information in a timely fashion.<sup>3</sup> We will now elaborate on these two points.

One useful way of categorising incoming information is based on the *level of abstraction*. It is useful when the information about some aspects of a bot’s surrounding

---

<sup>2</sup> Recall that we call intelligent virtual agents bots here.

<sup>3</sup> Note that the meaning of the word “rapid” differs from the MAS community’s use: in gaming industry, constant algorithms tend to be considered as fast while polynomial are considered to be slow in general, let alone exponential (of course, this depends on a particular situation). For instance, traversing expression trees of dynamic lengths built out of domain specific language checking for event triggers, which is linear, is found to be slow [22]. That work suggests using only expression trees that fit into a rather small pre-allocated array.

is provided in an abstract manner while other aspects are presented in a low level way, i.e., more akin to inputs from robotic sensors. Thus, at the same time, a bot can get information like “a projectile is coming from 170°”, “there is a wall or something 2.34 meters at 54°”, “this is a position from which you can shoot”. The important point is that the level of abstraction is not determined by an ideal of psychological plausibility or the logical coherence of the DMS, but by technical rationale, including the real-time constraints and the believability requirement.

Another useful categorisation divides information into *static* and *dynamic* [10]. Static information is bound to properties of the virtual environment that do not change in the course of the simulation. For instance, whether a place is suitable as a cover in respect to a sentry gun fixed inside a bunker depends on the outlook of the sentry gun and the landscape. In an environment that does not change its topology, this information is known during the design time, i.e., the information has a static context, and can be precompiled. On the other hand, the information where defenders of a bunker have the weakest spot depends on their current positions, their patrol behaviour etc., i.e., it has a dynamic context implying the necessity to compute or acquire it during runtime. This distinction is crucial both technically and conceptually.

Now, to be able to contemplate on how an agent DMS coupled with a game engine can access information, we need to know how data are provided by engines. In our experience, game engines tend to export only information that is available via regular bots’ access mechanisms (because providing different access mechanisms would present only additional unnecessary development). Thus, we have to take a look on how bots sense their worlds.

Due to real-time constraints, game developers must balance the necessity of bots having to perform *active* sensor querying against bots’ automatic, i.e., *passive*, event notifications whenever a respective event occurs in the game engine. Note that the active vs. passive distinction may not mirror the static vs. dynamic dichotomy. The rationale behind the active vs. passive mechanisms is to automatically notify bots about important events that the bots would check anyway, such as “bot has been killed” or “bot has hit a wall” – this is the passive sensing. On the other hand, there are many events that bots need to know only from time to time. It is more efficient when bots actively request information about these events only when the information is really needed.

What does it imply for an external agent DMS? Generally, there are three ways how information can be obtained. 1) The information that is passively sensed by bots is also automatically exported by the game engine (*push* strategy). 2) The information that can be actively requested by bots can also be requested from the engine (*pull* strategy). 3) The information that is not provided by the game engine itself but it may be inferred from existing information (*inference* strategy).

Here is a distinction between agents and bots concerning these information access strategies: While bot DMSs tend to access information by several special purpose mechanisms, capitalising on both push and pull strategies, agent DMSs tend to use one generic mechanism only. This fact alone poses technical troubles for some agent DMSs. For instance, many goal-oriented agent architectures, such as AgentSpeak(L) [23] derivatives, require for underlying agent worlds to keep them automatically informed about events, which corresponds to the push strategy. For these architectures, the lack of ability to cope with pull and inference strategies must be compensated.

Having the notion of key differences between bots and agents, the paper continues with discussing game engines in general. This will help the reader to better understand the prerequisites of connecting an agent DMS to a game engine.

### 3 Game Engines

This section presents a generic framework for understanding game engines (GE). The section starts in an informal tone and continues with a formal definition of a GE, functioning of a bot DMS, and relations between various data kept by the engine.

#### 3.1 GEs Informally

“A game engine is a software system designed for the creation and development of video games. Game engines provide a suite of visual development tools in addition to reusable software components. These tools are generally provided in an integrated development environment to enable simplified, rapid development of games in a data-driven manner.” [24, see also 25] A GE itself is not a game, rather a middleware used by the game developers, who may arbitrarily extend it to suit the game’s needs. As a middleware, it usually empowers the developers with the ability to script game rules using interpreted languages such as Lua [26], Python [27] or a proprietary language such as UnrealScript [28].

The high level task of a GE is to simulate a game’s virtual world in (nearly) real-time providing smooth visualisation to players while reflecting their actions. One of the challenges GEs are facing is to arbitrate available CPU and GPU power between:

- 1) game visualization, e.g., performing animations, managing polygons, textures and shader programs in the graphic card’s memory, etc.;
- 2) (simplified) physics simulation, e.g., computing the trajectories of moving objects, performing collisions and deformations, etc.;
- 3) game mechanics, e.g., triggering game events at correct time, executing scripted situations, etc.;
- 4) artificial intelligence, e.g., providing believable behaviour for interactive game objects such as bots.

Game engines tend to prioritize these issues in the given order. Smooth visualization is preferred over AI computations, which do not affect the simulated world every frame. Thus AI computations are interleaved with scene rendering, physics engine etc., and in most cases, are not given extensive computational facilities.

In the previous section, we discussed how information can traverse from a GE to a bot. Now, we are going to discuss how these data are stored within the GE and how they are managed.

**GEs as managers of facts.** GEs can be conceptualised as managers of game facts (grounded formulae of first order logic), which are true in the certain point of time. GEs represent them as data structures of a native programming language. GEs can also be seen as rule engines that transform given facts as the game proceeds. A GE maintains a virtual clock, which measures time in ticks. Each tick, the GE transforms



game facts according to game rules that are encoded inside the GE or written outside by game developers in a language the GE can interpret. We will refer to this transformation as *TICK* function.

The following (non-exhaustive) list presents examples of such game rules:

- 1) Physics. Change locations of objects, the speed of which is greater than 0, as if 0.05 seconds has passed, and compute collisions along the way.
- 2) Game mechanics. If a rocket explodes, compute damage and process the damage (lower health) to all bots and players in the radius of 500 m.
- 3) Trigger. When a player steps on a jump pad, apply force of 1000 N to his avatar in the direction perpendicular to the jump pad.
- 4) Bot API: If a bot issues a move command, change the bot's speed to 20 km/h.

Section 2 has divided a bot's information into *dynamic* and *static*. This dichotomy comes from the GE itself as it is the GE which defines, which information can change in the course of the simulation and which can not. Static information (facts) contain the GE's configuration, together with the underlying geometry of the land, game triggers, events or programs in a language that GE can interpret, etc. Examples of dynamic information (facts) includes the current number of bots and items in the game along with their state, i.e. position, current speed, animation in progress, etc.

The distinction between static and dynamic facts is technical as well as conceptual. Technically, a DMS must handle dynamic facts differently, i.e., it must be ready to handle their change and offer the developer a way to act upon such changes. Conceptually, the more the game world is dynamic, the more sophisticated algorithms must be employed. For example, there are games where objects forming the borders of the virtual world are destructible by bots and players, e.g., it is allowed to blast a hole into a building's wall, which makes the navigation mesh a dynamic fact. Thus a dynamic path finding algorithm, e.g., D\* [29], must be used instead of a classical one.

Now, let us proceed to the question how these facts are stored within a GE.

**Game facts division.** Game facts that are true about the game in a given time can be categorised into three groups. We will refer to these groups as *game facts classes*.

1) A GE usually provides an API to access all game facts it stores within its internal data structures, such as level geometry or a bot's level of health. Accessing these facts is computationally efficient. We will refer to these facts as *Class 1 facts*.

2) Additionally, there are facts that can be computed on request by invoking a method of an API that uses *Class 1 facts* to infer new facts. Such facts are, e.g., ray cast results or a path to a distant location. We will refer to these facts as *Class 2 facts*.

3) Finally, there are facts that can not be obtained through the GE's API but that are algorithmically inferable, such as the shortest path a bot should follow to collect all items in the world (leads to the well-known Travelling Salesman Problem). We will call these facts *Class 3 facts*.

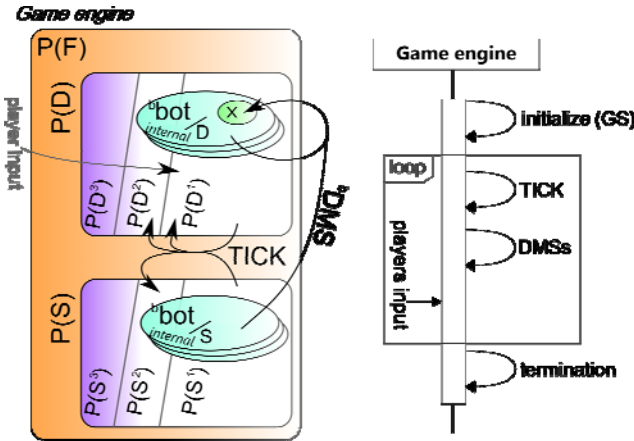
**Sending requests to GEs.** So far, only the information flow from a GE to a bot has been discussed. Fortunately, the other way is easier. A GE usually offers a set of methods that can be invoked by bots, e.g., `GoTo(Location)`, `Shoot(Actor)`, or `GetPath(Location)`. These methods are of two types: 1) actions the bot's body should carry out in the game world (`GoTo`, `Shoot`, etc.), and 2) computation requests

that induce facts not present implicitly in GE’s data structures, i.e. Class 2 facts (e.g. `GetPath`). Both commands and computation requests may last (and usually last) several *TICKS* before they are finished. The number of *TICKS* depends on the request’s complexity, e.g., say “Hello!” command will take less time to perform than `GoTo(Hospital)`. Similarly, a ray cast request will finish well before the A\* algorithm finds out that a requested path does not exist.

Two major issues linked to the requests are to be handled by an agent DMS. Firstly, the DMS should be able to handle the lag between a computation request and its returned result. Especially the path requests should be made in advance. Secondly, GEs may not report success or failure of an action. In these cases, a DMS must actively watch over a bot’s related facts to infer whether the action is being carried out as expected (e.g., has the bot just hit an obstacle?). If a deviation from an expected result is detected, actions to compensate should take place. Such success/failure reporting is required, e.g., by BDI systems. Whenever the bot has an intention to go to the hospital, it needs to know whether the `GoTo(Hospital)` command has been executed successfully or failed in order to maintain, delete or re-plan the intention.

### 3.2 GE (More) Formally

This part of the paper summarises formally description of GEs in order to facilitate thinking about connecting agent DMSs to GEs. Therefore, the definition intentionally emphasises management of game facts and their accessibility, but not the issue of bots’ action selection and carrying out action commands by GEs. Note also that GEs may differ in important implementation details; what we present here is only our view of GEs. We are not aware of any widely accepted formal definition of a GE.



**Fig. 1.** Visualization of TICK and DMS functions together with the sequence diagram of subsequent calls. The mark “X” (on the left figure) denotes a bot’s mental states and requests, i.e.,  $P^b(VM) \times P^b(R)$ . The bots in the figure are marked as “internal”, i.e., they are native to the GE and controlled by bot DMSs. This labelling will become important later on in Figure 2.

**Definition 1.** (*Game engine, managed facts*) *Game engine*

$GE = (F, I, P, B, TICK, GS, T)$  is characterized by:

- **Facts**  $F$ . This is the set of all ground formulae that can be possibly true about the game during some moments.
  - The set  $F$  consists of three classes:  $F = F^1 \cup F^2 \cup F^3$ .
    - **Class 1** game facts  $F^1 \subseteq F$  is a set of all ground formulae that (a) can be possibly true about the game and (b) they are stored in appropriate data structures iff they are true.
    - **Class 2** game facts  $F^2 \subseteq F$  is a set of all ground formulae that can be inferred by any inference function  $i \in I$  (see below), i.e.,  $F^2 = \bigcup_{i \in I} \text{range}(i)$ .<sup>4</sup>
    - **Class 3** game facts  $F^3 \subseteq F$  is a set of all ground formulae that can be possibly true about the game, but they are never stored within a  $GE$ 's data structure and they are not obtainable through any inference function  $i \in I$ .
  - Additionally,  $F$  can be divided amongst **dynamic** facts  $D = D^1 \cup D^2 \cup D^3 \subseteq F$  and **static** facts  $S = S^1 \cup S^2 \cup S^3 \subseteq F$ , where  $F^i = D^i \cup S^i$ ,  $D^i \cap S^i = \emptyset$ . Dynamic facts can be changed between  $TICK$ s (see below), whereas static facts remain the same throughout the whole simulation.
- **Inference functions**  $I$ . This is a set of all functions that the  $GE$  can use to infer facts of Class 2:  $i \in I : P(F^1) \times P(F^2) \rightarrow P(F^2)$ .
- **Embedded bots**  $B$ . This is a set of all possible bots in the game. See Def. 2.
- **Embedded players' abstractions**  $P$ . This is a set of all possible manifestations of players in the game. See below.
- **Function**  $TICK$ . This function  $TICK : P(S^1) \times P(D^1) \rightarrow P(D^1) \times P(D^2) \times P(S^2)$  is used to advance the game situation.
  - Concrete facts that can be changed by  $TICK$  function, i.e.,  $(\text{def}(TICK) \cup \text{range}(TICK)) \setminus S^1$  are called **managed facts**.
- **Game settings**  $GS$ .  $GS = (\omega, \beta, \pi)$  is characterized by
  - initial game situation  $\omega \in P(F^1)$ ,
  - list of bots  $\beta \subseteq B$  and players  $\pi \subseteq P$  connected to the game.
- **Terminal states**  $T$ . This is a list of game situations that terminates the  $GE$ ,  $T \subseteq P(F^1 \cup F^2)$ .

<sup>4</sup> In fact, a  $GE$  may cache results of inference functions making these facts Class 1 facts. However, these functions often compute dynamic facts, which are being quickly invalidated as the simulation advance forward, therefore we will omit caching.

**Definition 2.** (*Bot, bot's facts*) Bot  $b \in B = (VB, P, VM, R, DMS)$  is characterized by:

- the set of **virtual body facts**  ${}^bVB \subseteq F^1$ ,
- the set of facts that can be possibly **perceived**  ${}^bP \subseteq F^1 \cup F^2$ ,
- the set of **mental facts**  ${}^bVM \subseteq F^1$ ,
- the set of **requests** the bot can possible make  ${}^bR \subseteq D^1$ ,
- **decision making system** function  
 ${}^bDMS: P({}^bVB) \times P({}^bP) \times P({}^bVM) \times P({}^bR) \rightarrow P({}^bVM) \times P({}^bR)$ ,
- the set of all **bots' facts**  ${}^bbot = P({}^bVB) \cup P({}^bP) \cup P({}^bVM) \cup P({}^bR)$ .

Abstractions of players from Def. 1 can be defined similarly to the bot's definition (Def. 2) except the *DMS* function is unknown and brings uncertainty to the game.

Def. 1 and 2 result from the informal discussion from Section 2 and 3.1. The last thing to explain is how the function *TICK* and *DMS* work.

$TICK: P(S^1) \times P(D^1) \rightarrow P(D^1) \times P(D^2) \times P(S^2)$  is being used to compute a next game state ( $P(D^1)$  from  $\text{range}(TICK)$ ) as well as requests made by bots ( $P(S^1) \times P(D^1)$ ). Concerning  $\text{dom}(TICK)$ , inside  $P(S^1), P(D^1)$ , there lies currently true facts about all bots' virtual bodies, true facts bots are currently informed about, as well as their mental states and their active requests. The *TICK* function applies game rules to advance the game's progress a small fraction of time forward, i.e., it replaces the current  $t \in P(D^1)$  of true facts with a new set. Additionally, it handles the bots' requests by removing them and providing DMSs with  $P(S^2), P(D^2)$  facts that are accessible for a short period of time (they are invalidated in next few *TICK*s).

$DMS: P({}^bVB) \times P({}^bP) \times P({}^bVM) \times P({}^bR) \rightarrow P({}^bVM) \times P({}^bR)$  assesses the current state of the virtual body  $P({}^bVB)$ , facts that are known to the bot  $P({}^bP)$ , the current DMS state  $P({}^bVM)$  and unfinished bot's requests  $P({}^bR)$ , and produces zero or more requests while altering its own state. Note that facts  $P({}^bP)$  and  $P({}^bVB)$  are computed by the *TICK* function. Conceptually, the *TICK* function also computes the *DMS* functions; however, for intelligibility, it is better to conceive these two functions as separate.

The *DMS* function model suggests what needs to be done in order to connect an external agent DMS to a GE. This issue will be further elaborated in the next section.

## 4 Connecting an Agent DMS to a GE

Arguably, the final goal of the whole endeavour is that game industry starts using some ideas stemming from the subfield of agent reasoning in videogames or even employs a whole agent DMS for the purpose of controlling in-game bots. As already said, an agent DMS can be coupled with a game either internally or externally. In a final application, it is likely that bots will be controlled internally – this is more efficient than external connection both in terms of memory and processor requirements (see [10] for a different view). However, in our opinion, before this can

happen, it is necessary to connect several agent DMSs to a single GE and evaluate them. These DMSs should be compared with existing AI techniques currently used by the game industry, such as finite state machines [30], behavioural trees [31] and simple planning [22]. They should be compared along several lines, most notably in terms of computationally efficiency, improvement of bots' cognitive abilities, and design time. For instance, before these DMSs can start to compete with industry solutions, they should accommodate multiple bots at the same time and they should be intelligible for game designers who may not have strong AI knowledge.

All of this means that for the time being, the goal is not to employ an agent DMS within a videogame to be marketed, but to implement several prototypes. Should they employ the external or internal coupling? This section evaluates these two approaches and argues that the external one is better for prototyping purposes. It also makes it explicit what the external coupling means in terms of the formalism from Sec. 3.

Be it internally or externally, both approaches require researchers to write additional code binding a DMS and a GE together. It would be an advantage if the researchers can use a middleware facilitating this infrastructure work. It would be even better, for the purposes of evaluation, if all of these researchers use a common middleware. Section 5 proposes that Pogamut can be used for this purpose.

#### 4.1 Internal or External Coupling?

**Integrating a DMS into a game engine.** Integration of an existing DMS into a GE means to re-implement the existing solution inside the framework (code base) of the engine.

The advantages are as follows:

- The integration may utilise all features of the GE.
- The implementation may blend with the original code of GE resulting in optimal performance, which suits the needs of the game industry.

The disadvantages are as follows:

- The DMS must be re-implemented in the native programming language of the GE.
- The code of the engine must be opened.
- The solution will be GE dependent and may not be reusable with different engines.
- The implementation could not be developed over a common architecture that adapts the GE to the DMS, which will make empirical comparison of different DMSs troublesome.

Note that in [10], this is called a server-side approach. That work also claims that the DMS must be completely synchronised with the GE, being integrated in the default game loop. However, this is not the case. Even internally coupled DMS can perform decisions in several time steps if it is able to interrupt and resume its computations.

**Connecting DMS to GE externally.** The other way is to utilise existing DMS implementations and connect them to a GE externally via, for example, TCP/IP.

The advantages are as follows:

- The translation of game facts and requests on both the GE's and the DMS's sides can be reused.
- The creation of a general layer between the GE and the DMS can result in a common platform for empirical comparison of different DMSs.
- The general layer could be extended to comfort more GEs.
- The DMS and the GE may run on different computers, allowing for distributed simulations.
- The DMS can be implemented in a language favoured by the developer.

The disadvantages are as follows:

- Game facts and requests have to be exported and translated between the GE and the DMS, which results in worsened performance.
- The bot's reactions to events take longer because there is a round trip time between the DMS and GE.

Our experience gained during the implementation of Pogamut has shown that the disadvantages of the external approach are not so sever or troublesome. The translation of facts and requests between the GE and the DMS does not take such extensive time to harm the bot's reactive capabilities (i.e., Pogamut can easily communicate synchronously with several bots on 4Hz, while asynchronous messages are handled in milliseconds). Thus, we will continue only by assessing the external approach. We will briefly return to the internal approach in Sec. 6.

## 4.2 DMS as an External GE's Component

According to Def. 1, an internal bot DMS is a function  $DMS : P(^bVB) \times P(^bP) \times P(^bVM) \times P(^bR) \rightarrow P(^bVM) \times P(^bR)$ . To provide the same mechanism externally, we need to export facts about the virtual body and facts perceived by the bot from the GE to an external DMS and to provide the DMS with a way to pass requests to the GE. The bot's mental states need not be present inside the GE as the mental states are utilised only by a native DMS that will not be active. Instead, agent DMS may represent mental states derived by itself. Figure 2 depicts the desired model of a GE bound together with an external DMS.

When comparing Figure 2 with Figure 1, Figure 2 divides bots between internal and external. The internal bots are controlled by their bot DMS functions (native to the GE) whereas external bots are controlled by agent DMSs, which are modelled separately of GE. The sequence diagram on the right reflects this distinction. Figure 2 also contains many additional arrows between the GE and the agent: 1) load, 2) update, 3) requested, 4) requests, commands. These arrows denote the translation of facts and requests between the GE and the engine. The *load*, *update* and *requested* arrows are functions exporting facts from the GE and translating them to the representation used by the agent DMS. *Load* is invoked only at the beginning of the simulation, and it exports some (or all)  $S^l$  facts. Some  $S^l$  facts may stay managed by the GE and become later requested. Thus, the *requested* arrows realise the pull strategy (see Sec. 2) returning facts from  $D^l$ ,  $S^l$  and more importantly from  $D^2$ ,  $S^2$ . The *update* arrow realises the push strategy returning some facts from  $D^l$  regularly.

Note that technically, even external bots may use some AI algorithms implemented by the GE, algorithms operating with unexported facts, such as path-finding algorithms.

The *requests, commands* arrow depicts the way the agent DMS passes requests back to the bot inside the GE. Requests are managed inside  $D'$  class, i.e., as dynamic facts. Additionally, the agent DMS may implement supplementary inference algorithms to infer Class 3 facts as *inferred* arrow suggests.

Importantly, by introducing push and pull strategy, the GEs conceptually differ from simple worlds such as grid worlds, in which many AI algorithms are tested, where all information is usually present as  $F'$  facts but not  $F^2$  facts.

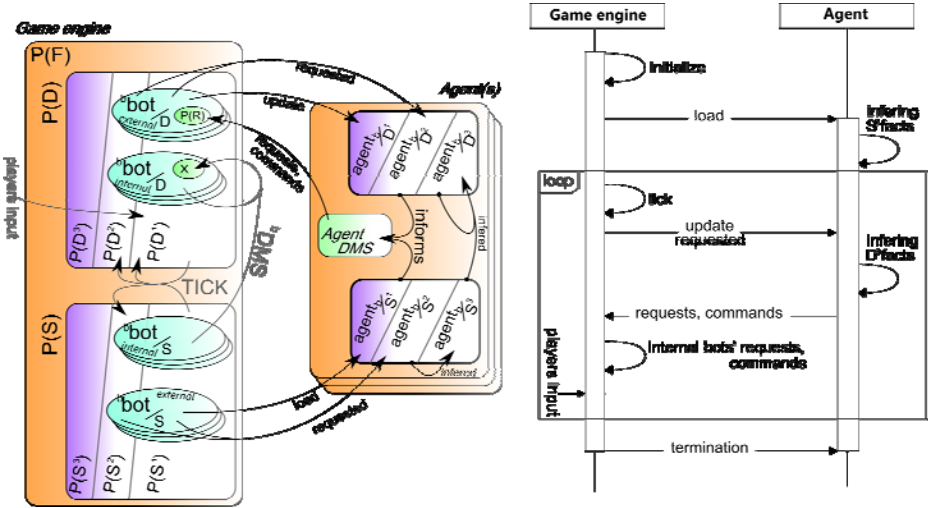


Fig. 2. The figure pictures the information flow between GE–DMS together with sequence diagram of GE–DMS interaction. The mark X has the same meaning as on Fig. 1.

### 4.3 Summary

This section has presented a model of a GE bound with an external DMS, capitalising on the conceptual framework from Section 3. As every abstraction, the model presented here may not fit an actual GE entirely, but we believe it is robust enough to embrace most of them and to provide useful guidance in coupling agent DMSs with GEs. The next section will look at this topic from more technical standpoint, introducing a particular implementation of the abstract model.

## 5 Pogamut 3 Platform, GE–DMS Mediation Layer

The purpose of this section is to present the architecture of Pogamut 3 as a mediation layer between a GE and an agent DMS. The development of such layer is technically hard by itself. First, developers need to understand the complex, often undocumented code of the GE before they can even start thinking about the translation of facts into

an agent DMS. Then, the developers' have to address many tedious technical issues, mostly out of the main scope of their work, such as development of a debugger. Many of such issues are addressed repeatedly, which is in most cases a waste of time.

For past four years, we have been developing the Pogamut platform, which provides general solutions for many of these issues, allowing developers to focus on their main goals – experimenting with various DMSs inside simulated 3D worlds provided by the game industry. We remark that Pogamut is already widely used [e.g., 12, 13, 14, 15]. A new major version, Pogamut 3, has been already released [11]. Importantly, Pogamut 3 is also suitable for education [32].

Pogamut 3 currently utilises the well-known Unreal Tournament 2004 action videogame [17]. The game features a lot of pre-built objects, maps, and a map editor, allowing for custom modifications of the original game content, including creating simple maps for experiments. Section 6 reviews our work in progress concerning bindings Pogamut 3 with more game engines.

Pogamut 3 is programmed in the Java language and is currently most suited for utilisation of DMSs implemented in Java, Python or Groovy language. The platform already allows to experiment with a few agent DMSs, namely POSH [33], ACT-R [34] and an AgentSpeak(L)-like system [16].

Pogamut 3 has been already discussed in depth in [11, 32, 35] where comparison with related work is given as well. This paper focuses more on the layers standing between DMSs and GEs. In the rest of this section, Pogamut's generic agent architecture implementing the ideas from Sec. 3 and 4 will be introduced and exemplified on an implemented AgentSpeak(L)-based agent [16], called here *AS agent*, demonstrating Pogamut's technical flexibility.

## 5.1 Architecture of Pogamut Agent

The generic architecture of Pogamut 3 agent is depicted on Figure 3. The architecture introduces a set of layers that shields an agent DMS from the low-level communication with a GE, taking care of the *load*, *update*, *requested* and *requests* arrows from Figure 2. These layers are: *WorldView*, *Working memory*, *Inference engine*, and *Reactive layer*. The layers were implemented by the AS agent using Java, tuProlog [36], RETE-engine Hammurapi rules [37], and again Java, respectively (tuProlog and Hammurapi were chosen due to their available Java implementation that fits nicely with Pogamut). Additionally, the AS agent employed our proprietary AgentSpeak(L)-like system [16] as the DMS.

The DMS selects actions to execute based on facts represented in Working memory. It is more comfortable to work with facts like “bot Tom is chasing bot Clara” at the level of DMS (and thus at the level of Working memory) rather than with facts like “bot Tom is at position  $\langle x, y, z \rangle$ ” and “bot Clara's speed is  $V$ ”. It is the job of Inference engine to infer the former kind of facts, i.e., Class 3 facts.<sup>5</sup>

The main component of the architecture, and the only mandatory, is *WorldView*. This component arose from the need to provide an abstraction of game facts for the

---

<sup>5</sup> We also tried to infer these facts inside the DMS directly, but, at least for our AS agent, it turned out that that was inefficient and specification of the inference rules was cumbersome. A separate RETE engine ([37]) turned out to be more suitable for this task. These practical aspects motivated the separate component for inferring Class 3 facts in our architecture.



other components of the architecture. WorldView has two major functions: 1) synchronising facts incoming from a GE as well as the other components, 2) providing these facts to multiple components at once. The component can be seen as a blackboard [38].

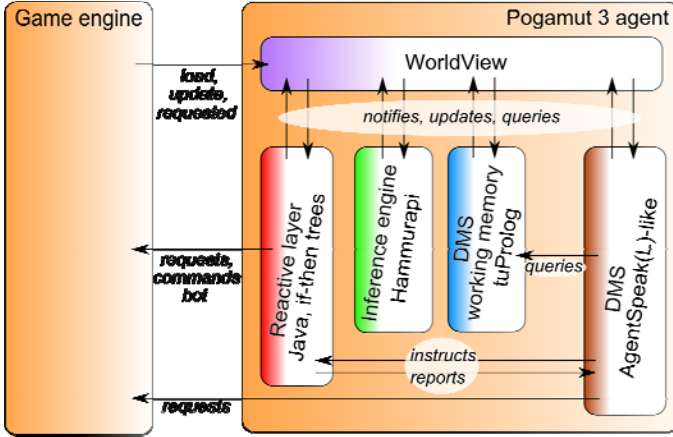


Fig. 3. High-level architecture of the Pogamut 3 agent implemented by the AS agent

It implements both *push* and *pull* strategy information retrieval, i.e., it is possible to query WorldView directly, or attach listeners that are informed whenever facts are updated. Importantly, this *update event model* of WorldView is flexible, in the sense that WorldView understands an ontology of Java objects that comes out of the Java class hierarchy. This works as follows. Let us assume that the virtual world simulated by the GE can be populated by items such as fruits and vehicles, but currently only apples and cars are supported. Concerning agents populating this world, this means that designers must provide a way for the agents to recognise items of these two kinds. When WorldView is used, designers have to create the following classes representing the objects' types and their categories: "item", "fruit", "vehicle", "apple", and "car". Utilising Java class inheritance, designers will further define that classes "vehicle" extends "item", "fruit" extends "item", "apple" extends "fruit", and "car" extends "vehicle". WorldView then propagates events according to this ontology; for instance, when an event happens on the object of class "apple", it is propagated also to "fruit" and "item". This makes the object model of WorldView flexible as any other agent's component may listen on all "fruit" events by attaching a listener on the class "fruit" instead of attaching the listener on every instance of fruit. This feature is not present in most GEs today.

The application of WorldView for various knowledge representations is straightforward. Whenever a different knowledge representation is required, it suffices to create a translator from Java objects into a desired representation and vice versa. The WorldView event model will then take care of propagation of fact updates to such a translator. For the AS agent, we have implemented this mechanism for all the four components WorldView communicates with (Fig. 3). For instance, the RETE

inference engine has its listeners attached to desired Java classes inside WorldView and when it infers a new fact, this fact is propagated via WorldView to other components when they have their listeners up.

The last component of the architecture is Reactive layer, which is useful for coping with situations like “projectile is coming” in a swift way. While it is possible to model this kind of reflexive behaviour within a high-level goal-oriented DMS, our experience is that it is better to have a separate module handling these reflexes (for instance due to time efficiency and due to the fact that reflexive behaviour is easier to manage in Java than in such DMS). This reflexive—deliberative decomposition is a reminiscence of layered control architectures [39].

The key point of the blackboard architecture is that individual components are, to a large extent, oblivious to existence of the other components. This is important for at least two reasons. First, a developer can use arbitrary Java libraries for different modules. Second, a new module can be added during development. In our case, we have added the inference engine in this way, but in the context of videogames, other modules come into mind, such as an emotion model or episodic memory.

## 6 Pogamut 3 beyond UT 2004

So far, this paper discussed predominantly coupling between a GE and an external agent DMS for controlling a single bot. Additionally, only a UT 2004 implementation has been presented. It is natural to ask whether our approach can be extended. This section discusses three possible directions of such extension.

- 1) Different game engine. Does our approach work well for a different GE?
- 2) Internal coupling. How the framework from Fig. 3 should be refined when an agent DMS is coupled internally?
- 3) Multiple agent generalisation. How our framework can be generalised when multiple agents connect to a same GE?

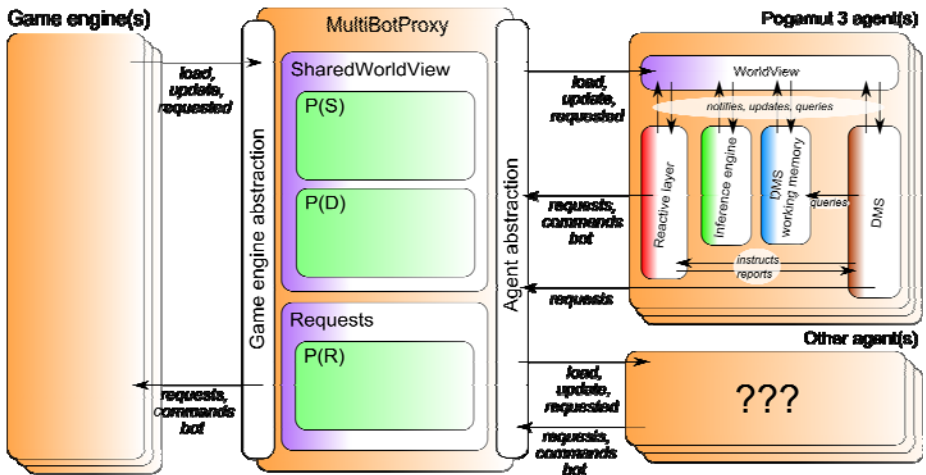
In fact, we already have prototypes of binding to Unreal Tournament 3 [40], Unreal Development Kit [41], Defcon [20], StarCraft [19] and Virtual Battle Space 2 (VBS) [18] which provides us a broad range of game engines and game types for the validation of our approach (Point 1).

Internal coupling (Point 2) is being tested mainly on binding to Defcon [20]. Defcon is a completely different game than UT 2004. It is a simulation of a global thermonuclear conflict played in real-time, where a player takes a role of a commander in charge of military forces under the flag of one nuclear nation. Secondly, this work demonstrates that Pogamut can be used also beyond the domain of 3D bots. Thirdly, Pogamut is connected to Defcon via the game’s internal API, demonstrating that internal binding is possible. In future, we plan to investigate the relation between the framework presented on Fig. 2 and the Defcon binding in detail.

Finally, connecting Pogamut to Virtual Battle Space 2 (VBS) [18], a military simulator, is important due to Point 3. Almost every 3D game features multiple bots. When these bots are represented as individuals, that is, each follows the architecture from Fig. 3, following observations can be made:

- a) Repetitive queries. Facts, both static and dynamic, are often queried repetitively.
- b) Concurrent access. Bots may access the game engine concurrently, creating synchronisation issues.
- c) Locality of facts. Nearby bots tend to acquire similar fact sets.
- d) Shared facts. Bots may share some static and dynamic information, e.g. discovered map and known topology, communication channels, etc.
- e) Bots communication. If it is believable, bots may be allowed to communicate directly each other with, which opens the possibility to bypass the GE concerning the communication.

Based on these observations, we designed the *MultiBotProxy* (MBP) architecture extending the architecture from Fig. 3. The MBP architecture covers the possibility of multiple externally connected DMSs to a single GE. This architecture is now being implemented using the VBS simulator.



**Fig. 4.** MultiBotProxy schema. The MBP may allow connection of agent DMSs developed by different means (i.e., not only by Pogamut) to the same GE.

The MBP is a 3-tier architecture (see Figure 4), where the MBP node is the place between the GE and the externally connected agents. The MBP provides, besides its obvious proxy functionality, an interface for communication and data flow management – information caching, request optimisation, and data processing. *SharedWorldView*, the main component of the architecture, can be conceived as a blackboard shared by the GE and all the agents.

## 7 Conclusion

This paper started with the question *Can knowledge accumulated in the MAS field concerned with agent reasoning be used for reasoning of individual bots or a couple of bots?* We argued that to answer this question, it is necessary to connect several

agent DMS to a single GE and compare them against existing AI techniques currently used by the game industry. To facilitate this process, we have presented a theoretical framework (Fig. 1, 2) for thinking about coupling external agent DMSs to GEs and the software toolkit Pogamut for practical development of such couplings. We have also presented an AgentSpeak(L)-based system fully connected to the UT 2004 game, demonstrating applicability of Pogamut. Finally, we reviewed our work in progress aiming at applying Pogamut beyond the domain of UT 2004: most notably for two strategy games, for new UT versions, and for a 3D game featuring teams of bots.

We conclude this paper with an interesting observation that our architecture from Fig. 3 implementing the theoretical framework from Fig. 2 could be, *per se*, conceived as a multi-agent system. In other words, our suggestion is that a “mind” of a *single* agent connected to a game engine can be perceived as a system of interacting, fully or partly autonomous agents. It would be interesting to elaborate on this idea in future, since this may bring the possibility to use, in games, knowledge the MAS field gained on negotiation; not for negotiation between different bots, but between different components of their “minds”.

**Acknowledgement.** This work was partially supported by grant P103/10/1287 (GA ČR), research project MSM0021620838 (MŠMT ČR) and by students grants GA UK No. 44910 and GA UK No. 21809.

## References

1. Champeandard, A.J.: Key Trends in Game AI – Are You Ready for These? In: AiGameDev.com online, <http://aigamedev.com/open/editorial/key-trends/> (April 17, 2009)
2. Orkin, J.: Three States & a Plan: The AI of F.E.A.R. In: Proceedings of Game Developer’s Conference (2006)
3. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Real-time Neuroevolution in the NERO Video Game. *IEEE Transactions on Evolutionary Computation* 9(6), 653–668 (2005)
4. Zubeck, R.: Introduction to Hidden Markov Models. In: *AI Game Programming Wisdom 3*, pp. 633–646. Charles River Media, Hingham (2006)
5. Wooldridge, M.: *An Introduction to MultiAgent Systems*. John Wiley & Sons, Chichester (2002)
6. Briot, J.-P., Sordoni, A., Vasconcelos, E., de Azevedo Irving, M., Melo, G., Sebba-Patto, V., Alvarez, I.: Design of a Decision Maker Agent for a Distributed Role Playing Game – Experience of the SimParc Project. In: Dignum, F., Bradshaw, J., Silverman, B., van Doesburg, W. (eds.) *Agents for Games and Simulations*. LNCS, vol. 5920, pp. 119–134. Springer, Heidelberg (2009)
7. Bordini, R.H., Hübner, J.F.: BDI Agent Programming in AgentSpeak Using *Jason* (Tutorial Paper). In: Toni, F., Torroni, P. (eds.) *CLIMA 2005*. LNCS (LNAI), vol. 3900, pp. 143–164. Springer, Heidelberg (2006)
8. Busetta, P., Ronnquist, R., Hodgson, A., Lucas, A.: JACK Intelligent Agents - Components for Intelligent Agents in Java, *AgentLink News* (2) (1999)
9. Braubach, L., Pokahr, A.: Jadex: BDI Agent System, <http://jadex.informatik.uni-hamburg.de> (27.9.2009)

10. Dignum, F., Westra, J., van Doesburg, W.A., Harbers, M.: Games and Agents: Designing Intelligent Gameplay. *International Journal of Computer Games Technology*, vol. 2009 (2009)
11. Gemrot, J., Kadlec, R., Bída, M., Burkert, O., Příbil, R., Havlíček, J., Zemčák, L., Šimlovič, J., Vansa, R., Štolba, M., Plch, T., Brom, C.: Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. In: Dignum, F., Bradshaw, J., Silverman, B., van Doesburg, W. (eds.) *Agents for Games and Simulations*. LNCS, vol. 5920, pp. 1–15. Springer, Heidelberg (2009)
12. Brom, C., Bída, M., Gemrot, J., Kadlec, R., Plch, T.: Emohawk: Searching for a “Good” Emergent Narrative. In: Iurgel, I.A., Zagalo, N., Petta, P. (eds.) *ICIDS 2009*. LNCS, vol. 5915, pp. 86–91. Springer, Heidelberg (2009)
13. Burkert, O., Brom, C., Kadlec, R., Lukavský, J.: Timing in Episodic Memory: Virtual Characters in Action. In: *Proceedings of AISB workshop Remembering Who We Are – Human Memory For Artificial Agents*, Leicester, UK (to appear)
14. Arrabales, R., Ledezma, A., Sanchis, A.: Towards Conscious-like Behavior in Computer Game Characters. In: *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pp. 217–224 (2009)
15. Hindriks, K. V., van Riemsdijk, M. B., Behrens, T., Korstanje, R., Kraaijenbrink, N., Pasman, W., de Rijk, L.: *Unreal GOAL Bots: Conceptual Design of a Reusable Interface (AGS 2010)* (May 2010)
16. Gemrot, J.: Joint behaviour for virtual humans. Master’s thesis. Charles University, Prague (2009)
17. *Epic Games: Unreal Tournament 2004* (2004),  
<http://www.unrealtournament2003.com/> (7.2.2010)
18. *Virtual Battlespace 2*. Bohemia Interactive,  
[http://www.bistudio.com/bohemia-interactive-simulations/virtual-battlespace-2\\_czech.html](http://www.bistudio.com/bohemia-interactive-simulations/virtual-battlespace-2_czech.html) (7.2.2010)
19. *StarCraft*. Blizzard Entertainment,  
<http://eu.blizzard.com/en-gb/games/sc/> (21.8.2010)
20. *Defcon: Everybody dies*. Introversion software,  
<http://www.introversion.co.uk/defcon/> (7.2.2010)
21. Loyall, B.A.: *Believable Agents: Building Interactive Personalities*. Ph.D. diss. Carnegie Mellon University (1997)
22. Orkin, J.: 3 States & a Plan: The AI of F.E.A.R. In: *Game Developer’s Conference Proceedings* (2006)
23. Rao, A.S.: *AgentSpeak(L): BDI agents speak out in a logical computable language*. In: Perram, J., Van de Velde, W. (eds.) *MAAMAW 1996*. LNCS, vol. 1038. Springer, Heidelberg (1996)
24. *Game Engine*. Wikipedia,  
[http://en.wikipedia.org/wiki/Game\\_engine](http://en.wikipedia.org/wiki/Game_engine) (7.2.2010)
25. Zerbst, S., Duvel, O.: 3D game engine programming. In: *Course Technology PTR* (2004)
26. *Lua programming language*, <http://www.lua.org/> (7.2.2010)
27. *Python programming language*, <http://www.python.org/> (7.2.2010)
28. *UnrealScript programming language*,  
<http://unreal.epicgames.com/UnrealScript.htm> (7.2.2010)
29. Stentz, A.: *Optimal and Efficient Path Planning for Partially-Known Environments*. In: *Proceedings of the International Conference on Robotics and Automation* (1994)
30. Houlette, R., Fu, D.: *The Ultimate Guide to FSMs in Games*. In: *AI Game Programming Wisdom 2*. Charles River Media, Hingham (2003)

31. Isla, D.: Managing Complexity in the Halo 2 AI System. In: Proceedings of the Game Developers Conference (2005)
32. Brom, C., Gemrot, J., Burkert, O., Kadlec, R., Bída, M.: 3D Immersion in Virtual Agents Education. In: Spierling, U., Szilas, N. (eds.) ICIDS 2008. LNCS, vol. 5334, pp. 59–70. Springer, Heidelberg (2008)
33. Bryson, J.J.: Intelligence by design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agent. PhD Thesis. MIT, Department of EECS, Cambridge, MA (2001)
34. Anderson, J.R.: How can the human mind occur in the physical universe? Oxford University Press, Oxford (2007)
35. Gemrot, J., Brom, C., Kadlec, R., Bída, M., Burkert, O., Zemčák, M., Přebil, R., Plch, T.: Pogamut 3 – Virtual Humans Made Simple. In: Gray, J., Nefti-Meziani, S. (eds.) Advances in Cognitive Systems. IET Publisher (in press)
36. TuProlog, <http://alice.unibo.it/xwiki/bin/view/Tuprolog/> (7.2.2010)
37. Hammurapi rules, <http://www.hammurapi.biz/hammurapi-biz/ef/xmenu/hammurapi-group/products/hammurapi-rules/index.html> (7.2.2010)
38. Hayes-Roth, B.: A blackboard architecture for control. *Artificial Intelligence* 26(3), 251–321 (1985)
39. Wooldridge, M.: An Introduction to MultiAgent Systems, p. 99. John Wiley & Sons, Chichester (2002)
40. Epic Games: Unreal Tournament 3, <http://www.unrealtournament.com/uk/index.html> (1.9.2010)
41. Epic Games: Unreal Development Kit (UDK), <http://www.udk.com/> (1.9.2010)

# Goal-Based Communication Using BDI Agents as Virtual Humans in Training: An Ontology Driven Dialogue System

Joost van Oijen<sup>1,3</sup>, Willem van Doesburg<sup>2</sup>, and Frank Dignum<sup>3</sup>

<sup>1</sup> VSTEP

Weena 598, 3012 CN Rotterdam, The Netherlands

joost@vstep.nl

<sup>2</sup> TNO Defence, Security and Safety

P.O. Box 23, 3769 ZG Soesterberg, The Netherlands

willem.vandoesburg@tno.nl

<sup>3</sup> University of Utrecht

P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

{oijen,dignum}@cs.uu.nl

**Abstract.** Simulations for training can greatly benefit from BDI agents as virtual humans playing the role of key players. Learning to communicate effectively is a key aspect of training to command a team that is managing a crisis. In this paper, we present a goal-based dialogue system which has been applied to a navy fire-fighting incident simulation, training the commanding officer. The system enables more natural training of communication because agents utilize specialized communication goals. Communication is data-driven where the content is defined by a shared ontology of domain knowledge and tasks. The use of this ontology enables reuse of agent capabilities within the system and enables a tight coupling between the agents and the simulation.

## Categories and Subject Descriptors:

I.2.0 [Artificial Intelligence]: General - *Cognitive Simulation*

I.2.1 [Artificial Intelligence]: Applications and Expert Systems

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence - *Intelligent Agents, Multiagent Systems*

I.6.3 [Simulation and Modeling]: Applications

**General Terms:** Design, Human Factors.

**Keywords:** BDI Agents, Virtual Humans, Communication, Ontology, Training, Simulation, Behavior Modeling.

## 1 Introduction

Modern society has ample systems where one decision maker controls the safety of many. For example, during a fire, the fate of fire-fighters, bystanders and victims largely depend on decisions of the fire officer. It is evident that for such

safety critical systems we need competent and experienced decision makers. Scenario-based simulator training is considered very appropriate for learning decision making in complex environments [1]. Common practice to realize this in simulation-training is to use Subject Matter Experts (SMEs) (usually staff members) to play the role of key players [2]. Training the fire-fighting commander requires developing his command and control competencies. An important aspect of command and control is the ability to communicate clearly with all the members of the fire-fighting organization. A reason why SMEs are required to play key roles, is that these can accommodate all the (possible variety of) communication that occurs during a fire-fighting scenario. However, the logistical effort to train (sometimes only one) student(s) becomes a great burden. The student versus key-player ratio can be, in more complex scenarios, 1:10. An alternative way of training command and control competencies such as communication that is not dependent on so many SMEs is sought after by the RNLN (Royal Netherlands Navy). One possibility is to use virtual humans in a training simulation to play the key roles [3].

The use of BDI agents for behavior modeling of virtual humans in games and simulations has gained a lot of interest recently. In some applications like virtual training simulations or interactive dramas the use of communication by agents is essential and can be the main driving force for their behavior and therefore the advancement of a story or scenario in terms of mental and environmental changes. Successful attempts have been reported that employ BDI agents in game engines as virtual humans for training or entertainment, though most are not yet industry-ready [4,5,6,7]. Usually the focus is on non-communicative agent behavior like navigation or object manipulation while interactions with the player or trainee are limited. On the other hand, systems originating from the field of embodied conversational agent (ECA) research pay a lot of attention to believable communicative behavior [8,9]. Still, we have seen few examples of where agents that communicate with a trainee play an independent role in a larger organization, pursuing their own goals. Most examples that we found show agents that take on a more supportive role and don't express much goal-directed behavior of their own.

Using the BDI paradigm for agents in these applications offers both advantages and additional challenges with respect to simulating human communication. BDI offers a more intuitive way to translate (expert) domain knowledge into agent knowledge. However, it offers a) no instruments to structure that knowledge beyond the boundaries of an individual agent, and b) no inherent guidance to model human communicative behavior. There is a need to specify how goals can be used in communication for practical reasons like information exchange or resolving conflicts of interest. Additionally, agents need a way to coordinate goals and actions that relate to communication with other activities. Lastly, it is not clear how the BDI paradigm guides the flow of conversations to achieve natural human-like simulated interactions.

In this paper we present our approach to modeling communicative behavior of agents. We report our experience in applying our dialogue system for use in



BDI agents. We will argue that using an ontology can not only structure the knowledge of BDI agents but also provide structure to information exchange throughout the system as a whole. We introduce specialized goals that support the BDI approach in modeling communicative behavior. We will show how specialized goals can also provide for a more natural flow between conversations. The proposed dialogue system has been applied in the CARIM<sup>1</sup> research project, a virtual training application for command and control, simulating the roles of military officers working together as a team fighting a damage control incident on board of a navy ship.

This paper is organized as follows. A review of related work is given in section 2. Section 3 provides an overview of the system that was developed focusing on the design of our BDI agents and their integration into the visualization engine. In section 4 we introduce the ontology-driven dialogue system and its integration in the agents. Requirements for the trainee to use the dialogue system are outlined in section 5. Finally, section 6 provides a conclusion and describes gained experiences and future aims.

## 2 Related Work

The scope of the presented work involves several research areas including the use of embodied conversational agents (ECAs) for tutoring systems or interactive dramas and the integration of multi-agent systems in games and simulations. Much work has been carried out in those fields.

Looking at ECA tutoring systems, examples include the Tactical Language Training System [8] where a trainee practices foreign communication skills in a simulated village. The trainee can communicate with the local village people and is assisted by a virtual aide. The system makes use of the THESPIAN architecture for interactive pedagogical drama [10]. Possible encounters and dialogue instances are defined by a set of scripts where a "fitting" algorithm automatically adjusts the goals of the agents so they perform their roles according to the scripts.

More wide-spread research on virtual humans for training environments is described in [11], where the goal is to build embodied agents providing a social human focus to training and serve as guides, competitors or teammates supporting interactive face-to-face interaction to train leadership, negotiation and cultural awareness. In this context dialogue modeling for communicative virtual humans has given much attention [12,13]. Both in the tutoring work and the virtual human work little attention has been given on how agent notions affect the design of a training system as a whole.

Considering the integration of existing multi-agent systems to game engines, several successful attempts have been made, where BDI agents are used for modeling human behavior using more complex goal-driven characters. In [4,5] BDI agents written in *Jadex* and *JACK* respectively were connected to a game engine to apply more sophisticated agent techniques in games to overcome the limited

---

<sup>1</sup> Cognitive Agents for Realistic Incident Management training.

reactive behavior usually found in game agents. In these attempts agent communication wasn't given much attention. In [14], it is argued that the coupling of agents to games requires a new design methodology including agent notions, focusing on the aspects of synchronization, information representation and agent communication. More experiences and design ideas can be found in [15] where a middleware is introduced for integrating AI controllers into a game engine.

We argue that there is a need for a more light-weight goal-based dialogue system to be employed by BDI agents and trainees giving those means to automatically perform dialogues satisfying their immediate needs.

### 3 System Overview

The proposed dialogue system was designed in the context of the CARIM project. The primary goal of the project is the development, implementation and evaluation of intelligent agents for team-based command and control simulation training. A trainee takes on the role of the commanding officer during a fire-fighting incident situation. Intelligent agents fulfill the roles of the remaining officers in the damage control organization. A director agent controls the scenario and evaluates the trainee's performance. An impression of the simulation environment is illustrated in Figure 1.



Fig. 1. CARIM Screenshot

A complete technical overview of the system is described in [16]. Ideas and design decisions for the director agent can be found in [17,18]. Below we describe the relevant design aspects of the intelligent agents and the trainee which are required as a basis for introducing the dialogue system.

### 3.1 Intelligent Agents

Modeling the behavior of the officers to be simulated, we employed the BDI agent paradigm. Using the BDI framework provides us with an efficient workflow converting cognitive models that were built with the help of subject matter experts (SMEs) to corresponding BDI concepts like beliefs, desires and intentions. The agents were created using *Jadex* [19], a multi-agent BDI platform. For the visual representation of the agents' embodiment and visualization of the simulation environment, VSTEP's in-house developed game engine was used [2].

The use of these two distinct software technologies, namely the multi-agent system and the game engine, forces us to create a distributed architecture of the intelligent virtual agent (IVA). The proposed architecture is shown in Figure 2. It is composed of two layers: the cognitive layer of the IVA, realized in the multi-agent system, and the physical layer, realized in the game engine.

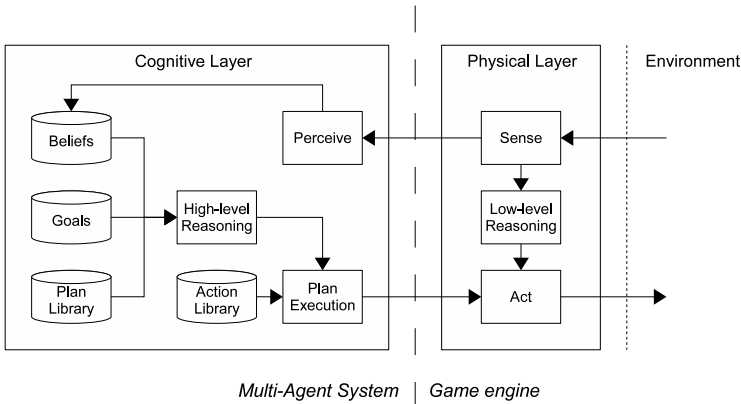
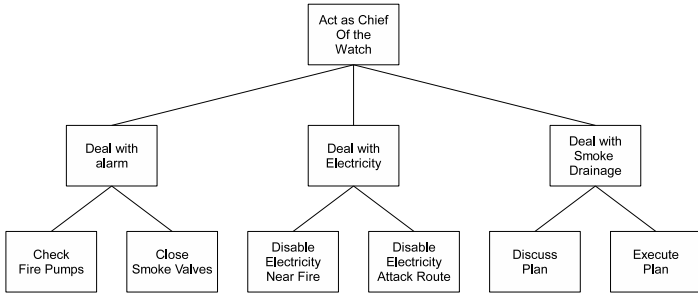


Fig. 2. IVA Architecture

In the cognitive layer the full behavior of an agent comprises two modules also known as capabilities: an agent-generic capability and an agent-specific capability. The agent-generic capability is shared by all agents and consists of basic humanoid behaviors required in the simulated domain like agent communication or object interactions like knowing how to use a phone. The agent-specific capability comprises task knowledge belonging to a certain role the agent can take on. To give a small impression, Figure 3 illustrates a simplified abstract model of such task knowledge for an officer in our example domain. High-level goals break down to lower-level goals and eventually to actions at the bottom. Each goal maps to a single plan which either adopts sub-goals or executes actions. Also more generic goals were defined, for example a goal stating to inform a superior officer about important information retrieved during the incident progress.

<sup>2</sup> www.vstep.nl



**Fig. 3.** Task knowledge of the officer *Chief of the Watch*

The agent’s physical layer executes the actions which have been scheduled by the cognitive layer, influencing the environment. Changes in the environment are sensed in the physical layer and communicated back to the cognitive layer. Due to the distributed nature of the architecture, both conceptually and technically, reasoning is performed at both layers in parallel only at different levels of abstraction. The cognitive layer reasons at a higher level using declarative information while the physical layer involves reasoning on procedural information, having close ties with the environment. For example, reasoning in the cognitive layer may form an intention of the agent to move itself to some other agent in the environment. Reasoning in the physical layer involves identifying the target agent, determining a path towards the target and reevaluating the path in case the target changes position while navigating towards it. Results of the intention(s) are sent back to the cognitive layer. In addition, intentions sent to the physical layer can be aborted if required.

The agents in our system have to operate in a complex environment with a wide range of domain specific concepts. These concepts are not only shared between agents, but also shared between the distributed components of the system. For example an important concept in our example domain is the *fire*. It is physically represented in the virtual environment in the game engine. In the multi-agent platform agents can form beliefs and goals about it and use the concept in communication. In turn, resulting agent actions concerning the *fire* concept have to be recognized in the game engine to apply physiological changes to the *fire* entity. A common approach to deal with the sharing and reuse of knowledge and concepts is the use of an ontology.

### 3.2 Ontology

To capture the relevant concepts involved in the target domain to be simulated, an ontology has been defined providing a formal representation of these concepts. This ontology is essential for the system as it forces an agreement on the used domain concepts throughout the system as a whole. Using such an agreement in software design is also known as *design by contract* [20], increasing robustness and reusability in software systems. By using the ontology as a design contract

not only can the agents exchange these concepts among themselves. Also the need to translate between intentions and perceptions in the agent system and the actions and events in the simulation system is removed.

In our agent architecture this ontological design approach is applied in several areas. First of all it is used in the connection between an agent’s physical and cognitive layer, representing all valid environmental perceptions and agent actions. Second, it limits the concepts the agents can reason about, the goals they can adopt and the actions available to them to execute. Lastly, it is used in the communication language of the agents ensuring valid semantics while communicating knowledge and motivations.

Designing the ontology according to the above requirements, three categories were identified:

- *knowledge ontology*: a complete reference of domain knowledge
- *goal ontology*: a complete reference of all goals used in the domain
- *action ontology*: a complete reference of all actions available to the agents

Items in each category adhere to the same structure. An ontological element is defined by a concept type and a list of key-value attributes. The attribute key corresponds to an enumeration type binding the possible attribute values. This structure was chosen as it provides a generic encoding of information usable for all required purposes. On one hand knowledge about objects or concepts can be defined as a class type with properties. On the other hand, the element type can be used to identify a specific goal or action where attributes are used for parameterization. Further, this structure allows the full ontology to be used as content for agent communication without a need for special-purpose parsing.

The use of these ontological concepts are illustrated in Figure 4. A simplified part of a scenario of our fire-fighting simulation is shown where a fire is present which needs to be extinguished using the right equipment. The left diagram illustrates part of the ontology with ontological elements on the outside and enumeration types in the middle. On the right side, an example conversation is given between two officers employing the ontology in their dialogue acts.

It is clear that an appropriate design of the ontology is essential and should be created early in the design-phase. We have experienced that any change to the ontology in a later stage of development has serious consequences, breaking functionality and requiring redesign throughout the system.

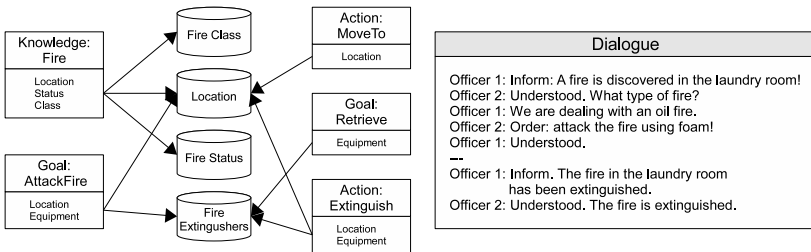


Fig. 4. Ontology Example

## 4 Ontology-Driven Dialogue System

As seen from the agent dialogue example in Figure 4.1, agents sharing the ontology can use simple dialogues to communicate knowledge and motivations among each other. We have argued that being able to perform such practical communication can be an important requirement when modeling human behavior in BDI agents. This becomes clear when considering the requirements of the CARIM fire-fighting training simulation. Here, the majority of the officers simulated take on a leading role, instructing a team of executing officers to perform their duties. For example, the commanding officer, head of the damage control organization and played by the trainee, has to formulate several plans like an attack plan and a boundary cooling plan. These plans have to be discussed with the officers responsible for executing those plans. These officers, in turn, have to communicate the plans to the executing teams and monitor the progress of these plans, which they are required to report back to their superiors. In addition, the commanding officer should not passively wait for information but also proactively retrieve information to obtain good situation awareness. Having a hierarchical organization of officers, communication often needs to flow up and down the hierarchy.

To simulate these communication flows we need to augment our agents with communicative abilities enabling them to exchange information and give or accept orders. As said earlier the ontology plays a central role, defining the possible communication content.

We propose an ontology-driven goal-based dialogue system for use in BDI agents simulating natural human-like interactions in virtual environments. The system combines two main aspects of human communication in terms of goals to achieve:

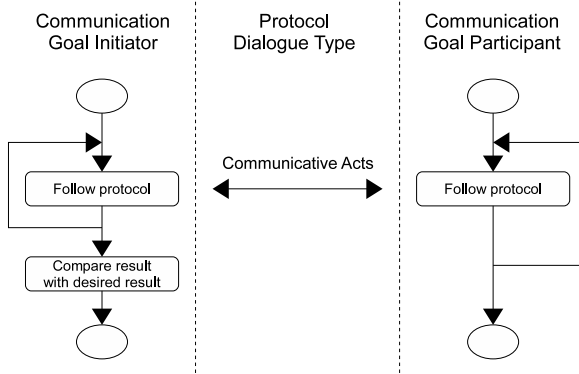
- **Communicating semantic content** through the execution of *communication goals*: An agent forms a *communication goal* when it desires to bring over a communicative intent to another agent. This goal comprises the execution of an initial speech act containing the intent to communicate and the determination of a corresponding result. In addition, an agent forms a *communication goal* as a response to a collaboration request to facilitate another agent in achieving his *communication goal*.
- **Providing a natural flow of conversation** through the management of an interaction instance in the form of a *conversation goal* controlling the execution of one or more *communication goals*: A *conversation goal* provides commencement and termination rules for an interaction instance. In addition, it relays incoming communicative acts to corresponding *communication goals*.

In the next sections we describe these two aspects in more detail.

### 4.1 Communication Goal

In our system, a *communication goal* denotes a single ontology-driven dialogue between two agents. A dialogue is an exchange of speech acts between two speech

partners in turn-taking sequence aimed at a collective goal [21]. Walton and Krabbe [22] have identified a set of dialogue types based on the initial situation of the participants, the goals of the individual participants and the goal of the dialogue itself. We employ this classification to design *communication goals*. Within one dialogue type, two goals can be identified based on the role an agent takes on during the dialogue: the initiator’s goal and the participant’s goal. Acts performed within each goal are part of a mutually agreed protocol used for a specific dialogue type. Semantic content inside each act is based on the shared ontology known by each participant. Figure 5 illustrates the process of a dialogue, achieving the communication goal of the initiator.



**Fig. 5.** Dialogue using Communication Goals

Whenever the initiator has adopted a communication goal of the desired type, he starts by sending the initial dialogue act to the participant. Receiving this act, the participant adopts a corresponding communication goal to support the initiator in achieving his communication purpose. Both participants then follow the protocol belonging to the type of dialogue. Finishing the protocol, the initiator successfully finishes the goal if the result of the dialogue is equal to the desired result. Otherwise the goal fails.

Currently the dialogue system supports three types of communication goals, representing three types of dialogues according to the Watson and Krabbe classification, namely persuasion, information seeking and deliberation. Each type uses a one-turn protocol. The system can be extended to allow for more types of dialogue. Also, the protocols used inside the communication goals can be extended to incorporate more standardized protocols like the FIPA interaction protocols [23]. Figure 6 outlines the implemented goal types with corresponding results at the initiator’s side and results of the dialogue as a whole.

The system assumes that agents have a norm stating they will follow the protocol required for the corresponding dialogue type. Violation of this norm by the participant results in a failure of the communication goal of the initiator.

Goal Type	Initial Act	Response Act	Goal Result	Dialogue Result
Persuasion/ Informing	inform(x)	agree(x)	success	x is added to the participant's belief base
		disagree(x)	failure	-
Information Seeking/ Querying	query-ref(x)	belief(x)	success	x is added to the initiator's belief base
		unknown(x)	failure	-
	query-if(x)	yes(x) or no(x)	success	x is added to the initiator's belief base
		unknown(x)	failure	-
Deliberation/ Ordering	order(x)	accept(x)	success	x is added to the participant's goal base
		reject(x)	failure	-

**Fig. 6.** Communication Goal types

Communication goals give BDI agents additional means to achieve their desires besides non-communicative acts. They can be treated similar to other goals allowing BDI agents to successfully employ communication in their plans to achieve a desired state. To give an example, looking at our fire-fighting training simulation, consider an agent taking on the role of the commanding officer in the technical control room who wants to know the current status of the fire. Without communicative abilities, the officer would have no other option but to investigate himself and go to the fire, which would be a violation of his role of being a commanding officer. The use of communication provides alternatives. He can ask one of the officers in the room about the status; he can order that same officer implicitly to retrieve the information for him or explicitly order him to investigate the fire and report back; at last, he can use a communication device like a headset or a phone to contact someone near the fire and ask for the status. In case he orders another officer to retrieve the information, this officer in turn has the same decision to make and the same options to choose from.

Dealing with agent communication in terms of goals provides advantages over implicitly modeled communication. Communication goals to retrieve knowledge can be adopted to satisfy preconditions of other goals. Communication goals transferring motivations can help to achieve a higher-level goal. For example, an agent who doesn't have a plan himself to achieve some goal is now able to delegate the goal to another agent. Additionally, agents can apply goal prioritization to communication, giving priorities to certain communication targets or topics.

An agent can have multiple communication goals active at the same time with one or more target agents. Managing the execution of these goals requires us to introduce a higher level of control which we propose to achieve using a *conversation goal*, representing an interaction instance with one participant.

## 4.2 Conversation Goal

With the help of communication goals the BDI agents are now able to use dialogues to achieve their communicative purposes. However, simulating natural human communication requires additional communicative behavior. Agents starting to perform acts cannot assume the intended interlocutor currently has



their attention or is even available for communication. Commencing and termination rules are required to ensure a valid open communication channel with the interlocutor. We don't think it suits to add these rules to the protocol used inside communication goals as it would result in unnatural communication when multiple dialogues are performed consecutively or in parallel with the same participant. We introduce the notion of a *conversation goal* to manage the execution of one or more communication goals as one interaction instance.

Using a *conversation goal*, BDI agents don't have to concern themselves with performing natural and believable communication during planning. The goal is automatically adopted and dropped based on the presence of one or more communication goals involving a participant.

A *conversation goal* for a participant is automatically adopted when:

- the agent has *communication goal(s)* to achieve with the participant.
- the agent receives a request to collaborate in a conversation.

A *conversation goal* for a participant is automatically dropped when:

- the agent has no more communication goals active with the participant of this conversation and there are no more unprocessed incoming communicative acts coming from the participant.
- the agent receives a termination act from the participant. All currently active communication goals will fail and cannot be achieved anymore within this conversation instance.
- the agent decides to abort the goal himself in case there are higher priority tasks to perform.

**Interaction States.** When active, the *conversation goal* can be in one of three interaction states:

- *Commencing State:* Establishes a valid conversation instance. If the agent is the initiator of the conversation, a collaboration request is sent to the participant. If accepted, the active-conversation state is entered. If rejected, the goal fails. If the goal was started based on an incoming collaboration request, the agent either accepts the request and enters the next state or rejects the request and aborts the goal.
- *Active-Conversation State:* Processes incoming communicative acts. Incoming acts belonging to an active communication goal are relayed to the corresponding goal for further reasoning. An incoming act representing the start of a new dialogue will result in a new communication goal.
- *Terminating State:* A termination act is sent to the participant after which the *conversation goal* will terminate. As a consequence, all active communication goals with this participant will fail.

The system allows one conversation with some participant to be active. *Communication goals* with other participants can be scheduled but they will only result in a new conversation when the current conversation has been terminated.

When an agent receives a communication request from another participant during a conversation, he is able to reject the request which results in a reactive communicative behavior in the visualization engine, expressed either verbally or nonverbally. This behavior is more in line with expectations of human communication. The attention of a simulated person is effectively capturing by being in a conversation. However, flexibility is not lost as by reasoning about the priority of communication goals the agent can still shift attention if necessary.

### 4.3 Communicative Action and Perception

To complete the functionality of the system, we need a way of handling the execution of communicative acts and the perception of communicative acts by other agents. Multi-agent systems that don't require the simulation of virtual humans are able to communicate directly with each other using standard agent communication languages like FIPA ACL [23]. In our system, agents are visually represented in a virtual environment. They are therefore bound to use the same communication channels as humans would: using verbal and nonverbal communicative behavior. Also, unlike in multi-agent systems, receiving a communication event is not trivial and a successful reception depends on the available medium from the source to target, bounded by the simulated laws of physics.

Looking back at the architecture of our IVA in Figure 2, an agent performs a communicative act by sending a parameterized communicative intent to its physical layer inside the game engine. This intent includes the intended target agent, the communicative act type and the content message containing semantics according to the ontology. Here, the intent is converted to a verbal communication action accompanied with appropriate nonverbal behavior like gazing, orientation and facial expressions. Finishing the communicative behavior, an event is generated in the environment originating from the source character containing the communicative intent. This event can be sensed by other characters and sent to their cognitive layers in the multi-agent system for further reasoning.

## 5 Trainee Communication

Allowing the trainee to use the proposed dialogue system imposes several requirements for the game engine. In contrast to the BDI agents who control their avatars from the multi-agent system, the trainee controls his avatar using input devices processed inside the game engine. Therefore all communication actions for the trainee need to be constructed in the game engine.

Two common approaches include the use of a dialogue interface from which valid acts can be selected or the use of actual speech expressed by the trainee. Either way, the trainee's action must result in a communication message with valid semantics according to the defined ontology. If the ontology can be accessed from inside the game engine the validity of the acts can be determined here. Using a dialogue interface, acts can be presented to the trainee in an intelligent way by categorizing acts based on communicative purposes and ontological

concepts. When using actual speech input, natural language must be mapped to an ontological representation to create valid speech acts. Approaches have been described in [24].

In the CARIM system, communication actions are performed using a dialogue interface with communication menus. Here, the ontology is only available from within the multi-agent system and acts presented to the trainee are dynamically determined from the trainee agent in the multi-agent system based on the current scenario situation and the current dialogue turn. Such an approach imposes additional requirements on the connection layer between the multi-agent system and the game engine: a protocol is required to dynamically add and remove dialogue acts available to the trainee for specific target agents. This approach shows how the ontology agreement can be used to define the simulation system as well as the agent system.

## 6 Conclusion and Future Work

In this paper, we discussed the development of an ontology-driven goal-based dialogue system for use in BDI agents as virtual characters. The main purpose of the system is to augment these agents with communicative abilities providing those means to exchange knowledge and motivations with other agents using content adhering to concepts defined in a shared ontology. Dialogues are goal-based, performed within the context of *communication goals*. Different goals exist for different communicative purposes. They are represented the same as other BDI goals, so that agents can use them in reasoning, specify rules for prioritization and specify alternatives for communication goal failure handling. A higher level of control over dialogues is provided in the form of *conversation goals*, managing the communication of multiple dialogues and participants to create more believable and natural interactions.

Agent-based simulations for training command and control can greatly benefit from the proposed dialogue system. In such simulations, inter-agent communication plays an essential role in the advancement of the story or scenario. For example, the system enables both agents and trainees to obtain situation awareness and to fulfill leadership roles by delegating goals and monitoring progress. Ontologies defined for capturing domain knowledge and tasks in agent role descriptions can be automatically employed in dialogues, enabling agents to communicate autonomously while ensuring believable interactions. By enforcing the ontology throughout the system as a whole there is little need to translate information between agents and the simulation. If the agreement is enforced over multiple simulations, agents will even be able to connect to other simulations and perform their role without any changes.

The dialogue system has provided the communicative behavior required for the CARIM fire-fighting training system, showing that the BDI approach to model communication for agents in training simulations is both useful and fruitful. By creating agents that can communicate naturally we reduce the dependence on many human key role players in training scenarios.

From our experience we see the need for further research concerning agent communication. Although communication can now be employed by agents to achieve a desired result, it is not always possible, desirable or the most efficient way to achieve some goal. Deciding whether or not to use communication and with who can be a very complex task and can be highly dependent on the current situation, the agent's role descriptions and available alternatives. Agents not only need to decide when to use communication, but also how to handle situations where communication fails. To ensure a believable and valid simulation, rules and norms have to be specified for the agents enabling them to make proper decisions. Some of these rules should be specified at a higher level of abstraction, more easily accessible to designers or domain experts giving them more control over the way agents make decisions.

More straightforward work to be done involves extending the current dialogue system by providing more types of communication goals, more advanced interaction protocols and the ability to create communication goals within other communication goals, thereby creating more complex dialogues.

**Acknowledgements.** This research has been supported by the Netherlands Department of Defence and by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

## References

1. Oser, R.L.: A structured approach for scenario-based training. In: 43rd Annual meeting of the Human Factors and Ergonomics society, Houston, TX, pp. 1138–1142 (1999)
2. van den Bosch, K., Riemersma, J.B.J.: Reflections on scenario-based training in tactical command. In: Schiflett, S.G., Elliot, L.R., Salas, E., Coovert, M.D. (eds.) *Scaled Worlds: Development, Validation and Applications*, Ashgate, Aldershot, pp. 1–21 (2004)
3. van Doesburg, W.A., van den Bosch, K.: Cognitive Model Supported Tactical Training Simulations. In: *Conference on Behavioral Representation in Modeling and Simulation (BRIMS)*, Universal City, CA, pp. 313–320 (2005)
4. Davies, N., Mehdi, Q.: BDI for Intelligent Agents in Computer Games. In: *Proceedings of the 8th International Conference on Computer Games: AI and Mobile Systems* (2006)
5. Norling, E., Sonenberg, L.: Creating Interactive Characters with BDI Agents. In: *Proceedings of the Australian Workshop on Interactive Entertainment IE 2004*, Sydney, Australia (2004)
6. Magerko, B., Laird, J.E.: Towards Building an Interactive, Scenario-based Training Simulator. In: *Proceedings of the Behavior and Representation and Computer Generated Forces Conference* (2002)
7. Magerko, B., Laird, J.E., Assanie, M., Kerfoot, A., Stokes, D.: AI Characters and Directors for Interactive Computer Games. In: *Proceedings of the 2004 Innovative Applications of Artificial Intelligence Conference*. AAAI Press, San Jose (2004)

8. Johnson, W.L., Beal, C., Fowles-Winkler, A., Lauper, U., Marsella, S.C., Narayanan, S., Papachristou, D., Vilhjálmsson, H.H.: Tactical Language Training System: An Interim Report. In: Proceedings of the International Conference on Intelligent Tutoring Systems, pp. 336–345 (2004)
9. Niewiadomski, R., Bevacqua, E., Mancini, M., Pelachaud, C.: Greta: an Interactive expressive ECA system. *AAMAS* (2), 1399–1400 (2009)
10. Si, M., Marsella, S.C., Pynadath, D.V.: THESPIAN: An Architecture for Interactive Pedagogical Drama. In: Proceedings of the 2005 Conference on Artificial Intelligence in Education: Supporting Learning through Intelligent and Socially Informed Technology, pp. 595–602, May 06 (2005)
11. Kenny, P., Hartholt, A., Gratch, J., Swartout, W., Traum, D., Marsella, S., Piepol, D.: Building Interactive Virtual Humans for Training Environments. In: Proceedings of Interservice/Industry Training, Simulation and Education Conference (2007)
12. Traum, D., Swartout, W., Marsella, S., Gratch, J.: Fight, flight, or negotiate: Believable strategies for conversing under crisis. In: Panayiotopoulos, T., Gratch, J., Aylett, R.S., Ballin, D., Olivier, P., Rist, T. (eds.) *IVA 2005. LNCS (LNAI)*, vol. 3661, pp. 52–64. Springer, Heidelberg (2005)
13. Traum, D.R., Marsella, S.C., Gratch, J., Lee, J., Hartholt, A.: Multi-party, Multi-issue, Multi-strategy Negotiation for Multi-modal Virtual Agents. In: Prendinger, H., Lester, J.C., Ishizuka, M. (eds.) *IVA 2008. LNCS (LNAI)*, vol. 5208, pp. 117–130. Springer, Heidelberg (2008)
14. Dignum, F., Westra, J., van Doesburg, W.A., Harbers, M.: Games and Agents: Designing Intelligent Gameplay. *International Journal of Computer Games Technology* (2009)
15. Riedl, M.O.: Towards Integrating AI Story Controllers and Game Engines: Reconciling World State Representations. In: Proceedings of the 2005 IJCAI Workshop on Reasoning, Representation and Learning in Computer Games, Edinburgh (2005)
16. Cap, M., Heuvelink, A., van den Bosch, K.: Using Agent Technology to Build a Real-World Training Application (2010) (submitted)
17. van den Bosch, K., Harbers, M., Heuvelink, A., van Doesburg, W.A.: Intelligent Agents for Training On-Board Fire Fighting. In: Proceedings of the 2nd International Conference On Digital Human Modeling (ICDHM 2009): Held as Part of HCI International 2009. *LNCS*, vol. 5620, pp. 463–472. Springer, Heidelberg (2009)
18. Heuvelink, A., van den Bosch, K., van Doesburg, W.A., Harbers, M.: Intelligent Agent Supported Training in Virtual Simulations. In: Proceedings of the NATO HFM-169 Workshop on Human Dimensions in Embedded Virtual Simulation. *NATO Human Factors and Medicine Panel*, Orlanda (2009)
19. Braubach, L., Pokahr, A., Lamersdorf, W.: Jadex: A BDI-Agent System Combining Middleware and Reasoning. In: *Software Agent-Based Applications, Platforms and Development Kits*, Birkhus, pp. 143–168 (2005)
20. Meyer, B.: Applying "Design by Contract". *Computer (IEEE)* 25(10), 40–51 (1992)
21. Walton, D.N.: Types of Dialogue, Dialectal Shifts and Fallacies. In: van Eemeren, F.H., et al. (eds.) *Argumentation Illuminated*, Amsterdam, SICSAT, pp. 133–147 (1992)
22. Walton, D.N., Krabbe, E.C.W.: *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. State University of New York Press, Albany (1995)
23. The Foundation for Intelligent Physical Agents, <http://www.fipa.org>
24. Hartholt, A., Russ, T., Traum, D., Hovy, E., Robinson, S.: A Common Ground for Virtual Humans: Using an Ontology in a Natural Language Oriented Virtual Human Architecture. In: Proceedings of the Language Resources and Evaluation Conference (LREC) (2008)

# Evaluation and Comparison of Multi-agent Based Crowd Simulation Systems

Bikramjit Banerjee and Landon Kraemer

School of Computing  
The University of Southern Mississippi  
118 College Dr. # 5106  
Hattiesburg, MS 39406-0001

Bikramjit.Banerjee@usm.edu, Landon.Kraemer@eagles.usm.edu

**Abstract.** We present a novel automated technique for the quantitative validation and comparison of multi-agent based crowd egress simulation systems. Despite much progress in the simulation technology itself, little attention has been accorded to the problem of validating these systems against reality. Previous approaches focused on local (spatial or temporal) crowd patterns, and either resorted to visual comparison (e.g., U-shaped crowd at bottlenecks), or relied on ad-hoc applications of measures such as egress rates, densities, etc. to compare with reality. To the best of our knowledge, we offer the *first* systematic and unified approach to validate the global performance of a multi-agent based crowd egress simulation system. We employ this technique to evaluate a multi-agent based crowd egress simulation system that we have also recently developed, and compare two different simulation technologies in this system.

## 1 Introduction

We describe a novel technique for the evaluation of multi-agent based crowd egress simulation systems in virtual environments. Crowd behavior simulation has been an active field of research [7, 24, 14, 4, 5] because of its utility in several applications such as emergency planning and evacuations, designing and planning pedestrian areas, subway or rail-road stations, besides in education, training and entertainment. The most advanced and realistic simulation systems employ intelligent autonomous agents [24, 14, 22, 19, 21, 25, 23, 11] with a balance between individual and group intelligence for scalability of the architectures. Although several systems have even been commercialized (e.g., the Evacuation Planning Tool from Regal Decision Systems [16], and CrowdFlow from Avalias [1] among others), little attention has been accorded to the problem of validating the outcomes of these simulations in a generalized manner, against reality. The extent of validation fails to go much beyond visual matching between the simulation and the actual scenarios (with recordings of human crowds), which can lead to highly subjective and often questionable conclusions. The existing numerical measures often rely on ad-hoc *applications* [6], e.g., local crowd densities are measured to verify patterns, without a systematic procedure to identify *at what times* in the simulation and the scenarios can the densities be compared. Furthermore, if there are multiple systems that

simulate crowd behavior in the same scenario in the same virtual environment, then no technique is currently known to quantitatively compare these systems in terms of realism.

In this paper, we present a principled automated technique for such evaluation and comparison. Our evaluation technique is systematic/algorithmic rather than an ad-hoc selection of metrics. Moreover, instead of comparing with the actual scenarios at some key points in the simulation where a close resemblance is detected, our approach automatically computes the comparison points in a principled way, based on the entire simulation run. In this sense, our approach is geared toward global validity. The comparison algorithm is closely intertwined with the selection of the comparison points, thus unifying two (previously) distinct facets of validation. We also evaluate a multi-agent based crowd egress simulation system that we have recently developed, and compare two different simulation technologies in this system using our approach.

The paper is organized as follows. In section 2 we present our literature survey for crowd simulation systems, with a focus on the existing evaluation methodologies and their limitations. In section 3 we present our algorithmic approach to evaluation and comparison of crowd egress simulation systems. In particular, we present two different methods for computing a key parameter for our technique. In section 4 we present our experimental results based on a stadium evacuation simulation system that we have developed, showing the differing evaluations of two different technologies for collision avoidance. We present discussions on the applicability of our evaluation and comparison approach in section 5 and we conclude in section 6.

## 2 Crowd Simulation

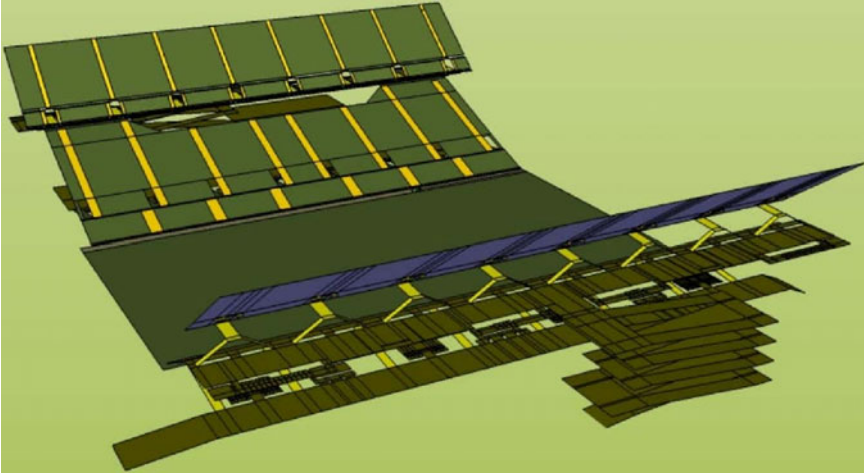
In most architectures for crowd simulation individuals are modeled as intelligent agents with (limited) perception and decision-making capabilities. Some of the earliest applications of simple agent-based behaviors were seen in Reynolds' flocking model – the “boids” [18]. In this and related work, each agent is endowed with a mix of simple steering behaviors, that produce complex macroscopic (group-level) behaviors as *emergent phenomena*. The basic idea of emergent behaviors has been extended to rule-based systems [24, 14] that offer the added advantages of efficiency and variety in behaviors. Thalmann, Musse and Kallmann [22] proposed to control agents having the same goal as a unit, for efficiency. Rymill and Dodgson [19] describe a system that exploits psychological studies in crowd behaviors relating to collision avoidance and overtaking, to simulate believable behaviors. Shao and Terzopoulos [21] have integrated motor, perceptual, behavioral and cognitive components through an *artificial-life* approach, to simulate autonomous agents in a large urban setting. In recent studies, McDonnell et. al. [10], have shed light on the perceptual impact of model cloning (different agents using the same underlying motion models) based on numerical (e.g., reaction times of the subjects in identifying clones) as well as non-numerical (e.g., agent appearance) measures.

With respect to integrating multiple agents within a distributed simulation, Wang and colleagues have contributed the High Level Architecture (HLA) [25]. A layering of social interaction on environmental simulation was also proposed to model interaction between humans and natural environments [23]. The problem of congestion has been addressed in context of amusement parks by using “social coordination” to reduce the time wasted in congestion [11]. The issue of time management in MAS was discussed to alleviate some of the problems that exist in the simulation by providing “Semantic Duration Models” to help developers [8].

Murakami et. al. [12] have conducted extensive experiments in agent-based evacuation simulation, and have used plots of the number of evacuees versus time, for various evacuation strategies, and compared with control experimental results. Participatory design has been used to iteratively refine agent and interaction models based on real-world human experiments, augmented with agents in virtual space [9]. While control experiments shed light on the validity of specific strategies in simulation, and help in the refinement of models, they are harder to perform (compared to scenario videos that are beyond the control of the evaluator) and do not address the problem of global validation of a simulation system that is only available as a closed product. This also makes the existing approaches to model verification and validation [20] inapplicable, since we only have access to the outcomes of the simulations, not the models underlying these systems. Pelechano et. al. [15] use the presence of some key desirable features such as absence of shaking (unrealistic, rapid change of location), continuous movements, absence of overlapping (agents merging in space and time), pushing etc, to compare various existing simulation methodologies such as social forces, rule-based systems etc. In contrast, we seek a quantitative validation methodology. In fact, a common limitation of the available systems is that none of them can be quantitatively compared to determine the “better” or the more accurate systems for various scenarios. Neither can we quantitatively evaluate the realism of a system, compared to the actual scenarios (i.e., with human crowds) in a general and global manner. Most of these systems have been validated subjectively through visual comparisons, looking for certain macroscopic patterns (emergent behaviors) such as congestions in key locations, or egress rates. The numerical measures that are often employed, such as statistics of local patterns [6], do not offer insights into global (both temporally and spatially) validity of the systems, and are also often applied in an ad-hoc manner. That is, there is hardly any motivation behind the selection of time-points on the simulation clock and the real-world clock when these numerical comparisons are made. Even if agent-speeds are initialized to visually match reality, microscopic emergent events can quickly lead to asynchronism, such that selecting such time-points is no longer a trivial problem.

Against this general backdrop, we present our technique for the quantitative validation and comparison of crowd egress simulation systems. We are particularly interested in crowd simulations where the crowd egresses (rather than only ingress, or a mixture of ingress and egress) a given virtual environment, since these are often associated with preparedness for emergency situations, thus constituting high priority applications of crowd simulation. However, due to absence of panic-data, we evaluate our techniques in a zero-panic scenario.





**Fig. 1.** The virtual environment surfaces on which agents can walk are partitioned into convex polygons shown in various colors

### 3 The Validation Algorithm

Almost all of the simulation systems identified above include a detailed geometry of the environment (such as a sport stadium, or a subway station), partitioned into a set of  $p$  convex polygons,  $\{R_1, R_2, \dots, R_p\}$ . These polygons are the surfaces on which the autonomous agents can move, as illustrated in Figure 1. The system simulates the navigation behavior of the agents in context of the environment's geometry. In an egress simulation, there is one or more *sink* of agents (i.e., where agents exit the virtual environment), but there is no *source* of agents (i.e., no polygon where agents are spawned). Instead, the agent distribution over the regions is pre-specified at the start of the simulation, and changes only by the simulated egress behavior.

In order to evaluate such an egress simulation system, we subscribe to the general idea of seeking macroscopic patterns, but in a quantitative manner. We compare the simulation run in the virtual environment, and the actual scenario with real people navigating the corresponding real environment, with identical initial conditions. This comparison is accomplished by calculating the distance between the distributions of agents over the  $p$  polygonal regions in the simulation (*sim*), and the distribution in the actual scenario (*scn*) over the same regions,  $D(sim_t, scn_{t'})$ .  $D$  can be calculated in a variety of ways, and in an earlier draft [3] of this paper we had reported using [1]

$$D(sim_t, scn_{t'}) = \sqrt{\frac{1}{p} \sum_{i=1}^{i=p} \left( \frac{C_{sim}^t(R_i) - C_{scn}^{t'}(R_i)}{X(R_i)} \right)^2} \quad (1)$$

<sup>1</sup> The measure reported before mistakenly had some omissions, which are corrected here.

where  $C_x^t(r)$  gives the count of the number of agents in region  $r$  at time  $t$ , and  $X(R_i)$  is the maximum capacity of the region  $R_i$ .  $X(R_i)$  can easily be estimated as the ratio of the total area of  $R_i$  to the (average) area occupied by an agent. It is important to notice that the time-points ( $t, t'$ ) at which the above comparisons can be made might be on different scales, due to difference in the speeds between agents and people. We emphasize that this distinction of the time-scales is crucial in validation, but has been ignored in the past. Consider for instance, that an event (e.g., congestion) occurs at a location in the map at 5 seconds in the simulation, and at 10 seconds in the scenario. However, they last for 3 seconds and 2.5 seconds respectively, i.e., the factor to adjust the time-scale going between the scenario and the simulation is not fixed. It is unlikely that even the most accurate simulation will possess a consistent scale factor, but it is essential to validation that we account for these discrepancies. Our solution is to “best-fit” the simulation to the scenario in terms of a fixed (determined off-line) or an adjustable (determined on-line) scale factor that is optimized through the entire duration of the validation run.

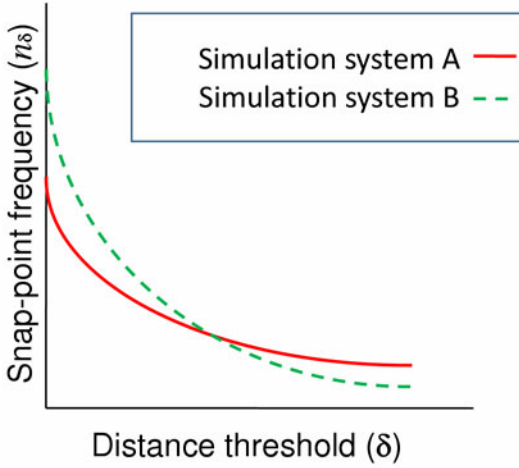
We assume that  $C_{scn}^t(R_i)$  values are given for certain discrete time points  $t_0, t_1, \dots, t_k$  at constant interval  $\tau_{scn}$ , i.e.,  $t_i - t_{i-1} = \tau_{scn}, \forall i = 1 \dots k$ . These could be generated from the snapshots of crowd video at regular intervals. We call these time points *snap points*. At the start of validation, the distribution  $C_{scn}^{t_0}(R_i)$  is replicated in the simulation (and a uniform random distribution followed within each region) and it is run for  $\tau_{sim}$  units of simulation time. Then  $D(sim_{\tau_{sim}}, scn_{t_1})$  is compared to a distance threshold  $\delta$  ( $0 \leq \delta \leq 1$ ). If the former is lower then the simulation continues to run, otherwise it “snaps”. This means that the distribution  $C_{scn}^{t_1}(R_i)$  is replicated in the simulation and it is run for another  $\tau_{sim}$  units of time. This process continues with the comparison of  $D(sim_{i*\tau_{sim}}, scn_{t_i})$  against  $\delta$  in the  $i$ th step, until we reach  $t_k$ , and at this point the total number of “snaps” so far – represented as  $n_\delta$  – is recorded as a function of  $\delta$ .

The need to check for “snaps” at a regular interval stems from the fact that most sophisticated crowd simulation systems incorporate complex agent-based models, producing non-linear behavior as a function of time. Such emergent behavior can quickly diverge from the behavior being modeled, if small discrepancies are allowed to accumulate, due to chaotic effects. Hence, any discrepancy that is determined to be sufficiently large needs to be reset, by re-matching the simulation distribution with the corresponding distribution from the scenario – a process that we call “snap”. The main idea is that a simulation system that needs to snap fewer times for a given  $\delta$ , compared to another simulation system in the same scenario, is more accurate for that scenario and for that  $\delta$ . A metric for comparison between two simulation systems in terms of accuracy for any given scenario could be the area under the curve of  $n_\delta$  vs.  $\delta$ ; the lower this area, the more accurate the simulation system. This is illustrated in Figure 2.

In order to score a system alone, we observe that there can be at most  $k$  snaps for all possible values of  $0 \leq \delta \leq 1$ . Therefore, a reasonable metric for scoring a system would be

$$Score(sys) = k - \int_0^1 n_\delta^{sys} d\delta,$$

producing a baseline score of 0 for any system that snaps at all  $k$  points.



**Fig. 2.** Example plots of  $n_\delta$  vs.  $\delta$  that might be produced by two simulation systems, A and B. The system with lesser area under the curve would be more accurate for the given scenario.

The main challenge underlying the above validation algorithm is in calculating the appropriate value of  $\tau_{sim}$ , since the timescale may differ between the simulation and the scenario, e.g., the agents may be navigating too fast compared to real people in the scenario, or too slow. Moreover, these speed-discrepancies may not stay constant, but vary from region to region and also with time. Any endeavor at such synchronization may also be affected by micro-level occurrences in reality that may be hard to reproduce spatially and temporally in simulation, such as people stopping to chat (in a zero-panic scenario), which ultimately affects the count distribution, and thus  $D$ . The premise of our validation approach is that the inevitable mismatches in micro-level occurrences can be overlooked as long as the macro-level patterns (e.g., agent count distribution over regions) match well enough. Clearly, the problem of finding appropriate times to match the simulation with the scenario at the macro-level is the key to this approach. We now describe two approaches – an off-line and an on-line approach – to calculating  $\tau_{sim}$ .

### 3.1 Off-Line (Pre-)Calculation of $\tau_{sim}$

Here we describe an off-line strategy to find the best value of  $\tau_{sim}$  that will enable the closest possible match between the simulation and the scenario. Let  $C_x^t = \sum_i C_x^t(R_i)$ , i.e., the total number of agents (either in the simulation or the scenario) at time  $t$ . Then we can represent the total number of people that have left the environment (i.e., the set of  $p$  polygons) between time  $t_0$  and  $t_j$  as  $L_{scn}^j = C_{scn}^{t_0} - C_{scn}^{t_j}$ . We can define a sequence of these values over the snap-points,  $\{L_{scn}^j\}_{j=1}^{j=k}$ , that specifies a time-series of the number of egresses between snap intervals in the scenario. Note that the sequence  $\{L_{scn}^j\}_{j=1}^{j=k}$  must be a non-decreasing sequence to match with the assumption of there being no agent source in the simulation.

In order to pre-calculate  $\tau_{sim}$  before the validation runs, we run the simulation once, populated with the distribution  $C_{scn}^{t_0}(R_i)$ , and record the total agent counts  $C_{sim}^t$  (i.e., the total count over all polygonal regions) at small intervals  $\Delta$ , thus producing a sequence  $C_{sim}^0, C_{sim}^\Delta, C_{sim}^{2\Delta}, \dots, C_{sim}^{m\Delta}$ , where  $m$  is such that

$$C_{sim}^{m\Delta} \leq C_{scn}^{t_k} \quad \text{and} \quad C_{sim}^{(m-1)\Delta} > C_{scn}^{t_k}. \quad (2)$$

The above condition is both necessary and sufficient, to terminate the above sequence. The existence of  $m$  is guaranteed by the assumption that there is no agent source in the simulation. Now, as in the case of the scenario, we can generate another non-decreasing sequence  $\{L_{sim}^j\}_{j=1}^m$ , where  $L_{sim}^j = C_{sim}^0 - C_{sim}^{j\Delta}$  is the number of agents that have egressed the virtual environment by time  $j\Delta$  from the start of the simulation. If  $\Delta$  is chosen sufficiently small, then it can be ensured that  $m \gg k$ .

Given the above set-up,  $\tau_{sim}$  can now be estimated as

$$\tau_{sim} \approx h^* \Delta$$

where integer  $h^*$  is found by solving the following discrete optimization problem

$$h^* = \arg \min_h \sum_{s=1}^k (L_{sim}^{sh} - L_{scn}^s)^2$$

under the constraint that  $kh \leq m$ . This optimization computes the best ‘‘skip’’ value (i.e., the number of consecutive points that can be skipped)  $h^*$ , for the sequence  $\{L_{sim}^j\}_{j=1}^m$ , so that the subsequence of size  $k$  built by picking points every  $h^*$  places from this sequence, matches most closely with the sequence  $\{L_{scn}^j\}_{j=1}^k$ . Thus  $h^*$  gives the optimal matching between the times scales of the scenario and the simulation, and the estimate of  $\tau_{sim}$  thus produced, clearly gets increasingly accurate as  $\Delta$  gets smaller, i.e., when  $m \gg k$ .

It is noteworthy that if the assumption of no agent source is relaxed, then the only part of the above scheme that might fail is the termination condition in equation 2. In this case,  $C_{scn}^{t_k}$  becomes a poor determinant of when the simulation must be stopped (i.e., to determine  $m$ ), as it is possible to meet the condition repeatedly. It is unlikely that some scheme of selecting a value of  $m$  (for instance, the lowest  $m$ ) from multiple possibilities will not impact the reliability of the resulting evaluation.

### 3.2 On-Line Adjustment of $\tau_{sim}$

Another technique for calculating  $\tau_{sim}$  is to adapt it on-line during evaluation of each segment between snap-points. In particular, the information generated during the run through the segment  $t_{j-1}$  to  $t_j$  is used to adapt  $\tau_{sim}$  for the segment  $t_j$  to  $t_{j+1}$ . Let  $\tau_{sim}^j$  be the value of  $\tau_{sim}$  calculated at snap-point  $t_j$ , and used for this segment. This is calculated as

$$\tau_{sim}^j = \tau_{scn} \cdot \left( \frac{C_{scn}^{t_{j-1}} - C_{scn}^{t_j}}{C_{sim}^x - C_{sim}^y} \right)$$

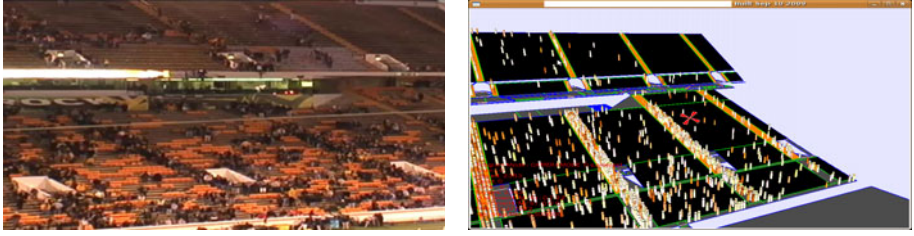
where  $x = \sum_{t=0}^{j-2} \tau_{sim}^t$ , and  $y = x + \tau_{sim}^{j-1}$ , for  $j = 1, \dots, k-1$ , with  $\tau_{sim}^{-1} = 0$ , and  $\tau_{sim}^0 = t_1 - t_0 = \tau_{scn}$ . The ratio  $\frac{C_{scn}^{t_{j-1}} - C_{scn}^{t_j}}{C_{sim}^{t_{j-1}} - C_{sim}^{t_j}}$  gives the relative number of egresses between the scenario and the simulation during the segment  $t_{j-1}$  to  $t_j$ , and acts as an estimate of how much faster (if  $< 1$ ) or slower (if  $> 1$ ) the simulation must be interrupted (and possibly snapped) for the current segment, to produce a better (expected) match with the following segment of the scenario. It is important to note that the resulting segment lengths in the simulation will not be a constant, viz.,  $\tau_{sim}^j$  for the segment between  $t_j$  and  $t_{j+1}$ , unlike in the off-line method.

Although this method is computationally cheaper than the off-line calculation of  $\tau_{sim}$ , the estimate is expected to alternate between overcompensation and undercompensation for the discrepancy in the number of egresses between the simulation and the scenario. Hence, unless  $k$  is large, the estimate may not find enough time to stabilize. Therefore we expect the off-line estimation method to produce greater accuracy for validation with a few snap-points. Generally speaking, the accuracy of the on-line method depends on the atomicity of the video data (spacing of the video shots), while the accuracy of the off-line method depends on the atomicity of the simulation data ( $\Delta$ ), and oftentimes it is easier to control the latter. This is mainly why we would prefer the off-line method in general.

## 4 Experimental Results

We have implemented both techniques for estimating  $\tau_{sim}$ , and evaluated a single segment of 7 minutes duration (with 7 possible snap-points after  $t_0$ , at 1 minute intervals) from the video footage of spectators from an actual football game at the Southern Miss stadium [13]. The segment starts with the final declaration of scores in the game, at which point spectators start to leave in large numbers. The segment ends (roughly 7 minutes later) when almost all spectators have left. We had recorded multiple games during a season, but all except one turned out to be rather one-sided which meant spectators left the stadium throughout the game and there was no coherent segment where most of those that were moving across regions  $R_i$  were actually egressing. Since the simulation system that we wanted to evaluate did not simulate a mix of casual movements and egress, but only movements with the purpose of egress, these recordings could not be used for evaluation. Consequently, we are only able to report results on one segment.

We have recently developed a multi-agent based crowd egress simulation system [2] which we used in the current evaluation against the chosen video segment. Figure 3 shows a pair of snapshots from a snap-point in the video, and the corresponding shot from our simulation system from a comparable camera position. A major challenge in comparing this system to existing open source systems was the remarkable difficulty of applying these systems to the stadium model for Southern Miss. Although this model was represented quite generally (as Google sketchups), we were unable to get another system to simulate egress behavior on this model. On the other hand, commercial softwares, such as those developed by EA Sports, do allow simulation on particularly the Southern Miss stadium (using their own model representation), but are inflexible on



**Fig. 3.** Left: A snap-shot from a game video segment. Spectators are leaving after a game, hence the stairs are more crowded. Right: The corresponding shot from the simulation at that snap-point. Stairs are shown in orange and bleachers in black.

snap points (which we believe is a useful feature and ought to be accommodated by commercial simulation softwares) and do not allow access to code. Therefore, we settled on evaluating just the simulation system that we developed, and compare the on-line and the off-line schemes in this system for two different simulation technologies.

We applied two competing simulation technologies for controlling the avoidance of collisions among the agents. Our simulation system was originally set up to work with a physics engine, and was tuned to produce the kind of movement behaviors that are observed in reality, e.g., agents moving from the bleachers (black regions in Figure 3 right) to the stairs (orange regions in the same figure), and then climbing down the stairs. Collision avoidance was also controlled by the same physics engine. To produce a variant, we turned off the collision avoidance part of the physics engine (other aspects of movement control still rested on the physics engine), and replaced it with steering behaviors [17], particularly *separation* among agents enforced. However, this variant using the steering behavior with separation actually forced the agents to separate and use the bleachers more often (since there was limited space on the stairs), which does not visually match the videos. Therefore, we expected to see this visual discrepancy between the two variants reflected in our validation measure, with the physics based collision avoidance evaluating more favorably.

Our initial experiments using equation 1 were inconclusive on which collision avoidance technique produced more accurate results. We tracked this down to the oversensitivity of the distance measure in equation 1 to small sized regions, i.e., those with low capacity. Note that there are many regions that have lower capacities than the stairs. Small regions contribute large relative errors to the sum, thus obfuscating significant differences over relatively larger regions. We addressed this by developing two new distance measures that reduced this sensitivity. One measure simply removes regions that have capacities below a threshold from the distance measure. Thus, if  $R_{trunc}$  is the set of resulting regions, the first alternative distance measure is

$$D_1(sim_t, scn_t) = \sqrt{\frac{1}{|R_{trunc}|} \sum_{R_i \in R_{trunc}} \left( \frac{C_{sim}^t(R_i) - C_{scn}^t(R_i)}{X(R_i)} \right)^2}$$

The original setting contained  $p = 101$  polygons (with a combined capacity of 26401 agents), while after removing all regions with capacities below 25,  $R_{trunc}$  contains only 35 polygons with a combined capacity of 16747.

A second distance measure explicitly weights the contributions of each region by its capacity, so that they contribute proportionally to the overall distance. This is given by

$$D_2(sim_t, scn_t) = \frac{1}{T} \sqrt{\sum_{i=1}^{i=p} (X(R_i) \cdot |C_{sim}^t(R_i) - C_{scn}^t(R_i)|)}$$

where  $T = \sum_1^p X(R_i)$  is the total capacity. As with the other alternatives, this measure is also guaranteed to lie in  $[0, 1]$ , since

$$\begin{aligned} \sum_{i=1}^{i=p} \left( X(R_i) \cdot |C_{sim}^t(R_i) - C_{scn}^t(R_i)| \right) &\leq \sum_{i=1}^p X^2(R_i) \\ &\leq \left( \sum_{i=1}^p X(R_i) \right)^2 \\ &= T^2. \end{aligned}$$

The plots of  $n_\delta$  vs.  $\delta$  resulting from the application of our validation algorithm using measures  $D_1$  and  $D_2$  are shown in Figures 4 and 5 respectively. Figure 4 shows that both the on-line and the off-line measures with the physics based collision avoidance has better accuracy (score) than those for the steering based collision avoidance. In further validation of the efficacy of the off-line measure compared to the on-line measure, we see that the physics based off-line measure has significantly better score than the physics based on-line measure, while the steering based off-line measure has a marginally better score than the steering based on-line measure. Figure 5 shows that both versions of the physics based collision avoidance have better scores than the corresponding steering based collision avoidance, once again numerically validating the visual discrepancy observed above.

It is interesting to note that with both distance measures,  $D_1$  and  $D_2$ , the on-line measure for physics based collision avoidance shows a better score than that of the off-line measure for steering based collision avoidance. In other words, the instability of the on-line measure is more than compensated by the departure from realism of the steering based collision avoidance. Another point to note is that the distance measure  $D_2$  displays a compressed range of  $\delta$  values, compared to  $D_1$  which shows a better spread of the  $\delta$  values. This is due to the fact that  $D_1$  involves normalization by smaller factors compared to  $T$  used by  $D_2$ .

Finally, Figure 6 shows the alternating overcompensation and undercompensation in the estimate of  $\tau_{sim}^j$  (shown as the ratio  $\tau_{sim}^j / \tau_{scn}$ , where the scale of  $j$  from 0 to 6 stands for  $t_1$  through  $t_7$ , i.e.,  $\tau_{sim}^0 / \tau_{scn} = 1$  is omitted from the plot) produced by the on-line method, in the experiment for physics based collision avoidance with  $D_2$  and  $\delta = 0$ . This plot also shows the 1-level for perspective. It verifies the intuition that with a few snap-points the estimate fails to stabilize to a fixed point.

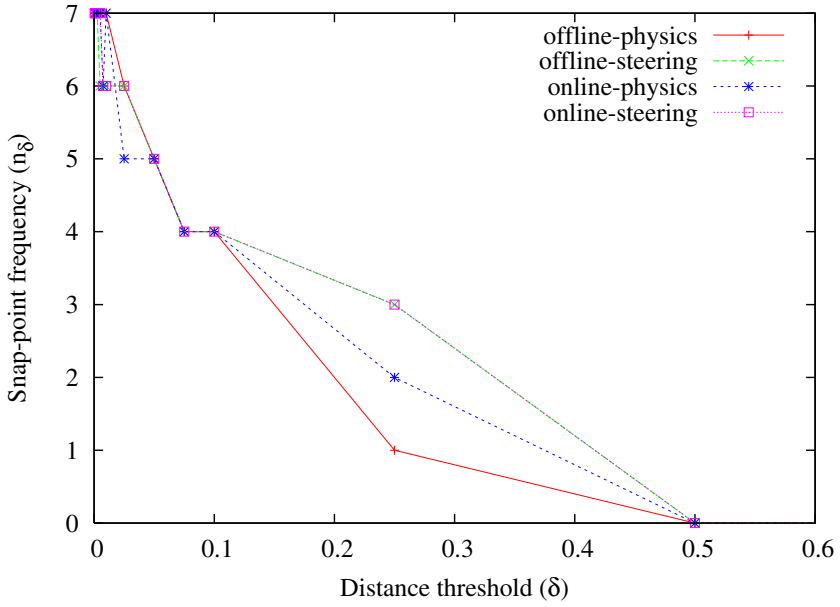


Fig. 4. Plot of  $n_\delta$  vs.  $\delta$  for the variants using distance measure  $D_1$

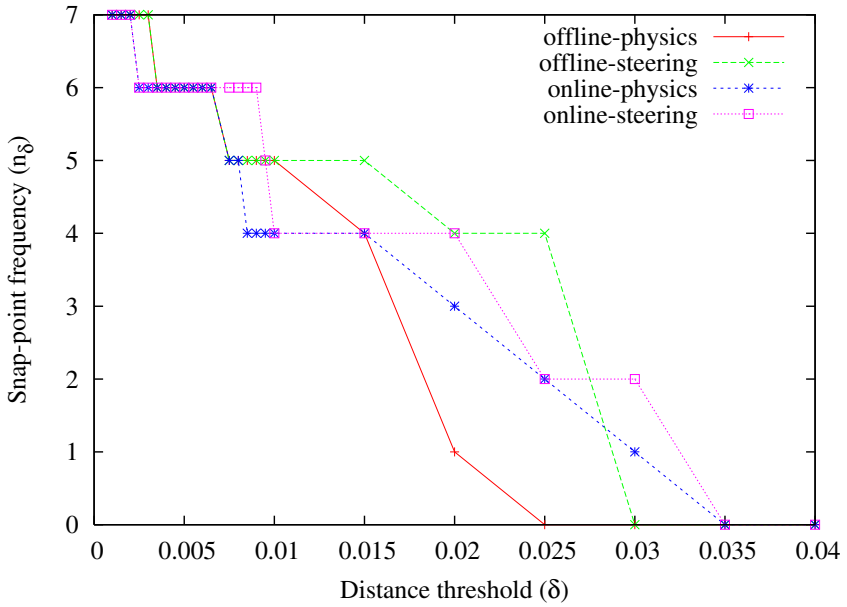
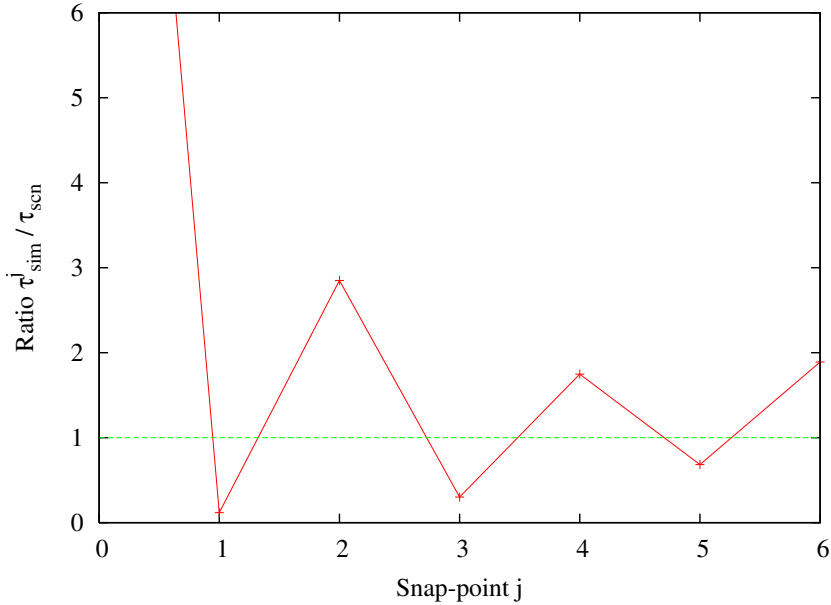


Fig. 5. Plot of  $n_\delta$  vs.  $\delta$  for the variants using distance measure  $D_2$





**Fig. 6.** Plot of the ratio of  $\tau_{sim}^j$  to  $\tau_{scn}$ , versus snap-points ( $j$ ), generated by the on-line method for physics based collision avoidance with  $D_2$  and  $\delta = 0$

## 5 Discussion

The evaluation and validation scheme presented in this paper is by no means the only possible, or the “best” (in some sense) scheme for crowd egress simulation. In fact, it is unlikely there will ever be a single scheme that will be appropriate for all such simulation systems. Our scheme is appropriate for systems

- that are closed in the sense that major changes to the underlying code are impossible, hence iterative refinement of models as in participatory/augmented design [9] is ruled out;
- that offers the evaluator the ability to stop and re-start (with user-selected agent distributions) the system at the times of the evaluator’s choice;
- that report the counts of agents on a region by region basis, as well as the capacities of all regions.

While many existing crowd simulation systems possess these features, not all systems necessarily do, and for even those that do, there may be other assumptions that may conflict with the intention behind our scheme. For instance, if the agents maintain a history of past events or an internal memory, then restarting the simulation at a “snap” would essentially eliminate a major capability of the system by which it might seek to approach reality.

It should also be noted that this scheme does not validate or compare the simulation *models* (as in [20]), only the observable behavior in the simulation's output. Furthermore, this scheme may be extended (with minor modifications) to any simulation system that involves *autonomous, intelligent* control, such as traffic simulation systems.

## 6 Conclusion

We have presented a novel technique for the validation and comparison of multi-agent based crowd egress simulation systems. Most existing systems rely on visual comparison or ad-hoc measures, whereas our technique is quantitative and principled. We evaluated a simulation system (that we have developed recently) using our technique, in particular, comparing two different technologies for collision avoidance and showing that the one with better visual match with reality also scores higher with our approach. We have also experimentally compared two approaches to calculating a key validation parameter and verified our intuition about the relative instability of one of them. The results also show a high degree of validity of our simulation system in an actual scenario of spectator egress from a football game. In the future, we would like to apply our approach to validate and compare other existing simulation systems in various scenarios. We would also like to address the issue of memory-based agents in future work.

## Funding Acknowledgement

This work was supported in part by the U.S. Dept. of Homeland Security through the SERRI (Southeast Region Research Initiative) Program at the Oak Ridge National Laboratory.

## References

- [1] Avalias. Crowdflow, <http://www.avalias.com/products/crowdflow>
- [2] Banerjee, B., Bennett, M., Johnson, M., Ali, A.: Congestion avoidance in multi-agent-based egress simulation. In: Proceedings of the 2008 International Conference on Artificial Intelligence (ICAI), Las Vegas, USA (2008) (to appear)
- [3] Banerjee, B., Kraemer, L.: Validation of agent based crowd egress simulation (extended abstract). In: Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, pp. 1551–1552 (2010)
- [4] Braun, A., Musse, S.R., de Oliveira, L.P.L., Bodmann, B.E.J.: Modeling individual behaviors in crowd simulation. In: Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA), pp. 143–148. IEEE Computer Society Press, Los Alamitos (2003)
- [5] Fukui, M., Ishibashi, Y.: Self-organized phase transitions in CA-models for pedestrians. *J. Phys. Soc.*, 2861–2863 (1999)
- [6] Helbing, D., Farkas, I., Vicsek, T.: Simulating dynamical features of escape panic. *Nature* 407, 487–490 (2000)
- [7] Helbing, D., Molnar, P.: Social force model for pedestrian dynamics. *Physical Review E* 51, 42–82 (1995)

- [8] Helleboogh, A., Holvoet, T., Weyns, D., Berbers, Y.: Extending time management support for multi-agent systems. In: Davidsson, P., Logan, B., Takadama, K. (eds.) MABS 2004. LNCS (LNAI), vol. 3415, pp. 37–48. Springer, Heidelberg (2005)
- [9] Ishida, T., Nakajima, Y., Murakami, Y., Nakanishi, H.: Augmented experiment: participatory design with multiagent simulation. In: IJCAI 2007: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 1341–1346. Morgan Kaufmann Publishers Inc., San Francisco (2007)
- [10] McDonnell, R., Larkin, M., Dobbyn, S., Collins, S., O’Sullivan, C.: Clone attack! Perception of crowd variety. In: SIGGRAPH (2008)
- [11] Miyashita, K.: ASAP: Agent-based simulator for amusement park. In: Davidsson, P., Logan, B., Takadama, K. (eds.) MABS 2004. LNCS (LNAI), vol. 3415, pp. 195–209. Springer, Heidelberg (2005)
- [12] Murakami, Y., Ishida, T., Kawasoe, T., Hishiyama, R.: Scenario description for multi-agent simulation. In: AAMAS 2003: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 369–376. ACM, New York (2003)
- [13] The National Center for Spectator Sports Safety and Security,  
<http://www.sporteventsecurity.com/>
- [14] Pan, X., Han, C., Dauber, K., Law, K.: A multi-agent based framework for simulating human and social behaviors during emergency evacuations. In: Social Intelligence Design, March 2005. Stanford University, Stanford (2005)
- [15] Pelechano, N., Stocker, C., Allbeck, J., Badler, N.: Being a part of the crowd: Towards validating VR crowds using presence. In: Padgham, M., Parkes, Parsons (eds.) Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, pp. 136–142 (2008)
- [16] Regal-Decision-Systems. Evacuation planning tool,  
<http://www.regaldecision.com/ept.php>
- [17] Reynolds, C.: Steering behaviors for autonomous characters,  
<http://www.red3d.com/cwr/steer/>
- [18] Reynolds, C.: Flocks, herds and schools: A distributed behavior model. In: Proceedings of ACM SIGGRAPH (1987)
- [19] Rymill, S., Dodgson, N.: A psychologically based simulation of human behavior. In: Eurographics UK Theory and Practice of Computer Graphics (2005)
- [20] Sargent, R.G.: Verification and validation of simulation models. In: WSC 2008: Proceedings of the 40th Conference on Winter Simulation, pp. 157–169 (2008)
- [21] Shao, W., Terzopoulos, D.: Autonomous pedestrians. In: Eurographics/ACM SIGGRAPH symposium on Computer Animation (2005)
- [22] Thalmann, D., Musse, S., Kallmann, M.: From individual human agents to crowds. In: Informatik/Informatique - Revue des organisations suisses d’informatique (2000)
- [23] Torii, D., Ishida, T., Bonneaud, S., Drogoul, A.: Layering social interaction scenarios on environmental simulation. In: Davidsson, P., Logan, B., Takadama, K. (eds.) MABS 2004. LNCS (LNAI), vol. 3415, pp. 78–88. Springer, Heidelberg (2005)
- [24] Ulicny, B., Thalmann, D.: Towards interactive real-time crowd behavior simulation. Computer Graphics Forum 21(4) (2002)
- [25] Wang, F., Turner, S.J., Wang, L.: Agent communication in distributed simulations. In: Davidsson, P., Logan, B., Takadama, K. (eds.) MABS 2004. LNCS (LNAI), vol. 3415, pp. 11–24. Springer, Heidelberg (2005)

# Towards an Architecture for Collaborative Human/AI Control of Interactive Characters

James Niehaus and Peter Weyhrauch

Charles River Analytics, 625 Mount Auburn St., Cambridge, MA 02138  
{jniehaus, pweyhrauch}@cra.com

**Abstract.** Interactive characters – widely used for entertainment, education, and training – may be controlled by human operators or software agents. Human operators are extremely capable in supporting this interaction, but the cost per interaction is high. Believable agents are software controlled characters that attempt natural and engaging interaction. Believable agents are inexpensive to operate, but they cannot currently support a full range of interaction. To combine the strengths of human operators and believable agents, this paper presents steps toward an architecture for collaborative human/AI control of interactive characters. A human operator monitors the interactions of users with a group of believable agents and acts to intervene and improve interaction. We identify challenges in constructing this architecture and propose an architecture design to address these challenges. We discuss technologies that enable the operator to monitor many believable agents at once and act to intervene quickly and on many levels of granularity. To increase speed and usability, we employ principles of narrative structure in the design of our architectural components.

**Keywords:** Believable agents, virtual humans, semi-autonomous agents, explainable AI, narrative generation.

## 1 Introduction

Interactive experiences with virtual or physically embodied characters are widely employed for entertainment, education, and training purposes. This interaction can be managed by human operators or software agents. Human operators can support the full range of human interaction and may provide deep and rich experiences to users. They can fully understand the speech, gestures, and actions of the users, and their responses can be novel and meaningful. A highly trained operator has the potential to draw users into the interaction with the character and allow users to suspend their belief for long periods of time. However, the cost of human operators is high. Operating a character in a believable and engaging manner requires advanced training, and an operator can only control a single character at a time.

Believable agents are software agents that attempt to make their communication and actions understandable to an audience. These agents are adept at managing limited interactions where users' speech, gestures, and actions often fall within a small set of possibilities [1]. Current believable agents are unable to handle the full scope of communication and action, including human-level reasoning, understanding

(e.g. natural language understanding, speech recognition, and gesture recognition), and expression (e.g. natural language generation, speech, and gesturing). However, believable agents are inexpensive to operate (though initial design and development may incur additional cost) and highly available; they can manage hundreds or thousands of characters at the cost of a single human operator.

To create highly engaging and realistic interactions with low cost and operating effort, human operators and believable agents may collaborate. For the purposes of this paper, we envision a single human operator managing a moderately sized group of believable agents who are, in turn, simultaneously interacting with a number of users in real time. We consider the number of believable agents to be on the order of tens – the maximum group size depends on the complexity of interactions, the number of simultaneous users, and the skill of the operator. The role of the operator is to improve the character interactions with the users by issuing commands to the agents, updating agent state, or taking full control of a character. With  $n$  agents a single agent can require at most  $1/n$  of the operator's time on average. Thus we assume that the believable agents are largely able to function autonomously and that they can complete interactions without operator support, though the quality of the interaction may be poor. We do not consider coordinated action by the agents, either through one-to-one communication or management by a director agent. The rest of this paper discusses unique challenges presented by this problem in Section 2, a proposed approach in Section 3, related work in Section 4, and conclusions in Section 5.

## 2 Challenges

Enabling a human operator to manage, guide, and control a group of believable agents presents several open research questions. The operator must decide when, where, and how to intervene for each agent and then execute the interventions. This task is made more difficult by the real-time nature of interaction – the characters cannot “miss a beat” or they may seem artificial and unrealistic. With these constraints, we identify a set of challenges which may be grouped into monitoring (i.e. presenting the operator with sufficient and easily understandable information to make decisions about when, where, and how to intervene) and acting (i.e., enabling the operator to execute the interventions by exerting control and influence over the agents and environment).

First, *the operator should be able to make informed decisions about when and where to intervene*. The system should provide information to the operator on the progress of each interaction. Overview information about the interaction, such as the interaction participants, when the interaction began, and the duration may aid the decision to intervene. Information about the state of the agents and whether the agents assess the interaction to be going well, going poorly, or at a stand-still may be critical to the operator's decision. The operator's time and attention will be divided between all of the agents, thus this information should be minimal and direct.

Second, *the operator should have context to decide how to intervene*. When the operator identifies a possible point of intervention, he should have the context to determine the content and style of the intervention. The operator should be able to examine a single agent and interaction with more detail. This may include a history of the interaction with topics discussed, agent state changes (e.g. emotional valences),

and actions taken. The operator may want to know additional information about the users and agents involved, such as their personalities, their abilities, or their knowledge about the other characters and environment. With this information, the operator can provide a cohesive interaction that does not break character, as well as an understandable interaction that meets the user's knowledge and interaction style.

Third, *the operator should have effective control for the intervention*. Once the operator has decided when, where, and how to intervene, the intervention must be executed through the controls afforded by the system. These controls should enable the operator to create deep and meaningful interactions, and hence, should be as unrestrictive and natural as possible. For instance, humanoid characters may benefit from an operator interface that enables full body control, mimicking the operator's stance and gestures while projecting the operator's speech, and puppeteering interfaces may be appropriate for both humanoid and non-humanoid characters. However, we stress that the control need not be complete to be effective. It may be equally as effective to control a character by sending its agent an instruction to dance as it would be to assume complete control and move each joint.

Fourth, *the operator should have efficient control for the intervention*. Because the operator's time and attention is divided between the agents, quick interventions that improve the interaction are preferable to longer interventions with the same effect. The overhead in time and effort for an intervention should be low; the operator should be able to switch between interventions rapidly. The operator should also be able to take advantage of the capabilities of the believable agents when appropriate. For example, instead of assuming control of a character and delivering a lengthy monologue, the operator should be able to instruct the agent controlling the character to deliver the monologue. The operator is then free to monitor and intervene elsewhere. Thus, the need for effective and total control must be balanced with the need for efficient and shallow control. No one scheme will work for every situation.

Fifth, *the believable agent should recover gracefully from the intervention*. Interventions will be ineffective if the agent cannot account for them once the operator turns his attention elsewhere. For example, suppose an operator causes the character to display a rising level of anger and aggression leading to an emotional outburst. Once the operator ends the intervention, the agent is unaware of the new emotional state and proceeds to calmly discuss the previous topic, presenting the users with an abrupt and incoherent emotional transition. This incoherence may be avoided if the operator can communicate the changes in interaction state to the agent as a final step in the intervention. The agent can then transition smoothly and reflect knowledge gained during the intervention.

### 3 Approach

We approach the challenges of the previous section by proposing a general purpose architecture for collaborative human/AI control of interactive characters. The architecture enables the operator to 1) monitor character interactions to determine when, where, and how to intervene, 2) act with effective and efficient control over the character's behavior, and 3) communicate new state information to the agent after an intervention. We discuss the technical and research challenges posed by each

component of the architecture and present possible techniques and approaches to address these challenges.

### 3.1 Architecture

The proposed architecture design is displayed in Fig. 1. A character in this architecture consists of an agent and a controller linked to an avatar representation in the environment. An agent has sensing capabilities which update its state, and action logic that determines the next speech, gesture, or action from this state. The commands are passed to the avatar which executes them in the environment with the users. When the operator does not send direct commands to the controller, commands from the agent’s action logic determine the avatar’s actions.

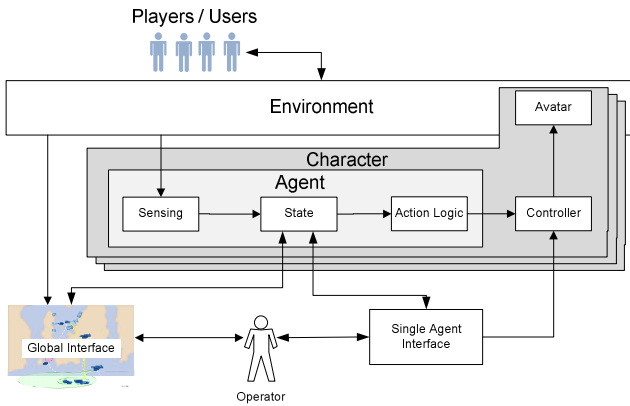


Fig. 1. Proposed Architecture

The operator monitors the characters and environment through the global and single agent interfaces. The global interface presents a high level overview of the states of all of the agents and the state of the environment. The single agent interface presents the details of agent’s state and current interaction. The operator may also act to intervene from the global and single agent interfaces. In the global interface, the operator may update the state of many agents by sending high level commands (e.g. character *X* must adopt goal *Y*). In the single agent interface, the operator may again update the state of the viewed agent, or the operator may send commands directly to the avatar through the character’s controller.

### 3.2 Explanation of Capabilities

This section describes why the components of the proposed architecture design address the challenges posed by the problem. To enable the operator to decide when and where to intervene, the architecture *provides global visualizations of agent and environment state* through the global interface. For the global summarization of agent state to be effective, it should be intelligently reported by the agent. Simple metrics such as duration of interaction will not provide enough information to form a

complete picture, and the operator will not have enough time to review detailed accounts of the interaction history (e.g. full text logs of a conversation).

Hence, the agent must process the interaction and report concise and meaningful metrics to the global visualization. For example, the agent may report rapidity of responses, confidence in interpreting user's speech, gestures, and actions, confidence in choosing correct responses, and assessment of the users' emotional states. Though none of these metrics are a comprehensive view of the interaction, combined they may provide the operator with a strong assessment of whether the intervention is necessary.

Many such metrics are concise, often reporting a single number or percentage, and readily available from current believable agent technologies. Hidden Markov Models for speech recognition compute the probability that a particular utterance matches a sequence of words [2]. This probability can be reported to the operator in the global interface as an indication of the functioning of the speech recognition, allowing the operator to intervene if the believable agent cannot understand the users' speech. Other possible interaction metrics for this purpose include distance measures for matching natural language text to stored sentences and phrases; discourse cohesion and coherence properties [3]; indicators of user psychological state and interest, such as galvanic skin response [4], posture, eye-gaze, facial expressions, and gestures [5]; and the agent's perception of its progress along its interaction goals such as number of critical information points reached or repetition of information.

To enable the operator to decide how to intervene, the architecture *provides detailed visualizations of agent state and interaction history* through the single agent interface. The interface must rapidly convey the interaction history, possibly including significant aspects of the agent state, perceived user state, and recorded speech, gestures, and actions by both character and users. For rapidly conveying this information to the operator, this causally related sequence can be structured as a *visual narrative*.

Narrative is a natural and prevalent form of communication, and narratives can present a large amount of information quickly by taking advantage of expectations and inferences about structure [6]. In this paper, a visual narrative incorporates visual elements into the presentation of the narrative to increase the speed at which elements can be recognized. We propose three principles, inspired by common narrative structure, for the organization of these visual narratives:

1. *Temporal organization* – relative time of elements is immediately apparent. Examples include a list of actions in temporal sequence or a network of a icons representing emotions connected by directed arrows.
2. *Causal organization* – the enabling causes of changes in elements are immediately apparent. An example may be short descriptions of an event next to icons for changes in emotional state.
3. *Prioritization* – only the elements contribute the most to the operator's understanding are included. An example may be when displaying a history of character gestures only displaying the gestures that proceed smiles or frowns by the users.

These principles for the design of the visual narrative enable the operator to use natural, narrative reasoning to quickly assess the agent and interaction state and to intervene



without causing an abrupt gap in the interaction. The operator knows the “what” and “how” of the interaction by the indication of elements in the temporal organization. The operator knows the “why” of the interaction by the causal organization. And the operator is focused on the most critical elements by the prioritization.

The specific design and presentation of the visual narrative depends on the internal structure of the believable agent and its type of interaction. For example, if the agent is acting as a tour guide and contains a model of emotion, a visual narrative might be a map displaying the agent’s path, an icon at each location for emotional state, and an image or short description of the main cause of each emotional transition. As another example, if an agent is a conversation agent that maintains beliefs about the world and goals for conversation topics, the visual narrative could present an ordered list of each statement by the agent or a user that caused a change in beliefs or an achievement of a topic goal alongside a set of topic goals yet to be achieved.

To enable effective and efficient control for the intervention, the architecture *allows the operator to intervene at multiple levels of resolution by updating the agents state and sending commands directly to the avatar* through the global and single agent interfaces. These interventions may be *direct interventions* in which the operator controls the avatar directly or *indirect interventions* in which the operator updates the believable agent’s state or otherwise sends commands to the agent to influence behavior. The range of interventions should match the capabilities of the agents and the demands on the operator. If the believable agents are capable in a wide range of tasks, indirect intervention can be given to an agent to execute while the operator is left free to monitor other agents. If the agents are only able to provide a minimum of interaction, then the operator may use direct intervention to control the agents during much of the interaction. In the latter case, the operator will be able to effectively manage only a few agents.

The form of the possible indirect interventions will be shaped by the internal structure of the believable agents. If the agents are goal driven, the operator may set or remove goals. If the agents maintain an emotional model, the operator may tweak the agent emotions. Similar interventions may be possible for agents that create plans, maintain beliefs, assign probabilities, or model conversation topics.

Direct intervention is shaped by the available control interfaces and the form of the avatar. At one end of the spectrum, the operator may push buttons on a controller to move and play character animations as one would control a video game character. At the other end of the spectrum, the operator may have a full body motion capture suit for the character to mimic his every gesture and facial movement. Speech may be projected through a microphone, or the operator may type text to be presented to the users.

To enable the believable agent to recover gracefully from the intervention, the architecture *allows the operator to update the agent state as the last step of the intervention* through the single agent interface. Understandable transition between emotional states, plans, beliefs, or goals is one way in which believable agents increase believability. For example, a character that is making a sandwich, turns to see the toaster on fire, and sprints from the kitchen is more believable than a character that is making a sandwich and then suddenly sprints from the kitchen for no apparent reason. For these transitions to be portrayed by the agent, the agent must know the previous action, the next action, and their relationship to each other. If an operator performs a direct intervention with an agent, the agent’s internal model of itself, the

user, and the interaction may no longer be valid by the end of the intervention. The agent must either watch and understand the intervention, updating its model as the interaction proceeds, or allow the operator to update its model to reflect the new state. In the first case, techniques the agent already uses when interacting with the users may be employed to attempt understanding of both sides of the interaction, the users' actions and the operator's actions.

However, the operator may often choose direct intervention because the agent is not capable of understanding the needs of the current interaction, and it may be preferable for the operator, with a clear understanding of the interaction, to update the agent on the intervention by relating the general "gist" of events and their meanings. For very simple believable agents, this may be a task of setting the values of one or two variables, but for agents of reasonable complexity we view this task as *storytelling*. Many of the same reasons for the visual narrative explanation apply here. The telling of a narrative is a concise representation of a sequence of events and the reasons for those events. The agent has access to the temporal and causal structure of the narrative, and only the most important events need to be related. The operator does not need to specify complete temporal or causal structure; instead, it may be inferred by the ordering and grouping of events. Thus the operator can quickly enter a brief narrative about the direct intervention in a format the agent can process before leaving the agent. The agent can process the narrative structure as if it were experiencing the interaction live and continue as if it had been managing the interaction the entire time.

## 4 Related Work

One of the first proponents of believable agents was the Oz Project [1]. Believable or broad agents differ from traditional AI agents. They focus on a broad range of shallow behavior instead of a restricted set of deep behavior, and they exhibit personality and manage user perception over striving for rationality. Loyall and Bates [7] describe an early implementation of believable agents using the Hap behavior language, and further extensions have incorporated models of emotion [8] and basic natural language generation [9]. Recent work has improved the capabilities of these agents, sometimes referred to as virtual humans or social agents, in the areas of speech, natural language processing, and emotional models [10] [11]. Applications have included military training and teaching of interpersonal skills [12] [13]. Much of this research has centered on fully autonomous believable agents whereas the focus for the present paper is on an architecture to support semi-autonomous believable agents.

Effective and efficient depictions of internal believable agent state is a largely unexplored research area. Agent systems such as Woggle World [7] and Soar [14] often use debugging output to aid in the development of the agents, but do not present the user with this information. This information may be graphed as in the Woggle World agent brain "EEG" [Weyhrauch and Neal Reilly, Personal Communication], or may be simply printed as text debugging statements. However, more research is needed to make this information meaningful to non-developer users. Explainable AI systems record the agent's decision process and can be later queried by the user to discover the agent's reasoning [15]. Of note in this area is the Debrief system [16],

which generates explanations from agent traces using Soar's rule mechanisms. These explainable AI systems are often intended to function when the operator's time is not constrained as a type of after-action review, but rapid assessment of the agent state is needed in our proposed architecture.

Collaborative control and adjustable autonomy have previously been explored for collaboration between humans and teams of robots. Fong et. al. [17] describe a method for controlling teams of driving robots using both dialog and sensor visualizations to pass information. An adaptation of this interface would be appropriate to fulfill a portion of the monitoring and command issuing capabilities discussed above. Olsen Jr. and Wood [18] investigate the effects of increasing the number of robots, termed "fan-out", on the effectiveness human and robot team collaboration, finding that adding robots steadily decreases performance. Performance with believable agents is yet to be tested, but would likely follow similar trends as increasing demand is placed on use of the operator. Thus, future implementations should consider carefully the time and attention of the operator. Goodrich et. al. [19] describe an evaluation of the effects of adjusting the level at which the human and robot collaboration occurs, using a range of autonomy: full autonomy, goal-biased autonomy, waypoints and heuristics (for navigation), and intelligent-teleoperation with a joystick while allowing the robot to change course if obstacles are met. This range of autonomy is similar to the control offered by our global and single agent interfaces, and may include appropriate approaches for believable agents. While research in robotic collaborative control and adjustable autonomy addresses some of the same issues required by collaborative control of believable agents, issues uniquely emphasized by collaborative control of believable agents are 1) expressing the state of dialog and interaction, 2) intervention to produce high quality interaction, and 3) communicating the results of intervention to the agent.

## 5 Conclusions

This paper has presented first steps in an architecture for human/AI collaborative control of interactive agents. We have identified technical challenges of this task, proposed an architecture for addressing these challenges, and discussed architectural components using narrative structure as design inspiration.

## Acknowledgements

Special thanks to Scott Neal Reilly for discussions on this topic and to Moira Racich for proofing revisions.

## References

1. Mateas, M.: An oz-centric review of interactive drama and believable agents. In: Veloso, M.M., Wooldridge, M.J. (eds.) *Artificial Intelligence Today*. LNCS (LNAI), vol. 1600, pp. 297–328. Springer, Heidelberg (1999)
2. Rabiner, L.R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE* 77(2), 257–286 (1989)

3. Graesser, A.C., McNamara, D.S., Louwerse, M.M., Cai, Z.: Coh-Matrix: Analysis of Text on Cohesion and Language. *Behavior Research Methods Instruments and Computers* 36(2), 193–202 (2004)
4. McQuiggan, S., Lee, S., Lester, J.: Predicting User Physiological Response for Interactive Environments: an Inductive Approach. In: *Proceedings of the 2nd Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pp. 60–65 (2006)
5. Kapoor, A., Mota, S., Picard, R.: Towards a Learning Companion That Recognizes Affect. *AAAI Technical Report FS-01-02* (2001)
6. Graesser, A., Singer, M., Trabasso, T.: Constructing Inferences During Narrative Text Comprehension. *Psychological Review* 101, 3.371–3.395 (1994)
7. Loyall, A.B., Bates, J.: Real-Time Control of Animated Broad Agents. In: *Proc. Fifteenth Annual Conference of the Cognitive Science Society* (1993)
8. Neal Reilly, W.S.: Believable Social and Emotional Agents (PhD Thesis) (Rep. No. CMU-CS-96-138). Carnegie Mellon University, Pittsburgh, PA (1996)
9. Loyall, A.B., Bates, J.: Personality-Rich Believable Agents That Use Language. In: *Proceedings of the First International Conference on Autonomous Agents*, pp. 106–113 (1997)
10. Gratch, J., Rickel, J., Andre, E., Cassell, J., Petajan, E., Badler, N.: Creating Interactive Virtual Humans: Some Assembly Required. *IEEE Intelligent Systems*, 54–63 (2002)
11. Loyall, A.B., Neal Reilly, W.S., Bates, J., Weyhrauch, P.: System for Authoring Highly Interactive, Personality-Rich Interactive Characters. In: *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2004)
12. Paiva, A., Dias, J., Sobral, D., Aylett, R., Sobreperez, P., Woods, S., et al.: Caring for Agents and Agents That Care: Building Empathic Relations With Synthetic Agents, pp. 194–201 (2004)
13. Hill, R., Gratch, J., Marsella, S., Rickel, J., Swartout, W., Traum, D.: Virtual Humans in the Mission Rehearsal Exercise System. *Künstliche Intelligenz* 4(03), 5–10 (2003)
14. Laird, J.E.: Soar: An Architecture for General Intelligence. *Artificial Intelligence* 33(1), 1–64 (1987)
15. van Lent, M., Fisher, W., Mancuso, M.: An Explainable Artificial Intelligence System for Small-Unit Tactical Behavior. In: *Proc. Sixteenth Conference on Innovative Applications of Artificial Intelligence (IAAI 2004)* (2004)
16. Johnson, W.L.: Agents That Learn to Explain Themselves, pp. 1257–1263 (1994)
17. Fong, T., Thorpe, C., Baur, C.: Multi-Robot Remote Driving With Collaborative Control. *IEEE Transactions on Industrial Electronics* 50(4), 699 (2003)
18. Olsen Jr., D.R., Wood, S.B.: Fan-Out: Measuring Human Control of Multiple Robots. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 231–238 (2004)
19. Goodrich, M.A., Olsen, D.R., Crandall, J.W., Palmer, T.J.: Experiments in adjustable autonomy. In: *Proceedings of IJCAI Workshop on Autonomy, Delegation and Control: Interacting with Intelligent Agents* (2001)

# An Architecture for Directing Value-Driven Artificial Characters

Rossana Damiano and Vincenzo Lombardo

Dipartimento di Informatica and CIRMA,  
Università di Torino, Italy  
{rossana,vincenzo}@unito.it  
<http://www.cirma.unito.it>

**Abstract.** Agent architectures have proven to be effective in the realization of believable characters, but they stay at odds with the notion of story direction, that is difficultly reconciled with the characters' autonomy.

In this paper we introduce the notion of character's values to mediate between agent architecture and story direction in storytelling systems. Modern theories of drama view story advancement as the result of the characters' attempt to maintain or restore their values, put at stake by unexpected events or antagonists. We relate characters' values with their goals; the activation and suspension of goals depend on the values that are put at stake by the progression of story incidents. Values and goals are integrated in a computational framework for the design of storytelling systems in which the direction is defined in terms of characters' values.

## 1 Introduction

In story-based artifacts, characters are the medium by which the story direction is conveyed to the audience. However, their behavior must be controlled strictly to achieve an effective story direction. This paper addresses this issue through the notion of characters' values. The notion of character's value has emerged in scriptwriting theory as a major propulsive force in story advancement. First stated in Egri's definition of drama premise [13], the notion of value underpins most of the subsequent work conducted in scriptwriting [8], until the recent formulation stated by McKee [19] about cinematographic stories. In McKee's reading, the plot must be designed so as to put at stake the values characters care for. The progression of the story follows a cyclic pattern: a character follows a line of action suitable to preserve its values, when some event (typically, a twist of fate or an antagonist's action) occurs that invalidates its line of action, and puts other, more important, values at stake, requiring the character to abandon or suspend its current line of action and devise yet another line of action to restore them.

For example, in Bond movies, the hero must defeat an arch-villain who threatens the human kind. As he devises a clever plan to neutralize his antagonist, the

value at stake, initially limited to the ‘security of the country’, becomes increasingly higher as an effect of the counter attacks of the antagonist, with a climax that invariably ends with the removal of the threaten. In courtroom dramas, a solitary lawyer fights against injustice; the climax puts at stake the lawyer’s self-achievement, then the fate of the victim of the injustice, until the lawyer becomes involved in some kind of direct opposition against the law institution.

The definition and formalization of the notion of story direction is still an open issue in storytelling systems, especially for the paradigm of emergent narrative, where the interaction with the user and the presence of autonomous characters stay at odds with the accomplishment of a direction. Although this notion is implicit in several models, for example, such as plot points [21], structural story models [16] or narrative mediation [28], an agreed-upon definition of direction is still lacking. Our claim is that the story direction can be conceptualized in terms of the sequence of characters’ values the story puts at stake. The values abstract from the actions actually carried out by the characters and this sequence forms the ‘direction’ of the story [32]. The direction conveys a type of universal meaning: for example, Bond movies tell the audience – through infinite variations – that evilness can be defeated by the courage of a solitary hero, courtroom dramas say that justice will eventually triumph.

In this paper, we address the following research questions:

- How can the story direction be formulated in terms of *characters’ values*?
- How can an architecture for interactive storytelling incorporate the notion of *characters’ values* to tie the plot to a specific story direction?

The paper is structured as follows. First, we describe the features that characterize a value-sensitive narrative system (Section 2). Then, we introduce the model of value-sensitive deliberation (Section 3) that we use to implement characters in such system (Section 4). In Section 5, we sketch a reference architecture for value-based narrative storytelling. Conclusions and future work end the paper.

## 2 Motivations for Value-Sensitive Storytelling

In this paper, we propose character’s values as a high-level guidance for the plot development. The notion of value belongs to the realm of ethics and economics [1]; a value is a subjective assignment of importance to some type of abstract or physical object. In scriptwriting, the moral nature of direction has been explicitly reaffirmed [19,36], following a tradition that dates back to [26]. The purpose of using values to drive the development of the plot is two-fold. On the one side, we want to give to authors a conceptual tool, the notion of value, they are familiar with. On the other side, we want to endow artificial characters with a way to drive the selection of goals. While intelligent agents have been acknowledged to provide a valid model for characters’ rationality, the mechanism by which they select their goals has not been investigated to depth. Here, we claim that character’s goals respond to values at stake, and provide a framework for driving

the development of the plot in which the direction is not directly expressed in terms of characters' goals, but stated in terms of values, which characterize the language of drama. The approach proposed in this paper is to augment the agent architecture with values and use embedded values as a drive for storytelling, with a story direction that results from the actions characters undertake to reestablish their values at stake, in line with the paradigm of emergent storytelling.

The paradigm of intelligent agents offers an operational way to design and implement characters in interactive storytelling, as shown by a number of applications [23,27,2,24,14]. From a theoretical perspective, it has been argued that characters, the primary medium for the audience identification [9], are expected to be rational agents by the audience [30] and must manifest an intentional behavior to acquire believability. So, characters must maintain their believability along the story, i.e., they must behave according to the basic requisites that are usually attributed to intelligent agents, who act in a social and physical world, according to some rationality constraints. For example, the generation of interactive stories by the system described in [29] relies on the assumption that a rational model of the characters' behavior is a precondition for equipping characters with an 'expressive behavior' [31], i.e. meaningful elements that help the audience making sense of the motivations underlying their behavior.

One of the mostly employed agent models is the Belief-Desire-Intention (BDI) characterization of agents [4]. BDI-based architectures have proven to be a solid and effective basis for the design and implementation of character-based storytelling systems and the availability of programmable agent frameworks [7,25] has promoted their utilization in practical applications. Recent work has explored the practical and theoretical issues of their use in games [12,17].

Agent architectures, that are effective in implementing characters' rational behavior, do not have the sufficient expressivity to account for the "divergent moral commitments and institutional obligations" that, according to Bruner [6], constitute the privileged object of stories. Since values detain an abstract and symbolic meaning, though, they do not exhibit a direct correspondence within the rationality [33]. Although the BDI model allows creating believable characters from a rational point of view, some decisional oppositions cannot be dealt with on a purely rational basis. In stories, a character's intentions are often traded-off against unexpected options, brought in by some external source of change (other characters' actions or unintentional events), that challenge the character's established commitment on moral grounds (see also the work by [3]). We believe that the notion of value can be an effective tool to model this kind of situations, characterized by a prominently moral nature. Values are subjective, and different individuals acknowledge different values, arranged into subjective 'scales of values' [33]. By providing the ranking of importance of some type of abstract or physical object for a character, values reveal the inner nature of that character. So, different characters react differently to values at stake, as a consequence of the values they care for and the importance they attribute to them.

The main feature of value-based storytelling is its capability to use the notion of characters' values to constrain the advancement of a narrative by affecting the characters' behavior. In order to design a system for value-driven interactive storytelling, we assume a character-based architecture. We tie the character's deliberation to values at stake, and we formalize the story direction in terms of the values it puts at stake for the characters, given the priorities of values set up by the author. Each story advancement occurs because a character's value is put at stake and, conversely, the character is required to execute actions that may put other characters' values at stake. If the system is provided with an environment simulator, unintentional events may also put values at stake.

The interactive system manipulates the characters' values by putting them at stake; characters react by forming value-dependent goals (or altering the status of their goals) and devise plans to achieve them. In the absence of relevant modification in the state of their values, characters behave autonomously, pursuing their intentions through the execution of their current plans.

### 3 Value-Sensitive Characters

We model a character as a BDI agent, with beliefs, high-level goals (or desires) and intentions (or plans) and we augment the BDI agent definition with the notion of value. Finally, we link the character's deliberation to its scale of values. A character is a 4-tuple  $\{B, D, I, V\}$ , where  $V$  represent character's values. Note that, here, we consider only the subjective dimension of values, and do not consider their relation with an external social system.

Each character features a **set of values**  $V$ . Each value is polarized, with a negative or positive **polarity**, and is associated with a **condition**. The negative polarity of a value means that, when the value condition holds in a certain state of the world, the value is violated; the positive polarity corresponds to the value being in force. In order to let characters arbitrate among their values, values are assigned a **priority ranking** that ranks them according to their importance.

A **value**  $v$  is defined by a set of constructs  $(p, c, r)$  where  $p$  is a negative or positive polarity,  $c$  is a ground logic formula and  $r$  is a priority (a real number). We pose the restriction that, for some value  $v$ , the conditions  $c$  cannot be inconsistent and the priority  $r$  must be the same for all constructs. If the condition  $c$  of a value  $v$  holds in a certain state of the world, that value is *at stake*. A character's **record of the values at stake**,  $VaS$ , is a set of constructs  $(c, p, r)$  for which  $c$  holds in the current state of the world or is expected to hold in the future, according to the character's beliefs. The character's record of the values at stake ( $VaS$ ) is a dynamic structure, updated along the progression of the story.

The function  $Character.VaSupdate(V, B)$  updates the  $VaS$  of a character by matching the conditions of the character's values  $V$  with the character's belief state  $B$ . From the point of view of agent architectures, the monitoring of values can be expensive. However, we assume that the set of character's values  $V$  has a limited size, since they are intended as general instruments for the



regulation of behavior. In this sense, values can also be seen as means to limit an agent's options to those that affect its values, in line with the theory of bounded-rationality [5].

The character's beliefs include not only the current state of the world, but also its expectations about how it may evolve. The expectations are computed by verifying if, from the current state of the world, any state can be derived in which any conditions of its values hold, as an effect of the character's own actions or other characters' actions. This mechanism is necessary to make the character proactive with respect to the compliance with its values, i.e., to model the fact that the character opportunistically monitors the possibility of bringing about its values when possible and anticipates the moves of the others when they may put its values at stake [4].

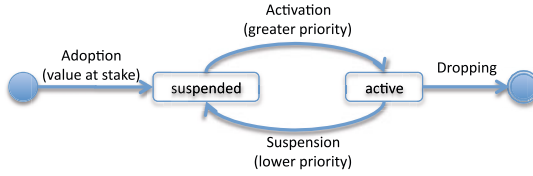
We assume that the character has multiple goals, whose formation is driven by the values listed in its value system, but that the character is committed to only one goal at a time (the one with the highest priority, determined by the ranking of values). Beside value-dependent goals, the character may also form instrumental goals, i.e., goals that are only aimed at establishing the conditions to achieve the value-dependent ones.

So, the character's beliefs include specific knowledge about its values and the dynamic record of the current relevance of its values (the record of the values at stake). The integration of values into the agent model requires that agent loop to have the following form:

1. **Monitoring.** The character updates its belief state by performing sensing actions; they include the character's expectations about the outcomes that may be generated by the behavior of the other characters, obtained by performing a (possibly limited) form of intention recognition.
2. **VaS update.** Then, it updates the record of the values at stake, *VaS*, given the new beliefs and expectations.
3. **Goal adoption.** If the values at stake have changed, the character forms new goals accordingly, following the rules for the formation of value-dependent goals listed below (Section 4).
4. **Meta-deliberation.** Through meta-deliberation, the character decides whether the state of its goals must be modified, depending on two factors: the ranking of the values that have determined the formation of the value-related goals and the achievability of the goals themselves. As a result of this process, the agent may commit to a new goal (new active goal), suspending or dropping the previously current goal, or adopt new goals without them to become active.
5. **Planning.** If the character commits to a new goal, it devises a plan to achieve it. If no plan can be found, the character drops the goal since it is infeasible.

---

<sup>1</sup> Since expectations pose problems for the computational complexity of practical systems, the look-ahead process may be limited in practical applications, for example by constraining it to few steps.



**Fig. 1.** A graphical representation of goal states and transitions including values (adapted from [30])

6. **Execution.** The next step of the character’s current plan is executed. Finally, the character evaluates the validity of its current plan: from time to time, the character may realize that the current plan is not valid anymore and re-plan (without modifying the related goal).

Two aspects of this loop require further comments. First, the formation of goals is governed by the notion of value; the agent’s proactivity is confined to its reactivity to the contextual relevance of values: this is a strong limitation for an agent, but a reasonable one for a character, whose behavior is only functional to a the communicative context set by the author. Second, the definition of the character’s values is kept separated from their influence on the character’s commitment, so that the choice of an appropriate policy to account for values in meta-deliberation is left to the autonomy of the designer of the storytelling architecture and can reflect the narrative design of the author.

## 4 Values at Work

When a character realizes that some value is at stake, it is expected to modify its commitment accordingly, by forming a goal (**value-dependent goal**) that contributes to re-establish the value at stake. The way goals arise and are affected by values can be grasped effectively by the framework by [35], which provides a unifying account of goal types and describes how goal state is transformed as a consequence of the modifications of an agent’s beliefs. According the operational architecture in [35], *adopted* goals remain *suspended* until they are ready for execution (i.e., they are in *active state*), than possibly suspended again if a more important goal is adopted. Goals can eventually be *dropped* if certain conditions hold, namely, when the rationality constraints stated by [10] are met.<sup>2</sup> In the following, we assume this framework, and we only specify the role of values at stake in the state transitions of goals, according to simple automaton represented Figure 1.

<sup>2</sup> In order to separate commitment to goals from commitment to plans, the model in [35] acknowledges different conditions to manage the status change of a goal in the presence or in the absence of a plan to achieve the goal. Since this difference is irrelevant for our purposes, here we consider only the overall set of goal conditions.

- Given a construct  $v(p, r, c)$  in  $VaS$  (the record of the values at stake), the character formulates and *adopts* a set of goals that have the conditions of its values at stake as an object.
  1. If the condition  $c$  holds in the character’s *expectations* and the associated polarity  $p$  is *positive* (i.e., it corresponds to the value being in force in that state), the character forms the goal to achieve that state of affairs (*achievement goal*). Expectations model the opportunity for the character to achieve the compliance with her/his own values. According to the framework in [35], this is an *achievement goal*.
  2. If the condition holds in the *present* or in the character’s *expectations* and the associated polarity is *negative*, the character forms the *reactive maintenance goal* to achieve any state of the world in which that condition does not hold (any state  $s$  such that  $c \notin s$ ). In most cases, this goal involves the need to contrast the behavior of an antagonist who is about to bring about a state of affairs in which the condition holds<sup>3</sup>

After a character adopts a goal, as a consequence of a value put at stake, the goal state can vary, as the story advances, in the following way:

- Goals become *suspended* immediately after their adoption. In our model, the activation of goals depends on the priority of the associated values, which reflects their importance for the character. Multiple matches with the same value cause multiple goals to be formed and *adopted*.
  - If the priority  $r$  of the value at stake  $v$  is higher than the priority of the current values at stake ( $\forall (p_i, c_i, r_i), (v \neq v_i) \rightarrow (r_i < r)$ ), the related goal becomes *active*. At this point, the character can *generate a plan* to achieve it and starts to *execute* it. The currently active goal may be have already been active in the past: if so, the character may resume the execution of a previously suspended plan, if still valid.
  - Or, the new goal may not become active immediately, and be activated later when no other goals with higher priority are left (because they have been achieved or abandoned, two conditions that equally result in the dropping of the goal) or never move to the active state. If a new goal is formed in reaction to a value at stake that has a higher priority, the active goal is *suspended*.
- A goal is finally *dropped* if the character recognizes that it is impossible to find a plan to achieve it or if it has been achieved, or if it is not relevant anymore, independently of its priority.

For an example of how a value-based framework can be applied to real stories, see [11].

<sup>3</sup> The number of states that match the definition above can be effectively limited by considering the specific events or actions that have established or are going to establish the condition  $c$ , and focusing the character’s deliberation on the possible ways to undo the effects of these actions or events.

## 5 Story Direction and Reference Architecture

In this section, we define the notion of story direction in terms of characters’ values and describe a reference architecture for value-sensitive interactive storytelling systems. This architecture is not intended for being implemented as it is, but to serve as the basis for testing the feasibility of constructing a value-based, interactive storytelling system.

To define the story direction, we start from the specification of the characters. It includes all the elements that are necessary to define the characters as rational, value-driven agents: beliefs, scales of values, planning knowledge. Characters are not initialized with goals, since all characters’ goals are formed along the story as a response to values at stake (see Section 4). The story direction is defined by specifying, for each character, a set of values to be put at stake and the order by which they must be put at stake. For each character, this order must be consistent with the ranking provided by her/his value specification, i.e., values with higher priority must be put at stake later, so as to comply with the well known notion of “dramatic climax” [15]. Values at stake challenge the character’s system of values. The direction specifies that a certain subset of the character’s values is put at stake and defines how the subset changes as the story advances (by doing so, we admit that values at stake may be brought in force again by other characters).

In summary, the **direction** is a sequence of pairs containing a value and a boolean flag to indicate if the character is expected to be successful in restoring that value:  $Direction = \{\{V_1, bool_1\}, \dots, \{V_n, bool_n\}\}$ .

The reference architecture (see Figure 2) takes as input the definition of a story world, a set of characters, and a story direction. The system interactively generates a plot (bottom left) in which characters’ values are put at stake, and the characters respond to values at stake by adopting new goals. Each character is represented as a BDI agent integrated with values (top left), according to the model described in Sections 3 and incorporates the value-sensitive deliberation model described in Section 4. The architecture also includes a simulator of the story world inhabited by the characters (bottom right), a set of “triggering events” (top right), i.e., a list that associates to each value condition a set of events that make the condition true in a given context. A full-fledged representation of the characters’ mental states is maintained only by the individual characters: so, the system relies on the characters’ processes to generate the plot incidents from the current state of the story world. The system represents the plot as a sequence of *incidents*, annotated with their relation to characters’ goals and values.

The core of the architecture is the Story Manager, a sort of drama manager in the sense established by [18], that drives the ongoing story towards the direction given by the author by coping with the interaction with the user. In order to make the plot advance, characters’ values are put at stake according to the story direction. When a character’s value is put at stake, the character reacts at the deliberation level, by forming a new value-dependent goal and devising a plan to achieve it if its priority overcomes its previous commitment. If no new value

is at stake, a character continues the execution of its current plan, replans if necessary, or else resumes the execution of a previously suspended plan.

Furthermore, the story manager enforces two requisites on the story. Given a character, the *incident-value consistency* holds when:

- For each character’s value-at-stake  $v$ , the character has formed (and not dropped) the value-dependent goals prescribed by the value-sensitive deliberation model in Section 4.
- The priorities of the character’s value-dependent goals match the scale according to which the characters’ values are ranked (with the highest priority goal being the currently active one).

In other words, after each incident, the character must be compliant with her/his values and their priorities. Moreover, the story must be consistent with the given direction. Given a character, the *story-direction consistency* holds when:

- A character’s values are put at stake according to the sequence prescribed by the story direction.
- The character is successful in restoring its values as prescribed by the direction.

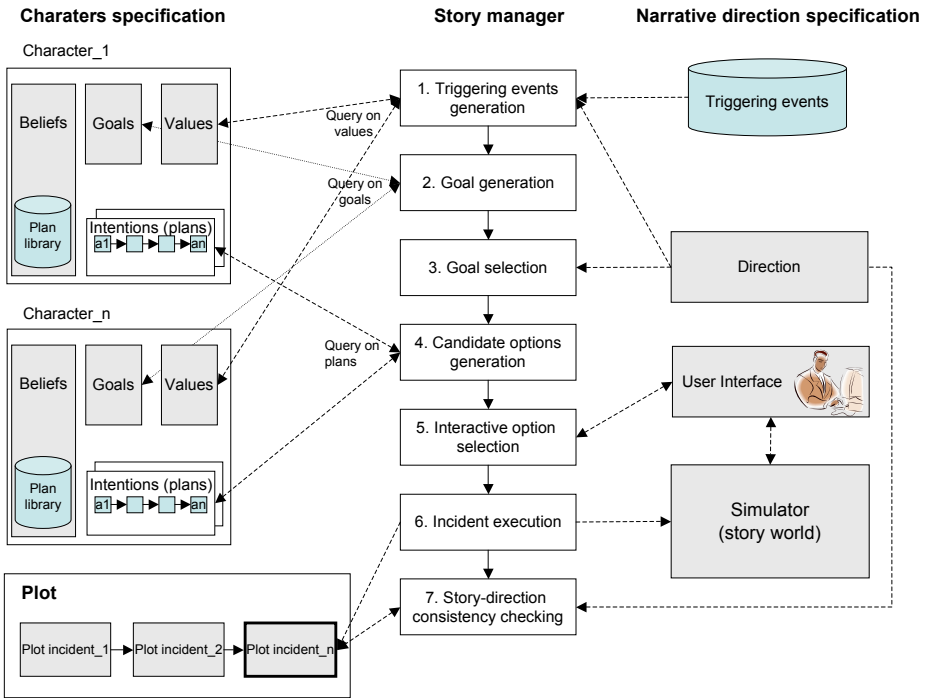
In other words, the system must correctly enforce the direction in the story.

The loop executed by the story manager is the following:

1. **Triggering events generation.** The system matches the value condition of each value against applicable triggering events, considering only, for each character, the next value to be put at stake according to the direction stated by the author (the first one when the system is initialized). When a value condition matches a triggering event, it means that the value can be put at stake by applying that triggering event in the current story world, so the story manager obtains a list of the candidate values at stake (*query on values* in the Figure 2). If no value can be put at stake (for any character), the story manager returns a failure.<sup>4</sup>
2. **Value-dependent goal generation.** Given the list of candidate values at stake, the system queries the characters to know, for each character, which candidate values at stake may lead that character to form a value-dependent goal and to make it active, given the current story world (*query on goals* in the Figure 2). From the goal dynamics described in Section 3, a goal becomes active only if there is a viable plan to achieve it. For each candidate value at stake, each character performs an anticipatory reasoning process in order to answer the query: assuming the adoption of the value-dependent goal in response to the candidate value at stake, it tries to devise a plan to achieve this goal. If this attempt is successful (at least one plan is found), the value is susceptible to become an active goal.

---

<sup>4</sup> In order to relax this constraint, the system could allow for a fixed number of cycles to be executed without new values at stake, but with the characters advancing with their plans.



**Fig. 2.** The reference architecture for a value-sensitive system for interactive storytelling. The architecture encompasses a set of characters (value-sensitive agents, left), a story engine that puts the characters' values at stake and generates the user options based on the response of the characters (center). User's choices determine the sequence of generated incidents (bottom left). On the right, the author-defined knowledge sources (direction, triggering events, story world). Dashed lines represent data flow, solid lines represent control flow.

### 3. Selection of candidate value-dependent goals.

After the characters communicate the value-dependent goals that may become active, the system verifies, for each of them, if it conflicts with the other characters' active goals. New characters' goals, in fact, are likely to conflict with self and other characters' goals.

The first source of *conflict* with the new active goal may be the previously active goal of the same character. Since we assume that, in the direction, the order by which a character's values are put at stake is consistent with that character's ordering of values, the new value-dependent goal is, by definition, related to a higher priority value than the current one. In a simplistic approach, the character suspends the previously active goal and the new one becomes active. However, in an ideal framework, characters should be able to merge the two plans, when compatible. No matter the chosen approach (simplistic or ideal), inner conflicts are not only tolerated, but even desirable for creating dramatic effects (see for example the work by [3]).

Secondly, a conflict may arise with the goals pursued by other characters. There are two main approaches to detect conflicts between goals. If the goal language is sufficiently expressive, conflicts may be detected by matching the definition of the potentially conflicting goals [34]. Alternatively, conflicts can be detected by examining the corresponding plans to verify if they are compatible. In this case, to detect conflicts, the system should examine the plans each character has devised to achieve its active goal. If a character's goal is in conflict with another character's value-dependent goal, the story manager consults the direction to verify if a failure to reestablish that value is compatible with the direction. If this is not the case, the goal is expunged from the candidate value-dependent goals.

Clearly, the strategy described so far only detects the interferences among a character's individual candidate goal and other characters' active goals, at some point of the story development. But this is not sufficient to detect the conflicts within the candidate goal set. The latter, in fact, may emerge or remain latent depending on the goals that will actually become active. Although the system we describe is intended as a reference model for designing value-sensitive storytelling systems, we believe that a complete cross-verification of the interferences among the candidate goals is not in the spirit of the approach we propose, that sees interactive storytelling as the shaping of an emergent plot according to a given direction. So, here we only state the instruments for dealing with control, but we consider the possibility that the behavior of the story characters leads to a system's failure to instantiate the direction later in the story generation. In order to address this issue, we think that the direction could be conceived of as a optimal solution, but suboptimal plots may be generated by relaxing the constraints in the direction or by allowing the system to backtrack (the latter solution may viable in an authoring scenario).

As a result of this survey, the story manager generates a list of *candidate characters' value-dependent goals*, each associated with a set of plans.

4. **Candidate options generation.** After the set of candidate value-dependent goals has been computed, the story manager queries the characters to obtain the plans by which each candidate goal can be achieved, getting a list of plans (*query on plans* in the Figure). At this point, the story manager needs to know if any of the candidate value-dependent goals or plans can put further values at stake – no matter if these values are acknowledged by same or by another character. Again, two situations are possible: a goal matches or logically implies the condition of a value, or the execution of any step of one of the alternative plans that are associated with the goal matches or logically implies the condition of a value. Notice that this situation is only relevant if the new value put at stake, for the involved character, has a higher-priority than its current value at stake (if any); otherwise, the value-related goal would be adopted but not become active and thus would not affect the enforcement of the direction. Conversely, if the new value put at stake has a higher priority than the current value at stake and it is not consistent with the direction given, the plans (or the plan) that are responsible for putting it at stake are removed from

plan set. Again, we do not consider here the emergent aspects of the goal-values interplay.

Plan actions, indexed by characters, constitute the *candidate options* for the creation of the subsequent story incident.

5. **Interactive option selection.** In an interactive storytelling scenario, the candidate options become the user options, with a separate set of options for each character. The available options are communicated to the user through the application-specific display environment (textual, 2D or 3D graphics, multimodal). The option selected by the user will be acted by the characters in the storyworld. Characters also adopt the value-dependent goals that have not been selected, if any, but they do not become active (although they are likely to become active later in the story). In an authoring scenario, for each character, the author would select a candidate option for the story prosecution.

Finally, characters who are not affected by new values at stake perform a planning step as defined in Section 3. In particular, if a character realizes that its current plan is flawed, it proceeds to repair it, committing to a new plan if possible. Or, if the plan cannot be repaired, and no other plan is available, it drops the current goal and selects a previously suspended goal, if any, as the active goal (for example, a character may address a value at stake of lower priority). Otherwise it remains with its current commitment and selects the next step for execution.

6. **Incident execution.** Finally, the story manager communicates the selected option to each character; the character executes the next step of the plan it is currently committed to, and the story world is updated. The plot is also update with the new incident, that contains the set of actions executed by the characters. At this point, each character updates its mental state, including the record of the values at stake. Here, emergent conflicts among goals and emergent interferences among goals and values become explicit. The character realizes that new values are at stake, and that some values previously at stake are not at stake anymore, because the character's efforts to establish them have been successful or because the activity conducted by some other character has brought about a state in which they are not at stake anymore. Or, the character may realize that a value-related goal has become unachievable, thus facing a failure to comply with that value.
7. **Story-direction consistency checking.** At this point, the story manager verifies whether the story constructed so far is consistent with the direction. While the incident-value consistency is automatically enforced by incorporating the value-sensitive deliberation model in the characters, the same does not hold for the requisite of story-direction consistency stated above. Due to the emergent aspects of the characters' interaction, in fact, it may be the case that, for a certain character, some value was put at stake too prematurely with respect to the ordering of values at stake prescribed by the direction. Or it may be the case that a character failed to achieve a value-dependent goal, while the direction requires it to succeed. In both cases, the system has failed to enforce the direction. Again, the direction may be



relaxed, for example by limiting a strict enforcement to one character (the main character).

8. **Story ending.** The story ends when the direction has been instantiated, i.e. all the prescribed values have been put at stake by leading the characters to form and actively pursue (successfully, if needed) the appropriate value-related goals.

According to the definition of open and closed interactive systems given by Meadows [20], the architecture we propose is *closed* for what concerns the definition of characters' values, so as to allow authors to enforce well-specified values in the story, and *open* for what concerns the plot generation, although the latter inherits the property of resource-boundedness that characterizes BDI agents [5].

From the system loop described above, it is clear that the abstract architecture has some important limitations. First, it is not guaranteed to find a compatible plot with the given direction, mainly due to the interactions among characters' goals and values that may emerge during the interactive plot construction.

The planning activity takes place in the characters, with the story manager making only the computation that is strictly necessary to control (but not to guarantee) the enforcement of the direction and to identify story failure and completion. This distributed design can be an advantage for the implementation of practical systems, but it is not suitable for modeling the characters' failure to restore their values in an explicit way, since a character's failure to comply with its values can happen only fortuitously.

The architecture and execution model presented above do not account for many sources of knowledge that take part into the design of characters and stories and into the expression of stories. For example, it does not address the characters' emotional aspects, an issue that has been recently addressed in virtual characters and virtual storytelling [2,24]. So, the basic architecture should be not only engineered to address its computational shortcomings (like the anticipatory mechanism required to characters) but also be integrated with specific modules to comply with storytelling issues. Finally, the perspective of emergent storytelling should not be intended as an alternative to integrating the proposed architecture with specialized techniques to manage and coordinate multi-agent systems.

## 6 Conclusions and Future Work

In this paper, we have proposed to use the notion of characters' values to embed values in interactive storytelling systems, and we have proposed a definition of value-based story direction and a reference architecture for value-based interactive storytelling. In our proposal, values are exploited to constrain the task of interactive generation of the plot to a smaller, value-compliant set of alternatives, and as a way to provide the procedural author [22] with a conceptual tool to define the "plot boundaries" [21] in terms of values. The whole architecture can be seen as a multi-agent system, in which the plot generation module exploits the individual characters' ability to perform value-sensitive deliberation to keep the story consistent with the values declared by the author.

The purpose of this architecture is to investigate the possibility of designing value-based interactive storytelling systems by relying on agent theories and technologies. In order to design and implement practical applications, the abstract architecture and execution model we propose need to be integrated with narrative knowledge required to manage all the story features the model does not deal with, both at the story modeling level (like story models or character definition), and at the story expression level (like story cadence, mood, etc). However, we think the specifying a value-based framework is a necessary step to verify its theoretical feasibility.

As future work, we envisage the implementation of a prototype to test the value-based paradigm and its appropriateness in the development of interactive storytelling systems and in story authoring, and the extension of the model to deal with dynamic values and their relations with models of emotions and personality.

## References

1. Anderson, E.: *Value in Ethics and Economics*. Harvard University Press, Cambridge (1993)
2. Aylett, R.S., Vala, M., Sequeira, P., Paiva, A.C.R.: FearNot! – an emergent narrative approach to virtual dramas for anti-bullying education. In: Cavazza, M., Donikian, S. (eds.) *ICVS-VirtStory 2007*. LNCS, vol. 4871, pp. 202–205. Springer, Heidelberg (2007)
3. Barber, H., Kudenko, D.: Generation of dilemma-based interactive narratives with a changeable story goal. In: *Proc. of 2nd INTETAIN*, Brussels (2008)
4. Bratman, M.: *Intention, plans, and practical reason*. Harvard University Press, Cambridge Mass (1987)
5. Bratman, M.E., Israel, D.J., Pollack, M.E.: Plans and resource-bounded practical reasoning. *Computational Intelligence* 4, 349–355 (1988)
6. Bruner, J.: The narrative construction of reality. *Critical Inquiry* 18(1), 1–21 (1991)
7. Busetta, P., Ronnquist, R., Hodgson, A., Lucas, A.: JACK Intelligent Agents-Components for Intelligent Agents in Java. *AgentLink News Letter* 2, 2–5 (1999)
8. Campbell, J.: *The Hero with a Thousand Faces*. Princeton University Press, Princeton (1949)
9. Carroll, N.: *Beyond Esthetics: Philosophical Essays*. Cambridge University Press, New York (2001)
10. Cohen, P.R., Levesque, H.J.: Intention is choice with commitment. *Artificial Intelligence* 42, 213–261 (1990)
11. Damiano, R., Lombardo, V.: Value-driven characters for storytelling and drama. In: Serra, R., Cucchiara, R. (eds.) *AI\*IA 2009*. LNCS, vol. 5883, pp. 436–445. Springer, Heidelberg (2009)
12. Dignum, F., Westra, J., van Doesburg, W., Harbers, M.: Games and Agents: Designing Intelligent Gameplay. *International Journal of Computer Games Technology* (2009)
13. Egri, L.: *The Art of Dramatic Writing*. Simon and Schuster, New York (1946)
14. Figueiredo, R., Brisson, A., Aylett, R.S., Paiva, A.C.R.: Emergent Stories Facilitated. In: Spierling, U., Szilas, N. (eds.) *ICIDS 2008*. LNCS, vol. 5334, pp. 218–229. Springer, Heidelberg (2008)
15. Freytag, G.: *Technique of the drama, an exposition of dramatic composition and art*. S.C. Griggs and Company, Chicago (1985)

16. Hartmann, K., Hartmann, S., Feustel, M.: Motif definition and classification to structure non-linear plots and to control the narrative flow in interactive dramas. In: Subsol, G. (ed.) ICVS-VirtStory 2005. LNCS, vol. 3805, pp. 158–167. Springer, Heidelberg (2005)
17. Hindriks, K., van Riemsdijk, B., Behrens, T., Korstanje, R., Kraaijenbrink, N., Pasma, W., de Rijk, L.: Unreal GOAL bots. In: Preproceedings of The AAMAS 2010 Workshop on Agents for Games and Simulations (2010)
18. Mateas, M., Stern, A.: Towards integrating plot and character for interactive drama. In: Working notes of the Social Intelligent Agents: The Human in the Loop. AAAI Fall Symposium Series (2000)
19. McKee, R.: *Story*. Harper Collins, New York (1997)
20. Meadows, M.: *Pause & effect: the art of interactive narrative*. Pearson Education, London (2002)
21. Medler, B., Magerko, B.: Scribe: A tool for authoring event driven interactive drama. In: Göbel, S., Malkewitz, R., Iurgel, I. (eds.) TIDSE 2006. LNCS, vol. 4326, pp. 139–150. Springer, Heidelberg (2006)
22. Murray, J.: *Hamlet on the Holodeck. The Future of Narrative in Cyberspace*. The MIT Press, Cambridge (1998)
23. Norling, E., Sonenberg, L.: Creating Interactive Characters with BDI Agents. In: Proceedings of the Australian Workshop on Interactive Entertainment IE 2004 (2004)
24. Peinado, F., Cavazza, M., Pizzi, D.: Revisiting Character-based Affective Storytelling under a Narrative BDI Framework. In: Proc. of ICIDIS 2008, Erfurt, Germany (2008)
25. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: a BDI Reasoning Engine. *Multiagent Systems, Artificial Societies and Simulated Organizations* 15, 149 (2005)
26. Polti, G.: *Les trente-six situations dramatiques*. Mercure de France, Paris (1895)
27. Rank, S., Petta, P.: Appraisal for a character-based story-world. In: Panayiotopoulos, T., Gratch, J., Aylett, R.S., Ballin, D., Olivier, P., Rist, T. (eds.) IVA 2005. LNCS (LNAI), vol. 3661, pp. 495–496. Springer, Heidelberg (2005)
28. Riedl, M., Young, M.: From linear story generation to branching story graphs. In: *IEEE Journal of Computer Graphics and Applications*, pp. 23–31 (2006)
29. Riedl, M., Young, R.: An intent-driven planner for multi-agent story generation. In: Proceedings of the 3rd International Conference on Autonomous Agents and Multi Agent Systems (2004)
30. Schank, R.C., Abelson, R.P.: *Scripts, Plans Goals and Understanding*. Lawrence Erlbaum, Hillsdale (1977)
31. Sengers, P.: Designing comprehensible agents. In: Proceedings of the Sixteenth International Joint Conference of Artificial Intelligence, Stockholm, Sweden (1999)
32. Styan, J.L.: *The Elements of Drama*. University Press, Cambridge (1963)
33. van Fraassen, B.: Values and the heart's command. *Journal of Philosophy* 70(1), 5–19 (1973)
34. van Riemsdijk, M., Dastani, M., Meyer, J.: Goals in conflict: semantic foundations of goals in agent programming. *Autonomous Agents and Multi-Agent Systems* 18(3), 471–500 (2009)
35. van Riemsdijk, M., Dastani, M., Winikoff, M.: Goals in Agent Systems: A Unifying Framework. In: Proceedings of AAMAS 2008 (2008)
36. Williams, S.D.: *The Moral Premise*. Michael Wiese Productions (2006)

# Implicitly and Intelligently Influencing the Interactive Experience

Michael J. O'Grady<sup>1</sup>, Mauro Dragone<sup>1</sup>, Richard Tynan<sup>1</sup>,  
Gregory M.P. O'Hare<sup>1</sup>, Jie Wan<sup>2</sup>, and Conor Muldoon<sup>1</sup>

<sup>1</sup> CLARITY: Centre for Sensor Web Technologies,  
School of Computer Science & Informatics, University College Dublin,  
Belfield, Dublin 4, Ireland

<sup>2</sup> Department of Computing & Networking, Institute of Technology Carlow,  
Carlow, Ireland

{michael.j.ogrady,mauro.dragone,richard.tynan,gregory.ohare}@ucd.ie,  
conor.muldoon@ucd.ie, jesse.wan@yahoo.co.uk

<http://www.clarity-centre.org/>

**Abstract.** Enabling intuitive interaction in system design remains an art more than a science. This difficulty is exacerbated when the diversity of device and end user group is considered. In this paper, it is argued that conventional interaction modalities are unsuitable in many circumstances and that alternative modalities need be considered. Specifically the case of implicit interaction is considered, and the paper discusses how its use may lead to more satisfactory experiences. Specifically, harnessing implicit interaction in conjunction with the traditional explicit interaction modality, can enable a more intuitive and natural interactive experience. However, the exercise of capturing and interpreting implicit interaction is problematic and is one that lends itself to the adoption of AI techniques. In this position paper, the potential of lightweight intelligent agents is proposed as a model for harmonising the explicit and implicit components of an arbitrary interaction.

**Keywords:** Implicit interaction, Social Signal Processing, Intelligent agents.

## 1 Introduction

A laudable objective of many computing applications and services is the provision of seamless and intuitive interaction. Ubiquitous computing is a case in point. The vision articulated by the proponents of this paradigm envisages a world saturated with electronic infrastructures, with the objective of making computing services available everywhere such that it may be accessed in an as-needed fashion. Indeed, the late Mark Weiser, the father of ubiquitous computing, likened ubiquitous computing to a common everyday signpost, both in its pervasiveness in the environment, as well as the ease, intuitiveness and lack of effort associated with its use. However, how such intuitiveness was to be achieved

in practice was not stated. To address this, the Ambient Intelligence (AmI) [1] concept was proposed. This explicitly acknowledged the interaction problem of practical pervasive or ubiquitous computing environments and proposed the adoption of Intelligent User Interfaces (IUIs) [7] as a means of addressing this. Again, the pragmatic issues of how such interfaces may be realised in practice remains unanswered.

It is well known that in commercial software, one prerequisite to success is ensuring that the user experience is a satisfactory one. For conventional workstation environments, many useful heuristics have been constructed pertaining to the effective design of interfaces and management of interactions. Given that such environments have been studied since the 1960s, it would indeed be disappointing if significant progress had not been made in this time interval. However, given the many form factors that computing frequently utilises, as well as the multitude of domains in which it is applied, it is questionable as to what degree conventional good practice HCI principles apply for non-workstation environments. In the case of mobile computing, one reason why such principles may not be applicable is that the nature of the context in which a mobile interaction occurs may differ radically from that of conventional interactions. For example, it has been demonstrated that there may be up to eight fold differences between the attention span that users give to tasks under both laboratory conditions and mobile contexts [13].

In this paper, it is argued that successfully harnessing the implicit interaction modality may offer significant potential for augmenting the interaction experience. By incorporating implicit interaction, the potential for sharing control of an application need be not seen as the exclusive preserve of either the human operator or the application in question. Rather it can be regarded as a collaborative effort. In circumstances where intelligent agents have been adopted as the software construct for capturing and interpreting implicit interaction, this collaboration may be one shared between the human and the agent.

## 2 Interaction Modalities

A number of modals of interaction have been proposed, those of Beale [3] and Norman [10] being well documented examples. For the purposes of this discussion, interaction is considered, albeit briefly, from both a unimodal and multimodal perspective.

### 2.1 Unimodal Interaction

Unimodal interaction refers to interactions that occur when only one modality is used, for example speech. Conventional interactions with computational devices of various genre are almost inherently unimodal. Though the prevalent approach, it is instructive to note that this is almost diametrically opposite to human communication, which is inherently multimodal.

## 2.2 Multimodal Interaction

Multimodal interaction involves a number of modalities being used in parallel, for example voice and gestures. This may be regarded as a human centric view of interaction. An alternative interpretation is based on the role of the computational artefact, making this central to the definition. Sebu [16] considers that the computational equivalent of the human senses is what makes a system multimodal. For example, the use of voice recognition using a microphone, and gesture recognition using a camera would constitute multimodality. Alternatively, using a suite of cameras, for example to identify gestures and facial expressions, would be regarded as unimodal. For this discussion, the human centric view is adopted.

Multimodal interaction is perceived as being more natural and intuitive [14]. However, this comes at a price: complexity and timeliness. Taken individually, gesture and voice recognition demand sophisticated complex solutions. In parallel, the difficulty is aggravated. Both modalities must be interpreted separately. Then the result must be considered in combinations such that a semantic meaning can be attributed to the interaction. All of this requires significant computational resources if the interaction is to be interpreted correctly, and, importantly, responded to, in a timely manner. Frequently, Artificial Intelligence (AI) techniques, for example, machine learning, are used to facilitate the process of interaction identification.

## 3 Explicit and Implicit Control

An alternative interpretation of interaction is to consider intent as the key parameter. Indeed, if intent was known with certainty in all circumstances, the issue of intuitive interaction would be less problematic, although its realisation in practice so as to meet user expectations might still raise particular difficulties. When considered in the light of user intent, we can consider the interaction, or by extension the control of the system, as being expressed in either an explicit or implicit fashion. However, an understanding of human communications is necessary before these can be considered.

### 3.1 A Reflection of Human Communications

For the most part, the vocal channel is the predominant one used everyday. However, this is usually accompanied by a variety of non-verbal cues that people interpret subconsciously. Thus, it might be concluded, irrespective of the particular utterance, that the speaker is sad, happy, busy or just indifferent. Though contentious, it has been estimated that nonverbal cues have up to four times the effect of verbal cues [2]. Indeed, a number of classifications of such cues exist, for example Ekman & Friesen [4] have identified 5 categories–

1. Emblems: actions that carry meaning of and in themselves, for example a thumbs up.
2. Illustrators: actions that help listeners better interpret what is being said, for example, finger pointing;

3. Regulators: actions that help guide communication, for example head nods;
4. Adaptors: actions that are rarely intended to communicate but that give a good indication of physiological and psychological state;
5. Affect: actions that express emotion without the use of touch, for example, sadness, joy and so on.

These give a flavour of the kind of cues that, if captured correctly, would lead to significant enhancements to the interactive experience.

### 3.2 Explicit Interaction

Explicit interaction is the normal method of interacting with and controlling software, and may be regarded as unimodal in nature. It is event or stimulus driven. For example, a button is pressed, and the system responds, ideally in some meaningful way. Its simplicity makes it easily understood by all, at least in principle. From a software engineering perspective, most programming languages make the managing of event handling relatively easy for software developers, for example through the Model-View-Controller pattern [18].

### 3.3 Implicit Interaction

Implicit interaction is a more subtle construct, and computationally challenging to implement. It is modelled on how humans communicate, and is essentially multimodal in character. The challenge is to capture the cues that invariably contribute to the human communication process, leading to a more complete understanding of the interaction. Even a simple explicit interaction, such as clicking a mouse, takes place within a context. For example, a mouse might be clicked in anger. Assuming this anger can be detected, and developments in affective computing make this increasingly likely, then the software can adapt. How? this will depend on the domain in question. Similar to multimodal interaction, the challenge with implicit interaction is to capture and interpret it in a meaningful fashion.

A question to be considered is whether implicit interaction occurs on its own, or should be regarded merely as augmented explicit interaction. In some cases, the lack of an explicit interaction event, if expected or available, may signify an implicit interaction. In many e-commerce WWW sites, users usually ignore the multitude of advertisements on offer. This they do subconsciously, for the most part, unless of course their attention is obtained and they explicitly decide to click on some advertisement.

## 4 Social Intelligence

Defining intelligent is one endeavour that has historically and continues to challenge research scientists in a number of disciplines. In many cases, intelligence is associated with IQ. Increasingly, this is being viewed as an extremely narrow

definition; indeed, in everyday life, it may be frequently observed that many individuals who are commonly perceived as being intelligent, may frequently indulge in behaviour that is anything but. Thus an increasing number of cognitive scientists believe that additional concepts should be incorporated into the intelligence construct. In short, additional abilities that are essential to success in life should be incorporated into any definition of intelligence. In essence these abilities define social intelligence [17] [6] [20].

Abilities that indicate social intelligence include empathy, sympathy, politeness and so on. In other words, social intelligence constitutes those abilities that aid people in the performance of their everyday duties (both in their personal and professional lives) and include negotiation, cooperation and collaboration, for example. Such skills are essential to success in life. While most of the research and discourse on social intelligence is naturally focused on human(s)-to-human(s), the question of human(s)-to-computer(s) is receiving increased attention. Recalling the previous discussion on Ambient Intelligence (AmI) and Intelligent User Interfaces, a natural question to ask is whether social intelligence can contribute to resolving the key problem of intuitive interaction. One emerging research domain that seeks to address this issue within a broader context is that of Social Signal Processing.

#### 4.1 Social Signal Processing

Social Signal Processing (SSP) [15] [19] has the singular objective of bringing social intelligence to the computing domain. Machine analysis is seen as the key enabler of SSP. Four key stages are envisaged in SSP:

1. Data capture - An array of sensors are necessary to capture selects aspects of an interaction. Obviously, cameras and microphones are the predominant sensors that would be harnessed in this context. However, it is easy to envisage that a wide variety of sensors could be fruitfully used, for example, biometric or physiological sensors. A critical point to remember is that communications and interaction always take place in a context. To determine a full picture of the prevailing context is usually problematic. Yet the data necessary even to practically achieve context recognition may be indispensable.
2. People identification - A key initial step is to associate captured sensed data with individuals. In cases involving a number of people, the difficulty and complexity is aggravated.
3. Social Signal identification - In this process, the individual social signals are extracted from the various data sets. A suite of techniques may be harnessed here including facial expression analysis and gesture recognition.
4. Social behaviour understanding - This stage of the process involves attaching semantic meaning to the signals and cues extracted from the data streams. Context is of vital importance at this stage.

Significant challenges must overcome before SSP becomes a reality in practical applications. However, it is important to recall that SSP is the summation of



many years research in a multitude of disciplines. In addition, it incorporates many of the issues described previously concerning implicit interaction. Indeed, computer games may offer a fertile domain for exploring SSP. Understanding SSP in terms of individual players offers significant potential for realizing games that are inherently dynamic, adaptive and personalised for individual players. It may also inform the situation where the control of the game can be exchanged between the player and the game as circumstances dictate.

## 5 The Locus of Control

Ultimately, control of any interactive application or service must rest with the user. However, in many cases, the application will perform in an autonomous or semi-autonomous fashion, resulting in control being shared. Thus the user is nominally in control, but is happy to remain outside the control loop as long as the application is performing to their satisfaction. This may happen in systems of all hues. Indeed, the objective of autonomic computing is to actually remove the human operator from the command chain as much as possible. Human time is perceived as a scarce and expensive commodity, as such must be used wisely.

With interactive entertainment systems in contrast, the motivation is different with the user being the key actor, as it were. In this case, the user (usually) wants to minimise their cognitive load, and with minimum interaction, leave the application follow its own cycle. Any significant intervention must be motivated. However, most entertainment systems react in a stimulus/response manner. No effort is made to monitor the user's reaction or perception to what is happening.

Considering the previous discussion on SSP and implicit interaction, it can be seen that there is significant opportunity available for enhancing the user experience, provided effort is expended to capture this interaction. By transparently observing the user as they interact, the potential for a more fulfilling experience emerges. In short, an arbitrary system could dynamically adapt its behaviour in response to observed cues. Obviously, the strategies and policies used for adaptation will be domain, and maybe user, dependent. However, how such adaptivity should be designed for remains an open question. It is envisaged that successfully adapting the behaviour of the application to the user is likely to minimise their need to explicitly take control of the application, and increase the likelihood that they will cede control to the application even if they should need to make an intervention occasionally.

A final issue that needs to be addressed is the characteristics of a software architecture for realising such adaptivity.

## 6 Software Architecture

Realising an adaptive application demands that the software inherently possesses certain traits. Autonomy is essential, as a capability to react to external events. For a dynamically adaptive solution, this is not sufficient. A capacity to act proactively and to plan ahead is also essentially. Ideally, some capacity to learn

would also be supported. Such characteristics immediately suggest the harnessing of the agent paradigm as this encapsulates these characteristics. However, not all agent architectures are sufficiently endowed with such capabilities. However, those that subscribe to the Belief-Desire-Intention (BDI) model [5] for example, could be reasonably expected to be capable of forming the basis of adaptive applications. In selecting an agent framework to support implicit interaction and realise an adaptive solution, it is essential that the attributes necessary for its realisation be kept in mind.

In our own research, the viability of lightweight embedded agents [12] [9] have been demonstrated in mobile computing contexts for managing and interpreting interactions, both explicit and implicit. In the mobile tourism domain, information has been adapted to tourists' contexts [11] while in the e-commerce domain, agents have negotiated deals for items on users' shopping lists [8].

## 7 Conclusions

In this paper, the potential of the implicit interaction modality as a means of augmenting the interactive experience was advocated. However capturing and interpreting such interaction is computationally complex. The harnessing of agent frameworks of sufficient power and complexity is suggested as a basis for realising applications and services that can dynamically adapt to implicit user input.

**Acknowledgments.** This work is supported by Science Foundation Ireland under grant 07/CE/I1147.

## References

1. Aarts, E., Grotenhuis, F.: Ambient intelligence 2.0: Towards synergetic prosperity. In: Tscheligi, M., de Ruyter, B., Markopoulos, P., Wichert, R., Mirlacher, T., Meschterjakov, A., Reitberger, W. (eds.) *AmI 2009*. LNCS, vol. 5859, pp. 1–13. Springer, Heidelberg (2009)
2. Argyle, M., Salter, V., Nicolson, H., Burgess, N.W.P.: The communication of inferior and superior attitudes by verbal and nonverbal signals. *British Journal of Social & Clinical Psychology* 9, 221–231 (1970)
3. Dix, A., Finlay, J., Abowd, G.D., Beale, R.: *Human Computer Interaction*, 3rd edn. Pearson, London (2003)
4. Ekman, P., Friesen, W.: The repertoire of nonverbal behavior: Categories, origins, usage, and coding. *Semiotica* 1, 49–97 (1969)
5. Georgeff, M., Pell, B., Pollack, M.E., Tambe, M., Wooldridge, M.J.: The belief-desire-intention model of agency. In: Papadimitriou, C., Singh, M.P., Müller, J.P. (eds.) *ATAL 1998*. LNCS (LNAI), vol. 1555, pp. 1–10. Springer, Heidelberg (1999)
6. Goleman, D.: *Social Intelligence: The New Science of Human Relationships*. Bantam, New York (2006)
7. Höök, K.: Designing and evaluating intelligent user interfaces. In: *IUI 1999: Proceedings of the 4th International Conference on Intelligent User Interfaces*, pp. 5–6. ACM, New York (1999)
8. Keegan, S., O'Hare, G.M., O'Grady, M.J.: Easishop: Ambient intelligence assists everyday shopping. *Information Sciences* 178(3), 588–611 (2008)

9. Muldoon, C., OHare, G., Collier, R.W., OGrady, M.: Towards pervasive intelligence: Reflections on the evolution of the agent factory framework. In: El Fallah Seghrouchni, A., Dix, J., Dastani, M., Bordini, R.H. (eds.) *Multi-Agent Programming*, pp. 187–212. Springer, US (2009)
10. Norman, D.A.: *The Design of Everyday Things*. Basic Books, New York, reprint paperback edn. (2002)
11. O'Grady, M.J., O'Hare, G.M.P., Donaghey, C.: Delivering adaptivity through context-awareness. *Journal of Network and Computer Applications* 30(3), 1007–1033 (2007)
12. O'Hare, G.M.P., O'Grady, M.J., Muldoon, C., Bradley, J.F.: Embedded Agents: A Paradigm for Mobile Services. *International Journal of Web and Grid Services (IJWGS)* 2(4), 379–405 (2006)
13. Oulasvirta, A., Tamminen, S., Roto, V., Kuorelahti, J.: Interaction in 4-second bursts: the fragmented nature of attentional resources in mobile hci. In: *CHI 2005: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 919–928. ACM, New York (2005)
14. Oviatt, S.: Multimodal interactive maps: designing for human performance. *Hum.-Comput. Interact.* 12(1), 93–129 (1997)
15. Pentland, A.: Social signal processing. *Signal Processing Magazine, IEEE* 24(4), 108–111 (2007)
16. Sebe, N.: Multimodal interfaces: Challenges and perspectives. *Journal of Ambient Intelligence and Smart Environments* 1(1), 23–30 (2009)
17. Thorndike, R.K.: Intelligence and its uses. *Harper's Magazine* 140, 227–335 (1920)
18. Veit, M., Herrmann, S.: Model-view-controller and object teams: a perfect match of paradigms. In: *AOSD 2003: Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*, pp. 140–149. ACM, New York (2003)
19. Vinciarelli, A., Pantic, M., Bourlard, H.: Social signal processing: Survey of an emerging domain. *Image and Vision Computing* 27(12), 1743–1759 (2009)
20. Wang, F.Y., Carley, K.M., Zeng, D., Mao, W.: Social computing: From social informatics to social intelligence. *IEEE Intelligent Systems* 22, 79–83 (2007)

# Creating Customized Game Experiences by Leveraging Human Creative Effort: A Planning Approach

Boyang Li and Mark O. Riedl

College of Computing, Georgia Institute of Technology,  
Georgia, USA  
{boyangli,riedl}@gatech.edu

**Abstract.** The task of entertaining people has, until very recently, been the exclusive domain of humans. However, recent advances in Artificial Intelligence (AI) suggest that intelligent systems may be used to create dynamic and engaging real-time entertainment experiences. In this paper we consider a novel technique called *Experience Adaptation*. Experience Adaptation is an offline process that leverages human creative ability by taking human-authored specifications of desired user experiences and autonomously “re-writing” them based on unique requirements of individual users. In this chapter, we illustrate Experience Adaptation in the context of computer-based role-playing games in which player experience is highly dependent on an unfolding plotline. Our approach uses a plan refinement technique based on partial-order planning to (a) optimize the global structure of the plotline according to input from a player model, (b) maintain plotline coherence, and (c) facilitate authorial intent by preserving as much of the original plotline as possible.

**Keywords:** Experience Adaptation, Narrative Intelligence.

## 1 Introduction

Artificial intelligence has long been used to automate certain tasks in order to perform those tasks faster, more accurately, more efficiently, more safely, or more often. However, the task of entertaining people has, until very recently, been the exclusive domain of humans. When it comes to commercial production of entertainment artifacts like TV shows, movies, novels, theatre, computer games, etc., the task of entertaining people has been the exclusive domain of “creative professionals” such as writers, actors, movie directors, theatre and improv performers, dungeon masters, and so on. The reason the task of entertaining people has been the exclusive domain of humans is that the creativity and intuition that human entertainers possess have not been reliably replicated in computational systems.

Currently, there are fewer professional and expert human “producers” of entertainment than there are human “consumers” of entertainment. This model works fine for mass-consumption entertainment such as film, TV, books, and, to a lesser extent, theatre performances. The *creative authoring bottleneck* refers to the situation where the cost of employing enough professional human producers to satisfy the demands of human consumers is prohibitively high, resulting in a situation where

there is more demand for quality content than production of quality content. (We use “authoring” to refer to the deliberate creation of any entertainment-related artifact, including an improvised performance created in real-time [9]). Recent work in the area of computational creativity, story generation, interactive storytelling, and autonomous believable agents lays the groundwork for a future where entertainment is fully automated. We are now at a unique point where modern computer technology, simulation, and computer games have opened up the possibility of that more can be done in the area of *on-demand* and *uniquely customized* entertainment.

- **On-demand entertainment** refers to the possibility that one can request, at any time, an entertainment experience that is significantly different from any previously consumed. For example, game players can exhaust game-play content faster than expansion packs and new releases can be produced. For an early case study in which consumers outpace producers of content in online virtual game worlds, see [11]. Ideally, there is a one-to-one relationship between producers and consumers so that content can never be consumed faster than it is produced.
- **Uniquely customized entertainment** means that entertainment artifacts should be customized or configured to suit every player’s unique motivation, tastes, desires and history. Usually this information is not available at the time the game is designed and implemented. The customization decisions can only be made in a just-in-time fashion because we need to know (a) who the user is, (b) what the user’s motivation, tastes, and desires are, and (c) what the user is doing at any given moment.

As we approach a world in which on-demand and uniquely customized entertainment is the expectation, the conventional consumer-producer model breaks down. To overcome the creative authoring bottleneck, we must consider automation. In this chapter we consider a technique called *Experience Adaptation* [8, 17]. Experience Adaptation is an offline process that leverages human creative ability by taking human-authored specifications of desired future experiences and autonomously “re-writing” them based on unique requirements of individual users.

To motivate the need for on-demand and uniquely customized entertainment, we explore these concepts in the context of generating plotlines for computer-based role-playing games. Computer based role-playing are believed to be highly dependent on individual differences such as play styles [1, 29, 26] and involve numerous tasks that may or may not be of interest to players. Rollings and Adams [21] argue that the core of gameplay in any game is “one or more causally linked series of challenges in a simulated environment.” These challenges often appear in units of role-playing game storytelling called quests. To accomplish the quests, players have to perform required gaming activities such as combat or puzzle-solving in a virtual world. Game designers usually use a main plotline, comprised of a set of quests, often ordered, that are sufficient and necessary to complete the game. The main plotline provides the player with a sense of meaningful progression through the game. Although the main plotline is mandatory, optional side-quests are often available to augment the gameplay experience, and to afford players a limited degree of customization through choice. Instead of supplementary side quests, we investigate intelligent systems that adapt and customize the primary plotline to satisfy player preferences, needs, and desires while maintaining narrative coherence and preserving the original plotline author’s intent.

We argue that customization of entertainment experience involves presenting *the right experience to the right person at the right time*. The significance of this claim is twofold. First, players usually possess diverse motivation, tastes, desires and history. A one-size-fits-all script may not cater to all types of players. Moreover, to achieve optimal game experience, challenges must adapt to the player's skill level. Secondly, preferences of players can change over time. Having experienced one story, the player may demand a new one. Therefore, the ability to generate customized plotlines may enhance replayability and improve player experience. By addressing the two implications, we are working toward the potential of games that continuously grow and change with the player over a long period of time by generating novel, customized plotlines.

The remainder of the chapter is organized as follows. In Section 2, we formulate the problem of game experience adaptation and ground the notion of experience on discrete computational representations of narrative. In Section 3, we provide a mathematical notion of narrative coherence based on our representation. Section 4 deals with the practical side of experience adaptation with a detailed planning algorithm, an example, and discussions of authoring and evaluation. Section 5 provides discussion of related work.

## 2 Experience Adaptation

We believe a computational system that scales up a human creator's ability to deliver customized experiences to a large number of consumers will provide a solution to the content creation bottleneck. Automated adaptation of experience is necessary when we can only learn about our intended customer at playtime. Unfortunately, the construction of autonomous systems capable of assuming responsibility for human users' entertainment experiences is largely an open research question. Until we have computational systems capable of creativity rivaling that of human experts, there is value in exploring *hybrid* approaches in which humans and computational systems share the responsibility of managing human users' entertainment experiences. Thus, such a computational system becomes a practical compromise: it should be able to facilitate human authors and scale up their authoring effort, so that a large number of customized variations of the original content can be produced easily without sacrificing the quality. Chen et al. [2] coined the term *authorial leverage* to indicate the ratio of quality of experience delivered by a computational system to authorial input. Hybrid Experience Adaptation systems leverage human knowledge for the purpose of creating novel experiences.

### 2.1 Leveraging Human Creative Effort

In the context of computer games, Experience Adaptation takes a few human-authored descriptions of experiences to be had in a virtual world and provides numerous experiences customized to individuals. The Experience Adaptation pipeline is shown in Fig. 1. A human author develops a plotline as a means of describing what a user should experience in the virtual world. The storyline determines events that will happen in the virtual world, including specifications for the behaviors of non-player characters. The plotline, provided in a computational format that facilitates

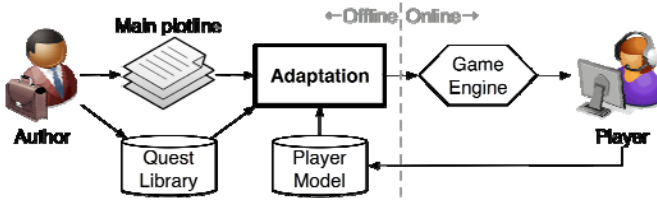


Fig. 1. The plotline adaptation architecture

automated analysis and reasoning, is combined with a player model and a world model. The world model describes what characters – human or virtual characters – can do in the world, and how the world is changed when actions are performed. The player model provides information about the user in terms of preferences over experiences. The player model also contains historical information describing the types of experiences the user has previously had. The player model is capable of generating a set of experiential requirements – the features of the experience the user should receive.

The plotline, player model requirements, and world model are inputs into the Experience Adaptor. The plotline is analyzed to determine whether it meets the experiential requirements from the player model. If it does not, the Experience Adaptor engages in an iterative process of making changes to the plotline until it meets the requirements of the user model. The result is a new creative artifact describing a customized narrative experience, which is sent to a game engine for interactive real-time execution. Note the cycle in Fig. 1 created by the Experience Adaptation process, resulting in improved replayability of authored experiences; as the player model evolves over time, the same human-authored storyline can be recycled into novel experiences.

The core component in the Experience Adaptation process is the Experience Adaptor. The Experience Adaptor has two functions, to interpret the requirements provided by the user model, and to “rewrite” the story provided by a human author. The Experience Adaptation Problem is as follows: given a domain model, a set of experiential requirements, and a storyline that does not meet the requirements, find a coherent storyline that meets the experiential requirements and preserves the maximal amount of original content. A coherent storyline is one in which all events have causal relevance to the outcomes [28]. The preservation of original content ensures that as much of the creative intuition of the human author remains intact as possible.

The plotline can be adapted in three different ways:

- **Deletion:** Events in the storyline can be removed because they are unnecessary or unwanted.
- **Addition:** Events can be added to the storyline to achieve experiential requirements, and to ensure narrative coherence.
- **Replacement:** a combination of deletion and addition, old events are swapped for new events that better achieve experiential requirements.

The application of these operations enables a refinement-search algorithm to incrementally tear down and build up a complete, human-authored narrative structure

until it meets the experiential requirements. Experience Adaptation can be online or offline; we have chosen to implement an architecture with an offline Experience Adaptor so as to optimize the overall global structure of the experience.

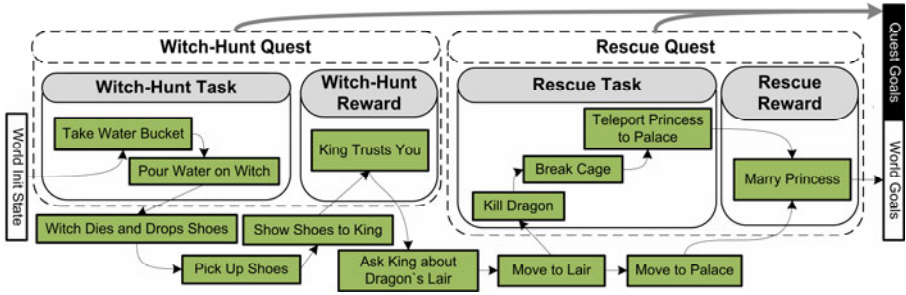
For the Execution Adaptor to function, it needs a set of experiential requirements that it can use to evaluate the current plotline and evaluate potential new plotlines. The player model is responsible for generating this set of experiential requirements. We model the player's preference as a function of previously selected quests. Each quest, in turn, is represented as a feature vector in a semantic space. We utilize a technique similar to that by Sharma et al. [25] to determine preferences over quests via ratings after gameplay concludes; similarity metrics allow us to extend preferences to quests not previously experienced by the user. In addition, a novelty model based on work by Saunders and Gero [23] favors quests that are appropriately novel to the player based on his or her history so that he or she would be neither bored nor unpleasantly surprised. Computing a weighted sum of utility by preference and utility by novelty, the result is the selection of the  $k$  quests with the greatest utility that should be included in the game plotline. Due to space constraints, a detailed description is beyond the scope of this paper.

## 2.2 Computational Representation of Plot

Experience Adaptation can only work if experience can be formally represented in a form that can be reasoned about and manipulated. As noted above, experiences are captured as narratives. Following others [30, 18, 15], we employ plan-like representations of narrative because they capture causality and temporality of action and provide a formal framework built on first principles, such as soundness and coherence, for selecting and ordering events. The plan representation provides a formal framework to explicitly represent causal relationships between events and reason about them on first principles (for example, we can ask if a narrative is sound). Further, plans closely resemble cognitive models of narrative. Graesser et al. [4] and Trabasso and van den Broek [28] in particular highlight the importance of causalities in stories. However, unlike a plan meant for execution, we use plans as descriptions of events expected to unfold in a virtual world; each action represents a formal declaration of an event that can be performed by the player or non-player characters, or occur as a consequence of physics laws in the virtual world.

Our specific representation builds on partial-order plans [14]. A partial-order plan consists of events and temporal and causal relations. Events encode preconditions, which must be true for the event to occur, and effects, which become true once the event completes. *Causal links*, denoted as  $e_1 \rightarrow^c e_2$ , indicate that the effects of event  $e_1$  establish a condition  $c$  in the world necessary for event  $e_2$ . Causal links act as protected intervals during which the truth of condition  $c$  in the world must be maintained. *Temporal links* indicate ordering constraints between events. Additionally, to capture semantic meaning of narrative subsequences, we allow for event abstraction hierarchies. Abstract actions are decomposed into sequences of equivalent, but less abstract events. The set of decomposition rules act as a grammar specifying legal configurations of narrative fragments. Decomposition rules must be authored *a priori* and are one way to leverage human authorial intuition; partial-order planning may discover causal and temporal relations based on the rules.





**Fig. 2.** An original game plotline before adaptation. The plotline contains two quest structures, represented as hierarchical decompositions.

In our system, quests are represented as top-level abstract events. A quest has a single effect, `quest-complete(quest-X)`, and may or may not have any preconditions. While not strictly necessary, we find the following authorial idiom to work well: decomposition rules break quests into an abstract task event and an abstract reward event, which are further decomposed into primitive events. Fig. 2 shows a complete plotline consisting of two quests. Primitive actions are shown as solid rectangles and abstract actions are shown as rounded rectangles. The hierarchical relationship between events is reflected in the containment relationships of rectangles. For example, one legal way in which a witch-hunt quest can occur is to kill the witch with water and earn the trust from the king. Arrows represent causal links. Note that not all causal links are shown for clarity’s sake. Temporal links are omitted.

The quest library (see Fig. 1) is a model of the dynamics of the virtual world. It is made up of primitive and abstract event templates plus decomposition rules. Event templates are parameterized events represented in a STRIPS-like (cf., [3]) format, allowing for specific characters, props, and location to be substituted in when an event is instantiated into a plan. The main plotline of the game, an example of which is illustrated in Fig. 2, is comprised of instantiated events, causal links, temporal links, and event relationships.

### 3 Narrative Coherence

We believe partial-order plans are effective representations of stories. Thus, a reasonable approach to solving the Experience Adaptation Problem is to use a form of refinement search that can manipulate partial-order plans. However, conventional planning is geared towards maximum efficiency, whereas the shortest or most efficient sequence of actions is rarely the best or most coherent story. Therefore, special care must be taken to maintain the coherence of the story generated.

Trabasso and van den Broek [28] proposed the idea of *narrative coherence* as a property of the causal structure of the story. A narrative is coherent when each event contributes significantly to the causal achievement of the main outcome. On each hierarchical level, a plan can be seen as a directed acyclic graph (DAG) with actions represented as vertices and causal links as edges. Whereas soundness is achieved if all

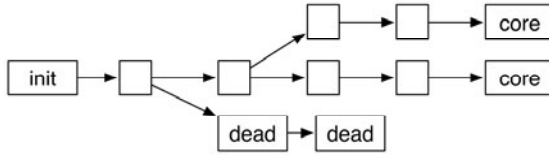


Fig. 3. Schematic of dead-end events

preconditions are on causal chains back to the initial state without creating logical inconsistencies, narrative coherence is achieved when each event has at least one effect on a causal chain to the outcome state. In this section, we elaborate on two types of story flaws that break narrative coherence: *dead ends* and *superfluous efforts* [7]. These flaws can happen even in a sound plan. The definitions of the two flaws rely purely on the abstract causal structure and performers of actions. In other words, the flaws are defined independently of the story domain, although they are dependent on how the preconditions and effects of actions are defined.

### 3.1 Core Set

First, we suggest that some events in a story are of special interest to the audience and more important than others. The significance of events can be perceived by human designers and audience. Other events set context for, revolve about, and eventually lead to these events, which form the core set of the story. The core set depends on the application. For example, when the player is interested in becoming filthy rich, the event where treasures are obtained is crucial, and other events should be subordinate. In this paper, we define the core set to include only the goal state of the plan. However, depending on the circumstances, one may want to choose other events for the core set. For example, complex authorial intent may be represented in the plan as intermediate goals, which can be negated after being achieved [16].

### 3.2 Dead Ends

An event is a dead end if it does not contribute in a meaningful way to the unfolding of events in the core set. It is believed that the presence of dead-end events directly harms the perception of narrative coherence. Following the previous example, suppose the primary interest of the player is to find treasure, then the event of obtaining a sword which is not useful for this purpose is not very relevant. Therefore, we consider the event to be a dead end. See Fig. 3 for an illustration of the causal structure of dead end, where a box represents an action and an arrow represents a causal link. The initial state, core events, and dead ends are labeled.

Formally, in a story DAG  $G = (V, E)$  where a vertex  $v \in V$  represents events in the plan and  $(u, v) \in E$  if and only if any effect of event  $u$  satisfies at least one precondition of event  $v$ . We use  $\text{path}(u, v)$  to denote the fact that there is a path from vertex  $u$  to vertex  $v$  in  $G$ . Given a core set  $S_C \subseteq V$ , the set of dead end actions  $S_D$  is defined by:

$$\forall u \in V, v \in S_C, \neg \text{path}(u, v) \Leftrightarrow u \in S_D. \quad (1)$$

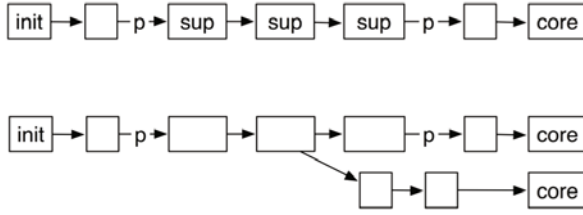


Fig. 4. Schematic of superfluous efforts (top) and non-superfluous efforts (bottom)

In general, it is recommended the core set be designed such that there are no dead ends in the original hand-authored storyline.

### 3.3 Superfluous Effort

Another breach of narrative coherence could happen when an event is part of a causal chain that contributes to the core set but at closer inspection appears to reestablish a world condition that is unnecessarily negated. For example, the player gives a sword to a stranger, and then has to steal it back to slay a dragon with it. The action of giving the sword is superfluous if, before the condition of the player having the sword, no other effects contribute to the core set. Fig. 4 (top) shows superfluous effort because the events serve no purpose other than re-establishing condition  $p$ . Fig. 4 (bottom) shows non-superfluous effort because the events that re-establish condition  $p$  serve an additional purpose. It is required that actions in the superfluous efforts are all performed by the same character.

Formally, a subset of vertices  $S \subseteq V$  is a superfluous effort if:

1.  $S$  is (weakly) connected.
2. The set of conditions annotating outgoing edges is a subset of the set of conditions annotating incoming edges.
3.  $\neg \exists a \in V, (\exists b, c \in S, \text{path}(b, a) \wedge \text{path}(a, c))$

Whereas, dead ends prevent interference with intentions of the author, superfluous efforts can be considered a heuristic guard against interference with intentions of story characters. The list of coherence flaws is by no means exhaustive, but the two examples illustrate two very important and complimentary aspects of the narrative coherence. We believe that the preservation of narrative coherence is important for any type of story adaptation.

## 4 The Experience Adaptor

The Experience Adaptor is the central component of the Experience Adaptation process (see Fig. 1). It leverages existing plotlines and promotes replayability by creating a cycle of play and adaptation. The Experience Adaptor module receives as input the following components:

- A complete plotline – a partially ordered, hierarchical plan – composed of events within and outside of quests.
- A set of plot requirements: `quest-complete(quest-X)` propositions specifying what quests should be included, and corresponding world-level outcome propositions.

The adaptation process involves two stages. In the first stage, a problem instantiation is created by rewriting the initial world state and desired outcome situation to match the plot requirements. When rewriting the outcome situation, any quests that no longer causally link to the outcome situation become dead ends and the plotline is no longer coherent. When rewriting the initial state, the preconditions of some events may no longer be supported by the initial state and the plotline may no longer be sound.

The second stage is plan refinement search process that progressively makes adjustments to the plotline until (a) all plot requirements are met, (b) the plotline is sound, and (c) the plotline is coherent.

#### 4.1 Experience Adaptation Planning Algorithm

Plan refinement techniques search a space where each node in the space is an instance of a plan (partial or complete) until a plan is found that has no flaws, or reasons why a plan cannot be considered a solution. Partial-order planning [14] is a form of plan refinement search that starts with the empty plan. For each plan visited, a flaw is detected and all repair strategies are invoked, each strategy resulting in zero or more new plans in which that flaw has been repaired. These new plans are successors to the current plan and are added to the fringe of the search space. A heuristic is used to determine which plan on the fringe visit next. Note that repairing a flaw may introduce new flaws.

Our adaptation algorithm is shown in Fig 5. The main loop is the standard plan refinement search loop. In addition to the pre-processing stage, we implement the following flaw types:

- **Open condition:** an event has a precondition not satisfied by any causal links from a temporally earlier event or the initial state.
- **Causal threat:** An event has an effect that undoes a condition necessary for another event to occur and there are no ordering constraints forbidding the interaction.
- **Un-decomposed event:** An abstract event has not been decomposed.
- **Dead end:** An event is not on a causal path to the outcome state.
- **Superfluous effort:** Events reestablish a redundant world state.

Each flaw type is paired with one or more repair strategies. Repair strategies can be additive or subtractive.

Additive strategies are as follows. An open condition flaw can be repaired by instantiating a new event with an effect that unifies with the open precondition or by extending a causal link from an existing event to the open precondition [14]. Thus

---

The algorithm takes a plotline plan, a set of rules to rewrite the goal and initial state, and a domain library  $A$  consisting of events specifications and quest decomposition rules.

**function** ADAPT ( $plan$ ,  $requirements$ ,  $A$ ) **returns** solution or failure  
 $plan \leftarrow$  REWRITE-GOAL-AND-INITIS( $plan$ ,  $requirements$ )  
 $fringe \leftarrow \{plan\}$   
**loop do**  
  **if**  $fringe = \emptyset$  **then return failure**  
   $plan \leftarrow$  POP( $fringe$ )  
  **if**  $plan$  has no flaws **then return plan**  
   $flaw \leftarrow$  GET-ONE-FLAW( $plan$ )  
   $newplans \leftarrow$  REPAIR( $flaw$ ,  $plan$ ,  $A$ )  
   $fringe \leftarrow$  INSERT-AND-SORT( $newplans$ ,  $fringe$ )

---

**Fig. 5.** The plotline adaptation algorithm

events are added to a plan in a backward-chaining fashion. A causal threat can be repaired by imposing ordering constraints between events [14]. An un-decomposed event can be repaired by selecting and applying a decomposition rule, resulting in new events instantiated, or existing events reused, as less abstract children of the abstract event [31].

Dead-end flaws can be handled in an additive fashion. We implement two additive dead-end repair strategies. First, if there is another event that has an open condition that unifies with an effect of the dead end, we can try to extend a causal link from an effect of the dead end to the open precondition of the other event. Second, we can shift an existing causal link to the dead-end event. This can happen if the dead end has an effect that matches the condition of a causal link between two other events. The dead-end event becomes the initiating point of the causal link, which may make the other event a dead end unless it has two or more causal links emanating from it. A third strategy is to ignore the flaw. This is used only as a last resort in the case that all other repair strategies, additive or subtractive, have proven to lead to failures. The intuition behind this strategy is that dead-end events are aesthetically undesirable but acceptable if necessary.

Superfluous effort flaws often occur when resolving other flaws. To repair a superfluous effort, one strategy is to extend causal links from events in the superfluous effort back to earlier events with effects that match. Extending causal links back to earlier events is a common technique used in continuous planning [22]. After the extension, some events in the superfluous effort become dead ends, and will be repaired accordingly. As with dead ends, a last-resort strategy is to ignore the flaw, favoring a narrative with superfluous efforts over no solution.

Subtractive strategies repair a flaw by deleting the source of the flaw from the plotline structure. Subtractive strategies are essential for plot adaptation because pre-existing events may interfere with the addition of new events, resulting in outright failure or awkward workarounds to achieve soundness and coherence. Deletion is straightforward. However, if an event to be deleted is part of a decomposition hierarchy, all siblings and children are deleted and the parent event is marked as

un-decomposed. This preserves the intuition authored into quests and decomposition rules.

Open condition flaws can be subtractively repaired by deleting the event with the open precondition. Causal threat flaws can be subtractively repaired by deleting the event that threatens a causal link. Dead end flaws can be subtractively repaired by deleting the dead end event. We implement a heuristic that prefers to retain events in the original quests as much as possible. Table 1 shows all the repair strategies available for each type of flaw.

The ability to add and delete events can lead to non-systematicity – the ability to revisit a node through different routes – and infinite loops. To preserve systematicity, we prevent the deletion of any event or link that was added by the algorithm. Events and links inserted by the algorithm are marked as “sticky” and cannot be subsequently deleted, whereas those in the original plotline are not sticky and can be removed.

## 4.2 Heuristics

As with all search problems, a powerful heuristic can significantly improve the efficiency of the search algorithm. Two types of heuristics are typically used in conventional partial-order planning. Here, we focus on the heuristic that determines which plan on the fringe to visit. Traditionally, such a heuristic favors plans with fewer flaws and shorter plans over longer ones.

**Table 1.** Additive and subtractive strategies for repairing flaws

Flaw	Description	Repair Strategies
<b>Open condition</b>	Event $e$ has a precondition $p$ that is not satisfied by a causal link.	<ol style="list-style-type: none"> <li>1. Instantiate new event <math>e_{\text{new}}</math> that has an effect that unifies with <math>p</math>. Extend a causal link from <math>e_{\text{new}}</math> to <math>e</math>.</li> <li>2. Select an existing event <math>e_{\text{old}}</math> that has an effect that unifies with <math>p</math>. Extend a causal link from <math>e_{\text{old}}</math> to <math>e</math>.</li> <li>3. Delete <math>e</math>.</li> </ol>
<b>Causal threat</b>	Event $e_k$ has an effect that negates a causal link between events $e_i$ and $e_j$ .	<ol style="list-style-type: none"> <li>1. Promotion: temporally order <math>e_k</math> before <math>e_i</math>.</li> <li>2. Demotion: temporally order <math>e_k</math> after <math>e_j</math>.</li> <li>3. Delete <math>e_k</math>.</li> </ol>
<b>Un-decomposed event</b>	Event $e$ is abstract but has no children.	<ol style="list-style-type: none"> <li>1. Select and apply a decomposition rule, instantiating new events or reusing existing events as children.</li> </ol>
<b>Dead end event</b>	Event $e$ is a dead end.	<ol style="list-style-type: none"> <li>1. Select an existing event <math>e_{\text{old}}</math> that has a precondition that is unsatisfied and that unifies with an effect of <math>e</math>. Extend a causal link from <math>e</math> to <math>e_{\text{old}}</math>.</li> <li>2. Select an existing event <math>e_{\text{old}}</math> that has a precondition that is satisfied by causal link <math>c</math> and unifies with an effect of <math>e</math>. Transfer the starting point of <math>c</math> to <math>e</math>.</li> <li>3. Instantiate new event <math>e_{\text{new}}</math> that has a precondition that unifies with an effect of <math>e</math>. Extend a causal link from <math>e</math> to <math>e_{\text{new}}</math>.</li> <li>4. Ignore the flaw.</li> </ol>
<b>Superfluous event</b>	Event $e$ is superfluous.	<ol style="list-style-type: none"> <li>1. Link effects of earlier steps to preconditions of <math>e</math>.</li> <li>2. Ignore the flaw.</li> </ol>

In order to preserve the original authorial intent of the plotline, deletion of events should be used with caution and guided by a good heuristic. One method is to favor the deletion of actions more relevant to the quests removed than to quests that remain. We propose two relevance criteria to build such a heuristic. The first criterion of relevance is causal relationships. Actions that immediately precede or follow actions within removed quest decompositions are more relevant to them than actions further away. The causal relevance between two actions is inversely proportionate to the length of the shortest path between them. The second criterion is the objects or characters the actions refer to. For example, actions in the `Witch-Hunt` quest refers to the witch frequently, where as other quests, as shown in Fig. 2 and 8, do not refer to her at all. We propose that the locality of character and object reference can be exploited to identify relationships between events in a plan.

### 4.3 Adaptation Example

In this section, we explain the working of quest-centric adaptation planning with an example of a simple role-playing game. As shown in Figure 6, the original game narrative consists of two quests. In the first quest, the player kills the witch, arch-enemy of the king, by pouring a bucket of water on her. In the second quest, the player rescues the princess from a dragon and marries her. However, suppose the player prefers treasures to marriage, we can remove the rescue quest and add an escape quest where the player is locked in a treasure cave and can only escape by solving a puzzle. The original storyline, an intermediate step, and the final result are shown respectively in Figs. 6, 7 and 8. The order of operations is denoted with numbers in circles. We do not intend to explain every detail due to space constraints. For the sake of simplicity, the search is assumed to be nondeterministic, which always makes the correct choice at every decision point. Backtracking will happen in real applications, even though not shown here.

Fig. 6 shows a given storyline of two quests. Thick gray arrows indicate the two quests satisfy quest-level goals `quest-complete(Witch-Hunt)` and `quest-complete(Rescue)` respectively. At the world-state level, the only goal is `married(player, princess)`, which is satisfied by the primitive action `Marry Princess` as shown by a thin black arrow.

We begin with requirements from the user preferring escape missions to rescues. The quest-level goal situation is updated accordingly by removing `quest-complete(rescue)` and adding `quest-complete(escape)`. The only outgoing causal link from the action `Rescue Quest` is used to satisfy this quest-level goal. As a result, this action becomes a dead end. The first step of planning is to remove it together with all descendant actions and all associated causal links. To fulfill the added goal `quest-complete(escape)`, the abstract action `Rescue Quest` is added and subsequently decomposed. New actions in the decomposition are added. They bring new open preconditions. We then deal with world-level goals. In the next few refinement iterations, dead ends, marked with number 3, are removed and actions marked with number 4 and 5 are added to fulfilling open preconditions. After these operations, we have obtained the plan in Fig. 7.

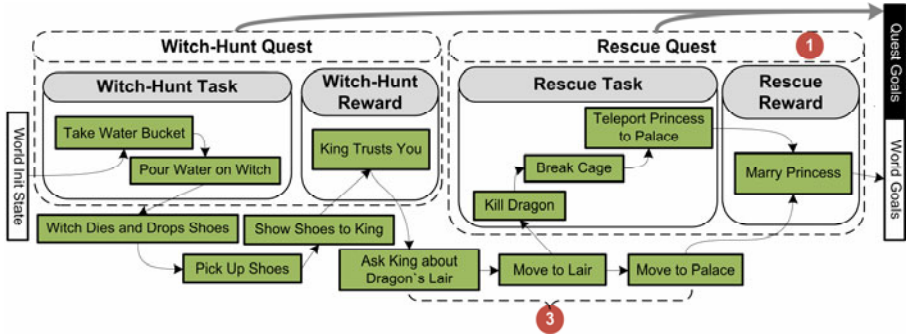


Fig. 6. The original plotline

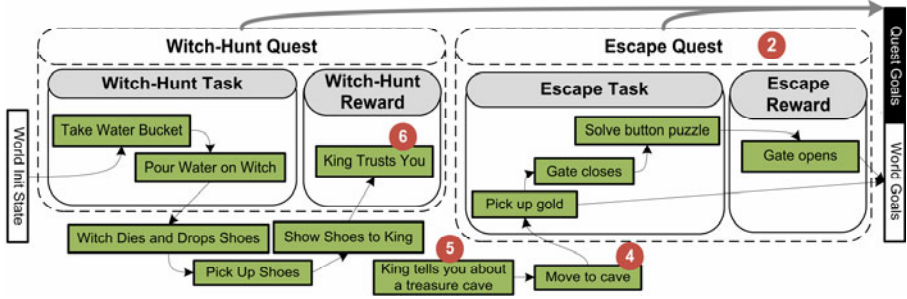


Fig. 7. Snapshot of an intermediate point in the adaptation process

The reward component of Witch-Hunt Quest is modified as follows. The action King Trusts You, marked with 6, becomes a dead end and removed. Its removal introduces two flaws: 1) the action Show Shoes to King has become a dead end, and 2) the Witch-Hunt Reward abstract action now has no decomposition. The relevance heuristic comes into play in resolving the dead end. The action Show Shoes to King is determined to be more relevant to the remaining quest than to the removed. Hence, we prefer establishing an outgoing link known-success(king, hero, witch-hunt) for action number 5 to removing it. Finally, we need a new decomposition for Witch-Hunt Reward, and we realize the decomposition can reuse action number 5. Having fixed all flaws, we have a complete and coherent narrative, shown in Fig. 8.

#### 4.4 Analysis of Authorial Leverage

Plotline adaptation scales up the ability to deliver customized experiences without significantly increasing the authoring effort. Chen et al. [2] defines authorial leverage as the quality of experience per unit of domain engineering, where quality is a function of complexity, ease of change, and variability of experience. We focus on variability – the number of distinct stories – as our metric.



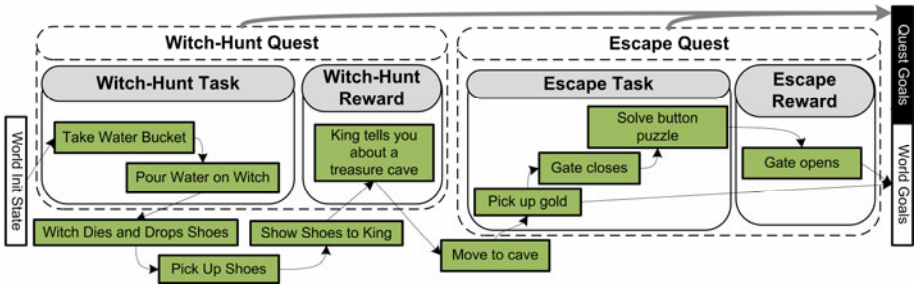


Fig. 8. Complete, Coherent Narrative after Adaptation

A one-time authoring cost by a domain engineer is incurred in the development of a world domain model, containing specifications for primitive events, abstract events (including quests), and decomposition rules. The payoff is a theoretically exponential leverage. The adaptation process can theoretically produce as many variations of a given plotline as the size of the power set of available quests. In practice, the number will be lower because a large fraction (e.g. 70%) of the original will be retained in each adaptation request. However, the scaling will still be exponential if the fraction remains constant. To manually achieve this scaling, one would have to author  $n(n-1)$  transitions between quests ( $n-1$  variations of each quest so it can be paired with  $n-1$  other quests). Thus, one strength of plotline adaptation is the ability to opportunistically discover new transitions between quests based on the world model. Future work is required to measure the pragmatic authorial leverage of the system.

#### 4.5 Evaluation

The principles of narrative soundness and coherence guide the adaptation process. To evaluate our approach to adaptation with respect to the necessity of detecting and resolving narrative soundness and coherence, we used an ablative technique whereby we determined degree of adaptation success on specific problems with several versions of the algorithm with different repair strategies disabled. Our hypothesis is that plotlines generated by the complete algorithm are preferred to stories generated when the system cannot repair dead ends or open preconditions.

Two adaptation tasks were performed based on a hypothetical player model. Each required the replacement of one quest with another in a two-quest plotline. The following versions of our algorithm were used to generate three versions of plotlines for each task:

- **N0:** Cannot repair flaws except un-decomposed events
- **N1:** Cannot repair dead-end flaws
- **N2:** The complete algorithm

Plotlines produced by N0 lacked events that establish required preconditions and seemed to contain gaps. Plotlines produced by N1 contained at least one dead end. Text descriptions of each plotline were hand-authored and participants were provided with the six descriptions arranged in two groups where each group contained

**Table 2.** Empirical results of the evaluation

<b>Plot Group 1</b>	<b>N2&gt;N1</b>	<b>N2&gt;N0</b>	<b>N1&gt;N0</b>
No. Participants	13	22	22
Percentage	52%	88%*	88%*
<b>Plot Group 2</b>	<b>N2&gt;N1</b>	<b>N2&gt;N0</b>	<b>N1&gt;N0</b>
No. Participants	19	25	15
Percentage	76%*	100%*	60%

adaptations generated by N0, N1, and N2 for one of the two tasks. Our hypothesis is confirmed if people prefer N2 to N1 (N2>N1) and N2 to N0 (N2>N0).

Twenty-five participants were involved in the study. The results are summarized in Table 2. All results were put to one-sided tests on binomial distribution at the significance level of  $p < 0.05$ ; asterisks (\*) mark significant results. In group 1, a significant number of participants preferred N2 to N0, but no significance was found about those who preferred N2 to N1. For plotlines in group 2, a significant number of participants preferred N2 to both N0 and N1.

Results from group 1 and group 2 should corroborate, suggesting a hidden independent variable. The N1 plotlines in both groups contained a dead end. However, the group 1 dead end appeared to be events that were never followed up, whereas the group 2 dead end directly contradicted the apparent intentions of other events. It is likely that our system, using formal definitions, is more sensitive to story incoherence than human game players. Thus, we believe that group 2 plotlines, consisting of more disruptive and noticeable dead ends, are more representative of worst-case situations. Group 2 results indicate that it may be beneficial to be cautious, erring on the side of being overly sensitive to story incoherence. Results of Group 2 validate our hypothesis, leading us to believe that enforcing narrative coherence is beneficial and that no harm is done by being overly sensitive to story incoherence.

## 5 Related Work

Automated adaptation of computer games has been explored in the context of player character attributes, difficulty adjustment, and game environment changes. Increasingly, player models are being used to adapt game content. Interactive storytelling systems demonstrate how players' behaviors can change the story content in virtual worlds on the fly. See Roberts and Isbell [20] for a general discussion of interactive narrative approaches. Of particular relevance to this work are interactive narrative approaches that leverage player models. Thue et al. [26] describe a technique whereby a player model based on role player types is used to select branches through an interactive story. Seif El-Nasr [24] attempts to infer feature-vectors representing player style, affecting changes in which dramatic content is presented to the player. Sharma et al. [25] use case-based reasoning to learn player preferences over plot points for the purposes of selecting the next best story plot

point. These approaches assume the existence of branching story graphs or pre-authored alternatives.

Note that our system is an offline process that effectively “re-writes” a plotline based on a player model before it is executed. As such, our system can afford to backtrack and make globally optimal decision, such as those about narrative coherence, whereas online adaptation systems can only make local decisions that cannot be undone. Our system is not an interactive narrative system; once execution of the plotline begins, our system does not make further changes. Indeed, interactive storytelling and plotline adaptation are complimentary: the adaptation system can be seen as a process that, based on knowledge about the player, configures the drama manager, which then oversees the user’s interactive experience online. Our system can be coupled with, for instance, the Automated Story Director [18], a planning-based interactive narrative system.

As an offline procedure, plotline adaptation has a strong connection with story generation. Story generation is the process of automatically creating novel narrative sequences from a set of specifications. The most relevant story generation work is that that uses search as the underlying mechanism for selecting and instantiating narrative events (cf., [10], [6], [15], and [19]). The distinction between our plotline adaptor and story generation is that plotline adaptation starts with a complete narrative structure and can both add and remove narrative content, whereas story generation typically starts from scratch. As with case-based planning, the adaptation of plotlines is, in the worst-case, just as hard as planning from scratch [12]. However, in the average case, starting from an existing plotline will require much fewer decisions to be made.

In a parallel effort, the TACL system [13] is designed to adapt and customize military training scenarios. Realistic military training is a highly rigorous process. Any automatic adaptation must preserve pedagogical correctness and the tolerance of modification is low. Game quests, on the other hand, can be modified extensively. In this paper, we apply the algorithm in the novel context of quests and games.

Work on adapting player experience in games has been addressed in terms of game level generation. Hullett and Mateas [5] have investigated generation of game level floor plans, and thus the narrative of moving through space, using HTN planning. HTN planning requires complete specification of how each task can be performed. In comparison, our approach is capable of opportunistic discovery of novel event sequences. Finally, others have explored game world generation and other non-narrative content generation using neural network models of players and evolutionary computation (cf., [27]). At the moment, we are ignoring the generation of landscape and environment in games.

## 6 Conclusions

As game players possess different motivations, tastes and needs, a one-size-fits-all approach to game plotlines may prove to be limiting. We treat adaptation as the optimization of plotlines based on requirements derived from a player model employing knowledge about player preferences and a model of novelty. As such, we find an offline approach to be beneficial in achieving global optimization of plotline structure.

The adaptation problem itself is solved by an iterative improvement search based on partial-order planning. However, in order to start from a complete plotline and arrive at a variation with different quests, we employ both additive and subtractive improvement mechanisms. To the extent that the player model is an approximation of player preferences, future work may pair our offline adaptation technique with online interactive storytelling engines.

As the world orients toward greater on-demand and customized entertainment experiences, overcoming the content authoring bottleneck will increasingly require automation on the level of creative production. We believe that a partnership between human authors and automated adaptation can scale up our ability to deliver the “right experience to the right person at the right time.”

## References

1. Bartle, R.: Hearts, Clubs, Diamonds, Spades: Players Who Suit MUDs. *Journal of Virtual Environments* 1 (1996)
2. Chen, S., Nelson, M.J., Sullivan, A., Mateas, M.: Evaluating the Authorial Leverage of Drama Management. In: Mehta, M., Louchart, S., Roberts, D.L. (eds.) *Intelligent Narrative Technologies II: Papers from the Spring Symposium (Technical Report SS-09-06)*. AAAI Press, Menlo Park (2009)
3. Fikes, R., Nilsson, N.: STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2, 189–208 (1971)
4. Graesser, A., Lang, K.L., Roberts, R.M.: Question Answering in the Context of Stories. *Journal of Experimental Psychology: General* 120, 254–277 (1991)
5. Hullett, K., Mateas, M.: Scenario Generation for Emergency Rescue Training Games. In: *Proceedings of the Fourth International Conference on the Foundations of Digital Games (2009)*
6. Lebowitz, M.: Planning Stories. In: *Proceedings of the 9th Annual Conference of the Cognitive Science Society (1987)*
7. Li, B., Riedl, M.O.: Planning for Individualized Experiences with Quest-Centric Game Adaptation. In: *Proceedings of the ICAPS 2010 Workshop on Planning in Games (2010)*
8. Li, B., Riedl, M.O.: An Offline Planning Approach to Game Plotline Adaptation. In: *Proceedings of the 6th Conference on Artificial Intelligence for Interactive Digital Entertainment (2010)*
9. Magerko, B.S., Riedl, M.O.: What Happens Next?: Toward an Empirical Investigation of Improvisational Theatre. In: *Proceedings of the 5th International Joint Workshop on Computational Creativity (2008)*
10. Meehan, J.: *The Metanovel: Writing Stories by Computer*. Ph.D. Dissertation. Yale University, New Haven (1976)
11. Morningstar, C., Farmer, F.R.: The Lessons of Lucasfilm’s Habitat. In: Bendikt, M. (ed.) *Cyberspace: First Steps*. MIT Press, Cambridge (1991)
12. Muñoz-Avila, H., Cox, M.: Case-Based Plan Adaptation: An Analysis and Review. *IEEE Intelligent Systems* 23 (2008)
13. Niehaus, J., Riedl, M.O.: Scenario Adaptation: An Approach to Customizing Computer-Based Training Games and Simulations. In: *Proceedings of the AIED 2009 Workshop on Intelligent Educational Games (2009)*

14. Penberthy, J., Weld, D.: UCPOP: A Sound, Complete, Partial-Order Planner for ADL. In: Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning (1992)
15. Porteous, P., Cavazza, M.: Controlling Narrative Generation with Planning Trajectories: The Role of Constraints. In: Iurgel, I.A., Zagalo, N., Petta, P. (eds.) ICIDS 2009. LNCS, vol. 5915, pp. 234–245. Springer, Heidelberg (2009)
16. Riedl, M.O.: Incorporating Authorial Intent into Generative Narrative Systems. In: Mehta, M., Louchart, S., Roberts, D.L. (eds.) Intelligent Narrative Technologies II: Papers from the Spring Symposium (Technical Report SS-09-06). AAAI Press, Menlo Park (2009)
17. Riedl, M.O., Li, B.: Creating Customized Virtual Experiences by Leveraging Human Creative Effort: A Desideratum. In: Proceedings of the AAMAS 2010 Workshop on Collaborative Human/AI Control for Interactive Experiences (2010)
18. Riedl, M.O., Stern, A., Dini, D., Alderman, J.: Dynamic Experience Management in Virtual Worlds for Entertainment, Education, and Training. *International Transactions on Systems Science and Applications* 3 (2008)
19. Riedl, M.O., Young, R.M.: Narrative Planning: Balancing Plot and Character. *Journal of Artificial Intelligence Research* 39 (2010)
20. Roberts, D.L., Isbell, C.L.: A Survey and Qualitative Analysis of Recent Advances in Drama Management. *International Transactions on Systems Science and Applications* 3, 61–75 (2008)
21. Rollings, A., Adams, E.: Andrew Rollings and Ernest Adams on Game Design. New Riders, Indianapolis (2003)
22. Russell, S., Norvig, P.: *Artificial Intelligence: A modern approach*. Prentice-Hall, Englewood Cliffs (2002)
23. Saunders, R., Gero, J.: Curious Agents and Situated Design Evaluations. *AI for Engineering, Design, Analysis, and Manufacturing* 18, 153–161 (2004)
24. Seif El-Nasr, M.: Interaction, Narrative, and Drama Creating an Adaptive Interactive Narrative using Performance Arts Theories. *Interaction Studies* 8 (2007)
25. Sharma, M., Ontañón, S., Mehta, M., Ram, A.: Drama Management and Player Modeling for Interactive Fiction Games. *Computational Intelligence* 26, 183–211 (2010)
26. Thue, D., Bultko, V., Spetch, M., Wasylishen, E.: Interactive Storytelling: A Player Modelling Approach. In: Proceedings of the 3rd Conference on Artificial Intelligence and Interactive Digital Entertainment Conference (2007)
27. Togelius, J., Yannakakis, G., Stanley, K., Browne, C.: Search-based Procedural Content Generation. In: Proceedings of the EvoStar Conference (2010)
28. Trabasso, T., van den Broek, P.: Causal Thinking and the Representation of Narrative Events. *Journal of Memory and Language* 24, 612–630 (1985)
29. Yee, N.: The Demographics, Motivations, and Derived Experiences of Users of Massively-Multiuser Online Graphical Environments. *PRESENCE: Teleoperators and Virtual Environments* 15, 309–329 (2006)
30. Young, R.M.: Notes on the Use of Plan Structures in the Creation of Interactive Plot. In: Mateas, M., Singers, P. (eds.) *Narrative Intelligence: Papers from the Fall Symposium* (Technical Report FS-99-01). AAAI Press, Menlo Park (1999)
31. Young, R.M., Pollack, M.E.: Decomposition and causality in partial-order planning. In: Proceedings of the 2nd International Conference on Artificial Intelligence and Planning Systems (1994)

# Guiding User Adaptation in Serious Games

Joost Westra<sup>1</sup>, Frank Dignum<sup>1</sup>, and Virginia Dignum<sup>2</sup>

<sup>1</sup> Universiteit Utrecht\*

<sup>2</sup> Delft University of Technology

**Abstract.** The complexity of training situations requires teaching different skills to different trainees and in different situations. Current approaches of dynamic difficulty adjustment in games use a purely centralized approach for this adaptation. This becomes impractical if the complexity increases and especially if past actions of the non player characters need to be taken into account. Agents are increasingly used in serious game implementations as a means to reduce complexity and increase believability. Agents can be designed to adapt their behavior to different user requirements and situations. However, this leads to situations in which the lack of coordination between the agents makes it practically impossible to follow the intended storyline of the game and select suitable difficulties for the trainee.

In this paper, we present a monitoring system for the coordination of the characters actions and adaptation to guarantee appropriate combinations of character actions that ensure the preservation of the storyline. In particular we propose an architecture for game design that introduces a monitoring module to check the development of user skills and direct coordinated agent adaptation. That is, agents propose possible courses of action that are fitting their role and context, and the monitor module uses this information together with its evaluation of user level and storyline progress to determine the most suitable combination of proposals.

## Categories and Subject Descriptors:

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence

**General Terms:** Design.

**Keywords:** Agents in games and virtual environments, Adaptation, Organisations.

## 1 Introduction

Dynamic difficulty adjustment is an important aspect in training applications that need to be suitable for a large variety of users. Current approaches of dynamic difficulty adjustment in games use a purely centralized approach for this adaptation [21,9]. This becomes impractical if the complexity increases and especially if past actions of the non player characters (NPC's) need to be taken

---

\* This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

into account. Serious games [19,20] often require teaching different skills to the trainee making centralized adaptation even more complex. The use of software agents has been advocated as a means to deal with the complexity of serious games [12]. Leaving the responsibility of staying believable and adjusting to game progress, to the non player characters creates a much more manageable situation. In serious games, quality is measured in terms of how well the components in the game are composed, how they encourage the player (or trainee) to take certain actions, the extent to which they motivate the player, i.e. the level of immersiveness the game provides, and how well the gaming experience contributes to the learning goals of the trainee [4].

Believability is a main driver of game development. The search for enhanced believability has increasingly led game developers to exploit agent technology in games [12]. Three requirements for online game adaptation have been identified [1]. First, the initial level of the player must be identified. Second, the possible evolutions and regressions in the player's performance must be tracked as closely and as quickly as possible. Third, the behavior of the game must remain believable. In this paper we will focus on how the tracking is used to find the most appropriate solution while making sure that the game stays believable.

In order to optimize learning, serious games should provide the trainee an ordered sequence of significantly different and believable tasks. Without a clear organization structure, adaptation can quickly lead to a disturbed storyline and the believability of the game will be diminished. Furthermore, characters in serious games are usually active for relatively long periods. This poses an extra burden on the believability of the game, namely coherence of long-term behavior [14]. When there are multiple NPC's that all have their own preferences and can all adapt to the trainee independently, it becomes almost impossible to create a coherent game that has a natural progression of the game and is the right difficulty for the user. The game progression is much more controllable if there is a monitoring system in the application where the desired progression (could have different paths) is specified. We propose a system where we do not only have a monitoring system that specifies the desired storyline but also keeps track of the current progression within the storyline. The NPC's are still programmed with their own preferences but they receive updates of the game progression. The agents can then easily be programmed to perform different plans, not only dependent on their own beliefs but also dependent on the game progression.

Coordination of agent actions also becomes a lot more manageable if there is a central control system that allows the designer to put restrictions on the possible plans performed by the agents. A very simple example is that the designer can specify that at a certain point in the game, one of the NPC's should always check the left hallway. A possibility would be that all the agents are programmed with all these restrictions in mind and that they communicate directly with each other to make sure that one goes left. We propose a coordination system where all the agents propose multiple actions with preference weights corresponding to each of these proposals. From these proposals the adaptation engine will select the optimal solution that also keeps the restrictions of the designer and the

preferences of the agents in mind. In this example, this still means that at least one agent should have the capability to check the left hallway but it puts a lot less burden on the designer to allow autonomy within the agents while making sure that certain critical criteria are always met.

In previous work [24,23] we proposed the use of multi-agent organizations to define a storyline in such a way that there is room for adaptation while making sure that believability of the game is preserved. In this paper, we discuss the effect of these approach to adaptation on the design of the agents.

The paper is organized as follows. In the next section the background and motivation for this model are discussed. In section 3 we describe the framework and the adaptation engine. We illustrate the use of the model with an example in section 4. Conclusions are discussed in the last section.

## 2 Background

We advocate bringing together three issues in order to adapt serious games to the user. The adaptation should be distributed over the separate elements that constitute the story line, these elements should adapt themselves online using some machine learning technique and they should do it in an organized fashion to maintain the general story line. In this section, we discuss the related work that can be used for these aspects.

### 2.1 Adaptation in Games

Even though many commercial games do not use any dynamic difficulty adaptation [17], already some research has been done on difficulty adaptation in games. Most of this research focuses on adaptation of certain simple quantitative elements in the game that do not influence the storyline of the game. For example better aiming by opponents or adding more or a stronger type of opponents.

Current research on online adaptation in games is based on a centralized approach [22,10]. Centralized approaches define the difficulty of all the subtasks from the top down. This is only feasible if the number of adaptable elements is small enough and if the separate adaptable elements have no separate time lines that need to be taken into account. In shooting games, for example, these requirements are not problematic. The games only adapt to the shooting skill of the trainee and most characters only exist for a very limited amount of time.

Another important aspect of adaptation in (serious) games is the distinction between direct and indirect adaptation. Direct adaptation occurs when the designer specifies possible behavior of the agents in advance and specifies how and when to change this behavior. The designer also specifies what input information should be used. Direct adaptation only allows adaptation to aspects that the designer has foreseen. No unexpected behavior can emerge when using direct adaptation. On the other hand, in indirect adaptation performance is optimized by an algorithm that uses feedback from the game world. This requires a fitness function and usually takes many trials to optimize. If indirect optimization is used the algorithm also needs to be able to cope with the inherent randomness



of most computer games. In this paper, we will use an approach that has the benefits of direct adaptation without the need for the designer to directly specify how the adaptation should be done. The designer is able to specify certain conditions on the adaptation to guarantee the game flow but does not have to specify which implementations are chosen after each state.

An added challenge for user adaptation in games is, that it can only be done while the user is playing the game [3,5]. Online adaptation requires that the algorithm adapts quicker with a lot less episodes and learning data. Because the game is adapting while the user is participating in the game, it is also important that no unwanted and unpredictable situations are introduced by the adaptation. This means that the adaptation should only try promising and believable solutions while exploring different options.

In this paper, we will focus on not supervised learning (we use this term to avoid confusion with unsupervised learning). That is, we do not expect the user or an expert to assess the performance of the user on certain tasks. The algorithm also needs to stay real-time when scaled, meaning that it also functions fast enough if a lot of adaptable elements are added.

One of the most important factors when performing online adaptation is to limit the size of the state space. In offline learning the algorithms are usually given a lot of different input variables and the algorithm is given enough time and is flexible enough to learn which have the most influence. In online learning this usually takes too much time.

Research has been done on using reinforcement learning in combination with adaptation to the user [22,2]. Most of these algorithms rely on learning relatively simple subtasks. Moreover, the aim of these adaptation approaches is learning the optimal policy (i.e. making it as difficult as possible for the user). In order to avoid that the system becomes too good for the user, some approaches filter out the best actions to adjust the level of difficulty to the user. This results in unrealistic behavior where characters that are too successful suddenly start behaving worse again. Little attention is paid to preserving the story line in present online adaptation mechanisms, because they only adjust simple subtasks that do not influence the storyline of the game. Typical adjustments are, for example, changing the aiming accuracy of the opponents or adding more enemies. Some work has been done on preserving the storyline with adapting agents [13] but they focus on preserving the plot, not on adapting to the trainee.

## 2.2 Agent Organizations

Adapting the game to the trainee for complex learning applications requires both learning capabilities and decentralized control. However, in order to guarantee successful flow of the game and the fulfillment of the learning objectives, the system needs to be able to describe global objectives and rules. Although many applications with learning agents exist, multi-agent systems with learning agents are usually very unpredictable [16]. In order to limit unpredictability in MAS, organization-oriented approaches have been advocated such as Opera [7] and [8]. In this framework it is possible to define conditions when certain plans are

allowed or not. The ordering of the different possible plans can also be defined in this framework. This allows the designer to make sure that the users are not exposed to tasks that are not suitable yet or would ruin the storyline. In previous work we have shown how to use agent organizations to specify the boundaries of the game [23].

The OperA model for agent organizations enables the specification of organizational requirements and objectives, and at the same time allows participants to have the freedom to act according to their own capabilities and demands. In OperA, the designer is able to specify the flow of the game by using landmarks. The different sub-storyline definitions of the game are represented by scenes which are partially ordered without the need to explicitly fix the duration and real time ordering of all activities. That is, OperA enables different scenes of the game to progress in parallel. In the scenes, the results of the interaction are specified and how and in what order the different agents should interact.

Such an interaction structure defines the ordering of the scenes and when it is allowed to transition to the next scene. The scenes are defined by scene scripts that specify which roles participate and how they interact with each other. The definition of the organization can be so strict that it almost completely defines the strategy, similar to what is done when using scripts. But it is also possible to specify the organization in such a way that all the agents in the game work towards achieving the goals of the game but are still able to do this, using different strategies. For example, you can specify that a certain amount of money needs to be earned by the agents before moving to the next scene, but that you do not specify how they need to do this. It is also possible to define norms in the scene description. This makes it possible to put extra restrictions on the behavior of the agent. In a scene script, it is also possible to define certain time constraints to make sure that the game progresses fast enough.

### 2.3 Adaptation with BDI-Agents

Autonomous game characters should be able to ensure that they maintain a believable storyline. They also have to make complex decisions during the game because not all information is known before the game is started. Using BDI agents is a suitable implementation because it allows us to create intelligent characters that are goal directed and able to deliberate on their actions. For the implementation of the BDI agents we will use the 2APL [6] language. 2APL is an effective integration of programming constructs that support the implementation of declarative concepts such as belief and goals with imperative style programming such as events and plans. Like most BDI-based programming languages, different types of actions such as belief and goal update actions, test actions, external actions, and communication actions are distinguished. These actions are composed by conditional choice operator, iteration operator, and sequence operator. The composed actions constitute the plans of the agents. Like the existing agent programming languages, 2APL provides rules to indicate that a certain goal can be achieved by a certain pre-compiled plan. Agents may select and apply such rules to generate plans to achieve their goals.

Most BDI architectures do not provide learning abilities. However, an extension of 2APL for adapting agents is available [11], in which multiple equivalent 2APL plans are available that are suitable for different skill levels. There are many forms of adaptation that could be used in conjunction with BDI agents. The approach we are using changes a preference ordering in the applicable plans of the agents. This means that all the plans that are performed by the agents are plans that are manually created. This results in adaptation that always selects plans that have a predictable outcome. This is important for online adaptation because the trainee observes all the plans that are performed by the agents. The agents are created in such a way that they have multiple applicable plans at the same time. The default 2APL deliberation cycle will always select the first applicable plan. This extension makes it possible to select the applicable plan with the highest preference. Plans with the same plan type but with a different difficulty for the trainee are created. This allows for adaptation of the difficulty level by changing the preferences. Not only is it possible to change the difficulty level but it is also possible to have the agent perform a plan of a different type if this is also an applicable plan.

### 3 Framework

To get a better understanding of the different elements of the whole framework we first briefly describe the different elements and the information that is passed between them. Figure 1 shows a schematic overview of all the different elements of the framework. We are currently using a custom Java environment as our *game world*, but our approach is also applicable to other games. The *NPC*'s and

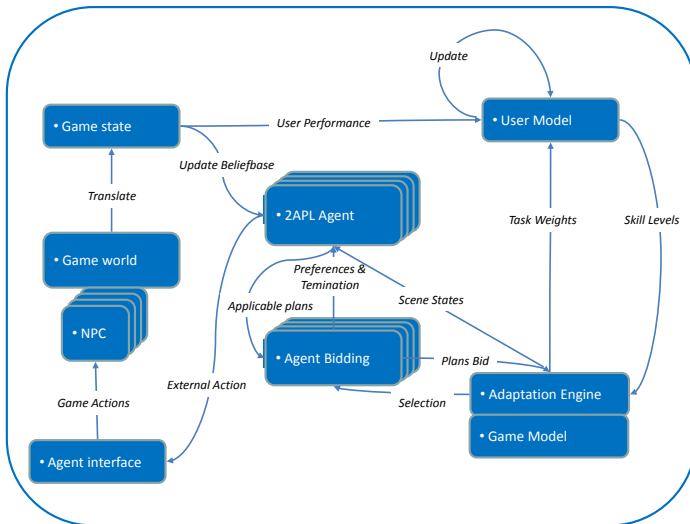


Fig. 1. Framework overview

other dynamic game elements in the game are controlled by *2APL agents*. The agents in the game have the capability to perform basic actions, like walking to a certain location or opening a door. The higher level behaviors are specified in the *2APL agents* which send the basic *external actions* to the *agent interface* which translates these commands to basic game actions.

The *game state* is used to update the beliefs of the agents, update the progression of the game and pass the performance of the trainee to the user model. The *user model* uses this information and the *task weights* from the adaptation engine to update the estimated skill level for each state. These updated *skill levels* can then be used again to find better matching agent behaviors.

The 2APL agents can perform different actions depending on their beliefs and dependent on the scene states. The *game model* contains information about the desired storyline of the game and keeps track of how far the game has progressed in the storyline. This information is passed to the *2APL agents* to influence the possible actions they can perform. The *agent bidding* module specifies the agent preferences for all the *applicable plans*. The *adaptation engine* uses this information and the information from the *user model* to find the plan assignment for the agents that best serves the situation for the trainee. The bidding module of the agent uses this information to control the plans that are selected by the agents.

The agents do not use the adaptation engine for all their plan selections. If there is no need for adaptation, then the agents will keep running their normal 2APL program with the current preferences. The adaptation engine will request a new bidding round if the deviation from the intended difficulty becomes too large. The bidding process is also started at fixed points in the game scenario where it is logical for the agents to start performing different actions. Updated preferences also do not mean that the agents have to stop performing their current plan but the selection of the first new plan is influenced.

### 3.1 Adaptation Engine

The adaptation engine consists of two different parts. One part selects the best combination of plans for all the different agents. The other part keeps track of the game progress and is responsible for checking if the combinations of plans are currently valid depending on the state of the game. The combinatorial auction has to optimize on two possibly conflicting objectives. On the one hand we want to optimize on the preferences of the agents while on the other hand we want to select the combination which is the optimal difficulty for the user. Because we focus on adapting to the trainee, we give the highest priority to finding the best match for the trainee. Remember that we optimize on different skills of the trainee. Slight variations in difficulty level are not problematic but do want to prevent large deviations from the desired skill levels for each separate skill. This means that we rather have deviations that are a bit larger for each skill than that we have multiple skill levels that are perfectly chosen but a large deviation in one remaining skill.

While optimizing on the skills of the trainee we also want to optimize on the preferences of the agents to keep their preferences into account to keep the game as believable as possible. This process uses a form of a combinatorial auction [18]. This needs to be a combinatorial auction because the agents can give a higher score for performing a certain action depending on which plans the other agents will perform. This preference dependence is only used for tasks that require coordination between the agents. For example, it is more believable for a medical NPC to go towards enemy lines if the offensive NPC's are going towards enemy lines. We try to limit the amount of preference dependences because it is much more labor intensive for the game designer to specify the preferences of the agents and it is also more computationally expensive to find the best solution. Similar to finding the best match for the skill level we also want to avoid large deviations from the preferences. This means that we do not optimize on the highest combination of preferences from the agents but on the smallest squared deviations from the preferred proposal. The formula below is the function that we currently use and corresponds with the properties that we specified. We will extend the framework to allow for more complex multi attribute functions.

$$f = w_u \frac{((d_1 - s_1)^2) + (d_2 - s_2)^2 + \dots + (d_n - s_n)^2}{n} + w_a \frac{((b_1 - a_1)^2) + (b_2 - a_2)^2 + \dots + (b_m - a_m)^2}{m} \quad (1)$$

With  $w_u$  specifying the influence of the user model preference,  $n$  the number of skill,  $s_i$  the desired difficulty level for skill  $i$ ,  $d_i$  the difficulty of the task combination for skill  $i$ ,  $w_a$  specifying the influence of the agent preferences,  $m$  the number of agents  $b_j$  the value of the highest preference of agent,  $a_j$  the preference value of agent  $j$  corresponding to the task combination.

A common point of critique is that using dynamic difficulty adjustment removes the challenges from the game. This is true up to a certain point; the user will always be able to play the complete game below his maximum skill level. But if the player is not purposefully sabotaging the game it does not mean that the challenges of the game have to be removed from the game. Controlling the relative difficulty of the game at specific parts of the game results in a more satisfying game experience than trying to optimize on a constant "optimal" relative difficulty. In the game model we do not only allow the designer to specify the progress of the game but we also allow the designer to specify different difficulties corresponding to certain phases in the storyline. We also allow the designer to specify an absolute difficulty level, this can be a desired option especially for serious games because one would like to be able to know that if the trainee finishes the training that the skill level of the trainee is high enough. Updating the user model can be done in different ways. Our proposed user model update function is beyond the scope of this paper but is described in [23].

### 3.2 Synchronization

Selecting the best combination of plans from the different agents is easiest if they all terminate at the same moment. If all plans are terminated and started at the

same time the optimal combination for the trainee can be selected. However, the time to execute the different plans by the agents is not always the same, and to keep the storyline flowing, it is not always a possibility to terminate plans of all the agents when a few agents have completed their task. In our framework we specify different subtasks of the game application by using scenes. The end of a scene usually is a natural time for all the participating agents to terminate their behavior. This gives enough control to make the necessary changes both for the gameflow and to optimize learning for the user.

Because multiple scenes can be active at the same time, it also does not mean that if a scene is finished that all agents have terminated their plans. The goal is to have the most suitable task combination for the trainee during the whole game. Our solution is to assume that all plans that have not terminated are fixed and that newly created plan combinations keep these active plans into account. This results in a good combination for the trainee when the new plans are started. If plans are terminated the difficulty of the task changes again (becomes easier most of the time), but this can usually be compensated very quickly with new plans from the same agents (instant correction) or new plans from other agents. This results in a system that adapts quickly while keeping the behavior of the agents realistic.

### 3.3 Agent Implementation

The high level actions of the NPCs are implemented using the 2APL [6] language. This allows modeling of the NPC's using the BDI architecture. The agents are created with the game model structure in mind. This is done in such a way that the applicable plans are not only dependent on the game state and the internal state of the agent but also on the scenes that are currently active. This process makes it a lot easier for the developer to ensure the certain behaviors are only performed at the right moment in the game progress. The 2APL agents are created in such a way that multiple plans are applicable at the same time. These applicable plans can vary in difficulty for the trainee but they can also have the NPC perform substantially different tasks in the game.

When the agents receive a request to perform a new behavior they reply with a number of different applicable plans according to the game state, the active scenes and the internal state of the agent. This bidding process is not part of the normal 2APL deliberation cycle but is a separate part of the agent. We separated these tasks because it would be very inefficient and unnecessarily complex if the agents use the BDI reasoning process to decide why they want to perform a certain plan. This separate bidding part of the agent is also responsible for estimating the believability of each action. One important factor in estimating the believability of a new plan is dependent on the difference compared to the previous plan. This transition believability can either be fully manually audited by the designer for each transaction but usually a hybrid solution is most suitable. In this hybrid solution the game designer can specify that a certain plan can never follow certain other plans and may not be included in the current bid. For example, it could be desired that a victim becomes more mobile to make

it easier for the trainee to extract the victim but we do not want the trainee to observe the victim NPC transitioning from having a broken leg to suddenly running away. In most plans transitions, multiple new plans are possible but one is more believable than another. Usually plans with the similar actions and similar difficulty have the best believability. The larger the change in difficulty and the more sudden the action types change, the less believable it becomes. For example, it would be very unlikely if the aiming accuracy of an opponent in a shooting game suddenly decreases. In these cases simple formulas are used to specify the believability of plans transitions. The bidding element of the agent influences the plans selected by putting a preference relation on the plans of the agents. These preferences are used by a modified version of the deliberation cycle which selects the plans with the highest preference from all the applicable plans.

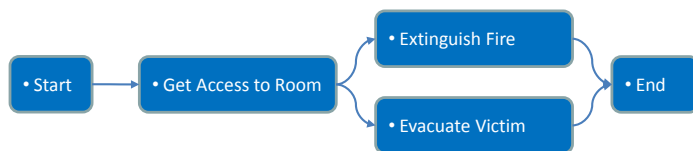
### 3.4 Prototype Implementation

We are currently developing a proof of concept of the model using java based 2D example. This particular game world is chosen to speed up the prototyping process. We are also working on a very similar coupling with a serious game created with the Quest 3D game engine.

The game model is specified using the Eclipse version of the Operetta [\[15\]](#) IDE. This tool facilitates the design, analysis and development of agent organizations using the OPERA Conceptual framework. Using this tool not only helps the designer to chose the righth elements in model but also allows to check the validity of the model. Operetta makes it possible to export the game model to a XML to make it easier for the adaptation engine to parse this information.

## 4 Example

In this section, we give an example of the possible interaction between the agents and the adaptation engine. This example is situated in a game world with three game characters, one adaptive game element, and a single building. Besides this we also have the human player (the trainee) in the role of a firecommander. The aim of the game is to train the fire commander on how to work under pressure, and we will focus on one scene where a fire needs to be extinguished in a room. The agents are adapting to optimize the difficulty best matching with the skill levels of the trainee. For simplicity we only use one partial ordering for this task. The partial ordering for this task is shown in figure [2](#). In this game, agents simulate firemen with their own preferences, skills and own memory of past games. The first evaluation on performance is done when the trainee has finished extinguishing one room or when another ending criteria is reached (trainee can fail the task completely). When this period is over we get feedback about the performance of the trainee. These performance measures are always very domain specific and need to be designed by an expert. Experts creating these kinds of training are used to create performance measurable tasks. An example of a



**Fig. 2.** Example Scene

performance measure could for example be the time it took to extinguish the fire (in reality this would be more complicated). Remember that we are getting the performance of the user for the complete tasks because different subtasks influence each other making it impossible to accurately measure independent performance.

One important element in fire commander training is the behavior of the fire. An agent that controls a certain element for the adaptation does not always need to be a character in the game, allowing us to model the behavior of the fire as a separate agent. It could be argued that the fire is also called a character. Finally the victim inside the room is also modeled as an adaptive agent. Assume that the trainee has to learn three skill categories: (1) extinguishing fire, (2) giving orders to his team and (3) extract victims.

Let's assume that the adaptation engine sends a request to all the agents to send new bids because the difficulty needs to be adjusted. These requests go to the bidding part of the agents. This bidding part uses the 2APL part of the agent to retrieve a list of applicable plans. Listing 1.1 shows simplified and shortened 2APL code for fireman agent1. One aspect that can be seen in this code is that different plans are applicable if different scenes are active. Remember that multiple scenes can be active at the same time. In this example only "scenea" is currently active. The code also shows that specific ordering of plans can be specified within the agent. This agent can only perform plans of the type "a2" after completing a plan of type "a1". Please also note that the game state also influences the applicable plans of the agent by using guard checks in the plan specifications. In this case the agent can only open the door with an axe if the character has an axe in possession. It is possible to write more advanced agents where the agent, for example, retrieves an axe to open the door with an axe.

The bidding element of the agents calculates the preferences for all the applicable plans and sends this as a proposal to the adaptation engine. For simplicity we will not use dependent preferences in this example.

The proposal of one fireman agent (*fireman agent1*) is shown in table 1(a). The other fireman agent (*fireman agent2*) proposal is shown in table 1(b). The proposal from the victim agent is shown in table 1(c). And the proposal for the agent controlling the fire is shown in table 1(d). Please note that agents from the same type can have different proposals. This difference grows as the game progresses because each agent has a different history. Also note that the agents can prefer different variants of one action type over and under another action, at the same time, depending on the difficulty.



**Listing 1.1.** FireMan1.2apl

```
abilityToAdd (FireMan1).
```

**BeliefUpdates:**

```
{true } SetDoorOpened(A) {dooropened(A)}
[... ]
{true } SetGoalStarted(A){goalstarted(A)}
{goalstarted(A)} SetGoalDone(A) {not goalstarted(A)}
```

**PG-rules:**

```
[... ]
sceneb() <- role(Fireman1) and not goalstarted(sceneb) |
{ [... ] }
```

**PC-rules:**

```
a1 <- state (FireMan1, axe, yes, -) |
{ @gameworld(action (FireMan1, "opendoor, axe1", "ok"), -); }
```

```
a1 <- true |
{ @gameworld(action (FireMan1, "opendoor, kick1", "ok"), -); }
```

```
[.. ]
a1 <- true |
{ @gameworld(action (FireMan1, "manhose, helpfull1", "ok"), -); }
[.. ]
```

Remember that their proposals not only contain different plans and different action types but also contain several variations with different difficulties for the user. Because there are no prior actions performed by the agents it is easier to keep their behavior believable in this phase. For example the fire could start at any level of intensity before the user arrives and the cooperative members could be as intelligent as you like. This allows the agents to propose more varied plans.

The adaptation engine receives these proposals and calculates the highest scores for the combinations. Starting with the highest score, the adaptation engine will then check if the scores are valid within the game model. This process continues until a valid combination is found (the valid combination with the highest score).

A possible requirement for the whole task is that one agent always needs to operate the water pump. Fireman agent1 is the only one capable of performing this task and therefore only combinations with fireman agent1 manning the pump are valid. The partial ordering specifies that the getting access to the room needs to be completed before the victims can be extracted and before the fire can be extinguished. This means that in the beginning of the task at least one agent needs to perform this subtask. Fireman agent2 has two possible actions ('open door' and 'break window') that fit this subtask. The finally selected combination will contain at least one of these options. For all the remaining combinations the expected difficulty for all the skills category of the trainee are calculated. From these estimations the combination best matching with the desired difficulty is selected.

The number of combinations grows exponentially in regards to the number of proposals of the agents. In practice this is not a big problem because the amount of proposals is of reasonable size and this calculation does not need to be done very often compared to other processes in a real time game. The action

**Table 1.**

(a) fireman agent1					(b) fireman agent2				
Plan id	Skill	Type	Diff	Pref	Plan id	Skill	Type	Diff	Pref
ID1	orders	man hose	0.5	0.7	ID1	orders	man hose	0.5	0.7
ID2	orders	man hose	0.6	0.7	ID2	orders	man hose	0.6	0.7
ID3	orders	open door	0.5	0.5	ID3	orders	open door	0.5	0.6
ID4	orders	man pump	0.5	0.5	ID4	orders	break window	0.5	0.5
ID5	orders	man pump	0.8	0.5					

(c) victim agent					(d) fire agent				
Plan id	Skill	Type	Diff	Pref	Plan id	Skill	Type	Diff	Pref
ID1	victim	normal	0.4	0.7	ID1	fire	ground fire	0.5	0.7
ID2	victim	disabled	0.8	0.3	ID2	fire	ceiling fire	0.8	0.7
					ID3	fire	ground fire	0.5	0.6

selection however is not a trivial task, the example would become too big if all combinations are shown.

Let's assume that the desired difficulty for each skill is 0.4, that the user model influence  $w_u = 0,8$ , the agent influence  $w_a = 0,2$ . Given these assumptions the best valid combination for the trainee is: Fire agent: ID1, Fireman agent1: ID4, Fireman agent2 : ID3 and victim agent ID1. Using equation 1 the corresponding result is  $\approx 0.02917$ .

The agents get a response from the adaptation engine to perform the selected plan variations. The agents always comply with these requests and should therefore never propose offers they do not want to execute.

After the fireman agent2 completes the task of opening the door he will send a new proposal to the adaptation engine. The adaptation engine assumes that the other agents continue with their tasks and selects an action from fire agent2 that best fits the requirements for the trainee given the difficulties and limitations given by the actions from the other agents. The proposed actions and preferences of fireman agent2 have changed because of internal changes and changes in the scenario. It would make no sense to propose to open the door because the door is already open. This selecting, matching and executing behavior is repeated throughout the course of the game, guiding the adaptation of the agents.

## 5 Conclusions

In this paper we discussed online adaptation in serious games. The adaptation is based on the use of learning agents. In order to coordinate the adaptation of the agents we use an organizational framework that specifies the boundaries of the adaptation in each context. We have shown how the game adaptation model, meets the requirements posed on adaptation of the game. I.e. it is done on-line, takes care of a natural flow of the game and optimizes the learning curve of the user. In order to fulfill the requirements the game adaptation model uses

a user model, the preferences of the agents and uses the guidelines from the organization model.

We argue that an agent based approach for adapting complex tasks is more practical than a centralized approach. It is much more natural when the different elements are implemented by separate software agents that are responsible for their own believability. For complex games where characters play roles over extended periods of time this increases the believability of the whole game. However, the system not only needs to be flexible, the designer also must be able to define the storyline and put certain restrictions on the combined behavior of the agents. The combination of using a game model to specify the desired game progress and the adaptation engine to coordinate, we have the ability to guide the behavior in the desired direction.

The proposed model for game adaptation selects tasks that are most suitable for the trainee while following the specification of the game model and taking the preferences of the separate agents into account. The combination of adaptations selected at each moment is done through a kind of auction mechanism that provides a balance between local optimization of the task and believability of the agent and overall difficulty of the situation for the trainee. We have shown using a small example how the system works in practice. A framework for reasoning agents that are able to select different variations of subtasks, the learning 2APL agents, is also presented.

## References

1. Andrade, G., Ramalho, G., Gomes, A.S., Corruble, V.: Dynamic game balancing: An evaluation of user satisfaction. In: Laird, J.E., Schaeffer, J. (eds.) *AIIDE*, pp. 3–8. AAAI Press, Menlo Park (2006)
2. Andrade, G., Ramalho, G., Santana, H., Corruble, V.: Extending Reinforcement Learning to Provide Dynamic Game Balancing. In: *Reasoning, Representation, and Learning in Computer Games* (2005)
3. Beal, C., Beck, J., Westbrook, D., Atkin, M., Cohen, P.: Intelligent modeling of the user in interactive entertainment. In: *AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, Stanford, CA (2002)
4. Brusk, J., Lager, T., Hjalmarsson, A., Wik, P.: Deal: dialogue management in sxml for believable game characters. In: *Future Play 2007: Proceedings of the 2007 conference on Future Play*, pp. 137–144. ACM, New York (2007)
5. Chen, J.: Flow in games. *Communications of the ACM* 50(4), 31–34 (2007)
6. Dastani, M.: 2APL: A practical agent programming language. *Autonomous Agents and Multi-Agent Systems* 16, 214–248 (2008)
7. Dignum, V.: A Model for Organizational Interaction: based on Agents, founded in Logic. SIKS Dissertation, series (2004)
8. Hübner, J.F., Sichman, J.S., Boissier, O.: S-moise<sup>+</sup>: A middleware for developing organised multi-agent systems, pp. 64–78 (2005)
9. Hunicke, R., Chapman, V.: AI for Dynamic Difficulty Adjustment in Games. In: *Proceedings of the Challenges in Game AI Workshop, Nineteenth National Conference on Artificial Intelligence (AAAI 2004)* (2004)
10. Hunicke, R., Chapman, V.: AI for dynamic difficulty adjustment in games. In: *Challenges in Game Artificial Intelligence AAAI Workshop*, pp. 91–96 (2004)

11. Kok, E.: Adaptive reinforcement learning agents in RTS games. Technical Report INF/SCR-2007-073, Master thesis, Utrecht University (2007)
12. Lees, M., Logan, B., Theodoropoulos, G.: Agents, games and HLA. *Simulation Modelling Practice and Theory* 14(6), 752–767 (2006)
13. Magerko, B., Laird, J., Assanie, M., Kerfoot, A., Stokes, D.: AI characters and directors for interactive computer games. *Ann Arbor* 1001, 48109–2110
14. Moffat, D.: Personality parameters and programs. In: Petta, P., Trappl, R. (eds.) *Creating Personalities for Synthetic Actors*. LNCS, vol. 1195, pp. 120–165. Springer, Heidelberg (1997)
15. Okouya, D., Dignum, V.: OperettA: a prototype tool for the design, analysis and development of multi-agent organizations. In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Demo Papers*, Systems Richland, SC, pp. 1677–1678 (2008)
16. Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems* 11(3), 387–434 (2005)
17. Rabin, S.: *AI Game Programming Wisdom*. Charles River Media, Hingham (2002)
18. Sandholm, T.: Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence* 135(1-2), 1–54 (2002)
19. Schurr, N., Marecki, J., Lewis, J.P., Tambe, M., Scerri, P.: The DEFACTO system: Training tool for incident commanders. In: Veloso, M.M., Kambhampati, S. (eds.) *AAAI*, pp. 1555–1562. AAAI Press / The MIT Press (2005)
20. Silverman, B., Bharathy, G., O'Brien, K., Cornwell, J.: Human behavior models for agents in simulators and games: part II: gamebot engineering with PMFserv. *Presence: Teleoperators and Virtual Environments* 15(2), 163–185 (2006)
21. Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I., Postma, E.: Adaptive game AI with dynamic scripting (2006)
22. Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I., Postma, E.: Adaptive game AI with dynamic scripting. *Machine Learning* 63(3), 217–248 (2006)
23. Westra, J., Dignum, F., Dignum, V.: Modeling agent adaptation in games. In: *Proceedings of OAMAS 2008* (2008)
24. Westra, J., van Hasselt, H., Dignum, F., Dignum, V.: Adaptive Serious Games Using Agent Organizations. In: Dignum, F., Bradshaw, J., Silverman, B., van Doesburg, W. (eds.) *Agents for Games and Simulations*. LNCS, vol. 5920, pp. 206–220. Springer, Heidelberg (2009)

# Using Agent Technology to Build a Real-World Training Application

Michal Cap, Annerieke Heuvelink, Karel van den Bosch, and Willem van Doesburg

TNO Defense, Security, and Safety  
P.O. Box 23, 3769 ZG Soesterberg, The Netherlands  
Tel.: +31 346 356 430; Fax: +31 346 353 977  
{michal.cap, annerieke.heuvelink, karel.vandenbosch}@tno.nl,  
willem.vandoesburg@tno.nl

**Abstract.** Using staff personnel for playing roles in simulation-based training (e.g. team mates, adversaries) elevates costs, and imposes organizational constraints on delivery of training. One solution to this problem is to use intelligent software agents that play the required roles autonomously. BDI modeling is considered fruitful for developing such agents, but have been investigated typically in toy-worlds only. We present the use of BDI agents in training a complex real-world task: on-board fire fighting. In a desktop simulation, the trainee controls the virtual character of the commanding officer. BDI-agents are developed to generate the behavior of all other officers involved. Additionally, agents are implemented to manage the information flow between the agents and the simulation, to control the scenario, and to tutor the trainee. In this paper we describe the design of the application, the functional and technical requirements, and our experiences during implementation.

**Categories and Subject Descriptors:** I.2.0 [Artificial Intelligence]: General – *Cognitive simulation*; I.2.1 [Artificial Intelligence]: Applications and Expert Systems; I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence – *Intelligent agents, Multiagent systems*; I.6.3. [Simulation and Modeling]: applications; J.7. [Computers in Other Systems]: Military.

**General Terms:** Design, Human Factors.

**Keywords:** Intelligent Agents, Virtual Training, Multi-Agent System, BDI, Jadex.

## 1 Introduction

Scenario-based simulator training is considered very appropriate for learning decision making in complex environments [1]. A simulation enables trainees to experience the causal relations between actions, events and outcomes in the simulated task environment. It is common practice in simulation-training that Subject Matter Experts (SMEs) (usually staff members) play the role of key players. However, the need for

SMEs elevates costs of training, and staff tends to be scarcely available. As a result, there are often too few opportunities for training to achieve the required level of competence. Organizations are therefore looking for more flexible forms of simulation-training that require fewer organizational and logistic efforts. One solution to reduce the need for staff is to use intelligent software to play the required roles.

Finite State Machines (FSM) are the traditional approach for developing characters acting as key players. In this approach, characters are controlled by defining list of rules and contingencies. FSM has proven its value for designing autonomous characters performing procedural and other constrained tasks. However, for 'open' tasks like tactical decision making, it is hard or even impossible to create a 'spanning set' decision matrix specifying appropriate behavior for all entities for all possible states that may occur during a scenario [2], as in even relatively simple scenarios the number of states tends to be very high [3]. A more promising approach is to develop agents whose behavior is a function of Beliefs, Desires, and Intentions (BDI) [4]. A BDI model represents the knowledge (both generic and situational) and the reasoning processes of an individual or entity in a certain domain, task or scenario. There is a growing conviction that BDI modeling is promising for developing agents being able to handle complex tasks [5]. A considerable amount of research effort has been spent on designing frameworks, tools and specialized programming languages that facilitate implementation of intelligent agents based on BDI [6]. However, application of BDI agents have so far typically been demonstrated in toy-world environments [7,8,9].

In this paper we present the design, development, and implementation of a scenario-based simulator training of a complex real-world task in which BDI agents act as virtual team members. Our aim here is to describe the design of the application, to account for our choices to fulfill the identified functional and technical requirements, and to report our experiences during implementation. In the next section we will briefly introduce the domain and the purpose of the training application.

## 2 Training Application

For the Netherlands Defence Organization we developed an agent-based desktop training simulation. The domain is on-board fire fighting, and the task to be trained is that of the commanding officer, the Officer of the Watch (OW). If a fire breaks out, the OW is in charge of handling the incident. Operating in the Machinery Control Room (MCR) of the ship, he contacts his team, develops a plan to contend the incident, gives orders, monitors the events, and adjusts plans if necessary. The OW communicates with four other officers: Chief of the Watch (CW), MCR-Operator1 (MCRO1), Leader Containment Party (LCP), and the Leader Main Party (LMP). The first two are for the greatest part also situated in the MCR, the last two are at or near the incident scene.

The task of the OW is a typical example of decision making in a complex environment. There are, of course, procedures for handling a fire accident. However, the OW also has to anticipate on possible complications, needs to respond to unforeseen actions, has to adjust plans when events require him to do so, and so on. In the same manner, his team members need to be able to react to unforeseen events and new actions of the OW.

In the desktop-simulation training, the trainee controls the virtual character of the OW. For an impression of the training simulation, see Figure 1. In addition to the OW, the aforementioned team members are also represented as virtual characters in the simulation. In order to achieve good training, the trainee must be able to practice the task in accordance with the way an OW does this in the MCR. This means, for instance, that all relevant information and communication equipment needs to be simulated, and team members needs to respond in a realistic fashion to the trainee and to each other. However, the application is not mere simulation, it concerns a *training* simulation. This implies that the scenario needs to be managed in order to make sure that the emerging situations support achievement of the learning objectives. Furthermore, the quality of the trainee's performance must be evaluated and appropriate feedback needs to be given. In the next section we elaborate on the functional requirements to ensure successful desktop-simulation training.



Fig. 1. Impression of the desktop simulation training

### 3 Functional Requirements

In previous research we identified several functional requirements the training simulation needs to adhere to [10, 11]. In this section we describe and extend these functional requirements. Some requirements follow from choices concerning the scope of the training, e.g., only the OW is trained, others from the motivation to implement a full-fledged stand-alone training system, which includes believable behavior of virtual team members and a tutoring functionality.

1. To ensure transfer of training, the trainee's task environment needs to be simulated in a functionally realistic way. During fire fighting, the Officer of the Watch (OW) is

situated in the machinery control room (MCR) of the ship, while several team members enter and leave the MCR. So in the training system several virtual characters representing the trainee and the team members are required, as well as a realistic simulation of the MCR.

2. A great part of the OW training concerns the interaction with team members. To train this realistically, it is required that the OW can interact with the other virtual characters with some degrees of freedom. On the other hand, to keep the training scenario under control, communication should be regulated.

3. The behavior of the virtual characters has to represent that of task experts, which entails that they should be able to autonomously perform their task. Moreover, they should be able to reason about their task and, e.g., propose an alternative plan if the trainee proposes a wrong one.

4. To execute their task, the virtual characters need to have knowledge and reason about what has happened in the past. E.g. the Chief of the Watch (CW) should decide to plot the fire location on the damage control board if the fire broke out more than 30 seconds ago and if it has not yet been plotted by anybody else.

5. Some team members need to perform tasks in other parts of the ship than the MCR. In addition, the trainee needs to receive information coming from outside the MCR. So besides the visualized model of the MCR, we need a functionality that models events taking place in the rest of the ship (which does not need to be visualized).

6. The team members that enter and leave the MCR during fire-fighting will execute tasks in the MCR, as well as on the rest of the ship. For them it should not make a difference whether their task environment is visualized (the MCR) or modeled by another functionality (the rest of the ship, see 5).

7. To ensure believable behavior, what a team member can perceive should be determined by its location. For example, initially only team members present in the MCR should know the status of the incident, based on the damage-control board. However, once the status has been broadcasted by the OW, everybody knows it, because broadcasts are audible at each location.

8. A stand-alone training system requires a tutoring functionality. For tutoring it is foremost required to evaluate the trainee's behavior. For this the system should be able to compare trainee actions with a specification of expert actions. Moreover, this comparison should lead to an appropriate tutoring action.

9. Because goal-directed, systematic training is more effective than learning by doing, we want to implement a scenario-based training system [12]. In order to control the training scenario, we need a functionality that (or inhibits) events at appropriate times during the scenario.

10. To function coherently, the virtual characters and other functionalities of the training system require up to date information about the state of the simulated environment and the actions of the trainee. Unfortunately, a simulated environment will generate so much data that, if unmanaged, this information exchange will slow down the training system. We therefore need a functionality to channel this data, ensuring that only relevant information is transferred and without significant delays.

11. Our current focus is the development of a prototype training system, with the intention to develop it further for operational use in naval staff training. We therefore require our training system to be based on a stable software platform.



12. Future operational use of the program requires that the current prototype containing one example scenario should be easily extendable with new scenarios. This makes it necessary to use a programming environment that is transparent, expressive, and easy to work with.

## 4 Technical Solutions

In this section we shortly discuss the technical solutions we selected for our training system for each of the functional requirements introduced above.

1. To develop a visually realistic representation of the environment, we decided for a 3D representation of the MCR generated by a 3D game engine developed by our industrial partner “VSTEP”<sup>1</sup>. All equipment that is normally used by the OW is simulated and available to the trainee (damage control board, information panels, communication equipment, etc). In addition five virtual characters are modeled, one representing the OW, the others representing his team members.

2. In reality, team members communicate by speech. However, using speech in this training application would introduce at least two major problems. First, speech recognition technology is not yet advanced enough to recognize and interpret spoken messages, certainly if the syntax and vocabulary is unrestricted and the system is untrained to pronunciation characteristics of the speaker. A second problem is that natural speech would introduce so much freedom that it would be very difficult to control the scenario. As a solution, we decided to allow the trainee to communicate with its team members using pre-established context-sensitive menus that are dynamically filled with communication acts based on the current state of the training simulation, see Figure 1. All the OW’s possible communication acts, as those of his team members, are pre-recorded using a speech synthesizer. For an extensive discussion on the ontology-driven dialog system underlying this feature of our training system, see [13].

3. To develop virtual characters that act as experts, domain knowledge is required. Because experts tend to explain their actions in terms of beliefs, goals and intentions, expert knowledge can be easily translated to a BDI model [14]. It has been demonstrated that software agents based on the BDI-paradigm can provide virtual characters with believable behavior in computer games [15], and in virtual training [16]. BDI agents commonly incorporate a plan-base that embeds the information on how to reach specific goals. Decision making in fire fighting is often procedural in nature: plans for achieving goals under given conditions are thus available. Because of all these benefits, we decided to implement software agents based on the BDI paradigm to control the non-player virtual characters. We refer to these agents as the *role-playing agents*.

4. Due to the notion of beliefs, the agents have knowledge about the current situation. However, having beliefs does not necessarily entail that the agent can remember what was previously the case. To allow for temporal reasoning, we decided to store time annotated beliefs about every change of the world state into the agents’ historical belief bases.

5. In order to model and simulate parts of the ship that are not visualized in the 3D simulation, we introduce a **world manager agent**, which is one of the implemented

---

<sup>1</sup> <http://www.vstep.nl>

*functional agents* as opposed to the *role-playing agents*. In contrast to a role-playing agent, a functional agent does not represent an individual in the scenario, but instead fulfills one or more functions on the background needed to manage the simulation-based training. The world manager agent does not only simulate events outside the MCR, it also simulates the actions (with time and effects) of role-playing agents not present in the MCR.

6. To ensure that the role-playing agents can interact in the same way with the visualized (MCR) as with the non-visualized part of the simulated environment (rest of the ship), we decided to channel all actions and observations through the **world manager agent**. The fact that this functional agent maintains the complete world state of the entire simulation also ensures that no inconsistencies arise between the visualized and non-visualized part.

7. Because the world manager agent is informed about the exact state of the world, it knows the location of each role-playing agent, and thus what that team member can see and hear. Below (see 10) we explain how this information is used to channel the information flow between the role-playing agents and the 3D simulation.

8. In order to keep track of the actions that the trainee executes in the simulated environment, to evaluate the trainee's performance, and to decide upon appropriate interventions (if any), we introduce an **OW/Tutor agent**. This agent is interesting in the sense that it is a functional agent, embedding a role-playing agent. The BDI task knowledge underlying the actions of role-playing agents can be reused in the tutor agent to evaluate behavior of the trainee. If a trainee's action coheres with the task model (in this case of the OW), it is evaluated as correct. If the trainee fails to do what is required according to the task model (e.g. "hail the fire within 30 seconds after a positive fire alarm"), the OW/Tutor Agent interprets this as an error (constraint broken). In addition, the OW/Tutor agent has a function deciding whether and which intervention to take (e.g. letting the Chief of the Watch give a reminder).

9. For starting the training scenario and keeping it on track by triggering events in the simulated environment we use a **scenario manager agent**. This agent triggers events based on a pre-defined scenario model (e.g. a fire of type A starts at location X). By separating this knowledge in a specific functional agent, we intend to facilitate the future extension of the training simulation with additional scenarios (see 12).

10. In order to keep the data flow between agents and the 3D simulation manageable, we decided that agents only receive information that is relevant to them. We achieve this by using a publish/subscribe paradigm in which information producers and consumers coordinate among each other what information to exchange. We introduce a functional agent called the **broker** to carry out this task. Publish/subscribe mechanisms are provided by common middleware standards like the Data Distribution Service (DDS), Java Messaging Service (JMS) and High Level Architecture (HLA). However, these mechanisms define static contracts, whereas our agents' information need is dynamic (e.g. information exchange is, for instance, dependent upon the agent's location in the ship). We introduce a dynamic filtering system that adjusts the subscriptions of the agent to information supplied by the simulation in such a fashion that it corresponds to its specific situation or context (the knowledge for this is provided by the world manager agent, see 7). In this way, the agent only receives information that it potentially can receive.

11,12. To be able to implement the system cost-efficiently, we looked for an existing BDI-based agent platform. Since the prototype under development will at a later stage be further developed for operational use in naval staff training, the stability and performance of the selected platform is of major importance. We have considered three agent platforms to implement the role-playing and functional agents: JACK, Jadex, and 2APL.

**JACK** is a mature, proven agent platform providing commercial support. However, we experienced the provided development environment as rather unintuitive and the predefined workflow too inflexible for our needs.

**Jadex** is an academic project aiming to develop an industrial strength agent platform. It has been shown to be applicable in a real-world medical planning application [17], which made us believe that the performance of the system would be sufficient for our system. Agents in Jadex are specified using a relatively complex and verbose XML-based language. Since agents in Jadex can manipulate arbitrary Java objects and the platform is fully open-source, we considered the provided framework extensible enough to allow implementation of our specific needs.

**2APL** is an academic, research-oriented agent programming language adhering more than the other platforms to the theoretical principles of BDI. However, the current implementation has not been designed for computational efficiency and it has not been tested yet in operational contexts.

The main criteria for selecting an agent platform were the stability of the platform, its performance, and the ability to customize the provided framework. In particular, we required the platform to support incorporation of historical belief bases and a publish/subscribe mechanism for messaging. We decided to drop 2APL because it does not provide production-level stability and performance. Next, after comparing JACK and Jadex, we decided to choose Jadex because of its extensible design and its open source code, as this would allow us to attach our own functionalities.

## 5 System Architecture

Following the functional requirements and technical solutions stated above, we implemented a training system that consists of two major parts: 1) a 3D visualization that serves as an interface to the simulation for the trainee, and 2) an agent system accommodating a) role-playing agents that model behavior of the virtual characters and b) functional agents that manage the simulation, the execution of the scenario, and tutoring. For an overview of the training system, see Figure 2. In this section we will elaborate on the functionality of each part of the training system.

### 3D Visualization

Once a training session starts, the trainee can control his virtual character in the 3D game environment using the keyboard and mouse (section 4.1). Each interaction with the simulated environment (such as: picking up the headset, leaving the MCR, talking to another character) generates a *cue* (a data structure that describes the change of the world state), which is then relayed to the agent system. Conversely, the agent system can send a variety of commands to the 3D engine in order to force some of the entities

to perform a certain action. E.g. the CW role-playing agent may order the virtual character of CW to sit on his chair; the virtual character immediately starts moving towards the chair and once there, he plays the animation of the sitting move; after the successful completion of the action, the system generates a cue "the chair of CW is taken", which is then forwarded to all the virtual characters (and subsequently their attached role-playing agents) currently present in the room. Further, the 3D engine provides a dialog interface that allows a trainee to communicate with his team members (section 4.2). Using the interface, the trainee can choose from a wide variety of sentences that can be communicated by clicking on one of them; the engine plays the corresponding pre-recorded speech and generates a communication cue which is subsequently forwarded to the interlocutor. Since all communication between two virtual characters should be audible by the trainee (if he is in the same room), the role-playing agents communicate through their virtual characters in the simulation. After the appropriate speech file is played, a communication cue is relayed to the corresponding receiver and other characters that may have overheard the communication (section 4.7).

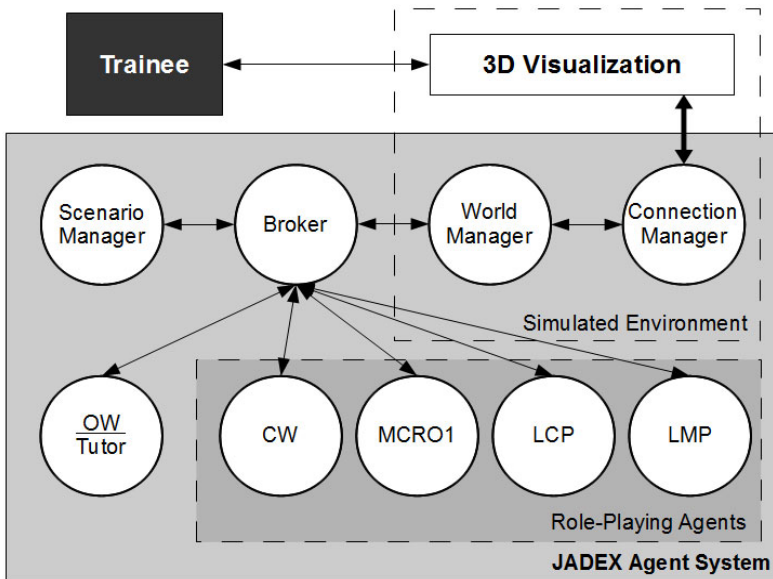


Fig. 2. Diagram of the system architecture

### Connection Manager – *functional agent*

When the **Connection Manager** is started it establishes a connection to the 3D visualization and starts all the other Jadex agents. The connection manager serves as the gateway between the 3D visualization engine and the agent system. Being the connection gateway between the simulation and the agent system the connection manager has one additional task: translating the cues originating from the 3D visualization into

a format understandable to the agent system, and vice versa, translating the actions coming from the agent system into the language of the 3D visualization. These translations are made based on a dedicated mapping of the ontology used within the 3D simulation to the ontology used within the agent system.

### **World Manager – *functional agent***

Since the virtual environment is divided into a visualized part (MCR), which is simulated within the 3D visualization, and a non-visualized part (the rest of the ship), the agent system contains a **World Manager** whose responsibility is to simulate the non-visualized part and to control the blending of both (section 4.5). As mentioned in section 4.6, the role-playing agents should not notice any difference between their interaction with the visualized, and non-visualized environment. Therefore all the cues from the visualized environment, denoting changes in the world state, are first send to the world manager that uses them to update its belief base containing the overall world state, ensuring that no inconsistencies exist. Next, these belief updates are forwarded to the rest of the agent system. Similarly, all role-playing agent actions are first send to the world manager that assesses whether or not they take entirely place in the MCR. If this is the case, the action is forwarded to the visualized environment to be executed, resulting in cues from this environment to the world manager, who will forward them to the appropriate agents. If an action takes (partly) place in the non-visualized environment, the world manager executes the action and generates (and forwards) the cues denoting the effects of the action.

To illustrate this function, consider the following example: During fire fighting, the MCR operator (MCRO1) is ordered to check the smoke valves. In reality this involves walking to a location near the fire, checking visually the status of the valves, then returning to the MCR. In our system, this action requires blending of the visualized and non-visualized world and is therefore executed by the world manager. The world manager first instructs the virtual character of the MCRO1 to walk out of the MCR (visually observable to those inside the MCR), and then lets the MCRO1 “wait” around the corner (not observable to those inside the MCR). Based on the real walking distance to the fire location, the world manager calculates how long the MCRO1 needs to wait outside the MCR before it can generate the cue denoting the result of the visual inspection, forwards this cue to the MCRO1, and instructs the MCRO1 to return into the MCR.

### **CW, MCRO1, ATL, CTL – *role-playing agents***

The four **Role-playing agents** (CW, MCRO1, LCP, LMP) represent the team members of the OW. The models underlying their behavior consist of expert task knowledge in the form of beliefs, goals and plans (section 4.3). There may be several reasons for a role-playing agent to initiate action. It may originate from his own task knowledge, for example if circumstances in the world form beliefs that result in a goal to become active, after which the agent selects a plan (series of actions) to achieve the active goal. It may also originate from a request for information, or an order, given by the trainee or other role-playing agents. Sometimes beliefs may trigger more than one goal. The role-playing agent then has to decide which goal to pursue based upon priority rules. The

agents store all messages concerning changes of the world state (received from the world manager) in their historical belief base (section 4.4). In our application, role-playing agents do not act directly in the simulated world. They do so by sending action messages via the broker to the world manager, who controls their virtual character accordingly. Because the models of role-playing agents are based on beliefs and goals, they are able to respond to questions from the trainee by inspecting their belief base, and can adopt any order as a goal.

### **OW/Tutor – functional agent**

The **OW/Tutor** agent embeds two models: one model is an expert model of the OW, the second model is a user model of the trainee. The expert task model is static and established before the training. The trainee user model is dynamic, and formed during training. For example, when the trainee asks about the state of the ventilation and receives an answer, the OW agent stores this action and the received information in its belief base, building up the user model. This information about the user, in combination with the task knowledge embedded in the OW agent, can be used to generate an evaluation of the performance of the trainee (section 4.8). Based on this evaluation, the agent can decide to adjust the scenario in order to achieve more efficient training. This can be done indirectly by ordering the role-playing agents to change their behavior.

### **Broker – functional agent**

The **Broker** represents the pivot of the publish-subscribe mechanism described in section 4.10. It contains a database of subscriptions determining which messages will be forwarded to each of the agents. It is the responsibility of the world manager agent to maintain the subscriptions for each agent, ensuring they are consistent with the constraints imposed by the world on what an agent can possibly sense. Consider the following example as an illustration of this mechanism: When the CW enters the MCR, the world manager subscribes him to all the beliefs related to the entities located in the MCR (status of a phone, availability of a chair etc.) Through the newly established subscriptions, the CW receives messages about any change in MCR. Analogically, as soon as the CW leaves the MCR, the world manager breaks all those subscriptions, after which the CW will not be informed anymore about changes in the MCR.

### **Scenario Manager – functional agent**

The **Scenario Manager** agent controls the course of events in the training simulation using a pre-specified scenario definition (section 4.9). The latter denotes, e.g., where a fire should start, what type of fire it is, and whether the ventilation of the ship will, or will not crash stop. Based on the scenario definition and generic knowledge about the environment stored in its own task model, e.g. about how the fire will react to certain events, the scenario manager agent will at appropriate times during training execute specific events like starting a specific type of fire at a specific location, expanding that fire, and crash stopping the ventilation.

## 6 Implementation

As mentioned in Section 4, we decided to implement the agent-system outlined above using the Jadex platform [9]. Due to space limitations, we will not discuss the implementation in full detail. Instead, we present a few examples of representative issues we had to deal with, and reflect on the suitability of features available in Jadex for implementing our agent-system.

### Defining an Ontology and Fixing the Message Format

After analyzing the domain, scenario and tasks, we found that we need about 40 types of beliefs to describe the states of the simulation environment (e.g. location of an agent, status of the fire siren etc.). Each belief has about 2 - 3 parameters. Similarly, we identified 52 actions required for the role-playing agents to perform their task. It is apparent that we need an appropriate ontology to handle the identified actions and belief types efficiently.

The Jadex platform supports use of ontologies created in the Protégé environment through the provided Beanalyzer plugin-in, which is able to generate JavaBeans for the modeled ontology. Consequently, each JavaBean may be serialized to XML and can easily be transferred between agents as the content of a FIPA message.

Although the provided framework is very powerful, we considered this approach as too heavy-weight for our needs and decided to define our ontology as a collection of classes that provide static constants representing the domain concepts. That way, we can enjoy facilities such as code auto-completion available in the modern Java IDEs. Moreover, most of the syntax errors are discovered already by the Java editor and other problems concerning incompatible use of beliefs or actions are reported as soon as the agent code is compiled.

Further, we feared that the XML representation of beliefs might become unpractical when one needs to debug the multi-agent system. Therefore, we have developed a more space-economic serialization format, which is able to carry all the information the agents need to exchange, namely belief updates, actions, behavioral instructions<sup>2</sup> and broker messages<sup>3</sup>. To illustrate the format we used: suppose the CW agent decides to check the status of the fire fighting pumps. Using the methods available in our ontology, he can create an object that represents the action, which may be serialized as follows:

```
(messagetype:action) (source:cw) (type:mcrpanelaction) (id
:checkpumps) (timeout:0) (location:panel_cw) (actiontype:c
heckpumps)
```

This representation proved practical for debugging purposes as one message usually fits to one line in a debugging output. Besides, we designed the message format

---

<sup>2</sup> A behavioral instruction is send by the Tutor agent to modify the behavior of a role-playing agent. The goal of intervening in the agent's goal state is to bring about behavior that helps the trainee in achieving a learning objective.

<sup>3</sup> Broker messages are used to indicate that a) an agent is capable to generate a certain message b) an agent is interested in messages of a certain type c) the broker has established a subscription between an information producer and an information consumer.

almost identical to the structure of cues and actions used within the 3D visualization engine, which makes the conversion of data exchanged between the two systems fast and straightforward.

Unfortunately, this simplified message format revealed its limitations in later stages of the project development. In particular, we found it problematic that the format is unable to naturally express nested data structures, such as a query about a belief, which forced us to come up with rather complicated workarounds. We now recognize that the built-in ontology support, most likely, would have served us better.

### **Historical Belief Base**

The domain of fire-fighting requires agents to reason about time in relation to actions and beliefs. For example, if the fire alarm has been active for more than 30 seconds and there is still no attack plan plotted on the damage control board, the tutor agent can decide to remind the trainee of his duties by sending a behavioral instruction to the CW to suggest to the trainee to plot an attack plan. In order to allow for such reasoning over states in time, the agents requires access to a history of beliefs (when was it first believed that there was a fire alarm?) and actions (is there not attack plan plotted since that time?). In addition, a history of the beliefs and actions of each agent can be used to off-line evaluate the trainee's performance.

To our knowledge, Jadex does not offer support for temporal reasoning. In fact none of the considered BDI-platforms do. However, Jadex allows to add a time parameter to each belief and keep all adopted beliefs in a dedicated set. That way an agent has access to the complete history of his beliefs and is capable of deriving time-related conclusions.

A historical belief base could be implemented in two ways, a) by employing the belief set concept available in Jadex or b) as a single Jadex belief that encapsulates (since it is a Java object) all the desired functionality. Since we considered it practical to be able to define custom methods to search and update the beliefs stored in the belief base, we chose for the latter solution. We found that the custom belief base specified in the above mentioned way enabled us to access the historical beliefs both from conditions in the agent definition files and from Java plans in a natural and effortless way.

### **Logical Inferences**

To perform their role as task experts, the role-playing agents need to be able to make logical inferences. For example, if an officer hears that the siren goes off, he needs to be able to derive that there is a fire alarm coming from an unknown source.

The ability to specify implication rules as a part of the belief base of an agent is one of the fundamental features of logic-based agent-oriented languages. However, since the belief base in Jadex is not logic-based, but specified as a collection of Java objects, it does not provide support for logical inferences.

To overcome this limitation, we have implemented a simple custom inference engine, which is executed each time a new belief is added or removed from an agent's



belief base. The inference engine is implemented as a Java class with a designated method that contains hard-coded implication rules that generally follow this structure:

```
if <query on belief update>/<query on belief base>
then <update the belief base>/<adopt a goal>
```

Besides its original use to deduce new beliefs from existing beliefs, we found that this system was also useful for generating reactive behavior (i.e. when an incoming belief update makes the agent adopt a new goal) and for decomposing complex beliefs (e.g., when a new attack plan is received, atomic parts of the plan are also added to the belief base). This turned out to be a robust approach: although our inference engine contains tens of inference rules, it did not noticeably decrease system performance.

## XML vs. Java

One of the distinguishing features of Jadex is its ability to program agents in both XML and Java. In fact, a programmer may decide whether he prefers to specify the major part of his agent declaratively in an XML-based agent definition file (ADF) or in a more imperative way using Java plans. However, from a theoretical point of view, it is more natural to program agents declaratively [18].

To explain the difference between using XML and Java to implement certain task behavior, consider the following example of desired role-playing agent behavior: When the siren goes off, the MCR operator must check whether the ventilation has been automatically crash stopped, and inform the CW about the result of this check. Only after that, he can use his work station to check the location of the fire alarm, and turn off the siren.

In this example we began by implementing this behavior in the XML-based agent definition file. For this, we first modeled that the MCR operator agent adopts the goal "deal\_with\_siren" as soon as it believes that the siren is active. When a goal becomes active, the agent constantly tries to execute all the associated plans. In the case of the "deal\_with\_siren" goal, multiple plans are associated: one for checking the status of the ventilation, one for telling this status to the CW, another for checking the location of the fire alarm, and yet another for turning the siren off. To ensure the correct execution of his task, the preconditions of these plans needed to be specified in such a way that in fact, they will be performed sequentially. We found that this led to very complex preconditions, e.g.:

```
<precondition>
  <!-- MCR01 knows the status of the ventilation. -->
  $beliefbase.worldbeliefs.contains(Ventilation
    .getFuncutor())
  <!-- It has not been communicated yet. -->
  $beliefbase.communicationbeliefs
    .getLastInformPattern($beliefbase.id, Officers.CW,
      null, Ventilation.getFuncutor()) == null
</precondition>
```

Next we implemented the identical behavior in a single Java plan that existed of a number of if-clauses that ensured that the "deal\_with\_siren" subtasks would be executed in the correct order. Based on this and further experiences we noticed that, although the XML-based approach fits the agent-orientation notion better, our

programming productivity significantly increased when employing the Java-based approach. This was mainly due to the availability of sophisticated productivity features for Java code such as code hints and instant source validation. None of these features are available to support the development of Java-like conditions in Jadex ADF files. Also the debugging of the Java code proved to be easier, as the error messages generated by the Jadex XML parser often provided insufficient details to identify the problem. For the above mentioned reasons, we found it easier to specify most of the agent's logic in the Java plans.

### **Jadex v. 1 vs. Jadex v. 2**

When we started the implementation of our agent-based training system (May 2009), the Jadex agent platform was available in two versions: a) Jadex v. 0.96 b) Jadex v. 2-beta2. As the latter version has undergone a complete re-design, which is expected to bring significant improvements in performance and flexibility [19], we aimed to base our system on Jadex v. 2. However, the problems stemming from the lack of documentation and instability of the platform we were facing in the first days of development made us switch to Jadex v. 0.96. This version has proven to be relatively well-documented and reasonable stable.

## **7 Conclusion**

In this paper we have reported our efforts to develop a stand-alone training system for a complex real-world task based on software agents. We have adopted a BDI-approach to develop role-playing agents that can act autonomously and intelligently, and selected the Jadex agent platform for implementation.

It took us about 80 man-days of implementation to build the agent system for the desktop simulation for training on-board fire-fighting. The agent system consists of 9 agents, 359 Java classes and 11 ADF files defined by 17 thousands lines of Java code and 2.5 thousands lines of XML code. The system satisfies our criterion that it should be able to react to events taking place in the simulation without any noticeable delays.

Although the coverage and quality of the programming and debugging tools bundled with Jadex are not comparable to mainstream development environments (e.g. Java + Eclipse), the Jadex agent framework proved to be a robust and well performing platform. We appreciated in particular that Jadex does not force programmers to work with the built-in notions of the BDI-components and allows users to extend or re-define them. For example, Jadex allowed us to bypass the platform's default belief base and switch to a custom-made historical belief base.

The work presented in this paper covers the implementation of the role-playing agents and their interaction with the 3D visualization. For the former the focus lay on the realization of valid expert behavior in a wide variety of emerging situations. For the latter we focused on an efficient, reusable manner to link BDI role-playing agents to a virtual simulation, for which we developed several specific functional agents.

Although we have far-stretching ideas on how to model the tutor agent conceptually [10, 11], we have as yet not been able to invest much efforts into implementation and integration of these ideas in the introduced OW/Tutor agent. Although all the

functionality is there for the OW/Tutor agent to record and reason about the behavior of the trainee, and to order other agents (role-playing ones, but also the scenario manager agent) to perform certain actions (which they can also already execute), we have not spend time on defining the rules to govern which actions should be ordered to support the trainee in reaching his training objectives. In future work we will extend the knowledge of the tutor agent with such rules of intervention to exploit the innovative opportunities offered by BDI agents to full extent: on-line control of agent behavior, ensuring it is supportive to the training goals. A short example may illustrate this point. Suppose that a tutor assesses that a trainee is not challenged to learn by the events when presented with a standard scenario. The tutor agent may decide to introduce events that do challenge the trainee in a useful way. He may for example send a behavioral instruction to the CW-agent prompting it to “forget” switching off the electricity at and near the incident scene. This forced event enables the trainee to achieve the learning objective of checking whether all safety precautions have been taken and to make corrections, if necessary. Such on-line control of autonomous training is the wish of many, and seem to come within reach by linking BDI agents to training simulations.

## Acknowledgement

This research project (032.13359) has been funded and supported by the Netherlands Department of Defense.

## References

1. Oser, R.L.: A structured approach for scenario-based training. In: Proceedings of the 43rd Annual meeting of the Human Factors and Ergonomics Soc., Houston, TX, pp. 1138–1142 (1999)
2. Silverman, B.A.: More realistic human behavior models for agents in virtual worlds: emotion, stress and value ontologies (Report No. Technical Report). Univ. of Penn/ACASA, Philadelphia, PA (2001)
3. Klein, G.: The source of power: how people make decisions. MIT, Cambridge (1998)
4. Bratman, M.E.: Intentions, Plans, and Practical Reason. Harvard University Press, Cambridge (1987)
5. Pew, R.W., Mavor, A.S. (eds.): Modeling human and organizational behavior: Application to military simulations. National Academy Press, Washington (1998)
6. Georgeff, M.P., Pell, B., Pollack, M.E., Tambe, M., Wooldridge, M.: The Belief-Desire-Intention Model of Agency. In: Papadimitriou, C., Singh, M.P., Müller, J.P. (eds.) ATAL 1998. LNCS (LNAI), vol. 1555, pp. 1–10. Springer, Heidelberg (1999)
7. Astefanoaei, L., Mol, C.P., Sindlar, M.P., Tinnemeier, N.A.M.: Going for Gold with 2APL. In: Dastani, M.M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) PROMAS 2007. LNCS (LNAI), vol. 4908, pp. 246–250. Springer, Heidelberg (2008)
8. Bordini, R.H., Wooldridge, M., Hübner, J.F.: Programming multi-agent systems in agentspeak using Jason. Wiley Series in Agent Technology, p. 180. John Wiley & Sons, Chichester (2007)

9. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A BDI Reasoning Engine. In: Bordini, R., Dastani, M., Dix, J., Seghrouchni, A. (eds.) *Multi-Agent Programming*, pp. 149–174. Springer Science+Business Media Inc., USA (2005)
10. Bosch, K., Harbers, M., Heuvelink, A., Doesburg, W.A.: Intelligent Agents for Training On-Board Fire Fighting. In: Duffy, G. (ed.) *Proceedings of the 2nd international Conference on Digital Human Modeling: Held As Part of HCI international 2009*. LNCS, vol. 5620, pp. 463–472. Springer, Heidelberg (2009)
11. Heuvelink, A., van den Bosch, K., Doesburg, W.A., Harbers, M.: Intelligent Agent Supported Training in Virtual Simulations. In: *Proceedings of the NATO HFM-169 Workshop on Human Dimensions in Embedded Virtual Simulation*. NATO Human Factors and Medicine Panel, Orlando (2009)
12. Blackmon, M.H., Polson, P.G.: Combining Two Technologies to Improve Aviation Training Design. In: *Proceedings of Human Computer Interaction (HCI)*, pp. 24–29. AAAI Press, Menlo Park (2002)
13. Oijen, J., van Doesburg, W.A., Dignum, F.: Goal-based Communication using BDI Agents as Virtual Humans in Training: An Ontology Driven Dialogue System. In: Dignum, F. (ed.) *Agents for Games and Simulations II*. LNCS (LNAI), vol. 6525, pp. 38–52. Springer, Heidelberg (2011)
14. Norling, E.J.: Folk psychology for human modelling: Extending the BDI paradigm. In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 202–209. IEEE Computer Society, Washington (2004)
15. Norling, E.: Capturing the Quake Player: Using a BDI Agent to Model Human Behaviour. In: *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Melbourne, Australia, pp. 1080–1081 (2003)
16. van den Bosch, K., van Doesburg, W.A.: Training Tactical Decision Making Using Cognitive Models. In: *Proceedings of the Seventh International NDM Conference*, Amsterdam, The Netherlands (2005)
17. Paulussen, T.O., Zöller, A., Heinzl, A., Braubach, L., Pokahr, A., Lamersdorf, W.: Patient scheduling under uncertainty. In: *Proceedings of the ACM Symposium on Applied Computing (SAC 2004)*, pp. 309–310. ACM, New York (2004)
18. Shoham, Y.: Agent-oriented programming. *Artificial Intelligence* 60(1), 51–92 (1993)
19. Pokahr, A., Braubach, L.: From a Research to an Industrial-Strength Agent Platform: Jadex V2 in: 9. Internationale Tagung Wirtschaftsinformatik (2009)

# Semi-Automated Dialogue Act Classification for Situated Social Agents in Games

Jeff Orkin and Deb Roy

MIT Media Laboratory, 75 Amherst Street,  
Cambridge, MA, USA 02139  
{jorkin,dkroy}@media.mit.edu

**Abstract.** As a step toward simulating dynamic dialogue between agents and humans in virtual environments, we describe learning a model of social behavior composed of interleaved utterances and physical actions. In our model, utterances are abstracted as {speech act, propositional content, referent} triples. After training a classifier on 100 gameplay logs from *The Restaurant Game* annotated with dialogue act triples, we have automatically classified utterances in an additional 5,000 logs. A quantitative evaluation of statistical models learned from the gameplay logs demonstrates that semi-automatically classified dialogue acts yield significantly more predictive power than automatically clustered utterances, and serve as a better common currency for modeling interleaved actions and utterances.

## Categories and Subject Descriptors

I.2.7 [Artificial Intelligence]: Natural Language Processing – language parsing and understanding.

## General Terms

Measurement, Performance, Design, Reliability, Experimentation, Human Factors, Languages, Verification.

**Keywords:** Social simulation, Modeling natural language, Virtual Agents, Agents in games and virtual environments.

## 1 Introduction

While mature graphics hardware, rendering engines, physics simulators, and path planners have leveled the playing field for near-photorealistic visuals in video games and simulations, artificial intelligence methods for social planning, interaction, and communication are poised to take the lead as the differentiating feature in games of the future. Though much progress has been made in navigation and action selection, natural language communication between agents remains a difficult problem, and communication between agents and humans even more so. Dynamic interactive dialogue poses numerous technical challenges, yet also holds the key to enabling entirely new genres of games, and broadening the reach of games beyond entertainment into new forms of social simulation.



Fig. 1. Screenshot from *The Restaurant Game*

Current approaches to implementing natural language dialogue systems in use in the video game industry are labor intensive, requiring designers to anticipate human input and hand-author responses. As von Ahn has demonstrated by labeling images with *The ESP Game* [20], the dramatic increase in popularity of online games provides an opportunity to teach machines by observing human gameplay. Multiplayer role-playing games and virtual worlds provide the opportunity for a potentially better approach to developing systems that can understand and generate natural language dialogue, by mining the enormous amount of data generated by thousands (or even millions) of human-human interactions. For example, as of 2008 *World of Warcraft* had over 10 million paying subscribers, *Habbo Hotel* had 9.5 million unique monthly visitors, and *Second Life* had 600,000 unique monthly visitors [4]. Clearly there is an opportunity to collect rich new forms of behavioral data from these players. What is less clear is how to maximize the utility of this data for agents to exploit at runtime, while minimizing the human labor required to structure and annotate corpora.

We are working toward a long term goal of generating dialogue and behavior for agents based on data collected from human-human interactions. Our approach, influenced by Schank, is to represent context in the form of socio-cultural scripts [16]. Due to the technological limits of the 1970s, Schank's scripts were hand-crafted, and thus subject to limitations associated with human authoring. Hand-crafted scripts are brittle in the face of unanticipated behavior, and are unlikely to cover appropriate responses for the wide range of behaviors exhibited in an open ended scenario. Today, we have the opportunity to do better by *discovering* scripts from human-human interaction traces of online gameplay.

With these ideas in mind, we launched *The Restaurant Game* (<http://theRestaurantGame.net>) as a platform for collecting rich physical and linguistic

```

WAITRESS: "welcome to our fine restaurant"
CUSTOMER: "thanks, it's just me tonight"
WAITRESS: "would you like the seat by the window?"
CUSTOMER: "sounds good"
WAITRESS: "follow me"
CUSTOMER SITSON chair3(Chair)
WAITRESS: "perhaps i should start you off with some water"
CUSTOMER: "that sounds good, can i check out a menu?"
WAITRESS: "sure thing, coming right up"
WAITRESS PICKSUP dyn1733(Menu)
WAITRESS GIVES dyn1733(Menu) TO CUSTOMER
CUSTOMER LOOKSAT dyn1733(Menu)
WAITRESS: "water please"
dyn1741(Water) APPEARS ON bar(Bar)
WAITRESS PICKSUP dyn1741(Water)
WAITRESS PUTSDOWN dyn1741(Water) ON table1(Table)
WAITRESS: "here's your water"
WAITRESS: "i'll give you a minute to look over the menu"

```

**Fig. 2.** Transcript from a typical interaction in *The Restaurant Game*

interaction [14]. To date we have collected over 9,400 game logs. The ultimate goal is to replace one of the players of *The Restaurant Game* with an automated agent that has knowledge of possible restaurant scripts at both a surface behavioral and linked deep intentional level. The agent will use this knowledge to guide its interpretation of the other human players' actions, and to plan its own physical actions and utterances while participating in the joint activity of completing a restaurant meal. As a step towards this goal, we address the problem of mapping surface forms of dialogue acts to underlying intentions and evaluate the quality of the resulting model in its ability to predict human dialogue acts at the intentional level.

Previously, we have demonstrated automating agents with data collected from *The Restaurant Game* [15]. In this first iteration of the system, dialogue between agents exhibited frequent non-sequiturs and incorrect responses, due to imitating human-human dialogues relying solely on matching surface forms of utterances. The system had no means of recognizing utterances with unique surface forms but the same semantic function, or utterances with the same surface form but different contextually-dependent semantic functions. There was no representation of the *intent* behind the humans' words. In contrast, players use a point-and-click interface to engage in physical interaction (e.g. highlight the dish and click the PickUp button on a pop-up menu). These interface interactions explicitly represent the player's intent. While the automated system constrained the agents' physical behavior based on learned patterns of these intentional actions, no analogous model existed for sequences of utterances. Ideally, a single model could capture patterns of interleaved physical actions and utterances. A precursor to learning such a model is an intentional representation of utterances that can be interleaved cleanly with physical actions.

In this paper we present results of semi-automated annotation of dialogue data collected from *The Restaurant Game*. We demonstrate that classified dialogue acts can function effectively as a common currency for modeling interleaved actions and words, given a domain-specific annotation scheme which classifies both illocutionary force and associated propositional content. We describe a dialogue act classifier that

we have trained on 100 log files, and leveraged to automatically annotate the remaining 5,100. Our results demonstrate how annotating 2% of the log files from a 5,200 game corpus can produce statistical models of dialogue act sequences with predictive power that outperform models based on surface forms. Finally, we show that dialogue acts can be integrated cleanly into a model of physical action sequences, preserving the predictive power of the original model of physical actions alone. While our current system filters out some percentage of utterances from the interleaved model, this is a step toward a solution, and the percentage of included utterances will increase in the future as the classifier improves.

## 2 The Restaurant Game

We designed *The Restaurant Game* to serve as both a data collection device, and a target platform for simulation of social behavior generated from the human data. Players are anonymously paired online to play the roles of a customer and waitress in a 3D virtual restaurant. Players can move around the environment, type open ended chat text, and manipulate 47 types of interactive objects through a point-and-click interface. Every object provides the same interaction options: pick up, put down, give, inspect, sit on, eat, and touch.

To date, 13,564 people have played *The Restaurant Game*, from which we have collected 9,433 log files of two-player games. This paper describes work with a subset of 5,200 game logs. A game takes about 10-15 minutes, and an average game consists of 84 physical actions, and 40 utterances with an average length of four words each. Player interactions vary greatly, ranging from games where players dramatize what one would expect to witness in a restaurant, to games where players fill the restaurant with cherry pies. While many players do misbehave, we have demonstrated that when immersed in a familiar environment, enough people do engage in common behavior that it is possible for automatic system to learn valid statistical models of typical behavior and language [14].

## 3 Related Work

We are working toward learning an interleaved model of actions and utterances in an everyday social situation, based on a large corpus of human-human interactions. Here we relate our work to previous research on dialogue modeling and learning from human data, and highlight significant differences.

Gorin et al [6] describe a system that learns to route calls in response to the prompt “How may I help you?”, by finding mutual information between routing decisions and n-grams in human speech. Satingh et al [18] developed a dialogue management system that uses reinforcement learning to learn an optimal policy for a phone-based information system about activities in New Jersey based on interactions with human callers. Huang et al [8] trained chatbots by extracting title-reply pairs from online discussion threads. Our work differs from these projects by collecting data from humans *situated* in a (virtual) physical environment, where players dramatize an everyday scenario through a combination of (typed) dialogue and physical interaction,



contributing to learning an interleaved model of actions and utterances, representing a commonsense script of restaurant behavior. Huang’s work may point toward an interesting direction for future work; incorporating knowledge extracted from external sources into our model.

McQuiggan and Lester [13] applied a similar methodology to ours (capturing demonstrations between humans in a game environment) to learn models of empathetic behavior, including gestures, posture, and utterances. Their work did not focus on learning open-ended natural language dialogue, and instead incorporated pre-recorded utterances. Gorniak and Roy [7] collected data from pairs of players solving puzzles in *Neverwinter Nights*, and constructed a plan grammar, which could be used to understand utterances between players. Similarly, Fleischman and Hovy [5] leveraged a task model of the game-based Mission Rehearsal Exercise to understand natural language input. In these projects, hand-constructed models of the situation (the plan grammar or task model) helped the system understand language. In contrast, we are training a classifier to understand language, and using classified utterances as building blocks to *learn* the situation model. While our data collection methodology is similar to previous work, we are working toward learning the structure of the situation from data, based on semi-automated annotation and automatic recurrence analysis, rather than hand-crafting a plan grammar or task model. Learning the structure has the potential of producing a more robust model through a less laborious process.

## 4 Dialogue Act Classification

Players of *The Restaurant Game* communicate by freely typing chat text to one another. While we can capture every utterance transmitted, there is no explicit representation of the intent behind the words of the player. In contrast, players use a point-and-click interface to engage in physical interaction (e.g. highlight the dish and click the PickUp button on a pop-up menu). These interface interactions explicitly represent the player’s intent. Human annotation is required to transform utterances into functional units that share a common currency with physical actions – atomic units with explicit representations of intent and semantic function. Unfortunately, human annotation is expensive; it is infeasible to annotate a corpus of thousands of game logs, let alone millions. In this section we describe our approach to semi-automating annotation.

We randomly selected 100 game logs from our corpus of 5,200 logs to serve as training data for a classifier, and we annotated these logs by hand. Each utterance is classified along three axes: *Speech Act*, *Propositional Content*, and *Referent*. Speech Acts categorize utterances by illocutionary force (e.g. question, directive, assertion, greeting, etc.), Propositional Content describes the functional purpose of the utterance, and Referent represents the object or concept that the utterance refers to.

Labels in the Speech Act axis are similar in function to those found in the widely used DAMSL annotation scheme [2]. Devising our own annotation scheme was necessary in order to also incorporate propositional content and referents, which will be critical to an interactive agent. It is not enough to recognize that an utterance is a

question or directive, the agent needs to understand what it is a question *about* (a problem with the bill, or the desire to see a menu) or a directive to *do* (prepare a steak, or have a seat at a table). Section 5.4 provides quantitative evidence that the inclusion of propositional content and referents maximizes the range of confidently recognized utterances while preserving an agent’s ability to predict future actions and utterances based on recent observations.

Our three labels are combined into a {speech act, content, referent} triple that serves as an abstraction allowing utterances to be clustered semantically, rather than by surface forms, and greatly compresses the space of possible dialogue acts. Below we provide the details of our annotation scheme, feature selection, classifier implementation, and classification results.

#### 4.1 Human Annotation

It took one of the authors 56 hours to annotate all 4,295 utterances (of average length four words) observed in 100 games. We developed the list of annotation-labels during the course of annotation. The Speech Act labels were based on Searle’s speech acts [17], expanded with Propositional Content and Referent labels to cover the range of utterances frequently observed in our corpus.

All three axes include an OTHER label, applied to utterances that fall outside the scope of typical restaurant conversation, such as nonsense, gibberish, and discussion of the players’ personal lives. We applied a NONE label to the Propositional Content and/or Referent axes for utterances that did not require any content or referent specification. For example, the utterance “yes” is annotated as {CONFIRMATION, NONE, NONE}. Table 1 provides the complete lists of labels for each axis, along with their distributions within the 4,295 utterances observed in 100 games. Table 2 provides a sampling of utterances from the 100 training games with their assigned label triples.

#### 4.2 Feature Selection

Each line of dialogue is transformed into a feature vector consisting of features derived from the surface text, and contextual features based on the physical situation of the speakers. Contextual features include the social role of the speaker (waitress or customer), the posture of the speaker (sitting or standing), who the speaker is facing (one or more of: customer, waitress, bartender, chef), and the containing spatial region of the speaker (one or more of the possibly overlapping regions: inside-the-restaurant, outside-the-restaurant, entrance, podium, table, counter, bar, behind bar, kitchen). The physical state of the players is reported explicitly in the game logs. The text-based features primarily consist of indicators for the presence of unigrams, bigrams, and trigrams of words observed to be salient for particular labels, as well as a smaller number of indicators for symbols and punctuation (‘?’, ‘!’, ‘\$’, emoticons, and digits). Salience is computed based on the mutual information between n-grams and labels, where mutual information is a measure of statistical dependence [3]. Mutual information has been applied for text-based feature selection previously [6].

The contextual feature set remains constant for each axis (speech act, content, and referent), while the salient indicators of the text-based feature set are customized for each axis. For each axis, we compute the mutual information between every label and every unigram, bigram, and trigram. The feature set for a classification axis is the compilation of the top 50 unigrams, bigrams, and trigrams for each label. We compute the mutual information between an n-gram and a label as:

$$MI(word, Class) = P(word, Class) * \log \left[ \frac{P(word, Class)}{P(word) * P(Class)} \right]$$

Where *Class* refers to a label (e.g. ASSERTION, DIRECTIVE, APPROVE, LAUGH, BILL, MONEY, etc.), and *word* refers to a unigram, bigram, or trigram of words from an utterance.

**Table 1.** Label distributions and classification accuracy, precision (Pr), and recall (Re)

Speech Act			Content			Referent		
	Dist.	Pr / Re		Dist.	Pr / Re		Dist.	Pr / Re
ASSERTION	338	0.6 / 0.5	APOLOGIZE	71	0.8 / 0.9	AGE	19	0.6 / 0.5
CONFIRMATION	354	0.9 / 0.8	APPROVE	267	0.7 / 0.6	BILL	106	0.9 / 0.9
DENIAL	90	0.7 / 0.7	BRING	413	0.8 / 0.8	CUSTOMER	5	1.0 / 0.2
DIRECTIVE	1,217	0.8 / 0.9	COMPLAIN	88	0.4 / 0.1	DIET	8	0.0 / 0.0
EXPRESSIVE	724	0.8 / 0.8	CONSOLE	11	0.8 / 0.3	FLOWERS	31	1.0 / 0.8
GREETING	302	0.9 / 0.9	CORRECT	11	0.5 / 0.2	FOOD	1,394	0.9 / 0.9
OTHER	517	0.5 / 0.4	DESIRE	363	0.8 / 0.8	GEOGRAPHY	51	0.9 / 0.3
PROMISE	136	0.9 / 0.8	EXCUSEME	25	0.8 / 0.8	MENU	52	0.9 / 0.9
QUESTION	617	0.8 / 0.9	FAREWELL	110	0.8 / 0.7	MONEY	75	0.8 / 0.6
			FOLLOW	24	0.9 / 0.8	NAME	24	1.0 / 0.3
			GIVE	170	0.8 / 0.7	OTHER	651	0.6 / 0.4
			HELLO	167	0.9 / 0.9	RESTAURANT	20	0.8 / 0.6
			INFORM	176	0.6 / 0.3	SPECIALS	12	0.9 / 0.6
			LAUGH	76	0.8 / 0.9	STAFF	22	0.9 / 0.5
			MOVE	32	0.4 / 0.2	TABLE	37	0.9 / 0.9
			OTHER	643	0.5 / 0.7	TIME	107	0.9 / 0.7
			PICKUP	29	0.5 / 0.3	WAITRESS	21	0.8 / 0.7
			PREPARE	627	0.9 / 0.9			
			REPREMAND	24	0.4 / 0.3			
			SIT	74	0.9 / 0.9			
			STATUS	149	0.7 / 0.4			
			THANK	290	0.9 / 0.9			
			UNDERSTAND	25	0.8 / 0.4			
			YRWELCOME	28	0.8 / 0.8			
CORRECT: 77.3%			CORRECT: 75.3%			CORRECT: 81.1%		
BASELINE: 28.3%			BASELINE: 15.0%			BASELINE: 38.6%		
OVERALL CORRECT: 60.9%			OVERALL BASELINE: 14.3%					

### 4.3 Classifier Implementation

There have been numerous approaches to automatically classifying speech acts, including neural network classification [12], maximum entropy model classification [1], and Hidden Markov Model (HMM) speech act classification [21]. Our classifier is composed of three independent HMM classifiers, one for each axis (speech act, content, and referent). An HMM classifier exploits transition probabilities in the temporal patterns that emerge in human dialogue to boost classification recognition beyond that of individual utterances. We employed the SVM<sup>HMM</sup> classifier [9], which combines a Support Vector Machine (SVM) for observation classification with an HMM for learning temporal patterns of hidden states. Words and contextual features function as observations, and the labels themselves are the hidden states. This combination of an SVM and HMM has proven successful for dialogue act classification previously [19].

**Table 2.** Example labels for utterances in corpus, sorted by classification precision (pr)

Annotation	Utterance	Pr.
{EXPRESSIVE, THANK, MONEY }	“thank you for the tip”	1.0
{ASSERTION, COMPLAIN, FOOD }	“excuse me, i didn't order the cheesecake”	1.0
{PROMISE, BRING, MENU }	“I'll be right back with your menu”	1.0
{DIRECTIVE, PREPARE, FOOD }	“one steak please”	0.9
{GREETING, HELLO, NONE }	“Welcome!”	0.9
{QUESTION, DESIRE, FOOD }	“Would you like a drink to start with?”	0.9
{CONFIRMATION, NONE, NONE }	“okey dokey”	0.9
{PROMISE, BRING, BILL }	“I'll be back with your bill in a moment.”	0.8
{DIRECTIVE, FOLLOW, NONE }	“follow me and i will have u seated”	0.8
{ASSERTION, GIVE, NONE }	“there we r sir”	0.8
{DIRECTIVE, SIT, NONE }	“have a seat wherever you want”	0.8
{DIRECTIVE, BRING, FOOD }	“Yes I'll start off with a soup du jour”	0.8
{EXPRESSIVE, YRWELCOME, NONE }	“no problem”	0.8
{QUESTION, DESIRE, TABLE }	“table for one?”	0.8
{EXPRESSIVE, LAUGH, NONE }	“lol”	0.8
{OTHER, OTHER, OTHER }	“i need to complete my quest”	0.5
{OTHER, OTHER, OTHER }	“donfdgdfgdfgdfgdfg”	0.5
{OTHER, OTHER, OTHER }	“some guy wanted to put 400 mb on floppies”	0.5
{OTHER, OTHER, OTHER }	“what are you a vampire?”	0.5
{EXPRESSIVE, APPROVE, FOOD }	“alrighty that was a satisfying dinner”	0.5
{EXPRESSIVE, APPROVE, FOOD }	“yum that lobster is too good”	0.5
{QUESTION, INFORM, SPECIALS }	“any specials today?”	0.0
{ASSERTION, COMPLAIN, NONE }	“its cold”	0.0

### 4.4 Classification Results

Despite the apparent freedom, players of *The Restaurant Game* tend to constrain their dialogue to social conventions associated with the mutually understood “scripts” of restaurant interaction. This contributes to strong classification results given the challenge of correctly classifying three independent axes capable of producing 4,050 unique triples.

Table 1 presents our classification results, evaluated with 10 fold cross validation. (each fold trained on 90 game logs and tested on 10). For each of the classification

axes, we report the precision and recall of each label, followed by the percentage classified correctly and a comparison baseline. All of the axes perform significantly better than baseline, contributing to 60.9% of the utterances being classified entirely correctly – correct on all three axes. It is notable that a human labeled at least one axis as OTHER in 11.7% of the incorrectly classified utterances. If we focus on the utterances that the human felt were relevant to the restaurant scenario, and ignore these degenerate utterances, the overall percentage correct increases to 70%.

For each label, we tabulate the number of instances in which the label was assigned by the classifier, the number assigned by the human annotator, and the number correctly classified (where the human and classifier agree). Precision is computed by dividing the number correctly classified by the total number assigned by the classifier. Similarly, recall is computed by dividing the number correctly classified by the total number assigned by the human. Baseline represents the percentage classified correctly if we always choose the most common label for each axis (DIRECTIVE, OTHER, and FOOD respectively).

In addition, we evaluated inter-annotator agreement among humans. A volunteer not involved with the development of the classifier annotated 10 game logs (422 utterances). We computed a kappa coefficient of {0.73, 0.70, 0.89} respectively for {speech act, content, referent}, with a mean kappa of 0.77. Kappa between 0.61 and 0.80 is considered substantial agreement [10].

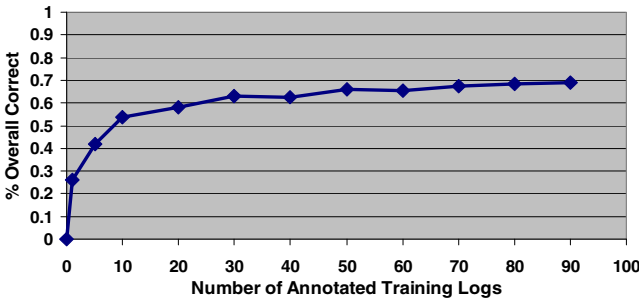


Fig. 3. Effect of training corpus size on classification

Figure 3 illustrates the effect of the training corpus size. For one particular cut of the data, we plot the overall percent correct when the classifier is trained on between 1 and 90 training log files, and tested on the same set of 10 logs. Given that the human labor involved in annotation is expensive, it appears as though annotating more than 30 game logs yields diminishing returns. Reviewing Table 1, we see that the labels with unsatisfactory results for precision or recall are most often due to sparse data – few examples for these labels in the training data. This suggests that continuing to annotate data is worthwhile, if we can focus human labor on appropriate selections of data. It is likely that a human-machine collaborative effort could lead to significant classification improvements, where the machine requests human assistance on assignments of low precision or recall, and efficiently classifies the rest independently.

## 5 Predictive Model Evaluation

The fact that we can correctly classify a large proportion of utterances does not guarantee that these dialogue acts are useful for modeling social interaction. In this section, we demonstrate quantitatively that dialogue acts are useful building blocks for learning patterns of interleaved utterances and physical actions. Our long term goal is to generate social behavior for agents based on models learned by observing human-human interactions. These models will guide agents to conform to expected social conventions, and predict future actions (physical and linguistic) of other agents based on recent observations. As a first exploration in this direction, we experimented with simple  $n$ -gram statistical models [11] applied to both the surface word level and the speech act “intentional level.” In our experiments, we replay the interactions observed between two humans up to some point in a particular game log, and then stop the simulation and predict the next human utterance or action. For each game log in the test set, we slide a window of size  $n$  over the entire log and count correct predictions of the next utterance or action. These predictions indicate what an agent would do, if guided by these models. In section 5.1 we only predict the next *utterance* based on recent utterances; in section 5.3 we predict the next *action or utterance* based on an interleaved model.

We evaluate our dialogue act classification quantitatively by learning three separate dialogue models – based on (1) classified utterances, (2) automatically clustered utterances, and (3) raw utterances – and comparing the predictive power provided by these models. We first evaluate models of utterance sequences alone. Next, we evaluate interleaved models of physical actions and utterances, in order to evaluate how well these utterance abstractions function as a common currency with physical actions.

### 5.1 Comparison of Utterance Abstractions

Our original corpus of 5,200 game logs was divided into 100 logs for annotating and training the dialogue act classifier, 4,800 logs for training  $n$ -gram models, and 300 logs for evaluating prediction accuracy. After training the dialogue act classifier on 100 logs, we automatically classified all utterances in the remaining 5,100 games in the corpus. There were 312 unique dialogue act triples observed in the 100 annotated logs, with 183 observed in more than one log.

There are 112,650 unique raw utterances observed in the corpus. We clustered these utterances automatically using the  $k$ -means algorithm, based on the Euclidean distance between feature vectors of unigrams, bigrams, and trigrams observed within the utterances. We chose  $k=300$ , as this number of clusters provides a fair comparison with the 312 unique dialogue act triples.

Figure 4 illustrates that dialogue acts are more predictive than raw utterances or clusters. Prediction accuracy is computed by counting the number of correct predictions of the next observed utterance, cluster, or dialogue act in a bigram, trigram, or 4-gram. The baseline prediction accuracy is computed by counting the number of correct predictions if we always choose the most likely utterance, cluster, or dialogue act found in the training corpus. Raw utterances yield poor prediction

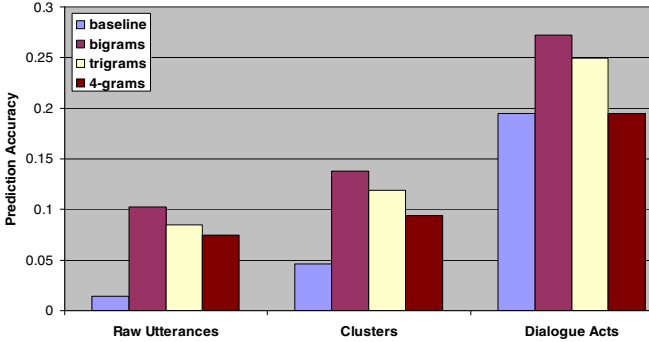


Fig. 4. Prediction accuracy for utterances

accuracy for n-gram models for all values of  $n$ , only achieving above 0.1 for bigrams. While clusters do achieve about a 30% increase in prediction accuracy over raw utterances, they fall below that of dialogue acts by over 50%.

### 5.2 Filtering to Improve Prediction

There is value in knowing what we don't know. Our classifier assigns labels with 60% accuracy. Ideally, we would train the n-gram model with only correctly labeled dialogue acts, rather than introducing noise with those classified incorrectly. Based on the statistics computed in section 4.4, we can interpret precision as our confidence that the classifier has assigned the correct label to an utterance, and exploit this to determine which labels to include in our model, and which to omit. Like a human traveler in a foreign country with limited understanding of the language, the system can grasp onto well understood utterances and exploit them to understand the gist of the interaction. There is no notion of confidence in automatically generated clusters, thus clusters cannot be filtered in the same meaningful way. For the sake of comparison, the best we can do is filter clusters by their observation frequency in the training corpus.

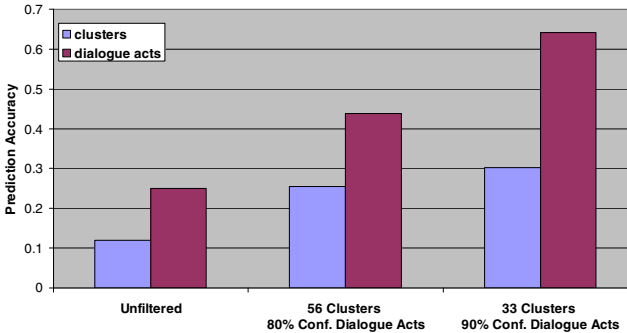


Fig. 5. Prediction accuracy for filtered utterances

If we train an n-gram model based only on dialogue acts with precision of at least 80%, trigram prediction accuracy increases from 0.25 to 0.44. With 90% precision, accuracy increases to 0.64. There are 56 unique dialogue act triples with at least 80% precision covering 51.7% of the training data, and 33 triples with 90% precision covering 28.4% of the training data. See Table 2 for examples of high precision dialogue acts for the restaurant domain – utterances related to ordering food, paying bills, getting seated, and bringing menus. It is not surprising to see an increase in prediction accuracy when we decrease the number of unique labels, and number of utterances labeled. However, Figure 5 illustrates that filtering clusters in a similar way does not yield the same dramatic increase that we observe with dialogue acts. We compare prediction accuracy of the 56 most likely clusters to the 56 dialogue acts with 80% precision, and the 33 most likely clusters to the 33 dialogue acts with 90% precision.

### 5.3 Integrating Utterances with Physical Acts

We followed the methodology described previously [14] for generating a lexicon of unique physical actions from a corpus of thousands of game logs. By observing the state changes that occur each time a player takes a physical action in each game log, we learn a lexicon of context-sensitive, role-dependent actions (e.g. waitress picks up pie from counter). In our 5,200 logs, we have observed 7,086 unique actions. Based on the learned lexicon, log files are transformed into a sequence of action lexicon indices interleaved with utterances, where utterances may be represented as either clusters or dialogue act triples.

In Figure 6, we illustrate the effect on prediction accuracy of integrating utterances into a trigram model of physical interaction. Based on recent observations of interleaved actions and utterances, the integrated model predicts the next action *or* utterance. This is a difficult problem, given that we are polluting the lexicon of 7,086 directly observable actions with 112,650 unique utterances, which we can classify with 60% accuracy. We find that interleaving physical actions with dialogue acts gives better prediction accuracy than with raw utterances or clusters, and if we filter

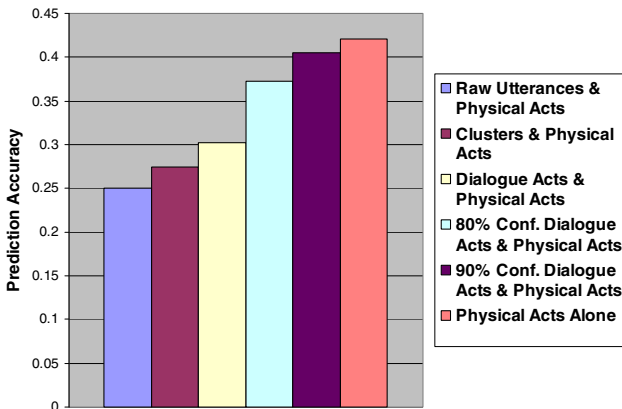


Fig. 6. Effect of interleaving physical acts with utterances



to only include dialogue acts with at least 90% confidence we can achieve a prediction accuracy negligibly lower than that of physical acts alone (0.41 vs. 0.42), demonstrating that dialogue acts function well as a common currency with physical acts. While 90% confidence only covers 28.4% of the utterances, these are highly salient utterances for the restaurant domain, and coverage should increase as more data is annotated and the classifier improves. This is a first step toward discovering a higher-level structure of the restaurant scenario, composed of interleaved sequences of actions and utterances.

Ideally integrating dialogue acts with physical actions would yield a higher prediction accuracy than either alone. The current representation of physical interaction clusters similar objects, and abstracts away timing information. In other words, the model does not differentiate between picking up *steak* or *salmon* (both clustered as *food*), and does not need to predict *when* the waitress will depart from the table to pick up food from the kitchen (perhaps after a three utterance exchange, concluding with “I’ll be right back with that”). Clearly these details will be important to an automated agent. We are working toward an agent guided by a model of physical interaction that retains these details, and we expect better prediction from the interleaved model of actions and words than from a model of either alone.

#### 5.4 Speech Acts vs. Dialogue Act Triples

Recall that our classification scheme classifies the propositional content and referent in addition to the illocutionary force of each utterance. In this section, we demonstrate that this difference makes a significant impact on our ability to integrate a maximal number and variety of utterances into the predictive model of physical interaction, while preserving predictive power. We compare the predictive power of dialogue act triples to that of speech acts alone, while scrutinizing the percentage of utterances covered as we filter by confidence (aka precision).

Initially, integrating speech acts into the model of physical interaction yields a slightly higher prediction accuracy than integrating dialogue act triples (0.34 vs. 0.30). As we filter out lower confidence speech acts and dialogue acts, the prediction accuracy of both comes closer to that of the physical interaction alone (0.42), and the difference between predictive power of speech acts and dialogue acts becomes negligible. However, as we raise the confidence threshold, the percentage of utterances covered decreases dramatically for the coarser grained speech acts, as seen in Figure 7. We preserve 28.4% of the dialogue acts with 90% confidence, and only 6.5% of speech acts. As seen in Table 1, only GREETING speech acts have above 90% precision, compared to 33 unique dialogue act triples with 90% precision, which employ the full range of speech act labels (see Table 2 for a subset). At 80% confidence, a higher percentage of speech acts are preserved than dialogue acts (63.4% vs. 51.7%), but this comes at the cost of a 5% decrease in predictive power from physical interactions alone. Annotating with finer grained dialogue act triples provides a means of recognizing a maximal number and range of salient utterances for the restaurant domain, while preserving predictive power.

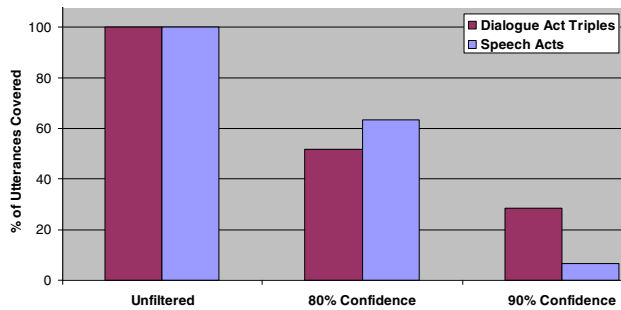


Fig. 7. Percent of utterances covered when filtering

## 6 Conclusion

Behavioral models generated by observing players of online games and virtual worlds have the potential to produce interactive socially intelligent agents more robust than can be hand-crafted by human designers. While it is possible to automatically learn statistically recurring patterns in surface level behavior, our results demonstrate that we can generate models with stronger predictive power by leveraging a minimal amount of human interpretation to provide annotation of the underlying intentions, in the form of dialogue act triples. The significant increase in predictive power with dialogue acts is evidence of progress towards discovering the socio-cultural scripts that guide social interaction in a restaurant.

It is likely that our dialogue act classifier could be improved by providing more training data guided by an active learning process, however intention of utterances can never be fully recognized without understanding their role in the higher level structure – the sub-goals of the restaurant scenario composed of interleaved physical and dialogue actions. Our evaluation of integrating physical actions with dialogue models demonstrates the potential for dialogue acts to function as building blocks of sub-goals. Discovering this higher level structure remains a goal for future work. Human annotation will be required to identify intentional sub-goals spanning multiple physical actions and/or dialogue acts, and based on our experience with semi-automated dialogue act annotation, we are optimistic that semi-automation of sub-goal annotation will be possible as well.

## References

1. Carvalho, V.R., Cohen, W.W.: On the Collective Classification of Email Speech Acts. In: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Salvador, Brazil (2005)
2. Core, M., Allen, J.: Coding Dialogs with the DAMSL Annotation Scheme. In: Proceedings of the AAAI Fall Symposium on Communicative Action in Humans and Machines, Boston, MA (1997)
3. Cover, T.M., Thomas, J.A.: Elements of Information Theory. John Wiley & Sons, Inc., Chichester (1991)

4. Edery, D., Mollick, E.: *Changing the Game: How Video Games are Transforming the Future of Business*. FT Press, Upper Saddle River (2008)
5. Fleischman, M., Hovy, E.: Taking Advantage of the Situation: Non-Linguistic Context for Natural Language Interfaces to Interactive Virtual Environments. *Intelligent User Interfaces* (2006)
6. Gorin, A., Riccardi, G., Wright, J.: How may I help you? *Speech Communication*, vol. 23, pp. 113–127. Elsevier Science, Amsterdam (1997)
7. Gorniak, P., Roy, D.: Speaking with your sidekick: Understanding situated speech in computer role playing games. In: *Proceedings of Artificial Intelligence and Digital Entertainment* (2005)
8. Huang, J., Zhou, M., Yang, D.: Extracting chatbot knowledge from online discussion forums. In: *Proceedings of IJCAI* (2007)
9. Joachims, T.: SVM<sup>hmm</sup>: Sequence Tagging with Structural Support Vector Machines (2008), [http://www.cs.cornell.edu/People/tj/svm\\_light/svm\\_hmm.html](http://www.cs.cornell.edu/People/tj/svm_light/svm_hmm.html)
10. Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. *Biometrics* 33, 159–174 (1977)
11. Manning, C.D., Schütze, H.: *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge (1999)
12. Marineau, J., Wiemer-Hastings, P., Harter, D., Olde, B., Chipman, P., Karnavat, A., Pomeroy, V., Graesser, A., The Tutoring Research Group: Classification of speech acts in tutorial dialog. In: *Proceedings of the Workshop on Modeling Human Teaching Tactics and Strategies at the Intelligent Tutoring Systems 2000 Conference*, pp. 65–71 (2000)
13. McQuiggan, S., Lester, J.: Learning empathy: A data-driven framework for modeling empathetic companion agents. In: *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Hakodate, Japan (2006)
14. Orkin, J., Roy, D.: The Restaurant Game: Learning social behavior and language from thousands of players online. *Journal of Game Development* 3(1), 39–60 (2007)
15. Orkin, J., Roy, D.: Automatic Learning and Generation of Social Behavior from Collective Human Gameplay. In: *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, Budapest, Hungary (2009)
16. Schank, R.C., Abelson, R.P.: *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*. Lawrence Erlbaum Associates, Mahwah (1977)
17. Searle, J.R.: *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge (1969)
18. Satingh, S., Litman, D., Kearns, M., Walker, M.: Optimizing Dialogue Management With Reinforcement Learning: Experiments with the NJFun System. *Journal of Artificial Intelligence Research* (2002)
19. Surendran, D., Levow, G.: Dialog Act Tagging with Support Vector Machines and Hidden Markov Models. In: *Proceedings of Interspeech* (2006)
20. von Ahn, L., Dabbish, L.: Labeling images with a computer game. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Vienna, Austria, pp. 319–326 (2004)
21. Woszczyzna, M., Waibel, A.: Inferring linguistic structure in spoken language. In: *Proceedings of the International Conference on Spoken Language Processing*, Yokohama, Japan (1994)

# Using Exclusion Logic to Model Social Practices

Richard Evans\*

RichardPrideauxEvans@gmail.com

**Abstract.** This paper introduces Exclusion Logic - a modal logic based on hierarchical finite-state machines. We show how Exclusion Logic can be used as a deontic logic to model social practices. We describe a computer implementation of Exclusion Logic in a multi-agent simulation, and explain how it efficiently resolves conflicts between incompatible norms.

**Keywords:** deontic, multi-agent, material incompatibility, exclusion, hierarchical finite-state machine, defeasible reasoning, conflict resolution.

## 1 Introduction

There are many good reasons, frequently rehearsed, for modelling norms in some sort of declarative formal language. Researchers working on normative multi-agent systems have used a large array of formal languages to model aspects of social practice: a variety of deontic logics, temporal logics, constraint logic programming, the event calculus, and so on.

But when software engineers build multi-agent simulations for large-scale industrial applications, they tend to eschew such formalisms, and fall back on the unglamorous but dependable finite-state machine. The hierarchical finite-state machine (hereafter, **HFSM**) is a tree of states and sub-states, allowing context-specific data and processes to be stored within each sub-state. Norm-violations can be detected in state-specific event-handlers, and can be responded-to by changing to a state which encourages some form of correction.

Despite the attractions of the HFSM, it has one fundamental weakness as a means of representing social practices: the reason why the machine moved from one state to another is hidden in the machine's processes. This information is compiled away into the run-time process, and is inaccessible to the agent. In other words: the state machine may have moved correctly from one state to another, but the reason why the machine changed state is not represented *declaratively*, and hence is not accessible to the agents themselves. Weaknesses in HFSM-based simulations are typically the result of this lack of informational transparency: the agents are participating in the practice without understanding the practice.

One way to try to get the best of both worlds is to recast HFSMs in a declarative language. Exclusion Logic, hereafter **EL**, is just such a language.

---

\* Thanks to Martin Berger and Jeff Orkin for detailed feedback on previous drafts. I am also grateful to the Berkeley Social Ontology Group.

We use EL to define a specifically *deontic* exclusion logic as a syntactic extension of EL. (A deontic logic is a type of logic which distinguishes between what is in fact the case and what *should* be the case [15]). We show how this deontic exclusion logic can be used to model social practices in a multi-agent simulation.

## 2 Exclusion Logic

**EL** is a simple language for constructing trees of data. It does not contain operators for negation or disjunction, and uses only a restricted version of implication. Instead, it has an operator for expressing the idea that one data-value *excludes* other values.

**Definition 1.** *Given a set  $S$  of symbols, the expressions in EL are defined as all terms of type  $G$  in:*

$$\begin{aligned} X &::= S \mid S:X \mid S.X \\ E &::= X \mid E \wedge E \\ G &::= E \mid E \rightarrow E \end{aligned}$$

The language is carefully stratified: terms are assembled into conjunctions, which are then combined into implications. In this way, it resembles disjunctive normal form. Note that  $\rightarrow$  is not recursively embeddable: although  $P \rightarrow Q$  is well-formed,  $P \rightarrow Q \rightarrow R$  is not.

The only unusual feature of this simple language is the use of two binary modal operators, ‘.’ and ‘:’, to build up trees of information. Asserting that  $A:B$  is claiming both that  $A$ , and that *one* of the ways in which  $A$  is  $B$ . Saying that  $A . B$ , by contrast, is to say that  $B$  is the only way in which  $A$  is the case. By saying  $A . B$ , we *exclude* other values of  $A$ .

Terms	Explanation
WhoseMove.Black	It is Black to move
Agents:Jack:Age.37	One of the agents is Jack, and his age is 37
Agents.Jack:Age.38	Jack is the only agent, and his age is 38
Wolf:Class.Mammalia.Theria	The wolf is of class Mammalia and sub-class Theria

**Fig. 1.** Examples of expressions in L

Both  $A:B$  and  $A.B$  imply that  $B$  is a way in which  $A$  is true. But  $A:B$  and  $A:C$  are compatible terms, whereas  $A.B$  exclude other values of  $A$ , such as  $A.C$ .

WhoseMove.Black	WhoseMove.White
Agents:Jack	Agents.Jill
Agents:Jack:Age.37	Agents:Jack:Age.38
Wolf:Class.Mammalia.Theria	Wolf:Class.Gastropoda

**Fig. 2.** Pairs of incompatible expressions

## 2.1 Expressive Power and Expressive Impoverishment

In some respects, Exclusion Logic seems woefully impoverished in comparison with more traditional formalisms such as predicate logic. But in other ways, it seems surprisingly powerful and versatile.

Let's look at the weaknesses first. At first glance, EL looks like a very impoverished logic indeed, hardly deserving of the name. How can we live without operators for negation and disjunction? Although Exclusion Logic does not contain these operators, it has resources for expressing *some* of what we want to say when we use them.

**Incompatibility and Negation in EL.** Asserting  $\neg P$  is a way of denying  $P$  - a particularly vague and non-committal way of denying it. Although EL has no negation, it has other, more specific, ways of denying a claim.

Given  $A.B$ , we can make an incompatible claim by saying  $A.C$  or  $A.D$ . EL has the ability to make contentful denials, but it doesn't have the resources for making the least contentful incompatible claim for any arbitrary complex expression.

We also have the ability in EL to express negation directly using  $\rightarrow$ . As long as EL contains at least two symbols  $A$  and  $B$ , we can define an explicit contradiction  $\perp$  as  $A.A \wedge A.B$ . Now define  $\neg P$  as  $P \rightarrow \perp$ . However, although we can express the negation of a simple term such as  $P$  or  $P.X$ , there is no way to express the negation of a *complex* term such as  $P \wedge Q$  or  $P \rightarrow Q$ , because  $\rightarrow$  is not recursively embeddable.

**Disjunction in EL.** Although EL does not contain a disjunction operator, it has other ways of achieving similar effects. Note that  $P \vee Q$  is the most specific of the claims which is entailed by  $P$  and by  $Q$ . If we see entailment as a partial ordering, then  $P \vee Q$  is the least upper bound of  $P$  and  $Q$ .

Now certain pairs of expressions in EL have a corresponding least upper bound. For example, the least upper bound of  $A.B$  and  $A.C$  is  $A$ .

## 2.2 Expressive Power

So even if EL does not contain negation and disjunction, it has ways of expressing *some* of what we want to do with these operators. Now let us focus on the ways in which EL is expressively powerful.

**Quantifying over Properties and Relations.** Because properties and relations are just first-class elements in a string of symbols separated by ‘:’ and ‘.’, we can quantify over them just as easily as we can quantify over symbols referring to objects.

So for example, if we have *Apple : Green* and *Book : Green*, then we can state that there is an  $x$  such that *Apple : x* and *Book : x*. Similarly, if we have *Jack : Likes : Jill* and *Jill : Likes : Jack*, then we can state that there is an  $x$  such that *Jack : x : Jill* and *Jill : x : Jack*. (In EL, constants are upper-case and variables are lower-case).

**Mixing Deontic and Epistemic Operators.** One of the motivations behind EL was the desire for a simple language in which deontic and epistemic operators could be mixed freely, so that we can express things like “Agent A believes that P, but he *should* believe that Q”.

EL, with its ability, for any term  $t$ , to produce further terms of the form *Operator : t*, allows us to mix and iterate deontic and epistemic operators freely. The  $\rightarrow$  operator is defined so that  $P \rightarrow Q$  and *Operator : P* together imply *Operator : Q*, which is a core requirement for being able to treat *Operator* deontically or epistemically.

### 2.3 Semantics

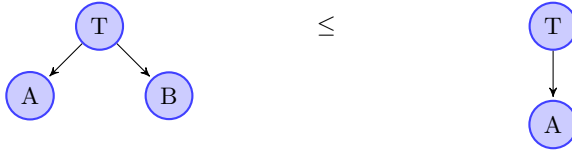
There isn’t space in this article to specify the semantics of EL in full detail. (For a complete description of the semantics, see [8]).

An expression in EL will be interpreted in a lattice of labeled rooted trees. In a labeled rooted tree (hereafter, **LRT**), each vertex is labeled with a symbol from  $S$ , and each edge is labeled with either ‘\*’ (suppressed) or ‘!’. If the edge from  $X$  to  $Y$  is labeled with \*, it means that  $Y$  is one of the children of  $X$  - but  $X$  may have other children also. But if the edge is labeled with !, it means that  $Y$  is the *only* child of  $X$ .

**Definition 2.** An **LRT** is a directed tree with a vertex- and an edge-labeling. Formally, an LRT is a tuple  $(V, E, L, M, R)$  where:

- $V$  is a set of vertices
- $E$  is a set of edges
- $L$  contains the vertex labels; it is a total function from  $V$  to the set of symbols  $S$
- $M$  contains the edge labels; it is a total function from  $E$  to  $\{*, !\}$
- $R \in V$  is the vertex for the root of the tree, where  $L(R) = T$  (here,  $T$  is a symbol not occurring in  $S$  used to label the root of each tree).

We define a partial-ordering on these trees, where  $t_1 \leq t_2$  if  $t_1$  contains at least as much information as  $t_2$ .



We define an operator  $\sqcap$  on trees, such that  $t_1 \sqcap t_2$  is the greatest lower bound: the least-specific tree that contains all the information of  $t_1$  and  $t_2$ , as long as they are compatible, and  $\perp$  otherwise.

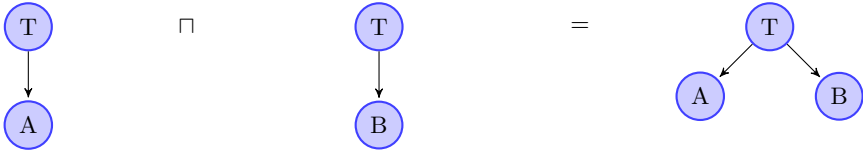


Fig. 3. Example of  $\sqcap$

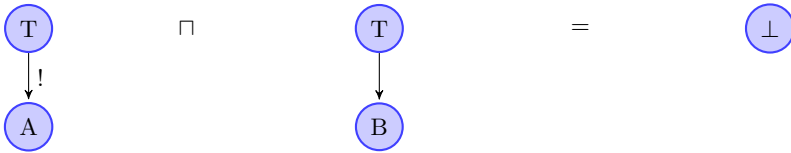


Fig. 4. Example of  $\sqcap$

We also define least upper bound  $\sqcup$  in the natural way, as the tree which contains all the information they share:

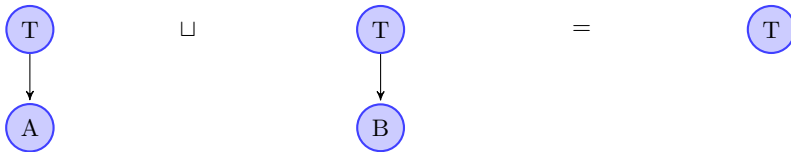


Fig. 5. Example of  $\sqcup$

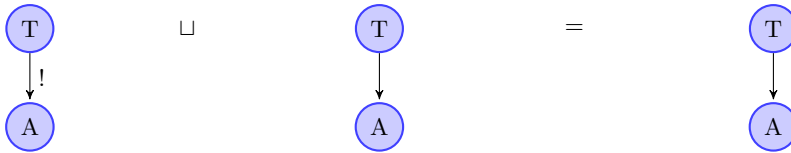


Fig. 6. Example of  $\sqcup$



Given this ordering, we can construct a lattice of labeled rooted trees, with  $\perp$  at the bottom, and the empty tree  $\top$  at the top.

Using this lattice, we can give EL a clean semantics and a straightforward decision procedure.

## 2.4 Decision Procedure

We will use the lattice of LRTs defined above to provide an efficient decision procedure. We will interpret an expression by the  $\sqcup$  of the LRTs which satisfy it.

**Definition 3.** Define  $[x]$  as the set of LRTs which satisfy  $x$ :

$$[x] = \{M \mid M \models x\}$$

Note that there are an infinite number of LRTs which satisfy an expression. Now because the LRTs form a lattice,  $[x]$  has a least upper bound  $\sqcup$ . We can use this least upper bound to calculate entailment directly.

**Proposition 1.**

$$\begin{aligned} X \models Y &\text{ iff } \forall M \ M \models X \Rightarrow M \models Y \\ &\text{ iff } [X] \subseteq [Y] \\ &\text{ iff } \sqcup[X] \leq \sqcup[Y] \end{aligned}$$

*Proof ((Left to right)).* We will show that  $[X] \subseteq [Y] \Rightarrow \sqcup[X] \leq \sqcup[Y]$ . Assume  $[X] \subseteq [Y]$ . Now  $\sqcup[X] \in [X]$ . So  $\sqcup[X] \in [Y]$ , as everything in  $[X]$  is also in  $[Y]$ . Now  $\sqcup[Y]$  is  $\geq$  everything in  $[Y]$ , including  $\sqcup[X]$ , as  $\sqcup[Y]$  is an upper bound. So  $\sqcup[Y] \geq \sqcup[X]$ .

*Proof ((Right to left)).* We will show that  $\sqcup[X] \leq \sqcup[Y] \Rightarrow [X] \subseteq [Y]$ . Assume  $\sqcup[X] \leq \sqcup[Y]$ . Take an  $x \in [X]$ . We must show that  $x \in [Y]$ . Now  $x \leq \sqcup[X]$ , as  $\sqcup[X]$  is an upper bound. So  $x \leq \sqcup[Y]$ , by transitivity of  $\leq$ . Now  $\models_{\sqcup[Y]} Y$ . As  $\models_M Y$  and  $M' \leq M$  implies  $\models_{M'} Y$ ,  $\models_x Y$ . Since  $x$  is one of the models which satisfies  $Y$ ,  $x \in [Y]$ .

Next, we will provide a direct way of computing  $\sqcup[X]$ . This will give us a direct and efficient way of determining if  $Y$  is entailed by  $X$ : look to see if  $\sqcup[X] \leq \sqcup[Y]$ . Instead of having to verify *all* the models of  $X$  to see if  $Y$  is true, we just look in one canonical model.

**Computing  $\sqcup[X]$  directly.** We will define a particular model  $m(X)$ , and then show that  $m$  is indeed the minimal model:  $m(X) = \sqcup[X]$ . Note that the expressions in L can be divided into two groups: the conjunctions and the implications.

**Definition 4.** First we will define  $m(X)$  for the fragment of  $L$  that doesn't include implications.

- $m(T) = \top = (\{1\}, \{\}, \{(1, T)\}, \{\}, 1)$
- $m(A \wedge B) = m(A) \sqcap m(B)$
- $m(A:B) = (V_{m(A)} \cup \{v\}, E_{m(A)} \cup \{v', v\}, L_{m(A)} \cup (v, B), M_{m(A)} \cup (\{v', v\}, *))$
- $m(A.B) = (V_{m(A)} \cup \{v\}, E_{m(A)} \cup \{v', v\}, L_{m(A)} \cup (v, B), M_{m(A)} \cup (\{v', v\}, !))$

Here,  $v$  is a new vertex not occurring in  $m(A)$  which is used to interpret  $B$ , and  $v'$  is the vertex which  $m(A)$  uses to interpret  $A$  i.e. the  $v'$  such that  $s_{m(A)}(v') = \text{symbols}(A)$ .

**Definition 5.** We extend  $m$  from individual expressions to sets of expressions:

$$m(X) = \sqcap \{m(e) \mid e \in X\}$$

**Definition 6.** We extend  $m$  to include implications. First, divide a set  $X$  of expressions into the conjunctions  $A$  and the implications  $G$ . We will interpret  $G$  as a function, and interpret  $m(G, A)$  as the result of repeatedly applying  $m(G)$  to  $m(A)$ . Define  $m(X \rightarrow Y)$  as the pair of LRTs  $(m(X), m(Y))$ . Next we will define the evaluation  $e$  of a pair  $(A, B)$  of LRTs to an LRT. The evaluation function  $e$  is defined as:

$$\begin{aligned} e &: LRT \times LRT \rightarrow LRT \rightarrow LRT \\ e(A, B) X &= \text{if } X \leq A \text{ then } B \sqcap X \\ &\quad \text{else } X \end{aligned}$$

**Definition 7.** We extend  $e$  to operate on sets of pairs in the natural way: Define  $e'$  as:

$$\begin{aligned} e' &: 2^{LRT \times LRT} \rightarrow LRT \rightarrow LRT \\ e' G X &= \sqcap \{e(h, X) \mid h \in G\} \end{aligned}$$

**Definition 8.** We define  $m(G, A)$  to be the least-specific model which is closed under the application of  $e'$  on  $m(G)$  over  $m(A)$ :

$$m(G, A) = \bigsqcup \{X \mid X \leq e'(m(G), m(A)) \wedge e'(m(G), m(X)) = m(X)\}$$

**Proposition 2.**  $m(X) = \bigsqcup [X]$  for all sets of expressions  $X$ .

*Remark 1.* In summary, to determine whether  $G, A \models X$ , we just need to check if  $m(G, A) \leq m(X)$ .

**The Complexity of the Decision Procedure.** We determine whether  $G, A \models X$  by checking whether  $m(G, A) \leq m(X)$ . This computation involves the following sub-tasks:

1. Computing  $m(G)$
2. Computing  $m(A)$

3. Computing  $m(G, A)$  by repeatedly applying  $\lambda z.e'(m(G), z)$  to  $m(A)$
4. Computing  $m(X)$
5. Determining whether the result of (3)  $\leq$  the result of (4)

Assume  $G$  is a set of judgments of size  $n$ , and  $A$  and  $X$  are individual conjunctive judgments. Assume the max number of conjunctions in  $G, A, X$  is  $m$ , and the max complexity of any conjunct is  $k$ . Now computing  $m(G)$  involves  $2n$  applications of  $m$  on a pair of conjunctions. Applying  $m$  to  $c_1 \wedge c_2 \wedge \dots \wedge c_n$  involves computing  $(c_1 \sqcap c_2) \dots \sqcap c_n$ . This involves  $\sum_{i=1}^n ik^2 \leq m^2 k^2$  operations. So computing  $m(G)$  involves at most  $2nm^2 k^2$  operations. Computing  $m(A)$  and  $m(X)$  involve another  $m^2 k^2$  each.

Computing the repeated application of  $\lambda z.e'(m(G), z)$  to  $m(A)$  involves at most  $n$  applications of  $e'$ . (Justification: a member of  $G$  can only be applied once. Each time through, we either run out of pairs in  $G$  to apply, in which case we are done, or we apply one, and start again, removing the one we used. There can be at most  $n$  applications before we run out of pairs to apply). Each application of  $e'$  involves at most  $n$  iterations to find a pair in  $G$  which applies to  $A$ , and an application of  $e$  if it finds one. So each application of  $e'$  involves  $\sum_{i=1}^n im^2 k^2 \leq n^2 m^2 k^2$  operations. There are at most  $n$  applications of  $e'$ , so repeatedly applying  $\lambda z.e'(m(G), z)$  to  $m(A)$  takes at most  $n^3 m^2 k^2$  operations.

Determining whether  $m(G, A) \leq m(X)$  involves comparing each vertex of  $m(X)$  with every vertex in  $m(G, A)$ . The size of  $m(G, A)$  is bounded by  $(n+1)mk$ . So computing  $\leq$  involves at most  $(n+1)mk \times mk = (n+1)m^2 k^2$  operations.

In summary, testing entailment in  $L$  is polynomial-time (unlike, say, propositional calculus which is NP-complete). A polynomial-time decision procedure is what we would expect, given that  $L$  is in a format very close to disjunctive normal form.

### 3 Using Exclusion Logic as a Deontic Logic to Model Social Practices

Deontic Exclusion Logic is a particular application of Exclusion Logic, designed for modeling social practices.

#### 3.1 Syntax

Given a set  $S$  of symbols, the **expressions** in Deontic Exclusion Logic are defined as the terms  $G$  in:

$$\begin{aligned}
 X &::= S \mid S:X \mid S.X \\
 J &::= X \mid \Box X \mid \bigcirc X \\
 E &::= J \mid E \wedge E \\
 G &::= E \mid E \rightarrow E
 \end{aligned}$$

In Deontic Exclusion Logic, like Standard Deontic Logic, the intended meaning of  $\Box P$  is that  $P$  should be the case. The  $\bigcirc$  operator comes from temporal logic:

$\bigcirc P$  means that  $P$  holds at the next state.  $\square P$  and  $\bigcirc P$  are just syntactic sugar for *Should* :  $P$  and *Next* :  $P$  respectively.

Note that the operators are carefully stratified. It is syntactically much more restricted than Standard Deontic Logic: we cannot iterate the  $\square$  (or  $\bigcirc$ ) operator, and we cannot put  $\square$  (or  $\bigcirc$ ) outside a conditional or conjunction.

Well-Formed	Ill-Formed
$\square A \wedge \square B$	$\square(A \wedge B)$
$A \rightarrow \square B$	$\square(A \rightarrow B)$
$\square A$	$\square \square A$
$\bigcirc A$	$\square \bigcirc A$
$A \rightarrow \bigcirc B$	$\bigcirc(A \rightarrow B)$

Fig. 7. Well-formed and ill-formed expressions

## 4 Computational Implementation

Having described Deontic Exclusion Logic in the abstract, we now focus on the computational implementation. We used this logic to power a multi-agent simulation, featuring tens of agents running in real-time in a Sims-style environment.

### 4.1 Overview

At any time, the system has a set of judgments in working memory. (In the current implementation, all agents share the same set of judgments, so there is no chance of their understandings of the social world diverging). Judgments are added to working memory by perceiving the world. Judgments are also added by making inferences. These inferences can involve free variables. For example:

`x:Apple -> x:Edible`

Some of these conclusions are further facts, and some conclusions are deontic judgments, to the effect that something *should* be the case. For example:

`x:Stomach:Filled.Empty /\ y:Edible -> [] x:Act.Eat.y`

Here, the conclusion is that  $x$  should eat  $y$ . In the code examples, the  $\square$  operator is represented by

`[]`

The  $\bigcirc$  operator is represented by

`Next`

Conjunction ( $\wedge$ ) is represented by

`/\`

Often, a number of incompatible conclusions will follow from the input data. In these cases, the system has to resolve the conflict before it can act.

Once conflicts have been resolved, the system is ready to act on the deontic judgments (converting judgements about what it should do into action). Some of the deontic judgments are primitive motor actions (body actions). These are automatically executed by the underlying system. The results of these primitive motor actions is that new facts are added to working memory. For example:

```
x:Agent /\ Motor:Animate:x.Eat.y.Finish.t -> x:Stomach:Filled.Full
```

These new facts suggest further inferences, which suggest further deontic judgments, which resolve to subsequent action, and so on.

## 4.2 An Ergonomic Language for Implementing Simulations

In EL, when an expression is added to the database, all incompatible expressions are removed.

This means that simulation rules can be expressed more economically than in predicate logic. For example: in the postcondition of the “Move block x from y to z” operator, we just need to specify that  $x : On.z$  - we do not need to specify that  $x : On.y$  is removed - because this removal follows *automatically* from the semantics of the ‘.’ operator.

There are more striking consequences of the fact that the semantics takes care of the removal of invalid data. Suppose that  $A.B : C$  and  $A.B : D.E$  and  $A.B : F$  were true, but then  $A.G$  was added to the database. This would invalidate the *whole tree* of data underneath  $A : B$ . In this way, the semantics of the ‘.’ operator takes care automatically of the removal of swathes of invalid data.

## 4.3 Motives Are Just a Type of Conditional Normative Judgment

In this deontic simulation, there is no separate system for goals, desires or motives. In this approach, having a desire to eat just is holding the conditional judgment that I should eat when my stomach is empty:

```
x:Stomach:Filled.Empty /\ y:Edible -> [] x:Act.Eat.y
```

In [11], Habermas distinguishes between four types of action: the teleological/instrumental (goal-oriented), the normatively-regulated (norm-following), the dramaturgical (expressive) and the communicative (dialogical). Different theorists have different understandings of which of these types of action is explanatorily fundamental. The instrumentalist, for example, takes teleological goal-directed action as most basic, and seeks to explain the other three types of action in terms of it.

In many multi-agent normative architectures, e.g. the Boid Architecture [5], obligations and desires are modelled separately. One of the things which is distinctive about the approach taken here, by contrast, is that the normative is taken as explanatorily fundamental. Teleological/instrumental reasoning is subsumed under the normative, as a special case.

## 4.4 Examples

To test the language and supporting framework, we used a variety of examples, ranging from the quotidian to the bizarre. For example:

- A case where eye-contact is used to communicate status - so an inferior must look away when his superior is looking at him
- A simple turn-taking board-game, where the participants notice if the rules are violated, and make suitable corrections
- A case where one agent is trying to kiss another, who is trying to run away
- A society where eating is private and shameful, and should not be done in the presence of others
- A set of tribesmen queuing up to lick a sheep

Deontic Exclusion Logic allows us to express these practices naturally and concisely.

### Example: Queueing.

```
// If x is the head of the queue, and x has finished doing his thing, and
// y is next in the queue after x, then it is y's turn next
Queue:Head.x /\ Animate:x.phi.z.Finish /\ Queue:After:x.y -> Next:Queue:Head.y

// If x wants to join an empty queue, then he becomes the head and tail
x:Participant /\ Queue:In:x.False /\ Queue:Tail.Null -> Next:AddHead.x
AddHead.x -> Queue:Head.x /\ Queue:Tail.x

// If x wants to join and the queue isn't empty, x is added at the end
x:Participant /\ Queue:In:x.False /\ Queue:In:y.True
/\ y:Participant /\ Queue:Tail.y -> Next:AddAfter:y.x

AddAfter:x.y -> Queue:After:x.y /\ Queue:Tail.y

// When x is the tail of the queue, there is nobody behind him
x:Participant /\ Queue:Tail.x -> Next:Queue:In:x.True /\ Queue:After:x.Null

// When x is the head of the queue, he is encouraged to do the action
// which he has been queuing up to do
Queue:Head.x /\ x:Participant /\ Queue:Target.y -> [] x:Act.phi.y
```

The queue keeps track of the head, the tail, and an associative array of who is behind whom. When somebody joins the queue, they are added at the back. The person at the head is tasked with performing the action *phi* on the target. When the head of the queue has finished performing the action, he leaves the queue, and the person behind becomes the head.

Note that the participants are preoccupied with place-work [10]: making the current state of the practice manifest. In this case, it means each member is tasked with looking at the person in front, thereby showing who is in front of him. Furthermore, if the head of the queue fails in his task of performing the

specified action on the target, the participants will gently remind him of his duty.

The result is a fluid practice involving multiple agents performing concurrent overlapping actions to maintain the state of the practice, defined concisely in a few declarative conditionals.

**Example: Tic Tac Toe.** A simple turn-taking game like Tic Tac Toe can be modelled in a few lines:

```
G:HasWon:x /\ G:Other:x.y -> G:Playing.F /\ [] Motor:Animate:x.Hooray.y

UserInput.x.t.z /\ G:Move.x /\ G:Symbol:x.t /\ z.Empty
-> Motor:Animate:x.t.z.Finish

UserInput.x.t.z /\ G:Symbol:x.u /\ G:Different:t:u /\ G:Other:x.y
-> [] Motor:Animate:y.NotYourSymbol.x /\ [] Motor:PointAt:y.x

UserInput.x.t.z /\ G:Other:x.y /\ G:Move.y
-> [] Motor:Animate:y.ItsNotYourMove.x /\ [] Motor:PointAt:y.x

G:Playing.T /\ G:Move.x /\ square.Empty /\ G:Symbol:x.t /\ G:Other:x.y
-> [] x:Act.Mark.square.t

[] x:Act.Mark.square.t -> [] Motor:Route:x.square /\ [] Motor:LookAt:x.square.At

[] x:Act.Mark.square.t /\ Engine:Test(IsVeryNear, x, square)
-> [] Motor:Animate:x.t.square

Motor:Animate:x.t.z.Finish /\ G:Symbol:x.t -> Next:G:HasMoved:x:z:t
G:HasMoved:x:z:t /\ G:Other:x.y -> G:Move.y /\ square.t

Square11.s /\ Square12.s /\ Square13.s /\ G:Symbol:x.s -> Next:G:HasWon:x
Square21.s /\ Square22.s /\ Square23.s /\ G:Symbol:x.s -> Next:G:HasWon:x
...
```

The game state  $G$  keeps track of whose move it is in the  $G : Move$  term.  $G : Move.x$  means that it is  $x$ 's turn to play. The game keeps track of which symbol each player is assigned:  $G : Symbol : x.t$  means that  $t$  is  $x$ 's symbol. Note that the player is physically capable of making illegal moves. If he makes a move when it is not his turn, or he writes the wrong symbol, his opponent will notice, and correct him.

In our prototype we have a number of such practices coexisting happily. For example, if two agents are playing Tic Tac Toe, and a wolf comes along, the agents will flee from the wolf. But if the wolf is killed, the agents will return to their game. The propositions storing the state of the game have been retained, so they carry on seamlessly from where they left off.

## 4.5 Inference Mechanism

The inference problem is: given a new piece of perceptual data, find all the consequences of that data. The naive approach to inference is to iterate through all the

rules, looking for a match with the input data. But this becomes prohibitively expensive as the number of rules increases. We chose the Rete algorithm to optimize inference, as it has been proven to work efficiently in large real-world applications.

The Rete algorithm works by maintaining a dependency graph of information which allows the system to know exactly which inference-rules to trigger when a new piece of information is added. The interested reader should consult [9] for details.

## 4.6 Conflict Resolution

As soon as we apply the logic to real situations, our inference rules generate inconsistent output. The incompatible conclusions may come from two different practices, or from one practice containing two or more inference rules yielding incompatible outputs.

Deontic Exclusion Logic uses a form of conflict-resolution which starts with the approach used in [11].

**Definition 9.** *If we divide our input theory into a set  $G$  of implications and a conjunction  $A$  of terms, where  $G, A \models \perp$ , then the **max-family**( $G, A$ ) is the maximal subsets  $G'$  of  $G$  such that  $G', A \not\models \perp$ . The **out-family** is the set of consequences of the members of the max-family:*

$$\text{out-family}(G, A) = \{\{C \mid (G', A) \models C\} \mid G' \in \text{max-family}(G, A)\}$$

At this stage, we have a set of candidate maximal consistent subsets to choose from. In order to act, we will need to choose *one* of these candidates. Alchourron and Makinson use a partial ordering over  $G$  to resolve conflict in [11]. But in our application, where we need agents to handle conflict-resolution immediately, we insist on a total-ordering  $\triangleleft$  over  $G$ . (In our implementation, the ordering of the inference-rules is determined simply by their relative position in the file).

Given this total ordering  $\triangleleft$ , there is a unique highest member of the max-family. Let  $G_{\triangleleft}$  be the member of  $\text{max-family}(G, A)$  such that every other member contains a member which is  $\triangleleft$  every member of  $G_{\triangleleft}$ .

This maximal consistent subset  $G_{\triangleleft}$  is constructed by going through the elements of  $G$ , in descending order, starting with the topmost. If adding the next member of  $G$  keeps the result consistent, we add it; otherwise, we skip it.

Applying this to our implementation, we need to generalize from the propositional case to the case of rules with universally-quantified free variables, by grounding out all instances of each rule. If our inference-rule has  $k$  free variables, and each variable can take  $n$  values, then there are  $n^k$  groundings of this rule.

Now it would be prohibitively slow to construct  $G_{\triangleleft}$  naively by grounding out the inference-rules, and going through all the elements in the right order. Instead, we use a form of *insertion-sort* on the current-set of justifications, using  $\triangleleft$  to choose between justifications.



The resulting conflict-resolution strategy is a form of justification-based truth-maintenance system. We keep track of all the justifications associated with each proposition. Each justification can be suppressed by a set of justifications, or it can be suppressing a set of justifications. (It cannot be both). A proposition is *in* if it has at least one justification which is not suppressed, it is *out* otherwise.

When the Rete inference engine wishes to add a new justification for a proposition to the database, it looks to see if there are any propositions in the in-list which conflict with it. If so, it looks through all the justifications for each conflicting term, and decides whether the conflicting justification is  $\triangleleft$  the newly added justification. If it is, then we add a new element to the Rete inference agenda: suppress the old justification with the new justification. Otherwise, we add the alternative pair: suppress the new justification with the old justification. The Rete inference agenda is a list of actions; each action is either adding a justification, suppressing a justification with another justification, or releasing a suppression. When a justification for a proposition is suppressed, this suppression travels through the Rete network, suppressing all the consequences of that proposition.

When we want to suppress a justification, we need to look to see if it is currently suppressing anything else. If it is, then these elements need to be released from this suppression. When a justification is released from suppression, this release also travels through the Rete network.

Conflict-resolution is thus interweaved with inference in the Rete engine, performing a continuous insertion-sort to respect  $\triangleleft$  as new propositions are added.

#### 4.7 Worked Example of Conflict-Resolution

Given the following rules:

```
P -> T.B
Q -> T.A
T.A -> M.P
C -> X.Z
B -> M.Q
M.Q -> X.Y
```

Suppose first the system receives  $B$ . Rete infers  $M.Q$  and then  $X.Y$ .

Next, suppose the system receives  $Q$ . First, Rete adds  $T.A$  from  $Q$ . But next it wants to add  $M.P$  from  $T.A$ , but it sees that  $M.P$  and  $M.Q$  are incompatible. At this point, it compares the justifications according to the  $\triangleleft$  ordering, and sees that  $B \rightarrow M.Q \triangleleft T.A \rightarrow M.P$  (ordering of rules is based on relative position). So  $T.A \rightarrow M.P$  suppresses  $B \rightarrow M.Q$ . Suppression propagates through the Rete network, and so our justification for  $X.Y$  is also suppressed by  $T.A \rightarrow M.P$ .

Next, suppose the system receives  $C$ . Rete infers  $X.Z$ . Now  $X.Z$  and  $X.Y$  are incompatible, but because  $X.Y$  has already been suppressed, it has already been removed from the in-list.

Finally, suppose the system receives  $P$ . Rete infers  $T.B$ . Now  $T.B$  is incompatible with  $T.A$ , and  $Q \rightarrow T.A \triangleleft P \rightarrow T.B$ , so the earlier justification for

$T.A$  is suppressed. This suppression propagates through the Rete network, and the earlier justification for  $M.P$  is also suppressed by  $P \rightarrow T.B$ . Now  $M.P$  was previously suppressing  $M.Q$  and  $X.Y$ , but now that  $M.P$  is suppressed, these two are now released from their suppression. However, when we try to add  $X.Y$  back again, we find another suppressing claim,  $X.Z$ , so  $X.Y$  is still excluded from the in-list.

## 5 Related Work

### 5.1 Related Work on Conflict Resolution

Xavier Parent [14] also handles conflict resolution by assuming an ordering on terms and uses the max-family and out-family constructs to find the best choices. He is considering conflict-resolution in the context of the Input/Output logics, rather than exclusion logic.

### 5.2 Comparison with HFSMs

In Deontic Exclusion Logic, a social practice is modeled as a set of declarative conditionals. This has a number of advantages over modeling the practice as an HFSM.

Firstly, the reason why the practice is in this particular state is accessible to the agents, so they can *justify* their judgements about the situation. In Deontic Exclusion Logic, the state-transition is a declarative conditional, which the agents can inspect themselves (unlike the state-transition in the HFSM, which is a compiled process). For example: one of the first practices modeled was a set of rules for making eating taboo in the presence of others. In this scenario, agents would not autonomously eat when other agents were around. But not only did they behave correctly - they knew *why* they were behaving as they were. The reason for their refraining was epistemically accessible to them: they could see that it was the presence of others, and the rule about not eating with others around, which prevented them from eating.

Secondly, when a practice is modeled as a set of declarative conditionals, it is easier to learn. Instead of having to construct a large HFSM to explain a block of behavior, the agent can learn individual declarative conditionals in a *piecemeal* manner. In a block of procedural code, such as an HFSM transition, an individual line does not even *compile* on its own, because of the various dependencies between the line and earlier lines. But in Deontic Exclusion Logic, each declarative conditional is intelligible on its own, so it can be learned on its own.

## 6 Summary

This paper has introduced yet another formal language for the representation of norms. The one thing that distinguishes this particular formalism is that it allows the declarative rearticulation of the Hierarchical Finite-State Machine.

Because Deontic Exclusion Logic is just a syntactic extension of EL, it is efficiently decidable. It has been used to power a real-time multi-agent simulation, to implement a variety of social practices. Our initial prototype suggests that expressing social practices in this language is natural and concise.

## References

1. Alchourron, C., Makinson, D.: Hierarchies of Regulations and their Logic. *New Studies in Deontic Logic*, pp. 125–148. D. Reidel, Dordrecht (1981)
2. Boella, G., van der Torre, L.: Permissions and Obligations in Hierarchical Normative Systems. In: *Proc. of ICAIL (2003)*
3. Brandom, R.: *Making It Explicit*. Harvard University Press, Cambridge (1994)
4. Brandom, R.: *Between Saying and Doing*. Oxford University Press, Oxford (2008)
5. Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., van der Torre, L.: The Boid Architecture: conflicts between beliefs, obligations, intentions and desires. In: *Proceedings of the Fifth International Conference on Autonomous Agents (2001)*
6. Evans, R.: The Logical Form of Status-Function Declarations. In: *Etica & Politica*, vol. XI, pp. 203–259 (2009)
7. Evans, R.: Re-expressing Normative Pragmatism in the Medium of Computation. In: *Collective Intentionality VI*, Berkeley (2008)
8. Evans, R.: Introducing Exclusion Logic as a Deontic Logic. *Deontic Logic in Computer Science*. Springer, Heidelberg (2010)
9. Forgy, C.L.: Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem. *Artificial Intelligence* 19, 17–37 (1982)
10. Garfinkel, H.: Autochthonous Order Properties of Formatted Queues. In: *Ethnomethodology's Program*. Rowman and Littlefield, Oxford (2002)
11. Habermas, J.: *The Theory of Communicative Action*, p. 85. Beacon Press, Boston (1984)
12. Makinson, D., van der Torre, L.: Input/Output Logics. *Journal of Philosophical Logic* 29, 383–408 (2000)
13. Makinson, D., van der Torre, L.: Permission from an Input/Output Perspective. *Journal of Philosophical Logic* 32, 391–416 (2003)
14. Parent, X.: Moral Particularism and Deontic Logic. *Deontic Logic in Computer Science*. Springer, Heidelberg (2010)
15. von Wright, G.H.: Deontic Logic. *Mind* 60 (1951)
16. von Wright, G.H.: *An Essay in Modal Logic*. New York Humanities Press (1953)

# Making Games ALIVE: An Organisational Approach


Sergio Alvarez-Napagao<sup>1</sup>, Fernando Koch<sup>2</sup>,  
Ignasi Gómez-Sebastià<sup>1</sup>, and Javier Vázquez-Salceda<sup>1</sup>

<sup>1</sup> Universitat Politècnica de Catalunya  
{salvarez, igomez, jvazquez}@lsi.upc.edu

<sup>2</sup> University of Melbourne  
fkoch@unimelb.edu.au

**Abstract.** The AI techniques used in commercial games are usually predictable, inflexible and unadaptive, causing a lack of realism for the player. In this paper, we introduce a proposal of integrating the ALIVE framework, based on Organisational theory, into commercial games. The objective of our proposal is to provide game AI developers with a methodology and tools to model gaming scenarios using social structures.

## 1 Introduction

Artificial Intelligence (AI) in commercial games (Game AI) provides the means to enhance the two-way communication with the human player by delivering the illusion of “intelligence” in the non-player characters’ (NPCs) behaviour. Usually, it encompasses a subset of academic AI techniques that implement *ad hoc* solutions in three groups :

- i Movement mechanisms, providing the decision process to control NPC’s motion, e.g. optimised real-time versions of  $A^*$  algorithms.
- ii Behaviour control used to control NPCs’ actions.
- iii Strategy techniques used to co-ordinate groups of NPCs.

Whilst algorithms in (i) have evolved to mature state-of-the-art, solutions commonly used in commercial games for (ii) and (iii) are far from aligned with academic AI and are based on simplistic, rule-, automata- or case-based methods optimised for performance. These domain-dependent approaches present the following limitations in most of the cases:

- Blind specifications: the NPCs are programmed on *how* to act in reaction to environmental and/or other players conditions, but not *why* to act in a given manner; hence, the actions are purposeful and, in most cases, not “natural” from the human player’s perception.
- Lack of flexibility and adaptiveness: the rule-based actions are limited and reactive to external conditions, not being able to evolve, and providing reduced pro-activeness.

- Strange behaviour: the behaviour of the NPCs do not reflect the aspects of sociability and “participating in a whole”, leading to unnatural actions from the human player’s perception.
- Predictable behaviour: NPCs’ tactics are easily discoverable by the human player and, after some time, predictable, leading to negative perception.
- Low reusability, as the solutions are commonly tailored to specific scenario domains and, therefore, not re-usable through different games even if they belong to the same genre.

We argue that it is possible to create elaborate solutions for the issues of (ii) behaviour control and (iii) strategy techniques by integrating models based on Organisation Theoretical methods to control NPCs’ behaviour. This theory contributes to the systematic study of how actors behave within organisations. Hence, the actors in a game are described as an organisation which behaviour is based on specific roles, norms, dependencies, and capabilities (services). Our aim is to provide a methodology and tools for Game AI developers to model gaming scenarios using social structures. We demonstrate how this approach tackles the limitations of the current domain-dependent approaches aforementioned.

This research is part of the Project ALIVE<sup>2</sup>, which combines ideas from organisational and coordination theories to create an integrated framework for the development of complex distributed systems. We describe this framework in the next section. Section 3 introduces our proposed architecture and how to integrate to existing games. Section 4 provides proof-of-concept case studies. Section 5 cross-relates our approach to other works. The paper concludes with Section 6, where we discuss our achievements and propose future work.

## 2 The ALIVE Framework

The ALIVE framework is being developed as part of the Project ALIVE<sup>2</sup>. It aims to combine existing work in coordination and organisational structures with the state-of-the-art in service-oriented computing. It will allow system architects to build service-oriented systems based on the definition of organisational structures and how they interact. This framework defines three structural levels, as depicted in Figure 1:

- The *Service Level* augments and extends existing service models in order to make components aware of their social context and of the rules of engagement with other services via semantic demarcation technologies.
- The *Coordination Level* specifies the high-level patterns of interaction (known as *workflows*) among services by using powerful coordination techniques from recent agent research. These *workflows* can be adapted at runtime, which is useful when the system has to react to unexpected events (such as failures and exceptions).
- The *Organisational Level* provides the *Service* and *Coordination* levels with a social context, specifying the organisational rules that govern interaction. This level makes services *organisational aware*, that is, services are aware

of system’s high-level objectives, structure and normative restrictions. This reflects in task allocation, *workflow* generation and agreement at the coordination level. For instance, the system will prevent *workflows* that violate normative restrictions from being generated and tasks are to be allocated to appropriate actors as defined on the organisational structure. This level also benefits from recent research in organisational dynamics to allow the structural adaptation of the system when changes on rules or restrictions happen.

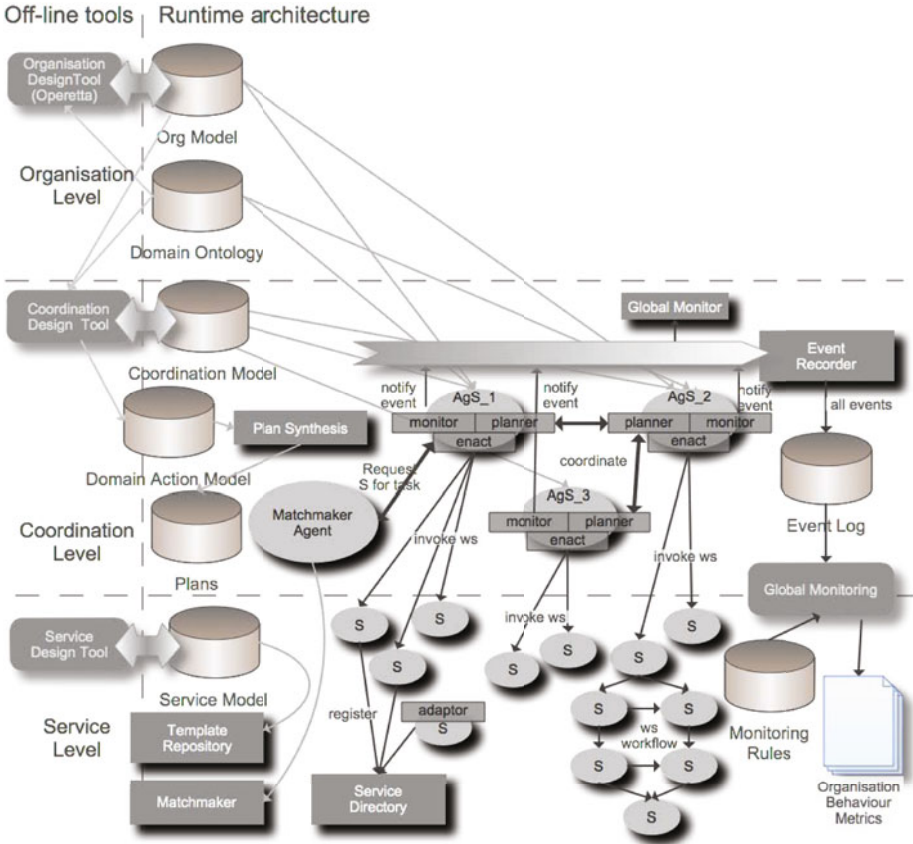


Fig. 1. ALIVE architecture (*S* stands for *Service*)

The ALIVE Framework allows Game AI developers to think in terms of why-what-how when defining the decision-making actions for NPCs. That is, at the Organisational level, the developer defines “why to do something” by describing the elements of the organisational structure in terms of organisation objectives, roles, norms, and restrictions. At the Coordination level, the developer defines

“what to do” based on possible solutions and tasks to realise in specific situations; finally, at Service level, the developer defines “how to do it” in terms of which actual, low-level actions to perform in order to realise those tasks.

Moreover, the ALIVE framework applies *substantive norms* that define commitments agreed upon actors and are expected to be enforced by authoritative agents, imposing repair actions and sanctions if invalid states are reached. Substantive norms allow the system to be flexible, by giving actors –human or computer-controlled– the choice to cause a violation if this decision is beneficial from an individual or collective perspective.

Finally, the ALIVE Framework provides useful tools to define these elements, such as *Operetta*, a visual tool implemented as an Eclipse IDE plugin, which allows to specify the organisational concepts of roles, interactions and norms. These structures are implemented as coordination agents, used to build coordination plans for groups of agents enacting roles within the organisation. Agents interact for enacting their roles either via direct communication or via service invocation. Monitors observe agent interactions. When these interactions are put together with the normative and organisational states – e.g. obligations, permissions, roles – they allow the agents to reason about the normative effects of their actions. The detection of normative states is a passive procedure that consists in monitoring past events and checking them against a set of active norms [1].

This set of tools and methods provides inherent support to the development of complex, re-usable Game AI solutions.

### 3 Proposal

Our argument is to create elaborate solutions for the issues of (ii) behaviour control and (iii) strategy techniques by integrating the ALIVE framework to academic and commercial games. This approach will provide extended flexibility to the elements that imply intelligent behaviour, e.g. actors and characters, teams of individuals, and narrative storylines. In addition, it will provide metrics that can be applied to evaluate the organisational behaviour using the games’ environments as simulation scenarios. Hence, it would be possible to compare, learn, and improve NPC’s behaviour with an approach based on organisation theoretical solutions for Game AI. This would contribute to overall flexibility and adaptiveness.

Figure 2 depicts the proposed architecture. We are providing in our solution:

1. A practical solution to couple agents to the Game Engine, by defining the Game Enactor programming interface.
2. A tool to describe the Organisation Ontology, which contains a representation of agent structures.
3. The elements to describe game actors’ behaviour via social structures based on norms, roles and their enactment, promoting the balance between autonomy and story direction.

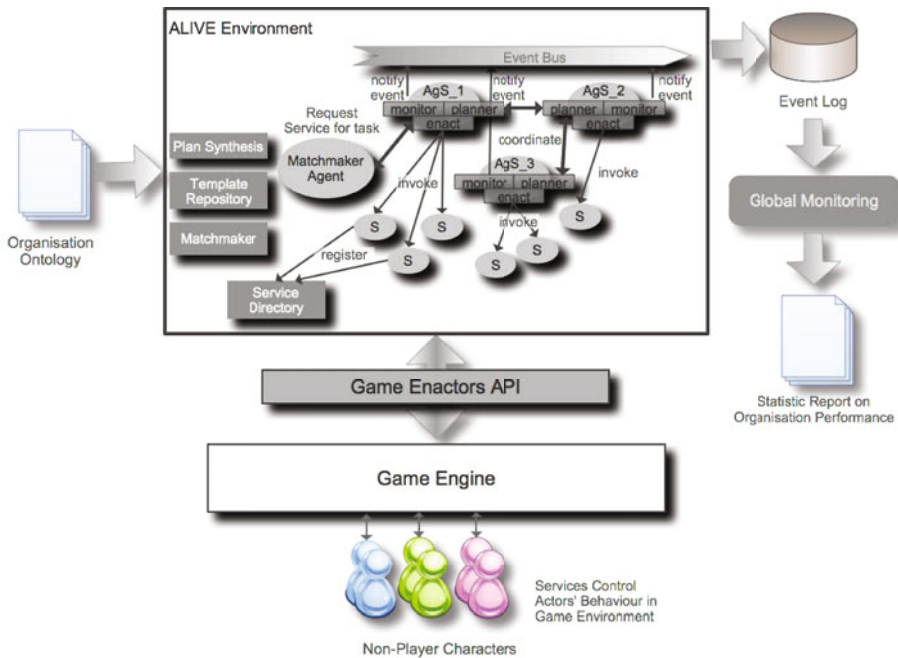


Fig. 2. ALIVE-Gaming coupling infrastructure

We propose that this solution is applicable to both *fun games* and *serious games*. For the former, we foresee our solution helping to improve the games' actors' functioning with more flexibility and promoting natural behaviour. For the latter, the model reflects the socio-environmental behaviour of human societies, providing the basis for games and simulations that can be used in the emerging field of Computational Social Sciences. Next, we present a number of case studies in the applicability field.

## 4 Case Studies

We are testing the solution in different games in order to validate our proposal, as depicted in Figure 3. We intend to analyse what is the advantage in terms of realism, flexibility and adaptability. Moreover, our application to simulation environments will provide results useful for organisational research. In the case of commercial games, it requires access to the internal game control structures. We selected three representative examples for our case studies, considering the complexity and validity of achieved results:

- i Grand Theft Auto IV, from Rockstar Games;
- ii Warcraft III, from Blizzard, and;
- iii Lincity, an open-source, SimCity like city building simulation.





Fig. 3. Games used as case studies

### 4.1 Sandbox Game: GTA IV

First we will test our environment on sandbox games, also known as free-roaming games. In these kind of games, players are given a large amount of freedom, with non-linear storylines and different paths to completion. For example, the Grand Theft Auto (GTA) series allows the player to wander around a whole city and interact with hundreds of NPCs and objects.

In *free roaming* games such as GTA, most of the interactions with characters are scripted, giving the player a feeling of repetitiveness after a few hours of play. On the other hand, the higher-than-normal freedom given to the player also provides less realism.

Our objective is to define a high level social structure, simulated by the ALIVE coordination layer, with dynamic adaptation of interaction patterns, using GTA as the graphic interface of such a social environment. For example, in GTA the player is almost free to behave in a violent way while driving a car. Passing red lights, driving in the wrong direction, and running over people are actions that have no consequences in the vast majority of cases. We have already implemented a prototype by designing, at the organisational level, traffic norms and roles defining authority figures, i.e. police (see Figure 5). Police agents plan and reason about

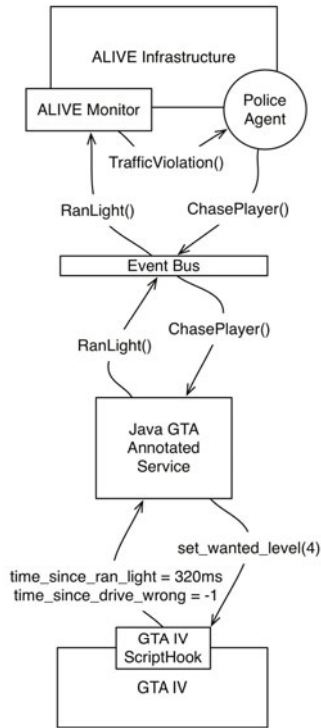
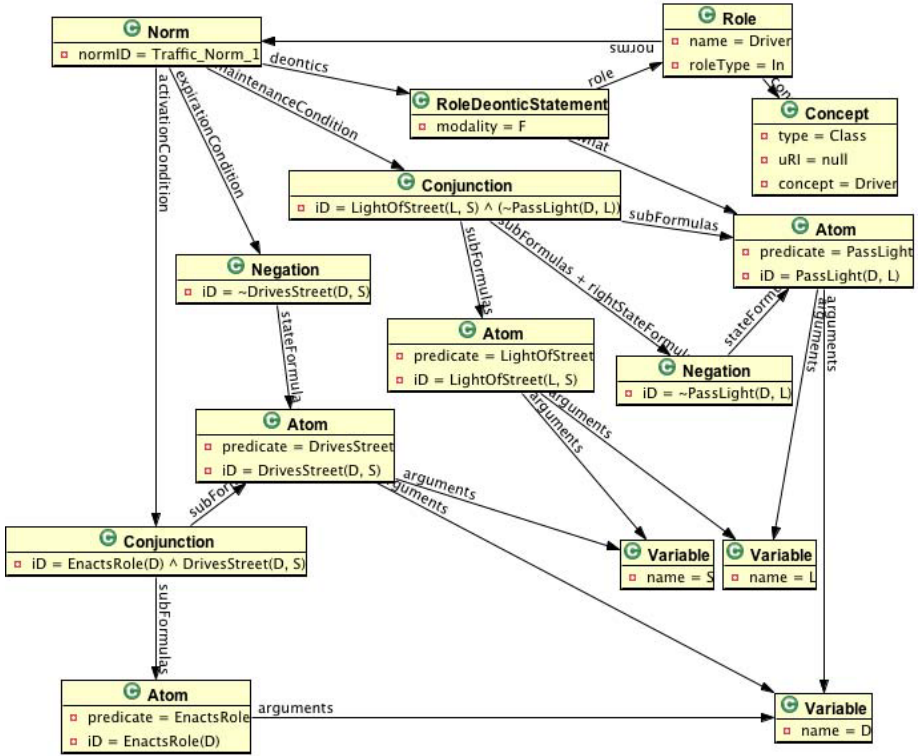


Fig. 4. ALIVE-Gaming coupling infrastructure for GTA

the sanctions to apply when detecting a traffic norm violation, which can consist on imposing a fine or initiating a car chase, depending on the gravity of such violation.



**Fig. 5.** Graphical representation of the norm *it is forbidden to pass under a red light* (Operetta Tool)

For the GTA connection with ALIVE we have used *GTA ScriptHook*, an open-source tool which allows us to capture all possible events and execute all possible actions in the game running environment, including the control of NPCs’ behaviour. The following is a typical flow of information on our system (see Figure 4):

1. An event, i.e. running past a red traffic light, happens on the game.
2. *GTA ScriptHook* captures the event and provides it to the *Java GTA Annotated Service*.
3. The service interprets the game event as a low-level event and puts it on the *Event Bus*.
4. The *ALIVE Monitor* captures the event from the *Event Bus* and infers its high-level interpretation.

5. This interpretation triggers the generation of a new event *ChasePlayer* that is registered in the *Event Bus*.
6. The *Java GTA Annotated Service* captures the *ChasePlayer* event from the *Event Bus* and, via *ScriptHook*, modifies the game.
7. This will effectively make something happen on the game, i.e. player being chased by police forces, as a response of having run a traffic light.

As we have already seen in Section 2, norms modelled using the ALIVE tools are not regimented but substantive, which means that the player –as well as any NPC– can decide not to fulfill them. Thus, a player can decide to break traffic rules if the police is not around or at line of sight, or if the player has no concerns about the possible sanctions enforced by the NPCs. This is a simple example, but more complex examples can be designed to create obstacles or motivations for the player, by reasoning at runtime about the social-environmental context in the game at a certain point of time.

Our intention is to design a full set of organisational constraints, i.e. norms, individual objectives and roles, in order to define high-level social structures in the game, therefore improving realism through sensible and adaptive interactions with NPCs.

## 4.2 Real-Time Strategy Game: Warcraft III

For many years, computer wargames have been designed as turn-based games. Real-time Strategy (RTS) games are an evolution of turn-based wargames, in which the player has to command a team of virtual individuals with diverse capabilities to achieve a common objective, commonly to defeat the teams of the human- or computer-controlled rivals. Other (sub)objectives include the capture and micro-management of resources, technological evolution, and so on. RTS games are interesting for our purpose in the sense that the concepts they deal with can be directly mapped to the ALIVE domain, i.e. organisational structure, roles, role hierarchy, objectives, and coordination.

At the moment, we have modelled the organisational specification of an abstract RTS game (see Figures 6 and 7), including roles, e.g. worker, defender, attacker, objectives – e.g. gather resources, defeat the enemy armies –, and norms – e.g. it is forbidden to create military units until there are enough workers to support them. This specification can be reused through different RTS implementations. For our tests we are using Warcraft III.

We have coded a Java service that is connected to Warcraft 3 game, allowing for bidirectional communication via sockets. This Java service is used by the ALIVE framework. With this implementation, agents are able to:

- Perceive the “state of the world”, reacting to events happening in the game at runtime, e.g. a unit being created, or a soldier spotting an enemy.
- Reason about which actions should be taken in the game taking into account the current state of the world.

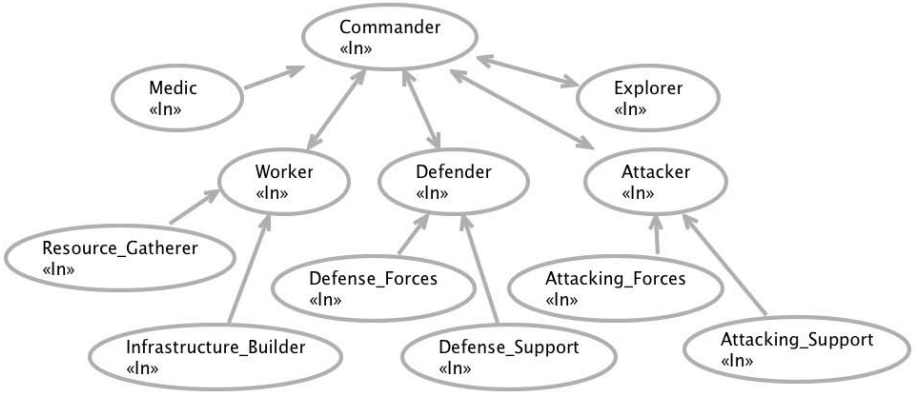


Fig. 6. Social structure for generic RTS games (*Operetta* Tool)

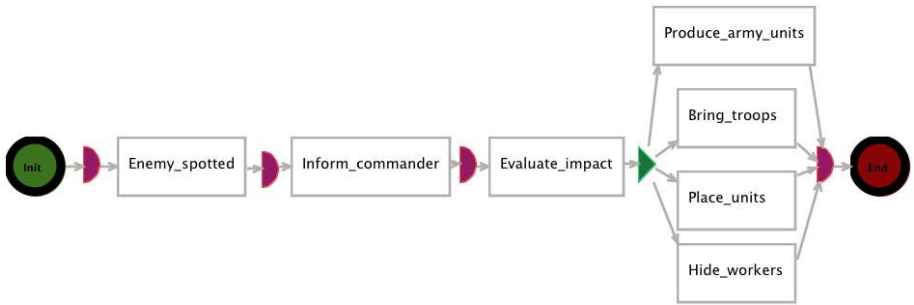


Fig. 7. Interaction structure for *Defend city* (*Operetta* Tool)

- Include the ALIVE specification in their reasoning. Agents will take into account the organisational structure: roles, plans and norms, defined in the ALIVE model, and proactively decide at each moment which actions to enact in order to accomplish the organisational objectives. The agents may also decide to discard some actions to be enacted if an organisational norm is forbidding to enact them given the current state of the world.
- Enact actions in the game (see Figure 8). Once the reasoning process has decided which are the next actions to be performed, agents are able to communicate with the game, making the unit responsible of each action to enact it according to the role and plan structures defined in the organisational specification.

This is the scenario that could best benefit from the adaptability offered by the ALIVE infrastructure. A common issue of RTS games is that after some amount of time spent on it, computer opponents are predictable and easily defeatable by using simple yet optimal strategies. We aim to produce computer opponents capable of adapting to unpredictable scenarios by dynamically improving at the



Fig. 8. Warcraft III units enacting actions sent from the ALIVE platform

organisation and coordination layers. Moreover, this type of game would provide us a clear visual interface to execute simulations of organisations in real-time.

### 4.3 Macro-economic Simulation: Lincity

One of our main objectives, as noted earlier, is to use games as an interface for visualising the results of simulations of organisational specifications. Real-time simulation (RTS) games can be used for this, but they are only useful for scenarios where the range of events types are limited. For discrete-event and off-line simulations we propose the use of games that deal with macro-economic variables.

In our case, we have chosen *Lincity*, which is an open-source city-building simulator<sup>[1]</sup> developed in C++. The idea behind this is to simulate the actions taken by large amounts of virtual individuals and present the results at the macro-economic level, i.e. the evolution of the city. Reorganisation techniques will be intensively used in this scenario.

For instance, we intend to simulate crisis management scenarios during natural disasters<sup>[2]</sup>. City-simulations are perfect environments to describe intuitive

<sup>1</sup> More particularly, it is a clone of the SimCity series.

disaster situations, such as fire and floods. In this situation, the diverse stakeholders have to coordinate their activities to handle such large scale crises. There are various dynamic aspects that must be dealt with, as the crisis may escalate, which require elaborate coordination mechanisms among the various stakeholders involved: (i) Emergency Call Centre; (ii) Fire Station; (iii) Fire fighter Team; (iv) First-Aid Station; (v) Police Station, and; (vi) Fire Fighter Truck.

We intend to introduce “unexpected events”, such as resources becoming unavailable, escalation of the problem, road block, and so on, that would require the handling of exceptions in ongoing coordinations. In this case, modelling of organizational knowledge has a crucial role, where role and objective concepts define *why* the stakeholders must operate and letting the system define *what to do* and *how*. Changes in the environment and in stakeholders needs are inherently reflected in a top-down approach. When the service and coordination levels fail, the system turns to the organizational level for continuing the operations.

Therefore, this elaborated simulation environment could also be used as a tool to model real world situations, allowing for testing and evaluating different response approaches to natural disaster situations.

## 5 Related Work

The current issues of commercial games AI introduced in Section 1 are related to high-level concepts of gaming such as realistic virtual actors, automatic content and storyline generation, dynamic learning, or social behaviour. Tackling these issues could represent a qualitative improvement on gaming experience from the player perspective and academic research on AI has good opportunities to provide solutions to these challenges [3,9].

Adaptiveness in games has been already explored in academic AI research. However, existing approaches are either focused on individual reasoning [5,6], or do not take into account high-level definitions that would allow for reasoning *why* to make a particular decision on a specific context [12]. These approaches can get advantage of ALIVE by extending individual agents’ reasoning cycle with *organisational awareness*.

Organisational frameworks such as OperA [4] are already being explored for their use in *serious games*. In [13], organisational specifications are used to create a distributed intelligent task selection system that adapts to the player skill level and to model the storyline. With our work we intend to advance on this line of work by generalising the use of organisational models for *fun games*, more focused on the realism of gaming experience, rather than on user modelling and learning.

There are already examples showing that higher levels of abstraction can be successfully used in commercial games’ AI. Actually, some recent important commercial games such as *F.E.A.R.* [10], *Fallout 3*, or *Empire: Total War*, have started to apply more complex cognitive patterns by using *GOAP* (Goal-Oriented Action Planning), a simplified and optimised version of *STRIPS* that allows for real-time planning of actions with pre- and post-conditions, even outperforming *Finite State Machine*-based algorithms in some scenarios [7]. Thus, these games

execute complex symbolic reasoning not only about *how* to execute certain actions, but also about *what* to execute at each moment. We believe that, by using an even higher level of abstraction in order to reason also about *why* actions have to be performed, methods such as GOAP can be complemented and improved.

## 6 Conclusions

Our research intends to address a common problem of commercial Game AI solutions by providing an approach based on the integration of an organisation theoretical control system for NPC. We suggest that this combination contributes to Game AI solutions by providing an adaptive, extensible and flexible solution to game development industry.

The main advantage of this approach is that now developers can specify NPCs' behaviour in terms of "why" they should do something, not only "what" and "how" to do it. That is, the actors in a game are described as an organisation which behaviour is based on specific roles, norms, dependencies, and capabilities (services). Our aim is to provide a methodology and tools for Game AI developers to model gaming scenarios using social structures.

We proposed an architecture for the integration of Project ALIVE's organisation specification and coordination framework to existing commercial games. We propose the introduction of a middleware 'Game Engine Interface' that proxies information in two-ways: from the game environment to the ALIVE-based Game AI component, allowing developers to plug the proposed solution to existing games, as long as the basic interface methods to control NPCs actions are available. We demonstrate the architecture, steps and expected improvements of promoting this solution in three representative commercial games.

We conclude that this approach contributes to the issues of improved behaviour control and advanced strategy techniques, tackling some of the main issues in Game AI solutions, by providing:

- open specifications where NPCs are programmed in terms of *why* they must act in a certain way;
- enhanced flexibility and adaptiveness by describing NPC's behaviour based on specific roles, norms, dependencies, and capabilities;
- more "natural behaviour" as NPC will act autonomously, respecting environmental conditions and organisational objectives that will be perceived as "natural" by human player's perception;
- broader behaviour range, as NPCs' behaviour is more autonomous and driven by overall organisation objectives, and;
- improved reusability, as the proposed solution is generic and can be attached to a variety of existing commercial games through a common interface and customised organisation models.

As future work, we intend to complete our implementations and extend the models with realistic social descriptions. We are also analysing the integrability and extension of our approach by exploiting the integration to other commercial games, such as World of Warcraft, and The Sims 3.

**Acknowledgments.** This research has been funded by the FP7-215890 ALIVE project, funded by the European Commission. Javier Vázquez-Salceda's work has been also partially funded by the "Ramón y Cajal" program of the Spanish Ministry of Education and Science.

## References

1. Aldewereld, H., Alvarez-Napagao, S., Dignum, F., Vázquez-Salceda, J.: Making Norms Concrete. In: Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (2010)
2. Alvarez-Napagao, S., Cliffe, O., Vázquez-Salceda, J., Padget, J.: Norms, Organisations and Semantic Web Services: The ALIVE approach. In: Workshop on Coordination, Organization, Institutions and Norms at MALLOW 2009 (2009)
3. Charles, D.: Enhancing gameplay: Challenges for artificial intelligence in digital games. In: Proceedings of the 1st World Conference on Digital Games (2003)
4. Dignum, V.: A model for organizational interaction: based on agents, founded in logic. PhD Thesis, Utrecht University (2004)
5. Laird, J., Newell, A., Rosenbloom, P.: SOAR: An architecture for general intelligence. Carnegie-Mellon University Technical Report (1987)
6. Leite, J., Soares, L.: Evolving characters in role playing games. *Cybernetics and Systems* (2006)
7. Long, E.: Enhanced NPC behaviour using goal oriented action planning. PhD Thesis, University of Abertay-Dundee (2007)
8. Millington, I., Funge, J.: *Artificial Intelligence for Games*. Morgan Kaufmann, San Francisco (2009)
9. Nareyek, A.: Game AI Is Dead. Long Live Game AI! *IEEE Intelligent Systems* (2007)
10. Orkin, J.: Three states and a plan: the AI of FEAR. In: Proceedings of the 2006 Game Developers Conference (2006)
11. Quillinan, T.B., Aldewereld, H., Wijngaards, F.N., Brazier, F., Dignum, F., Dignum, V., Penserini, L.: Developing Agent-based Organizational Models for Crisis Management. In: Decker, K.S., Sierra, C., Castelfranchi, C. (eds.) Proceedings of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), pp. 45–52 (2009)
12. Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I.: Adaptive game AI with dynamic scripting. *Machine Learning* (2006)
13. Westra, J., van Hasselt, H., Dignum, V., Dignum, F.: On-line Adapting Games using Agent Organizations. In: *IEEE Symposium on Computational Intelligence and Games (CIG 2008)* (2008)



# Building Quests for Online Games with Virtual Institutions

Gustavo Aranda<sup>1</sup>, Tomas Trescak<sup>2</sup>, Marc Esteva<sup>2</sup>, and Carlos Carrascosa<sup>1</sup>

<sup>1</sup> Universidad Politécnica de Valencia  
Departamento de Sistemas Informáticos y Computación  
Camino de Vera, s/n – 46022 Valencia, Spain  
{garanda,carrasco}@dsic.upv.es

<sup>2</sup> Artificial Intelligence Research Institute (IIIA)  
Spanish Scientific Research Council (CSIC)  
Campus Universitat Autònoma de Barcelona  
08193 Bellaterra, Catalonia, Spain  
tomi.trescak@gmail.com, marc@iia.csic.es

**Abstract.** This document describes how to re-purpose an existing agent technology called Virtual Institutions as a mechanism to define new “quest” elements in Massively Multiplayer Online Games based on Multi-Agent Systems. Quests are a very important part of most Massive Online Games as they wield to flow and narrative of the game in a linear or non-linear manner.

## 1 Introduction

Massively Multiplayer Online Games (MMOGs) are an important focus of research, not only because they are economically attractive, but also because a MMOG involves many fields and a large amount of data that is generated by the interactions of many individuals: configuring a MMOG is a relevant source of research. In the field of AI, to model such systems and its dynamics is nowadays a very relevant task [17,11].

Massively Multiplayer Online Games, by nature, are played “in the cloud”, i.e. in a virtual world away from the players’ computers and game devices that are hosted in a large array of servers. The complex and distributed nature of these kind of games makes them impossible to be played relying just in the players’ hardware and resources, as some other types of online games do. This distributed nature is also one of the many parallel factors between MMOGs and Multi-Agent Systems, as it has been described in some previous works [13].

Also, MMOGs usually spot a very open-ended nature and narrative, basically allowing the players to roam the virtual game world free doing largely whatever they want to do. However, these games also use quite often the concept of **quest** or instance: A quest is a specific mission designed to be fulfilled by the players of a game. It is played in a different flow than the open-ended virtual game world, a more straightforward and linear flow, similar to the flow of classic offline games. This mission may include the involvement of other characters in the game

(corresponding to players or not) and different sequences of actions to fulfill in different places in the game.

For example, suppose a player is playing a science fiction game (similar to the well-known “Star Wars Galaxies”<sup>TM</sup> or “Star Trek Online”<sup>TM</sup> games) in which players may travel freely with a spaceship through the cosmos. Players may encounter another character in the game, a computer-controlled character called the “quest-giver” which gives them the opportunity to embark on a singular quest looking for a specific item on a planet. The moment the players accept this endeavor, a new linear narrative opens just for them (and their potential companions), and a new set of sequential goals and rewards becomes available for the players.

This paper presents a new addition to the existing architecture of *MMOG based on Multi-Agent Systems* [3]: the making of quests using Virtual Institutions [8], detailing, not only the architecture (ontology and agent taxonomy), but also a prototype applied to a concrete game example.

In section 2 Virtual Institutions are presented. Later, in section 3 MMOG based on MAS architecture is also presented. Following, the concept of quest in this kind of systems is introduced in section 4. Additionally, section 5 explains how to develop these quests using VI technology. Finally, some conclusions and future lines of work are presented in section 6.

## 2 Virtual Institutions

Virtual Institutions are a 3D Virtual Worlds with normative regulation of interactions [8]. This concept appeared as a combination of electronic institutions [12] and 3D virtual worlds. In this context, electronic institutions are used to specify the rules that govern participants’ behaviors, while 3D virtual worlds are used to facilitate human participation in the institution. The design of Virtual Institutions is divided in two separate steps: i) specification of the institutional rules, and ii) generation of the virtual world.

The institutional rules are defined using Electronic Institution model composed by the following components:

- *Dialogical Framework*. It defines a common ontology and communication language to allow humans with different cultural backgrounds, as well as, agents to exchange knowledge. This ontology and language for humans will be further transformed into actions that are allowed to be executed in the Virtual World. Those actions are connected to 3D models in the environment, the affordances of which will help in eliminating the cultural barrier. Due to the further provided translation of the communication language into actions and vice-versa, the agents will be able to interact with humans and understand their actions. The dialogical framework also fixes the organizational structure of the society, that is, which roles can participants play, and relationships among them.
- *Scene*. Interactions between agents in order to jointly perform an activity are articulated through agent group meetings, which we call *scenes*, with a

well-defined communication protocol. We consider the protocol of a scene to be the specification of the possible dialogues agents may have. Hence, a scene is specified as a deterministic finite automata, whose states represent interaction states, while arcs are labelled with messages (illocutions) or timeouts. Notice however that the communication protocol defining the possible interactions within a scene is role-based instead of agent-based. In other words, a scene defines a role-based framework of interaction for agents.

- *Performative Structure*. Scenes can be connected, composing a network of scenes, that we call performative structure, to capture the relationships among scenes. The specification of a performative structure contains a description of how agents can legally move from scene to scene by defining both the pre-conditions to join and leave scenes. Satisfying such conditions will fundamentally depend on the roles allowed to be played by each agent and its acquired commitments. The execution of a performative structure equates to the execution of the multiple, possibly simultaneous, ongoing activities, represented by scenes, along with the agents participating in each activity. Agents within a performative structure may be possibly participating in different scenes, with different roles, and at the same time.
- *Norms*. They determine the consequences of user actions. These consequences are modeled as commitments that participants acquire as a consequence of their actions and have to fulfill later on. These commitments may restrict future activities of the users.

Furthermore, for each role activity and performative structure the designer can define an information model, which is composed of a set of attributes that will be used to keep the state of the agent or an activity. For instance, the information model of a player can contain its credit, points, or the objects it have. The values of such attributes are modified depending to the evolution of the institution. That is, when an action is executed some of the attributes are modified. For instance, the points of a player can be increased after successfully completing a quest.

Once the institutional rules have been specified it is time to generate the virtual world. This can be automatically done taking into account the activities can engage on defined in the specification. Specifically, a 3D room is represented for each activity (scene). As a result a mapping is created between the activities defined in the specification, and where these activities occur within the virtual world. In addition, messages specified in scene protocols are mapped to actions supported by the virtual world. For instance, in the context of an auction house raising a hand can be mapped to the message for submitting a bid.

In contrast to Electronic Institutions, the normative part of a Virtual Institution does not represent all the actions that are allowed to be performed in a Virtual World. Specifically, those actions that require institutional verification are those mapped to scene messages. The rest of the actions provided by the virtual world software can be freely executed. The specification of the institutional rules can be regarded as valid sequences of actions among the ones that require institutional verification. In addition, the attributes associated to the

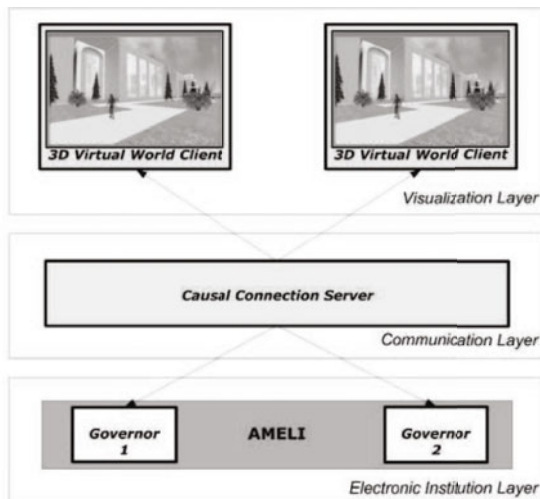


Fig. 1. Runtime Architecture

different roles, and activities help to keep participants or activities state. They keep information of past actions which are relevant to determine the validity of future actions.

Virtual Institutions are deployed by a 3-layered infrastructure presented in Figure 1.

First layer is the *Electronic Institution Layer*. It uses the AMELI system [4] for enforcing the institutional rules established on the specification step. AMELI keeps the execution state of the institution and uses it along with the specification to guarantee that participants' actions do not violate any of the institutional constraints.

Second layer is the *Communication Layer*. Its task is to causally connect [16] the institutional infrastructure with the visualization system transforming the actions of the visualization system into the messages, understandable by the institutional infrastructure and the other way around. This causal connection is done via the Causal Connection Server using the mapping between institutional messages and virtual world actions. The causal connection is happening in the following way: an action executed in the 3D Virtual World (that requires institutional verification) results in a change of the institutional state in the AMELI layer, as well as every change of the institutional state is reflected onto the 3D Virtual World and changes its state. The Communication layer conceptually and technologically connects two metaphors: Electronic Institutions and Virtual Worlds and we see it as one of our major scientific contributions.

The third layer is called *Visualization Layer*. It is used to visualize the 3D Virtual World for the users.

### 3 Massively Multiplayer Online Games Based on MAS

Deploying a game like a MMOG is like deploying a major software project to act as a service in the cloud. It presents all the issues and hazards one could expect from deploying a large software system to solve a big and distributed problem: need for good scalability, distribution of knowledge, user load balance, network bottlenecks, long development cycle and asynchronous events, just to name a few.

There have been other approaches to use agents in online gaming, albeit not exactly in the MMOG space. Most of them are oriented towards achieving better behaviors in Non-Player Characters. Dignum et al. [10] propose a more natural (long-term) behavior of Non-Player Characters through the use of Multi-Agent Systems, and clarify that game design should be adjusted to incorporate the possibilities of agents early on in the process, a statement also fundamental to this line of research. Also, Gemrot et al. [15] take a more practical approach by developing a full framework, called *Pogamut*, to integrate distributed intelligent agents as synthetic opponents and allies (bots) in games powered by the “Unreal Engine”™ technology. The main objective of the *Pogamut* project is to provide new AI-driven players that can bestow new challenges to the players and learn from their actions, using a distributed AI network that runs outside of the game clients and server. Both approaches take online gaming in general as a domain for agents achieving good results, so it is a natural step forward for agent technology to enter the MMOG space.

#### 3.1 Architecture

A MMOG (like most complex systems) can be seen as a system split into several layered subsystems, with each layer being relatively independent and taking care of one aspect of the whole MMOG experience. From the perspective of this work, a MMOG is split into three layers:

**HCI Layer:** It is the *client-side* of the system, the part of the game running on the players hardware (PC, mobile phone, game console, ...). It is the user interface that the game provides to the player, i.e. the game *client* software, and it provides the player with a gaming experience (i.e. 3D graphics, sound...). It is the framework of the *InterfaceAgent* [2].

**Intelligent Virtual Environment (IVE) Layer:** It is the virtual representation of the game environment itself. It is part of the *server-side* of the system, the part of the game that runs mostly on the game provider’s hardware, a controlled environment. The synthetic *place* and scenario where the game takes place: the virtual world. This world is independent of the type of game or simulation it must give support. Also, the IVE is designed following an agent-based approach, so it can be seen as Multi-Agent System embedded into another, larger, Multi-Agent System. The IVE layer is thoroughly described in [6].

**MMOG Layer:** It is a complex subsystem where all the game logics and mechanics are implemented and must be solved at run-time. It operates in conjunction with the IVE layer, but it is not dependent of that subsystem. It implements

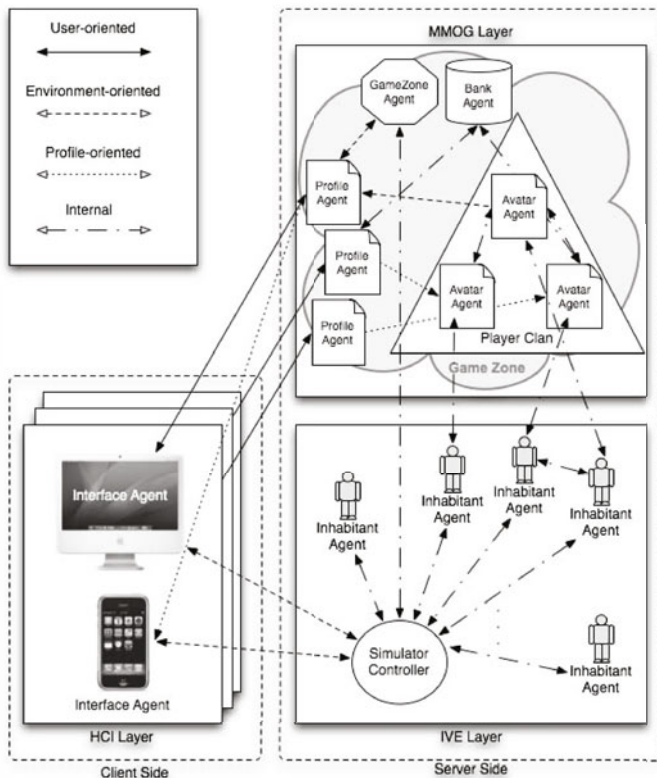


Fig. 2. The architecture of a MMOG based on MAS

the game rules / norms controlling the game development. It is the *place* where all the game clients connect to play and, along with the IVE layer, it must facilitate game server scalability. In this line of research, this subsystem is seen as the core of the MAS and requires, at least, one agent platform as its foundation. The MMOG Layer is the place that groups all the elements from the game which are independent both from the user space and the IVE, and thus is the core piece of the whole system, as it is where the actual game takes place. Section 3.2 describes this layer in more detail.

### 3.2 The MMOG Layer

As stated before, the **MMOG Layer** is essentially a dedicated, open MAS which runs the game. This MAS uses agent technologies like agent services, Electronic Institutions [13,7] and Agent Organizations [14,9,5] to model some game mechanics, and translates the common issues and situations found in MMOGs into problems that can be solved using classic software agent features, such as agent

interactions, agent communication protocols (like auctions or call-for-proposals), service-oriented computing, event-driven behaviors or role models.

Like any other agentification process, one of the key ideas is to identify the agents and types of agents that will conform the system. In this case, the agents are based on the concepts and entities that form the whole game experience of a MMOG, and are explained in more detail in [2]:

**ProfileAgent:** a personal agent which manages the player status and profile within the game community. It manages the player’s preferences in the game world, which avatars the player uses and the **role** that the player plays in the system (*Spectator*, *Player* or *GameMaster*).

**AvatarAgent:** an agent which represents an avatar of a human player within the game (a PC or Player Character). It is a persistent kind of agent: is not deleted and re-spawned often, it bears a long life cycle. It is the agent that holds the PC *stats* (server-side), and so, a malicious player cannot modify them locally (cheat). The AvatarAgent is the kind of agent that actually performs the actions for the player in the virtual world.

**NPCAvatarAgent:** an agent which represents an avatar of an AI-controlled character. It is similar to the AvatarAgent, as both populate the game world, but it does not obey nor represent a player in the game.

**GameZoneAgent:** a kind of agent which implements the logics of the game environment and works as a nexus between the *MMOG Layer* and the IVE Layer’s Simulation Controller (see figure 2).

## 4 Defining Quests for MMOG

Quests are a very important part of most Massive Online Games as they wield to flow the narrative of the game in a linear or non-linear manner. Quests present the players with the opportunity to improve their virtual characters and their playing experience by grouping together players with the same objectives and guiding them through a segment of the overall game experience, rewarding players for their performance in the game. Quests also offer the designers of these open-ended games, an opportunity to develop more narrow-focused “levels”, similar to those found in traditional offline games, without sacrificing the social aspects of online play and the overall goals of the game. Alas, the whole “career” of a virtual character can be seen as a series of linked quests towards an open-ended conclusion.

From the grand perspective of the game as a big Multi-Agent application, quests are a part of the system. They are seen as smaller Electronic Institutions with a semi-linear flow of the agents where custom and more strict norms exist than those from outside (those in the “open” areas of play). However, quests present some particular aspects that are not seen in other types of institutions.

Let’s present the concept of “sub-quest” in the context of quests, which has a lot in common with the concept of scene in Electronic Institutions. A sub-quest

is one of the steps that compose a quest. A sub-quest can be defined as a set  $S = (G, O, L, P, E)$ , where:

- **G**: is the set of objectives (goals) player agents (P) must complete in the sub-quest. These objectives are expressed in the semantic ontology of the game itself using the OWL-DL language, which in turn is based on “MMOG Ontology” [31]. This set of goals cannot be empty, there must be at least one goal to play in the sub-quest.

Goals are defined using the properties and data types present in the ontology, and usually make reference to modifying the state or the properties of an object of the game. For instance, let’s presume a game uses an ontology in which the game characters (i.e. Avatars) have a numeric “health” stat. When a designer wants to create a goal that means “kill this enemy”, it can be defined by declaring the “health” stat of the instance of NPCAvatar that represents that enemy to zero:

```
Goal0: Avatar MyEnemy.health == 0.0
```

Another example: let’s say a designer wishes to express a goal that means “take this chest to the vault”, it can be expressed by the *placedAtZone* property of the *chest* item:

```
Goal1: GameItem MyChest.placedAtZone == "Vault"
```

Initially, goals should not need to be decomposed into subgoals, but this is a feature that may be added in future iterations of this work.

- **O**: is the set of system agents (opponents) that oppose player agents (P) and keep them from completing the sub-quest by a process of conflict. Their view is the opposite of the player agents’ view and its actions counter those of the players. However, contrary to classic game theory, these agents do not seek the Nash equilibrium [18] in the system of the quest. They seek to maintain the goals (G) in the initial state of the sub-quest, that is, without being fulfilled. This may be an empty set (i.e. no opponents).
- **L**: is the set of virtual locations (also called *Dungeons* and *Game Zones* in the literature) that serve as the environment around players (P) and the opponents (O). They are comparable to the rooms and transitions present in Electronic Institutions as they each have a maximum (and a possible minimum) concurrent agents of a particular type and in each one of the locations changes according to an interaction protocol, although the same protocol can be shared by more than one location. This may be an empty set, meaning that the sub-quest is not tied to a specific location.

Locations can be specified using lists of derivatives of the *GameZone* and *GameBeacon* classes of the “MMOG Ontology”. The designers of the game are expected to define and tag the virtual places where the game takes place using instances of those classes (or some sub-classes). Therefore, in practice, each L set will be a sequence of some of those instances, and the same instance may appear in different sub-quests (or quests) for different purposes.

<sup>1</sup> <http://gti-ia.dsic.upv.es/ontologies/mmog.owl>



- **P**: is the set of agents who play the role of players. These agents are usually controlled by human players that are playing the game. Sometimes an agent of the group **P** can be controlled by the platform, but it is rare. The aims of these agents are twofold: make sure the goals of the scene (**G**) are fully accomplished and maximize the profits (**E**) during the process. Ideally, an organization of player agents (**P**) wishes to complete all of the objectives of a quest and to get the maximum number of potential profits. Player Agents (**P**) are the antagonists of the Opponent Agents (**O**). This set cannot be empty, there must be at least one agent that plays the sub-quest.
- **E**: the function of earnings player agents (**P**) can obtain during the quest. In each sub-quest, this function changes to reflect the ratio of risk / profit present. Based on this ratio, the quests can be dimensioned. For example, a quest with a very large **O** component and a very low **E** component is seen as a “High Risk and Low Gain” quest and is less desirable for the players than a quest with a higher **E** component and lower **O** component, which would be “Middle Risk and High Gain”. For a quest that has no earnings, this function can be equal to zero.

So then, a quest can be defined as a non-linear (or non-deterministic) sequence of connected sub-quests linked through outcomes. These outcomes are determined by the completion (or failing) of the goals in each sub-quest. Specifically, a quest can be seen as a directed acyclic graph where the nodes are sub-quests (as can be seen in figure 3). The designers of the quest must also specify which sub-quest (or sub-quests) serve as an entry point to the quest and which sub-quest (or sub-quests) serve as an exit point (i.e. end the quest). So, in the end, a quest can be defined in a very similar fashion to a non-deterministic finite automata:

$$Q = (SS, OC, I, F)$$

- **SS**: The set of sub-quests that compose the quest. The nodes of the graph.
- **OC**: The outcomes that connect the sub-quests. The strings of the graph.
- **I**: The subset of **SS** that are starting sub-quests for the quest.
- **F**: The subset of **SS** that are ending sub-quests for the quest.

Quests are semi-linear structures in nature. That means that a quest can follow a straight path from beginning to end, or that it can branch its path one or more times through its course. Nevertheless, every path that the players take will eventually lead them to ending the quest in one of the ending sub-quests of the quest. This gives the designers of the quest some sort of “elasticity” towards the development and narrative of the quest, as well as the ability to add some interesting gameplay elements.

For instance, suppose that a quest involves the players getting inside a locked room to retrieve an object (i.e. a treasure). The designer can create a sub-quest located in a contiguous room where an opponent (which has the door key) is guarding the entrance to the room. The designer may also create another sub-quest with no opponents that takes place in a backyard where a window leads

to the important room. Both sub-quests have similar objectives (i.e. get inside the important room), but they are presented as a choice to the players: Will the players take the indoor or outdoor path? If they take the indoor path, will they fight the opponent to get the key by force or will they try to bribe or sweet-talk the guard into give them the key? If they take the outdoor path, do they have the ability to open and jump off the window from the outside into the room? These are all gameplay choices that the developer may present to the players and implement them as branching paths in the quest design, but in the end, all of these choices lead to the same conclusion: the achievement of the ultimate quest goal and the completion of the quest.

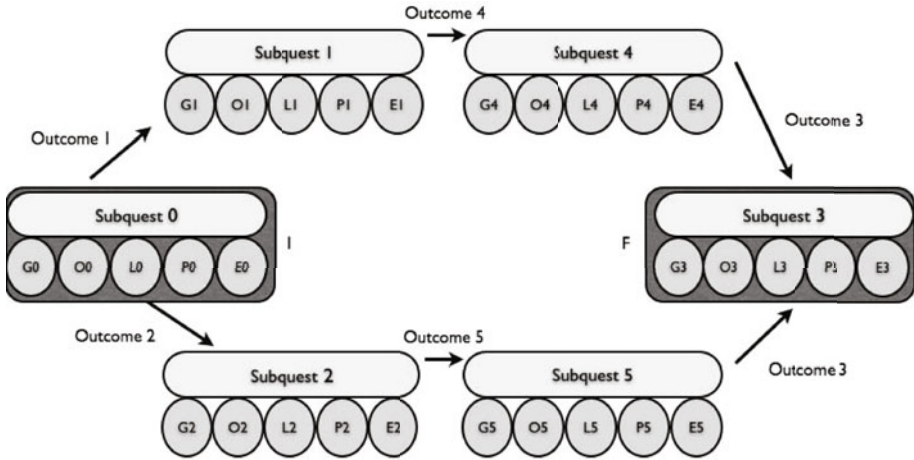


Fig. 3. Components of a quest

## 5 Building Quests with VIs

In this section the use of the VI framework and tools to develop quests for MMOG based on MAS is explained (specially using the Islander [12] editor). The initial approach followed in this work takes a formal specification of a quest and expresses it in practice using a VI. The transformation is implemented using the Islander tool from the Electronic Institutions Development Environment [4]. The Islander tool was developed as a user-friendly graphic interface for specifying Electronic and Virtual Institutions. In this work, it has been used in the first steps of developing quests for MMOG based on MAS (what is usually called the development *pipeline* in the videogame industry). In order to successfully implement a quest, the following points need to be taken care of: be able to express the knowledge using an ontology (this basically includes goals and earnings); be able to control the flow of the agents participating in the quest and the roles they play (this basically includes players and opponents); be able to specify the

flow and connections of the sub-quests; be able to express the different outcomes of a sub-quest; and be able to specify the sub-quests themselves.

Regarding the use of an ontology, the Islander tool allows the definition of custom ontologies with classes and properties, like the MMOG Ontology, that are used in the definition of the Performatives Structures and Dialogic Frameworks. So this need is fulfilled.

Regarding the agents, VI provide a unique type of internal agents called the Governors. These Governors are paired with the agents participating in the institutions and act as their real interface towards the system, preventing them from executing illegal actions based of their roles and characteristics (or stats, in MMOG terms). Besides, Governors “move” the agents through the scenes and transitions of the institutions as they interface with them. So this need is fulfilled.

Regarding the flow and connections of the quest, it’s worth noting that the main element of a VI is its Performative Structure which, as seen in section 2, is essentially a directed graph that connects scenes through transitions. Although they are different concepts, a Performative Structure and a quest share the same kind of graphic representation. In fact, a quest may be seen as a subset of a whole Performative Structure, which contains all the possible sub-quests and all the possible paths that any agent can follow through the quest at any given time (or attempt).

Regarding the possible outcomes of a sub-quest, VIs use diagrams called “*Protocols*” to define the inner workings of a scene. Protocols are essentially simplified nondeterministic finite automata, with an arbitrary number of starting and final states, and where the state changes are triggered by common interactions between the agents of the scene or timed events. So, the different outcomes of a scene can be directly mapped to the final states of the automaton. A scene will have at least as many outcomes as final states has its inner Protocol (since more than one final state can lead to the same outcome). So this need is fulfilled.

Regarding the sub-quests, the goals (G), opponents (O), players (P) and earnings (E) have already been explained, as well as the outcomes. The location is the missing item. Locations are normally defined logically through the use of the MMOG Ontology (or one of its derivatives) by using the *GameZone* and *GameBeacon* classes and their possible subclasses. In VI there is not a explicit “location” field associated with a scene of the Performative Structure. Fortunately, these scenes may have as many additional *properties* as needed. A property is a semantic “key-value” pair (which may be mandatory to define in each scene) where the “value” part is a semantic expression (or a list of semantic expressions). By creating a “location” property in each scene that needs locations, this need is partially fulfilled. The other half of the question is the connection between the VI location and the actual representation of the virtual place in the IVE layer. This is done through the Causal Connection Server (as seen in figure 1), but this problem falls outside the scope of this work and will be explained in detail in future articles.

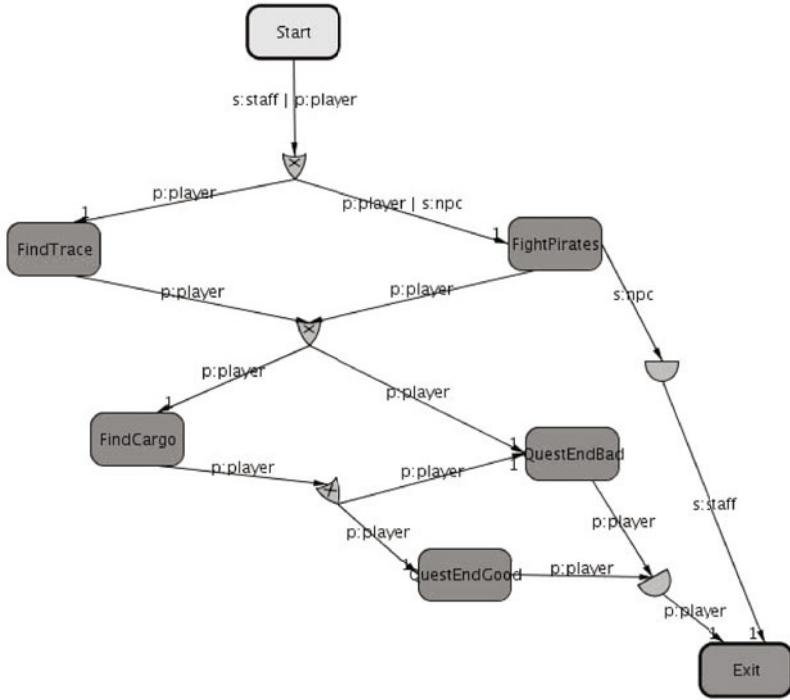


Fig. 4. Performative Structure of a quest as seen in Islander

In order to better explain this process, an example quest has been developed. This example quest introduced for this work is depicted in figure 4 and takes place in a fictional *science-fiction* MMOG game similar to the already mentioned examples in section 1. This quest follows the flow of a deed in which the players must track down a treasure stolen by space pirates. Players first receive the information of the deed by a non-player character (an agent of the class NPCAvatarAgent), which plays the role of “quest-giver” prior to accepting the quest (and entering the VI). This information presents the problem to the players (“*The evil space pirates have stolen a treasure. Their last known location are the ruins of an ancient base, but they are rumored to be in the orbit of a nearby planet.*”) and the players must decide whether or not to embark on the quest. If so, the players are faced with an immediate choice: they can go to the ruins to look for clues or they can go to the nearby planet to confront the pirates. These two different paths correspond with a gameplay choice that the game designers wish to present the players with: will they use subtlety and insight or will they use brute force and a direct approach? Either way, players will eventually learn the true location of the treasure (the drifting remains of an old ship) and proceed there to try to find it. If they succeed, all ends well and the players “win” the quest and retrieve the treasure. If they are not able to find it, the quest has a bitter end as the players “fail” the quest and obtain no treasure whatsoever.

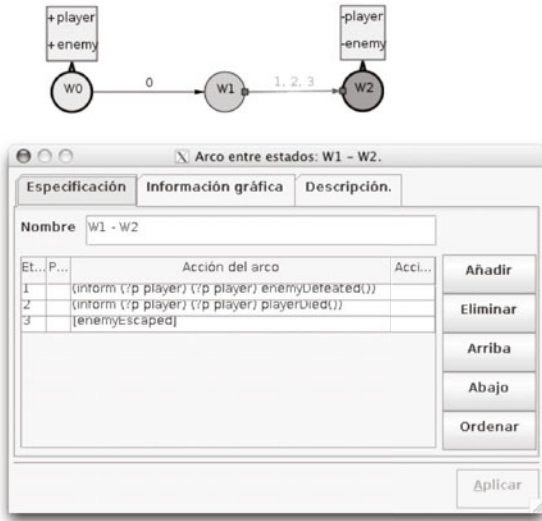


Fig. 5. FightScene Protocol

This quest has three types of scene: *FightScene*, with a protocol designed for the fighting; *FindScene*, for searches and tracking treasure; and *QuestEndScene* to resolve the end of the quest. The protocols of the scenes are quite simple. For example, *FightScene* is described in figure 5.

Internally, when the players accept the quest their AvatarAgents enter the VI. And the different gameplay branching choices are represented by different sub-quests and paths in the quest definition, very similar to what can be seen in figure 3. When the agents enter the VI, the first choice is represented by a XOR transition ( $t0$ ) which either leads them the “FindTrace” scene or to the “FightPirates” scene.

If the players choose the first path, their agents transition to the “FindTrace” scene. In this scene, the goal is to find the clue (or clues) to the real location of the treasure. When the clue is found, the scene ends and its outcome brings the players to the next transition ( $t1$ ).

However, if the players chose to confront the pirates, their agents are lead to a scene called “FightPirates”. This scene may end in two ways: the players beat the pirates and obtain the location of the treasure by force, or the pirates kill the players and escape with the treasure. This dichotomy is represented by two possible outcomes, both passing through the  $t1$  transition: the first one is the normal flow of the quest and leads to the “FindCargo” scene. The second one, terminates the quest and leads to the “QuestEndBad” scene (and later to the “Exit” scene) of the institution.

When the agents arrive at the “FindCargo” scene, they have the chance to look for the hidden treasure, but they have to find it in a short amount of time. That limitation is represented using a timer in the protocol of the “FindCargo”

scene. When the timer expires, the protocol ends with an outcome. However, if the players find the treasure, the protocol ends with a different outcome. If the players are unable to find the treasure, the quest ends in a negative way through the “QuestEndBad” scene. Nevertheless, if the players are able to find the treasure, the quest ends and the players receive a positive reward in their “QuestEndGood” scene. After that, the players’ agents leave the institution and so the quest ends.

## 6 Conclusions and Future Work

In this work, a new addition to the existing architecture of *MMOG based on Multi-Agent Systems* [3] has been presented and successfully implemented: the making of quests using Virtual Institutions [8], detailing, not only the architecture (ontology and agent taxonomy), but also a prototype applied to a concrete game example.

In the future of this line of work lie at least two new developments. The first one is to define all the possible interactions that can happen between agents populating a MMOG based on MAS and to integrate that knowledge into the definition of quests, in order to have better control and configuration of sub-quests based on agent interactions. The second one is to develop a methodology, addressed to game developers, to guide them in the use of this architecture for building their games.

## Acknowledgements

This work has been partially funded by TIN2009-13839-C03-01, TIN2008-04446, PROMETEO/2008/051, GVPRE/2008/070 projects, CONSOLIDER-INGENIO 2010 under grant CSD2007-00022, project EVE (TIN2009-14702-C02-01), EU-FEDER funds, the Catalan Government (Grant 2005-SGR-00093) and Marc Esteva’s Ramon y Cajal contract.

## References

1. Aranda, G., Botti, V., Carrascosa, C.: Mmog based on mas: The mmog layer (extended abstract). In: Decker, S., Sierra, C. (eds.) Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), pp. 1149–1150 (2009)
2. Aranda, G., Carrascosa, C., Botti, V.: Characterizing Massively Multiplayer Online Games as Multi-Agent Systems. In: Corchado, E., Abraham, A., Pedrycz, W. (eds.) HAIS 2008. LNCS (LNAI), vol. 5271, pp. 507–514. Springer, Heidelberg (2008)
3. Aranda, G., Carrascosa, C., Botti, V.: The MMOG layer: MMOG based on MAS. In: Dignum, F., Bradshaw, J., Silverman, B., van Doesburg, W. (eds.) Agents for Games and Simulations. LNCS, vol. 5920, pp. 63–78. Springer, Heidelberg (2009)
4. Arcos, J.L., Esteva, M., Noriega, P., Rodríguez-Aguilar, J.A., Sierra, C.: Engineering open environments with electronic institutions. Journal on Engineering Applications of Artificial Intelligence 18(2), 191–204 (2005)

5. Argente, E., Palanca, J., Aranda, G., Julian, V., Botti, V., Garcia-Fornes, A., Espinosa, A.: Supporting agent organizations. In: Burkhard, H.-D., Lindemann, G., Verbrugge, R., Varga, L.Z. (eds.) CEEMAS 2007. LNCS (LNAI), vol. 4696, pp. 236–245. Springer, Heidelberg (2007)
6. Barella, A., Carrascosa, C., Botti, V.: Agent architectures for intelligent virtual environments. In: 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp. 532–535. IEEE, Los Alamitos (2007)
7. Bogdanovych, A., Berger, H., Simoff, S.J., Sierra, C.: Narrowing the Gap between Humans and Agents in E-commerce: 3D Electronic Institutions. In: Bauknecht, K., Pröll, B., Werthner, H. (eds.) EC-Web 2005. LNCS, vol. 3590, pp. 128–137. Springer, Heidelberg (2005)
8. Bogdanovych, A., Simoff, S.J., Esteva, M.: Virtual institutions prototype. In: 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10-15, vol. 2, pp. 1373–1374 (2009)
9. Criado, N., Argente, E., Julian, V., Botti, V.: Organizational services for spade agent platform. In: IWPAAMS 2007, vol. 1, pp. 31–40. Universidad de Salamanca (2007)
10. Dignum, F., Westra, J., van Doesburg, W.A., Harbers, M.: Games and Agents: Designing Intelligent Gameplay. In: International Journal of Computer Games Technology, 2009 (2008)
11. Ducheneaut, N., Yee, N., Nickell, E., Moore, R.: Alone together?: exploring the social dynamics of massively multiplayer online games. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 407–416 (2006)
12. Esteva, M.: Electronic Institutions: From Specification to Development. PhD thesis, Artificial Intelligence Research Institute (IIIA-CSIC), Spain (2003)
13. Esteva, M., Rosell, B., Rodriguez-Aguilar, J., Arcos, J.: AMELI: An Agent-Based Middleware for Electronic Institutions. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, vol. 1, pp. 236–243 (2004)
14. Garcia, E., Argente, E., Giret, A.: Issues for organizational multiagent systems development. In: Sixth International Workshop From Agent Theory to Agent Implementation, AT2AI-6 (2008)
15. Gemrot, J., Kadlec, R., Bida, M., Burkert, O., Pibil, R., Havlicek, J., Zemcak, L., Simlovic, J., Vansa, R., Stolba, M., Brom, C.: Pogamut 3 Can Assist Developers in Building AI for Their Videogame Agents. Proceedings of the First International Workshop on Agents for Games and Simulations (2009)
16. Maes, P., Nardi, D.: Meta-Level Architectures and Reflection. Elsevier Science Inc., NY (1988)
17. Matskin, M.: Scalable Agent-Based Simulation of Players in Massively Multiplayer Online Games. In: Eighth Scandinavian Conference on Artificial Intelligence, SCAI 2003 (2003)
18. Nash, J.: Non-cooperative games. *The Annals of Mathematics* 54(2), 286–295 (1951)

# Author Index

- Alvarez-Napagao, Sergio 179  
Aranda, Gustavo 192
- Banerjee, Bikramjit 53  
Behrens, Tristan 1  
Brom, Cyril 19
- Cap, Michal 132  
Carrascosa, Carlos 192
- Damiano, Rossana 76  
de Rijk, Lennard 1  
Dignum, Frank 38, 117  
Dignum, Virginia 117  
Dragone, Mauro 91
- Esteva, Marc 192  
Evans, Richard 163
- Gemrot, Jakub 19  
Gómez-Sebastià, Ignasi 179
- Heuvelink, Annerieke 132  
Hindriks, Koen V. 1
- Koch, Fernando 179  
Korstanje, Rien 1  
Kraayenbrink, Nick 1  
Kraemer, Landon 53
- Li, Boyang 99  
Lombardo, Vincenzo 76
- Muldoon, Conor 91
- Niehaus, James 67
- O'Grady, Michael J. 91  
O'Hare, Gregory M.P. 91  
Orkin, Jeff 148
- Pasman, Wouter 1  
Plch, Tomáš 19
- Riedl, Mark O. 99  
Roy, Deb 148
- Trescak, Tomas 192  
Tynan, Richard 91
- van den Bosch, Karel 132  
van Doesburg, Willem 38, 132  
van Oijen, Joost 38  
van Riemsdijk, Birna 1  
Vázquez-Salceda, Javier 179
- Wan, Jie 91  
Westra, Joost 117  
Weyhrauch, Peter 67