First International Workshop on

Agents
for
Games and Simulations

Budapest

May 11, 2009

Table of Contents

Preface

Multi Agent System research offers a promising technology to implement cognitive intelligent Non Playing Characters. However the technologies used in game engines and multi agent platforms are not readily compatible due to some inherent differences of concerns. Where game engines focus on real-time aspects and thus propagate efficiency and central control, multi agent platforms assume autonomy of the agents. Increased autonomy and intelligence may offer benefits for a more compelling gameplay and may even be necessary for serious games. However, it raises problems when current game design techniques are used to incorporate state of the art Multi Agent System technology.
A very similar argument can be given for agent based (social) simulations.
In this workshop we bring people together that address the particular challenges of using agent technology for games and simulations. Submissions were invited for the following three main themes:

1. Technical:
   Connecting agent platforms to games and simulation engines; who is in control?, Are actions synchronous or asynchronous? How to monitor results of actions? Can agents communicate through the agent platform? How efficient should the agents be?
2. Conceptual:
   What information is available for the agents from the game or simulation engine? How to balance reaction to events of the game or simulation with goal directed behavior. Ontological differences between agents and game/simulation information.
3. Design:
   How to design games/simulations containing intelligent agents. How to design agents that are embedded in other systems

Of course we also welcomed papers about experiences on the use of agents in games and simulations. Both successes as well as "failures" were welcome as they both can help us better understand what are the key issues in combining agents with game and simulation engines.
We received 17 submissions of high quality covering many of the aspects mentioned above. We accepted 12 papers for presentation, which can be found in this proceedings. Among the papers we find a strong example of a description of middleware that is used to connect agents to Unreal Tournament (an extension of Gamebots). But we also have a paper describing how to design agents for games that have to behave according to bounded rationality theory in order to mimic human behavior. Several papers describe experiences with particular agent models used for games and simulations. All in all we are very happy with the papers contained in this volume. We are sure they form a valuable starting point 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 that gave valuable comments on all papers allowing for a tough selection of the best paper.

Frank Dignum, Jeff Bradshaw, Barry Silverman, Willem van Doesburg.

Organizing Committee

Frank Dignum, Utrecht University
Jeff Bradshaw, Florida Institute for Human & Machine Cognition
Barry Silverman, University of Pennsylvania/Systems Engineering Dept.
Willem van Doesburg, TNO Defence, Security and Safety

Program Committee

- Elisabeth Andre (DFKI, Germany)
- Ruth Aylett (Heriot-Watt University, UK)
- Andre Campos (UFRN, Brazil)
- Bill Clancey (NASA, USA)
- Rosaria Conte (ISTC-CNR, Italy)
- Vincent Corruble (LIP6, France)
- Yves Demazeau (CNRS-LIG, Grenoble)
- Virginia Dignum (Utrecht University, The Netherlands)
- Alexis Drogoul (LIP6, France)
- Bruce Edmonds (MMU, UK)
- Corinna Elsenbroich (University of Surrey, UK)
- Klaus Fischer (DFKI, Germany)
- Hiromitsu Hattori (Kyoto University, Japan)
- Koen Hindriks (Delft University, The Netherlands)
- Wander Jager (Groningen University, The Netherlands)
- Stefan Kopp (University of Bielefeld, Germany)
- Michael Lewis (University of Pittsburg, USA)
- Scott Moss (MMU, UK)
- Emma Norling (MMU, UK)
- Anton Nijholt (UT, The Netherlands)
- Ana Paiva (IST, Portugal)
- Michal Pechoucek (CTU, Czechia)
- David Pynadath (USC, USA)
- Geber Ramalho (Brazil)
- Gopal Ramchurn (University of Southampton, UK)
- Avi Rosenfeld (JCT, Israel)
- David Sarne (Bar Ilan University, Israel)
- Pjotr van Schothorst (VSTEP, The Netherlands)
- Maarten Sierhuis (NASA, USA)
- Pieter Spronck (Tilburg University, The Netherlands)
- Katia Sycara (CMU, USA)
- Duane Szafron (U of Alberta, Canada)
- Max Tsvetovat (George Mason University, USA)

Modeling Agents through Bounded Rationality Theories

Avi Rosenfeld[1] and Sarit Kraus[2]

[1] Jerusalem College of Technology, Jerusalem 91160, Israel
rosenfa@jct.ac.il
[2] Bar-Ilan University, Ramat-Gan 52900, Israel
sarit@cs.biu.ac.il

Abstract. Effectively modeling an agent's cognitive model is an important problem in many domains. In this paper, we explore the search agents people wrote to operate within optimization problems. We claim that the overwhelming majority of these agents used strategies based on bounded rationality, even when optimal solutions could have been implemented. Particularly, we believe that many elements from Aspiration Adaptation Theory (AAT) are useful in quantifying these strategies. To support these claims, we present extensive empirical results from over a hundred agents programmed to perform in optimization problems involving solving for one and two variables.

1 Introduction

Realistic modeling of individual reasoning and decision making is essential for economics and artificial intelligence researchers [1, 3–6, 10, 12, 13]. In economics, validly encapsulating human decision-making is critical, for instance in predicting policy effects. Within the field of computer science, it is critical for mixed human-computer systems such as entertainment domains [3], Interactive Tutoring Systems [6], and mixed human-agent trading environments [4]. In these and similar domains, creating agents that effectively understand and/or simulate people's logic is particularly critical. In both economics and computer science the perspectives of unbounded rationality based on notions such as expected utility, game theory, Bayesian models, or Markov Decision Processes (MDP) [7, 9], have traditionally been the foundation for modeling agent's behavior.

While many important insights have been gained by these perspectives, it often does not provide a descriptively correct model of human decision-making. Indeed, previous research in experimental economics and cognitive psychology has shown that human decision makers often do not adhere to fully rational behavior. For example, Kahneman and Tversky [2] have shown that individuals often deviate from optimal behavior as prescribed by Expected Utility Theory. Furthermore, decision makers often do not know the quantitative structure of the environment in which they act. But even if the quantitative structure of the environment is known to the decision maker, finding the optimal path of actions is often a problem with intractable computational complexity [8]. Thus, even in the best of circumstances, modeling behavior based on full rationality may be impractical.

A research direction called Bounded Rationality initiated by Simon [15] focuses on investigating observed rationality of individuals in decision making and problem solving. Simon presumes that people – except in the most simple situations – lack the cognitive and computational capabilities to find optimal solutions. Instead they proceed by searching for non-optimal alternatives to fulfill their goals. Simon proposes that real-world decision makers satisfice rather than optimize seeking a "good enough" solution instead of the optimal one. In this tradition, Sauermann and Selten propose a framework called Aspiration Adaptation Theory (AAT) [10, 12] as a boundedly rational model of decision making.

Recent experimental evidence from economics researchers provides support that people apply boundedly rational procedures, such as Aspiration Adaptation Theory (AAT) [10, 12], in real-world domains [11]. In this paper, we provide empirical evidence that the computer agents people write to search within optimization problems also contain several key elements from these models. This realization allows us to effectively model the decisions by these agents. Additionally, we present several guidelines by which realistic agents can be built based on this result. We begin by presenting the basics of AAT.

2 Aspiration Adaptation Theory

Aspiration Adaptation Theory was proposed by Selten as a general economic model of nonoptimizing boundedly rational behavior [10, 12]. We frame this theory within the specifics of the optimization problems presented in the next section. We refer the reader to the full paper [12] for a complete and general presentation of this theory.

As originally formulated, AAT is not a learning theory or a learning algorithm. Instead, it presents an approach by which bounded agents address a complex problem, $\mathcal{G}$. The complexity within $\mathcal{G}$ prevents the problem from being directly solved and instead an agent creates m goal variables G_1 , ..., G_m as means of solving for $\mathcal{G}$. These goals are incomparable, and no aggregate utility function, known as a substitution rate in economics, can be constructed for combining the m goals into $\mathcal{G}$. The lack of a utility function may be because the problem is simply too complex to quantify such a function, or because the agent lacks the resources to properly calculate the aggregate utility function for $\mathcal{G}$. In attempting a solution, the agent has a group of s instrument variables which represent base actions that can be used to pursue the goal variables. We define a *strategy* $x = (x_1, \ldots, x_s)$ as a combination of instrument values for the goal variables G_1 , ..., G_m. These values can and do typically change over the life of the agent. For example, assume a company has a goal $\mathcal{G}$ to be optimally profitable. Examples of goal variables might be to create more sales, create higher brand awareness or to invest in the company's production infrastructure. Here, the instrument variables may include lowering the product's price, investing more money in marketing, or hiring more skilled workers. The agent might have one strategy at the beginning of its operation (e.g. an opening sale to entice buyers) and then use a different strategy after the business matures. An action *A* is a rule for changing the strategy. Formally, we define x' = A(x) for every strategy $x_1, \ldots, x_s$. Examples of actions in this context include:

- Raising the product's price by 5%.

– Lowering the product's quality by 10%
– Lowering product's price by 10% in conjunction with raising the marketing expenditure by 15%
– Making no change to the strategy

A finite number of n actions, $A_1 \ldots A_n$ are considered. The agent can only choose one action per time frame.

Despite its lack of utility to quantify rational behavior, the model provides several guidelines for how bounded agents will behave. Agents decide about goal variables as follows: The m goal variables are sorted in order of priority, or the variables' *urgency*. Each of the goal variables has a desired value, or the *aspiration level*, that the agent sets for the current period. The agent's search starts with an initial aspiration level and is governed by its *local procedural preferences*. The local procedural preferences prescribe which aspiration level is most urgently adapted upward if possible, second most urgently adapted upward if possible, etc. and which partial aspiration level is *retreated from* or adapted downward if the current aspiration level is not feasible. Here, all variables except for the goal variable being addressed are assigned values based on ceteris paribus, or all other goals being equal, a better value is preferred to a worse one. Borrowing from Simon's terminology [15] there is only an attempt to "satisfice" the goal values, or achieve "good enough" values instead of trying to optimize them. Note that this search approach is in contrast to traditional A.I. methods such as Hill-climbing or gradient descent learning techniques which can search for optimal values of all variables simultaneously [9].

While this theory has existed since the early 1960's [10], there are few empirical studies of how well it explains observed behavior [11]. As Selten's paper states, "AAT was meant as a useful point of departure for theory construction. The theory as it stands cannot claim to be a definite answer to the problem of modelling bounded rationality. Probably, one would need extensive experimental research in order to modify it in a way which fits observed behavior." [12]

In this paper, we study how AAT provides such a point of departure for studying optimizing search agents. While several differences do exist between AAT theory and the search agents we studied, overall we found many key similarities in their search process to those within this bounded rationality theory. To the best of our knowledge, this paper represents the first study that bridges between the fields of experimental economics and computer science to demonstrate the applicability of bounded rationality theory and in describing the model used by computer agents. Our next section details the exact methodology used to study the link between these fields.

3 Research Methodology

Central to our methodology is the strategy-method [13] from experimental economics. The assumption behind this method is that people can effectively implement their own strategies within a certain period of time, without additional aids such as handbooks or other information sources. Underlying this assumption is that people will execute a task to the best of their abilities if they are properly motivated, monetarily or otherwise.

In our study, all people writing agents were upper level undergraduates (seniors), masters and PhD computer science students and were given a firm deadline of 2 weeks to complete their agents. As motivation, we told the students that their grade was based on their agent's decisions. Once these programs were written, the mental model of the person's agent can be directly evaluated from its performance. This approach is well accepted in experimental economics and it had also begun to be used within artificial intelligence research [1] as well.

In order to ensure that people were able to effectively encapsulate their own strategies, several steps were taken. First, we took care to provide students a well designed Java framework in which methods were provided for most computations (e.g. finding the average of the agent's past performance). Thus, full knowledge of Java was not necessary, as strategies were meant to be encoded in only a few lines of code. This approach mimics the strategy-method variant previously used in economics [13] where people program their own strategies. The Java language was chosen as all students had experience using this programming language in multiple previous courses. Finally, after the first 2 week deadline, we required all students to submit a "draft" agent after which we reported back to all students with the relative performance of their agents to others in the group. The students were then allowed an additional week to fix bugs in their agents, or to improve their implementation without any penalty.

We used two tools to study the agent's design. First, we studied the code of the agent itself. By analyzing the agent's logic and comments, one can often understand the search process used. Additionally, after an assignment had been completed, we distributed questionnaires to the students themselves asking them directed questions about the strategy their agent used, confirming particulars of their approach. This allowed us to definitively determine what search mechanisms were used by these agent.

Our general methodology is as follows. We first study a relatively simple optimization problem, to understand the mental model used by agents to solve the problem. Next, we study progressively more difficult problems. Eventually, we hope to reach real-world types of optimization problems. By studying progressively less difficult problems first, we believe it will be easier to understand the general behavior of these agents and the applicability of our results. In this paper, we report on the first stage of this research.

Specifically, we studied two optimization problems – a commodity search problem where the optimal solution could be found based on solving for one cost instrument variable, and a more complicated domain where the optimal solution requires solving for price and quality instrument variables. In both domains an optimal solution can be constructed, and thus bounded rationality theories such as AAT theory are potentially unnecessary. However, the optimal solution is far from trivial in these domains. Thus, these problems allow us to explore issues surrounding the strategies and heuristics implemented by people's agents and questions of performance and optimality of these agents.

3.1 Commodity Price Optimization

In the first optimization problem, we consider a problem where a person must minimize the price in buying a commodity (a television) given the following constraints. In this problem, a person must personally visit stores in order to observe the posted price of

the commodity. However, some cost exists from visiting additional stores. We assume this cost is due to factors such as an opportunity cost with continuing the search instead of working at a job with a known hourly wage. For any given discrete time period, the person must decide if she wishes to terminate the search. At this point, we assume she can buy the commodity from any of the visited stores without incurring an additional cost. The goal of the agent is to minimize the overall cost of the process which is the sum of the product cost and the aggregated cost of the search.

From a strategic point of view, the game is played under a time constraint rather than against an opponent. An optimal solution to this optimization problem can be found as an instance of Pandora's problem [16] resulting in a stationary threshold below which the search should be terminated. Formally, we can describe this problem as follows:

We assume that there is a finite timeline $\mathcal{T} = \{1, 2, ..., k\}$. In each time step t, $t \leq k$ the agent observes a cost and needs to decide whether to end the search. All of the observed costs, regardless of the time step, are drawn from the same distribution. We denote c_t as the lowest price the agent observed up to and including the time period t (i.e., $c_t \leq c_{t-1}$). At the end of the game the agent's cost is $cost(t, c_t) = c_t + \lambda * t$, $\lambda > 0$. The agent's goal is to minimize this cost. As has been previously proven, the optimal strategy in such domains is as follows: exists $\bar{c}$ such that if $c_t \leq \bar{c}$ the agent should stop the search [16].

Intuitively, it seems strange that the decision as to whether the agent should stop the search does not depend on how much time is left, i.e., $\bar{c}$ does not depend on $k - t$. However, the reason for this is as follows. If the agent's overall expected benefit from continuing the search (i.e., the reduction in price that it will obtain) is lower than the overall cost due to the added search time, the agent clearly should not continue the search. Furthermore, it was proven that it is enough for the agent to consider only the next time period, i.e., it should stop the search if and only if the expected reduction in the price in the next time period is less than the cost of continuing one time period (λ) [16]. To understand why, consider the following sketch of the proof: Suppose, to the contrary, that the agent is in time step t, the expected benefit from continuing to $t + 1$ is less than λ, but it will still be beneficial for the agent to continue until time $t' > t + 1$. The agent should then continue the search to time $t' - 1$. However, it is given that $c_{t'-1} \leq c_t$. Thus, given that the price in each time period is drawn from the same probability, the relative expected reduction of the price when moving from $t' - 1$ to t' is smaller than the expected reduction when moving from t to $t + 1$. Nonetheless, the cost per time step, λ is the same. Thus, we demonstrate that if it is not beneficial for the agent to continue from t to $t + 1$, it is also not beneficial to continue from $t' - 1$ to t'; contradicting our assumption that it is beneficial to the agent to continue until time period t'.

In our implementation, the prices are distributed normally with a mean μ and a standard deviation σ. We denote by x the price for which the expected reduction in the price for one time period is equal to λ. For a given price p the benefit is $x - p$ and the

probability[3] for p is

$$\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{p-\mu}{\sigma})^2}$$

Given these definitions we must generally solve:

$$\int_0^x (x-p)\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{p-\mu}{\sigma})^2}dp = \lambda$$

In our specific implementation, $\mu = 1000$, $\sigma = 200$ and $\lambda = 15$. Thus we specifically solve,

$$\int_0^x (x-p)\frac{1}{200\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{p-1000}{200})^2}dp = 15$$

Solving this equations yields a solution of $x = 789$.

Note that as an optimal solution exists, ostensibly there is no need for bounded rationality theories such as AAT. However, we believe that not only people, but even the agents they write on their behalf, do not necessarily effectively harness a computer's computational power to find optimal strategies. Thus, we predict that agents will use non-optimal search strategies involving instrument variables such as the current price of the commodity (x_1) and the elapsed time (x_2) as measured by the number of visited stores.

3.2 Price and Quality Optimization

In the second problem, we consider an environment of a company with a monopoly for a certain product. The owners of the company must set several important variables that will impact how much money the company will make. In this environment, there are no external factors to these decisions. Thus, the outcome of these decisions is not influenced by factors such as how other companies perform, what are other people's decisions, or random effects.

We formally present this problem as follows: We assume that there is a finite timeline $\mathcal{T} = \{1, 2, ..., k\}$. The agent needs to set two instrument variables, the price and quality of the product for any $t \in \mathcal{T}$, denoted p_t and q_t, respectively. The values of p_t and q_t can be set to any positive integer. The profit of the agent at a given time t depends on its choice for the price and quality until the current time. We denote by $\bar{p}^t = (p_1, ..., p_t)$ and $\bar{q}^t = (q_1, ..., q_t)$, the price and qualities determined by the agent until time t. The profit of the agent at a given time, $PT(\bar{p}^t, \bar{q}^t)$ consists of the gross profit and the expenses due to the quality. The part of the profit that is influenced by the price is:

$$PG(\bar{p}^t) = \bar{p}_t^t e^{-(\bar{p}_t^t-\mu)^2/\lambda_p^1}$$

The part of the profit that is influenced by the quality is:

$$QG(\bar{q}^t) = \lambda_q^1 QG(t-1) + \lambda_q^2 QG(t-2) + \lambda_q^3\sqrt{\bar{q}_t^t}$$

[3] In the domain, when a negative price was drawn, we drew a new price. Since the probability of such an event is extremely small, we did not take it into consideration in our analysis.

The profit at time t is

$$PT(\bar{c}^t, \bar{q}^t) = PG(\bar{p}^t) * QG(\bar{q}^t) - \gamma \bar{q}_t^t$$

The profit of the entire time period is: $\sum_{t \in \mathcal{T}} PT(\bar{p}^t, \bar{q}^t)$

All of the constants but μ and γ are known to the agent. In our experiments we set the constants λ_p^1 = 1000, $\lambda_q^1 = 0.7$, $\lambda_q^2 = 0.3$, and $\lambda_q^3 = 0.4$. For initialization purposes, q_{-1} and q_0 are set to 0. The value for μ is a randomly selected integer from a uniform distribution between 25 and 75 and γ is a randomly selected integer from a uniform distribution between 40 and 60. Finally, we set k=50 indicating that the company will only exist for 50 time periods.

The goal of the agent is to maximize the company's profit over the course of one trial. The agent operates within a one-shot environment which resets the values of μ and γ after every trial. Thus, no learning could be performed between these trials to learn the values of μ and γ. However, throughout one trial, for every time period the agent was given the values of $PT(\bar{c}^t, \bar{q}^t)$, $PG(\bar{c}^t)$, $QG(\bar{q}^t)$

Note that in this environment as well, an optimal solution can be found. Here, the only problem parameters with unknown values are μ and γ. It is possible to construct a table offline with the optimal values of price and quality given all possible permutations for μ and γ. Note that the size of this table will be 50 (as per k) times 50 (as per as possible values for μ) times 20 (as per as possible values for γ). Once this table has been constructed, the online agent only needs to identify what the values for μ and γ are so it may use the optimal prelearned values. As such, an optimal solution is as follows: In the first time period, the agent uses a predetermined value for γ that will yield the highest average profit. Once the agent observes the company's profit after the first iteration, it is able to solve for the unknown value of μ. In the second iteration we can similarly solve for γ as it is known to be the only remaining variable. After this point, the agent sets the price and quality for every remaining time step as per the prelearned optimal values for these values of μ and γ. Alternatively, another optimal solution involves first solving for the unknown value for γ in the first iteration, for μ in the second iteration, and using the prelearned optimal values after this point.

As an optimal solution is again possible in this domain, bounded rationality theories such as AAT seem irrelevant. However, we generally believe that two significant factors contributed to the student's inability to optimally solve these problems. First, both problems were verbally presented and thus students needed to properly model the problems before solving them. Second, even after these problems were properly modeled, correctly solving for these problem parameters was far from trivial and required significant algebraic knowledge. As a result, we again hypothesize that people will use non-optimal strategies here involving search within instrument variables of the company's price (x_1) and quality (x_2) parameters.

While the commodity search optimization problem and the slight more complex monopoly domain are relatively simple, the similarities and differences between them allows us to generalize our findings. The one commodity search problem is characterized by complete information, but a certain level of randomness (non-deterministic behavior) exists in what the price of the commodity is in a given store. Also note that the optimal solution involves making a decision based on the current price alone. In

the first domain, other instrument variables, such as the number of visited stores or the length of the time horizon, are not part of the optimal solution. In contrast, the monopoly game is characterized by deterministic functions with two unknown parameters (μ and γ). While this problem is more straightforward, the introduction of a second variable makes the problem seemingly more complex. Furthermore, the optimal value for quality changes over time, and can only be found by solving for γ. Nonetheless, both domains are generalized representations of real-world problems [1, 11] and thus serve as good domains for studying the models of search agents.

4 Results and Analysis

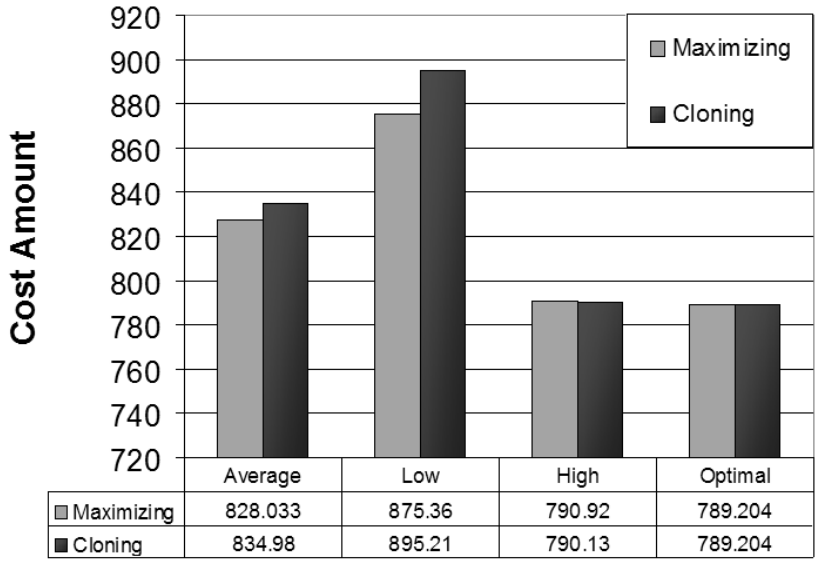
We studied how people's agents performed in the above commodity search and monopoly domains. Within the commodity search domain, we studied the agents from 41 senior undergraduate and graduate students. Within the monopoly domain, we studied the agents from a different group of 57 senior undergraduate and graduate computer science students.

4.1 Non-Optimal Performance

In both domains, a minority of the agents did in fact exhibit performance near or close to that of the optimal agent. However, the vast majority of these the agents deviated significantly from optimal behavior.

As per previous work by Sarne et al. [1], the 41 agents from the commodity search domain were divided into 14 "maximizing" agents and 27 "cloning" agents. The "maximizing" agents were written by students who were asked to create as high performance as possible (optimal). The "cloning" agents were written by people who were instructed to mimic their own personal strategies. As one focus of the experiment was how effectively people could clone their own strategies, the maximize group served as the control group with a ratio of 1:2. As random effects do exist in this environment, we ran each of these agents 50 times, and averaged the agent's performance. We then compared the average performance from the "cloning" and "maximizing" groups, the best performing agent from each of these groups, and the worst performing agent from each of these groups. Finally, we compared the performance of the optimal agent to the agents the students wrote.

Figure 1 shows the performance of these agents. Note that the goal of this domain was to minimize the search price. As such, the low search cost of the best agent (column 3) closely approximated the performance of the optimal agent (column 4). It is important to also note that the average cost of both the "cloning" and "maximizing" agents (approximately 830 units) were quite far from the optimal agent (approximately 790) with p-values testing for significance being much less than 0.0001. However, the differences between the "cloning" and "maximizing" agents were not significant (p-value 0.48). These results imply that most people asked to write optimal agents fall well short of this amount, and do in fact, closely replicate their own non-optimal strategies. This result validates the use of the strategy method [13] from experimental economics as people were typically successful in implementing their own strategies.



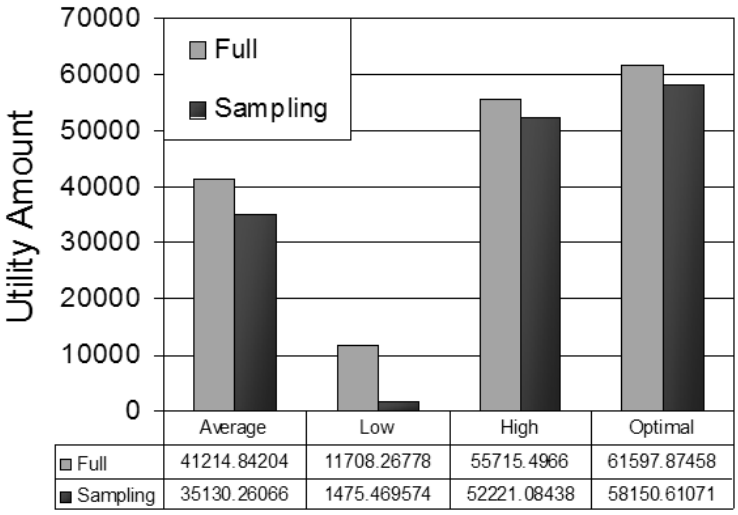
	Average	Low	High	Optimal
Maximizing	828.033	875.36	790.92	789.204
Cloning	834.98	895.21	790.13	789.204

Fig. 1. Comparing the Average, Worst, and Best Utility Value of "Maximizing" and "Cloning" Commodity Search Agents to Optimal Values. Lower costs are better.

Once we verified that the strategy method could be applied to agents written to act within optimizing problems, we studied a second group of 57 maximizing agents written for the monopoly domain. We again studied the average, highest, and lowest performance across the agents, and compared this performance to that of the optimal agent. In this domain, many different values were possible for the previously described price and quality functions. As such, we applied two different evaluation approaches. Within the *Full* evaluation, we studied the average performance of the agent across all possible permutations of price and quality. In the *Sampling* evaluation, we studied the average performance of the agents across six randomly selected value pairs of these values. Realistically, the second type of evaluation seems more appropriate as people typically build small numbers of companies. However, the fuller evaluation in the *Full* group is useful for statistical testing.

Figure 2 displays the performance of the agents from the monopoly domain. Note that again in this domain, the performance of the best agents (third column) in both the *Full* and *Sampling* evaluation methods reached near optimal levels. However, the agents' average performance (first column) again fell well short of the optimal (p-values between the optimal performance and the Full and Sampling evaluation groups were both well below 0.0001). This again strongly supports the claim that people's agents typically fall far short of optimal values. Note that in the more complex monopoly domain, the average agent's performance was over 30% less than the optimal value, while in the simpler commodity search problem, this difference was closer to only 5%.

This seems to imply that as problems become progressively harder, bounded agents seem to perform progressively further from the optimal values.



	Average	Low	High	Optimal
Full	41214.84204	11708.26778	55715.4966	61597.87458
Sampling	35130.26066	1475.469574	52221.08438	58150.61071

Fig. 2. Comparing the Average, Worst, and Best Utility Value of People's Monopoly Agents to Optimal Values. Higher utilities are better.

4.2 Elements of AAT to Quantify Behavior

Our goal was not just to verify the non-optimality of people's agents, but to generalize what non-optimal strategies are in fact being used. To the best of our knowledge, this paper represents the first of its kind that demonstrates that many agents designed to solve optimizing problems in fact implemented strategies consistent with bounded rationality, and specifically key elements of AAT.

It is important to note that several key differences exist between classic AAT theory, and the behavior exhibited by the search agents in the domains we studied. First, AAT assumes that the m goal variables used to solve $\mathcal{G}$ are incomparable. Here, we consider optimization problems where some function between $\mathcal{G}$ and the m goal variables clearly exists, but we hypothesize the agent will not attempt to calculate it due to its bounded nature. Second, AAT is based on the premise that the agent's search will be based on an *aspiration scale* which sorts the m goal variables and attempts to satisfice values for these goals. As we consider concrete optimization problems, it is more natural for agents to consider optimizing the instrument variables that constitute the basis of these goals rather than the more abstract general goal variables. For example, we would expect an agent in the monopoly domain to focus on variables such as price and

quality instead of higher level abstract goals such as "brand awareness" or "company infrastructure".

Nonetheless, we hypothesize that bounded search agents would still exhibit three key elements inspired by AAT. First, we expect the agents to prioritize certain instrument variables to be solved first. This concept parallel's AAT's concept of urgency between its m goal variables. Second, we expect the agent to stop its search within a given instrument variable once it has satisficed this value. Finally, we expect that at times an agent will change its satisficing threshold during the search process. This parallels AAT's concept of retreat and should be expected if the agent deems its original goal is no longer realistic. For example, within the commodity search domain an agent may begin by first attempt to find a commodity below a certain threshold price. However, assuming a certain time elapses (or a number of stores have been visited), it may revise downward this threshold as being a "good enough" solution. In the monopoly domain, the agent might set a priority between which variable, price or quality will be searched for first. After this variable has been satisficed, the agent will then proceed to the other variable.

Note that these three qualities are not an optimal optimizing approach. These problem solving approaches make no attempt to calculate the optimal solution through means such as solving for the unknown parameters within the problems (see the previous section). Furthermore, in contrast to more traditional A.I. methods [9], they do they attempt a simultaneous search on multiple variables.

To study this point we constructed a short list of several questions by which we determined the model of the agents. These questions included: How many variables did the agents attempt to solve for? What were they? Was a search process used to set these instrument variables, or was a predefined strategy (independent of actual performance) used instead? If search was used, were the variables searched for simultaneously, or sequentially? If sequential search was used, did the agent revisit variables after it had originally set a value for it (such as to retreat, or revise downward the previously set threshold value). We recognize that despite the wealth of log files and strategy descriptions provided by the agents' authors, at times some ambiguity may exist in an agent's model as how to answer these questions. To overcome this issue, we had a total of three people judge each of the agents in our results. Of these three people, two were not authors on this paper, and thus had no bias.

Instrument Variable	Judge 1	Judge 2	Judge 3	Average
Price	6	6	7	6.33
Stores Visited	5	5	5	5
Both w/ Retreat	30	30	29	29.67

Table 1. Goal Variables in the Commodity Search Domain

Table 1 presents the number of agents categorized by each of these judges in the commodity search problem. This table depicts how the 3 judges categorized the number and search variables of the agents. The optimal solution within this problem involves

search within only one variable, the current price of the commodity. Nonetheless, the judges found that on average 6.33 of the 41 agents made decisions based on this variable alone (see row 1, column 4). While none of these students actually used the optimal strategy (buy if the price in the current store is less than 789 – as generally dictated by [16] within our implementation), several of the students did have similar strategies such as buy if the price is less than 800. Another 5 students actually used another search variable, the number of stores visited, to make decisions (see row 2). For these students, the strategy was to visit a predetermined number of Y stores and to buy the commodity in the cheapest store from the group of Y. Both of these strategies only contained one instrument variable. As such, they can be viewed as basic search strategies involving only one variable (e.g. search until price $< X$, or visit Y stores and buy in the cheapest store).

Surprisingly, approximately 73% of the agents (see row 3 column 4, average 29.67 of 41) use combination strategies. For these students, the strategy was to immediately buy the commodity if the price was less than X, otherwise, they visit a maximum of Y stores and buy in the store with the cheapest found price. While non-optimal, this strategy has key elements of AAT. Originally, agents search based on price alone. However, if the desired price is not found, the agent downward revises its aspiration. This can be seen as being similar to the retreat process within AAT's goal variables. Note that the use of *urgency* here is not even justified based on optimal behavior as the second goal variable (Stores Visited) is not even part of the optimal solution! Furthermore, the combination strategy of settling on a price after visiting Y stores if the no commodity was found with a price less than X is a good example of *retreat* values. Here, the price less than X is aspired for. However, assuming this cannot be found after visiting Y stores, the satisficing threshold within this variable is revised and a lower value is accepted.

Case	Variable 1	Variable 2	Judge 1	Judge 2	Judge 3	Average	AAT?
1	Optimal	Optimal	0	0	0	0	No
2	Simultaneous Search	Simultaneous Search	9	13	8	10	No
3	Predetermined	Predetermined	3	3	4	3.33	No
4	Search Price	Search Quality	9	14	23	15.33	Yes
5	Search Quality	Search Price	4	0	1	1.67	Yes
6	Search Price	Predetermined Quality	27	22	15	21.33	Trivial Search
7	Search Quality	Predetermined Price	0	1	0	0.33	Trivial Search
8	Alternating	Alternating	5	4	6	5	Yes

Table 2. Comparing the Agent Strategies within the Monopoly (Price and Quality) Domain

Similarly, the results in Table 2 present the analysis of the three judges of the monopoly agents' cognitive models. None of the students used the optimal strategy to solve for both price and quality variables (line 1). A number of agents did simultaneously search for both the price and quality variables (line 2), and a smaller number of students did use predetermined heuristics for setting both variables (e.g. price always

equals 10 and quality equals time elapsed). Both of these strategies do not contain elements of AAT as no urgency exists between variables (in line 2), or no search for goal variables is performed (in the predefined case).

In the vast majority of the agents (approximately 77%) search was conducted with elements of AAT. For most agents (lines 4 & 6) the price variable was searched for first (e.g. had the highest urgency), after which quality was either searched for (in line 4) or set by a predetermined function (in line 6). A very small number of agents had the opposite *aspiration scale* with quality being searched for first (in lines 5 & 7). Similarly, only a relative small number of agents (line 8) consistently alternated between searching for price and quality. In this approach, an agent would set one value (say price), in the next time frame search for the optimal value of the second goal (quality), only to return back to the first goal variable (price) and adjust downward (retreat from) its original value. The reason why fewer agents made use of retreat in this domain seems as follows. According to AAT theory, retreat occurs once an agent realizes it must change its aspiration based on the infeasibility of satisfying multiple goals. In the monopoly game students typically did not see any infeasibility element and therefore did not retreat between variables. However, in the commodity search problem, students (albeit wrongly) saw infeasibility and therefore retreated back on their values. Thus they made use of *retreat* variables to refine their aspirations.

We hope to further study what specific mechanisms were used by agents to determine when it had satisficed a given goal variable. This direction is motivated by several points in Tables 3 & 4. For example, note that in the simpler cost search domain agents revisited previous values (and revised the other goal variable downwards through retreat), but in the more complicated domain they typically did not (except for the agents in line 8 of Table 2). Instead, most agents in the monopoly domain first satisficed the price value (albeit typically with a non-optimal value), and then tried to satisfice the quality goal, never to return to the price goal. Second, note that most differences between the judges in Table 2 revolved around differences in classifying an agent's model as one that is predefined or based on search (see differences in lines 4 & 6). We instructed the judges to assume an agent used search if it changed or retreated from its goal because of previous performance, but not search if it changed its goal because of some predefined value. For example, if an agent perceived that its performance dropped because it raised its quality value, and then decided to lower its quality value, search was used. However, if the agent decided to lower its quality value, even by the same amount, because of some predefined constant function, they were instructed to categorize the agent as having a predefined strategy without search. This definition is based on Learning Directional Theory (LDT) [14] whereby agents change the search process for goals based on previous performance. However, questions arose in cases where goal variables seemed to be intertwined (e.g. an agent set its quality as a function of the price value it was searching for). Additionally, this definition required the judge to read the agent's code and strategy files, and not just observe its performance. Furthermore, many monopoly agents that simultaneously searched for both goal variables (line 2 of Table 2) and thus were not classified as using AAT often instead used the boundedly rational model of LDT to search for these values. Consequently, we are currently studying if LDT can be extended to better describe why and how goal variables are satisficed, and

when an agent will reorder its *aspiration scale* to revisit previous goal variables during search.

Overall, several conclusions can be drawn from the results in both of these domains. First, nearly all agents written to "maximize" performance fell far short of doing so. Within the commodity search domain 30 of 41 agents of all agents used strategies consistent with elements of AAT's *urgency* and *retreat* concepts, while the remaining agents considered a trivial search case where only one goal variable was searched for. Within the monopoly domain, on average approximately 44 of 57 agents used AAT based strategies. Thus, we conclude that optimal approaches cannot properly model most people's agents, and bounded rationality models such as AAT should be used instead.

5 Conclusions and Future Work

In this paper we report our findings about the model used by people's agents to operate in two general optimization problems. These problems can be generalized to many real-world domains [1, 11] and thus our findings contribute two significant findings for Artificial Intelligence researchers. First, we empirically demonstrate that people, or even the agents they write on their behalf, are poor optimizers. Even when we explicitly asked two different groups of over 70 people to write agents to optimally solve a problem, and an optimal solution existed, they instead chose to use approaches that fell well short of optimal behavior. Thus, one must conclude that encapsulating human behavior based on optimal strategies is not effective for certain domains. Second, we find that key elements of Aspiration Adaptation Theory (AAT) do effectively encapsulate many people's search strategies. However, AAT was original formulated for domains where utility cannot be measured and thus did not make any guarantees about performance, or how close to optimal this behavior is [12]. Thus, the results in this paper is particularly important, and indicates the importance and greater generality of using bounded rationality models even in problems where optimal solutions exist and its applicability to search problems.

While the focus of this paper is quantifying the cognitive model of agents, our results lead us to the following conclusions about how to write agents that better interact with people or simulate human performance. First, optimal methods should not be used as they do not realistically encapsulate most human's behavior. Instead, bounded methods should be created such as those based on AAT. In understanding the strategies used by people, we propose that a small pilot be used based on the strategy method [13]. This should identify the ordering (urgency) for search variables and a range of aspiration values within these variables. For example, in the domains we studied, such a pilot would clearly identify price as the variable first searched for in both domains. Finally, any pilot should be focused on the range of aspired for values in each goal variables such that some distribution can be constructed to realistically model the range of problem solving approaches.

For future work, several directions are possible. First, while we found that people's strategies fell short of optimal behavior in both optimization problems we studied, we assume that people will write rational and optimal agents in simpler problems. We

hope to study the level of problem complexity that motivates people to abandon optimal solutions for those based on bounded rationality. Second, we hope to study how effective the above general conclusions are in simulating human behavior in these and other domains. Finally, we also hope to study how people interact with agents based on AAT versus those based on optimal or other predefined heuristic strategies. Specifically, we hope to study agent-human interactions in an emergency response domain. We are hopeful that AAT and other theories of bounded rationality can be applied to these and other agent-based domain problems.

References

1. M. Chalamish, D. Sarne, and S. Kraus. Programming agents as a means of capturing self-strategy. In *AAMAS '08*, pages 1161–1168, 2008.
2. D. Kahneman and A. Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47:263–291, 1979.
3. Pattie Maes. Artificial life meets entertainment: lifelike autonomous agents. *Commun. ACM*, 38(11):108–114, 1995.
4. Efrat Manisterski, Raz Lin, and Sarit Kraus. Understanding how people design trading agents over time. In *AAMAS '08*, pages 1593–1596, 2008.
5. Yohei Murakami, Kazuhisa Minami, Tomoyuki Kawasoe, and Toru Ishida. Multi-agent simulation for crisis management. In *IEEE Workshop on Knowledge Media Networking (KMN02)*, 2002.
6. Yohei Murakami, Yuki Sugimoto, and Toru Ishida. Modeling human behavior for virtual training systems. In *AAAI*, pages 127–132, 2005.
7. John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.
8. David V. Pynadath and Milind Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *JAIR*, 16:389–423, 2002.
9. S. J. Russell and Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003.
10. Heinz Sauermann and Reinhard Selten. Anspruchsanpassungstheorie der unternehmung. *Zeitschrift fr die Gesamte Staatswissenschaft*, 118:577–597, 1962.
11. R. Selten, S. Pittnauer, and M. Hohnisch. Experimental results on the process of goal formation and aspiration adaptation. In *Technical Report – Bonn Laboratory for Experimental Economics*, 2008.
12. Reinhard Selten. Aspiration adaptation theory. *Journal of Mathematical Psychology*, 42:1910–214, 1998.
13. Reinhard Selten, Michael Mitzkewitz, and Gerald R. Uhlich. Duopoly strategies programmed by experienced players. *Econometrica*, 65(3):517–556, 1997.
14. Reinhard Selten and Rolf Stoecker. End behavior in sequences of finite prisoner's dilemma supergames. *J. of Economic Behavior and Organization*, 7:47–70, 1986.
15. Herbert A. Simon. *Models of Man*. John Wiley & Sons, New York, 1957.
16. Martin L Weitzman. Optimal search for the best alternative. *Econometrica*, 47(3):641–654, 1979.

Design of a Decision Maker Agent for a Distributed Role Playing Game – Experience of the SimParc Project

Jean-Pierre Briot[1], Alessandro Sordoni[1], Eurico Vasconcelos[2], Gustavo Melo[3], Marta de Azevedo Irving[3], and Isabelle Alvarez[1]

[1] Laboratoire d'Informatique de Paris 6 (LIP6), Université Pierre et Marie Curie – CNRS, Paris, France
[2] Computer Science Department, Pontifícia Universidade Católica (PUC-Rio), Rio de Janeiro, RJ, Brazil
[3] EICOS Program, Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, RJ, Brazil

Abstract. This paper addresses an ongoing experience in the design of an artificial agent taking decisions in a role playing game populated by human agents and by artificial agents. At first, we will present the context, an ongoing research project aimed at computer-based support for participatory management of protected areas (and more specifically national parks) in order to promote biodiversity conservation and social inclusion. Our applicative objective is, through a distributed role-playing game, to help various stakeholders (e.g., environmentalist, tourism operator) to collectively understand conflict dynamics for natural resources management and to explore negotiation management strategies for the management of parks. Our approach includes support for negotiation among players and insertion of various types of artificial agents (virtual players, decision making agents, assistant agents). In this paper, we will focus on the architecture of the decision making agent playing the role of the park manager, the rationales for his decision, and how it takes into account the preferences/votes from the stakeholders.

1 Introduction

In this paper, we are discussing our experience of inserting artificial agents in a role-playing game populated with humans. The role playing game we consider may be considered as a serious game, as our objective is educational and epistemic. In this game, humans play some role and discuss, negotiate and take decisions about a common domain, in our case environment management decisions.

We are currently designing and inserting different types of artificial agents into this human-based role-playing game [1]. More precisely, we are considering three types of artificial agents:

– artificial players – A first motivation is to address the possible absence of sufficient number of human players for a game session [2]. But this will also allow

more systematic experiments about specific configurations of players profiles, because of artificial players' objective, deterministic and reproducible behaviors.
- artificial decision maker – The park manager acts as an arbitrator in the game, making a final decision for types of conservation for each landscape unit and it also explains its decision to all players. As for artificial players, using an artificial manager in place of a human manager will allow reproductible experiments with controllable levels of participation and of manager profile (see Section 6.1).
- assistant agents – These agents are designed to assist a player by performing tasks, such as orientation within games steps and actions expected, and also support for negotiation, e.g., by identifying and suggesting potential coalitions.

In this paper we focus on the design of the artificial decision maker agent. Its objective is to take decision based on its own analysis of the situation and on the proposals by the players. The agent is also able to explain its decision based on its chain of argumentation.

The structure of this paper is as following: after introducing the SimParc project, its role playing game and its computer support, and the insertion of artificial agents, we describe the decision maker agent objectives, architecture and implementation.

2 The SimParc Project

2.1 Project Motivation

A significant challenge involved in biodiversity management is the management of protected areas (e.g., national parks), which usually undergo various pressures on resources, use and access, which results in many conflicts. This makes the issue of conflict resolution a key issue for the participatory management of protected areas. Methodologies intending to facilitate this process are being addressed via bottom-up approaches that emphasize the role of local actors. Examples of social actors involved in these conflicts are: park managers, local communities at the border area, tourism operators, public agencies and NGOs. Examples of inherent conflicts connected with biodiversity protection in the area are: irregular occupation, inadequate tourism exploration, water pollution, environmental degradation and illegal use of natural resources.

Our SimParc project focuses on participatory parks management. (The origin of the name SimParc stands in French for "Simulation Participative de Parcs") [3]. It is based on the observation of several case studies in Brazil. However, we chose not to reproduce exactly a real case, in order to leave the door open for broader game possibilities [4]. Our project aim is to help various stakeholders at collectively understand conflicts and negotiate strategies for handling them.

2.2 Approach

Our initial inspiration is the companion modeling (ComMod) approach about participatory methods to support negotiation and decision-making for participatory management of renewable resources [5]. They pioneer method, called MAS/RPG, consists in coupling multi-agent simulation (MAS) of the environment resources and role-playing games (RPG) by the stakeholders [5]. The RPG acts like a "social laboratory", because players of the game can try many possibilities, without real consequences.

Recent works proposed further integration of role-playing into simulation, and the insertion of artificial agents, as players or as assistants. Participatory simulation and its incarnation, the Simulaciòn framework [6], focused on a distributed support for role-playing and negotiation among human players. All interactions are recorded for further analysis (thus opening the way to automated acquisition of behavioral models) and assistant agents are provided to assist and suggest strategies to the players. The Games and Multi-Agent-based Simulation (GMABS) methodology focused on the integration of the game cycle with the simulation cycle [2]. It also innovated in the possible replacement of human players by artificial players. One of our objectives is to try to combine their respective merits and to further explore possibilities of computer support.

3 The SimParc Role-Playing Game

3.1 Game Objectives

Current SimParc game has an epistemic objective: to help each participant discover and understand the various factors, conflicts and the importance of dialogue for a more effective management of parks. Note that this game is not (or at least not yet) aimed at decision support (i.e., we do not expect the resulting decisions to be directly applied to a specific park).

The game is based on a negotiation process that takes place within the park council. This council, of a consultative nature, includes representatives of various stakeholders (e.g., community, tourism operator, environmentalist, non governmental association, water public agency...). The actual game focuses on a discussion within the council about the "zoning" of the park, i.e. the decision about a desired level of conservation (and therefore, use) for every sub-area (also named "landscape unit") of the park. We consider nine pre-defined potential levels (that we will consider as types) of conservation/use, from more restricted to more flexible use of natural resources, as defined by the (Brazilian) law. Examples are: Intangible, the most conservative use, Primitive and Recuperation.

The game considers a certain number of players' roles, each one representing a certain stakeholder. Depending on its profile and the elements of concerns in each of the landscape units (e.g., tourism spot, people, endangered species), each player will try to influence the decision about the type of conservation for each landscape unit. It is clear that conflicts of interest will quickly emerge, leading

to various strategies of negotiation (e.g., coalition formation, trading mutual support for respective objectives, etc).

A special role in the game is the park manager. He is a participant of the game, but as an arbiter and decision maker, and not as a direct player. He observes the negotiation taking place among players and takes the final decision about the types of conservation for each landscape unit. His decision is based on the legal framework, on the negotiation process among the players, and on his personal profile (e.g., more conservationist or more open to social concerns) [4]. He may also have to explain his decision, if the players so demand. We plan that the players and the park manager may be played by humans or by artificial agents.

3.2 Game Cycle

The game is structured along six steps, as illustrated in Figure 1. At the beginning (step 1), each participant is associated with a role. Then, an initial scenario is presented to each player, including the setting of the landscape units, the possible types of use and the general objective associated to his role. Then (step 2), each player decides a first proposal of types of use for each landscape unit, based on his/her understanding of the objective of his/her role and on the initial setting. Once all players have done so, each player's proposal is made public.

In step 3, players start to interact and to negotiate on their proposals. This step is, in our opinion, the most important one, where players collectively build their knowledge by means of an argumentation process. In step 4, they revise their proposals and commit themselves to a final proposal for each landscape unit. In step 5, the park manager makes the final decision, considering the negotiation process, the final proposals and also his personal profile (e.g., more conservationist or more sensitive to social issues). Each player can then consult various indicators of his/her performance (e.g., closeness to his initial objective, degree of consensus, etc.). He can also ask for an explanation about the park manager decision rationales.

The last step (step 6) "closes" the epistemic cycle by considering the possible effects of the decision. In the current game, the players provide a simple feedback on the decision by indicating their level of acceptance of the decision.[4]

A new negotiation cycle may then start, thus creating a kind of learning cycle. The main objectives are indeed for participants: to understand the various factors and perspectives involved and how they are interrelated; to negotiate; to try to reach a group consensus; and to understand cause-effect relations based on the decisions.

[4] A future plan is to introduce some evaluation of the quality of the decision. Possible alternatives are: computable indicators (e.g., about the economical or social feasibility), more formal model (based on viability theory), or a multi-agent simulation of the evolution of resources. Note that a real/validated model is not necessary as the park is fictive and the objective is credibility, not realism.

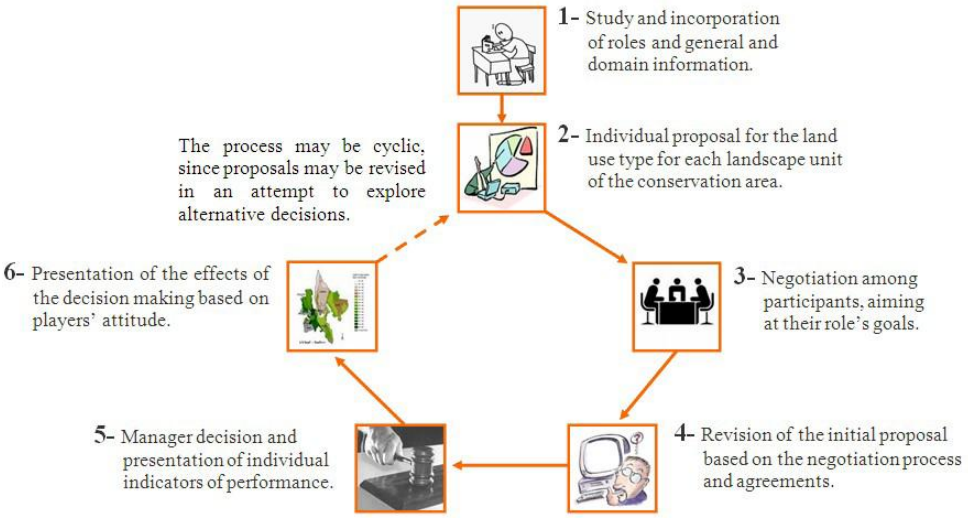


Fig. 1. The six steps of the SimParc game.

4 The SimParc Game Support Architecture

Our current prototype benefited from our previous experiences (game sessions and a first prototype) and has been based on a detailed design process. Based on the system requirements, we adopted Web-based technologies (more precisely J2E and JSF) that support the distributed and interactive character of the game as well as an easy deployment.

Figure 2 shows the general architecture and communication structure of SimParc prototype version 2. In this second prototype, distributed users (the players and the park manager) interact with the system mediated internally by communication broker agents (CBA). The function of a CBA is to abstract the fact that each role may be played by a human or by an artificial agent. A CBA also translates user messages in http format into multi-agent KQML format and vice versa. For each human player, there is also an assistant agent offering assistance during the game session.

During the negotiation phase, players (human or artificial) negotiate among themselves to try to reach an agreement about the type of use for each landscape unit (sub-area) of the park. The interface for negotiation is shown at Figure 3 and the interface for players decision about the types of use at Figure 4. A Geographical Information System (GIS) offers to users different layers of information (such as flora, fauna and land characteristics) about the park geographical area. All the information exchanged during negotiation phase, namely users' logs, game configurations, game results and general management information are recorded and read from a PostgreSql database.

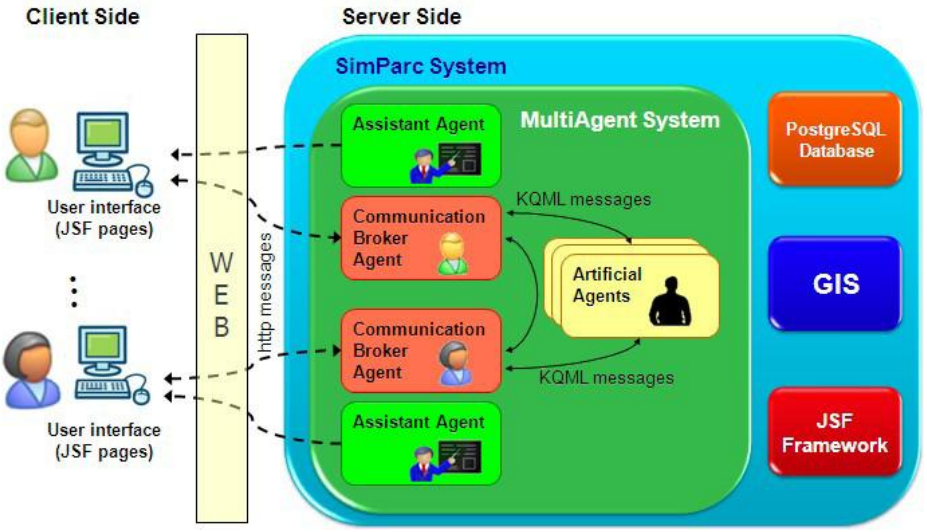


Fig. 2. SimParc version 2 general architecture.

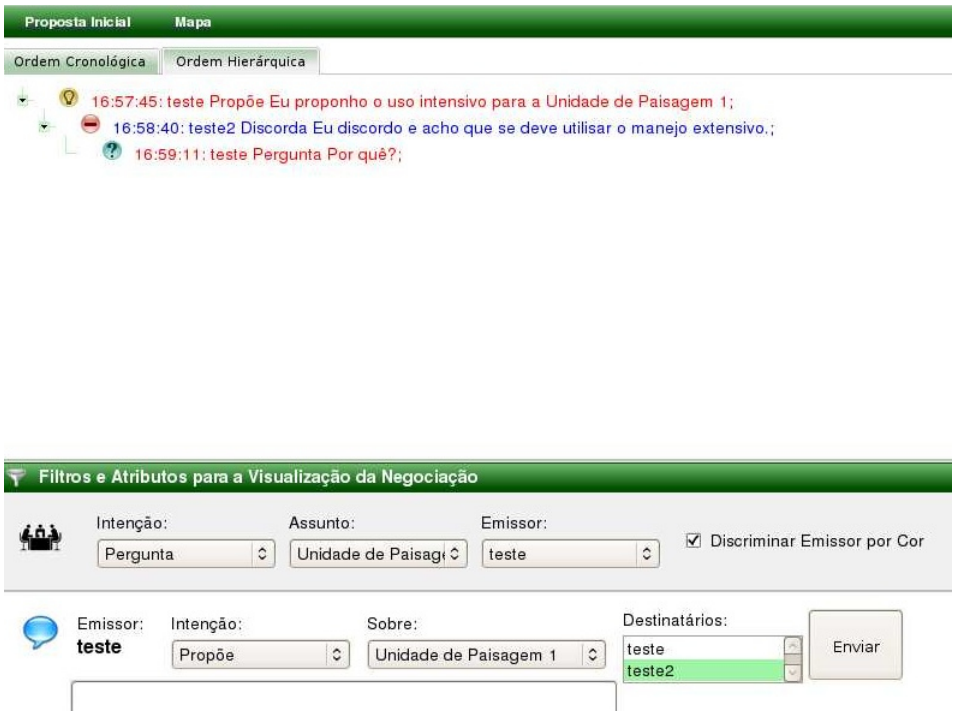


Fig. 3. Current prototype's negotiation graphical user interface.

Fig. 4. Current prototype's decision graphical user interface.

5 Inserting Artificial Agents into the SimParc Game

We are currently inserting artificial agents into the prototype. We consider three types of artificial agents: the artificial players, the park manager and the assistants. For the artificial players, we build up on previous experience on virtual players in a computer-supported role playing game, ViP-JogoMan [2]. The idea is to potentially replace some of the human players by artificial players (artificial agents). The two main motivations are: (1) the possible absence of sufficient number of human players for a game session and (2) the need for testing in a systematic way specific configurations of players' profiles. The artificial players will be developed along BDI (Belief, Desire and Intention) architecture [7] and implemented in Jason [8]. We are currently planning the first versions of the artificial players agents, based on profiles models identified by our domain experts, and with a basis similar to the park manager individual decision architecture (see next paragraph and also Section 6.3), with the addition of an automated negotiation model [1]. In a next stage, we envisage to use automated analysis of recorded traces of interaction among human players in order to infer models of artificial players. In some previous work [6], genetic programming had been used as a technique to infer interaction models, but we will also plan to explore alternative induction and machine learning techniques, e.g., inductive logic programming.

The park manager acts as an arbitrator in the game, making a final decision for types of conservation for each landscape unit and explains its decision to

all players. He may be played by a human or by an artificial agent. We have implemented a prototype implementation of an artificial park manager, based on 2 steps: (1) internal/ individual decision by the park manager, based on some argumentation model; (2) merging of the decision by the manager with the votes by the players, based on decision theory (social choice). Traces of argumentation may be used for explaining the rationale of the decision. The artificial park manager architecture is detailed in Section 6.

The last type of artificial agent is an intelligent assistant agent. This agent is designed to assist a player by performing two main tasks: (1) to help participants in playing the game, e.g.: the assistant agent tells the player when he should make decisions; what are the phases of the game; what should be done in each phase; etc.; (2) to help participants during the negotiations. For this second task, we would like to avoid intrusive support, which may interfere in his decision making cognitive process. We have identified some actions, e.g., to identify other players' roles with similar or dissimilar goals, which may help the human player to identify possible coalitions or conflicts. The general main idea for an assistance for negotiation is thus to combine and classify important information to help participants to make analysis and do it faster than they would do alone, while keeping their focus on the game proposal. An initial prototype implementation of an assistant agent has been implemented.

Further details about SimParc artificial players and assistant agents may be found in [1] and in future publications. Some advanced interface has also been designed for human players dialogue and negotiation support and is detailed in [9]. Now we will detail the rationale and architecture of our automated park manager who makes the final decision.

6 The Park Manager Artificial Agent

In this section, we describe an agent architecture to implement park manager cognitive decision rationale. As we summarized before, our decision model is based on two mechanisms. These mechanisms could be viewed as modules of decision subprocesses. We believe that complex decision making is achievable by sequential organization of these modules. Before proceeding to the description of our agent architecture, we present some more detailed motivation for it.

6.1 Objectives

Participatory management aims to emphasize the role of local actors in managing protected areas. However, park manager is the ultimate arbiter of all policy on devolved matters. He acts like an expert who decides on validity of collective concerted management policies. Moreover, he is not a completely fair and objective arbiter: he still brings his personal opinions and preferences in the debate. Therefore, we aim to develop an artificial agent modeling the following behaviors.

Personal preferential profile; park manager decision-making process is supposed to be influenced by its sensibility to natural park stakes and conflicts. In decision theory terms, we can affirm that park manager's preferential profile could be intended as a preference relation over conservation policies. One of the key issues is to understand that we cannot define a strict bijection between preferential profile and preference relation. Agent's preference relation is partially dependent on natural park resources and realities. Moreover, this relation is not likely to be an order or a preorder. Hence, our agent must be able to define dynamically its preference relation according with its preferential profile. We distinguish two preferential profiles:

- Preservationist, aims to preserve ecosystems and the natural environment.
- Socio-conservationist, generally accepts the notion of sustainable yield - that man can harvest some forest or animal products from a natural environment on a regular basis without compromising the long-health of the ecosystem.

Taking into account stakeholders' decisions; a participative decision-making leader seeks to involve stakeholders in the process, rather than taking autocratic decisions. However, the question of how much influence stakeholders are given may vary on manager's preferences and beliefs. Hence, our objective is to model the whole spectrum of participation, from autocratic decisions to fully democratic ones. To do so, we want the park manager agent to generate a preference preorder over conservation policies. This is because it should be able to calculate the distance between any two conservation policies. This way, we can merge stakeholders' preference preorders with manager's one to establish one participative final decision. Autocratic/democratic manager attitude will be modeled by an additional parameter during the merge process.

Expert decision; park manager's final decision must consider legal constraints related to environmental management; otherwise, non-viable decisions would be presented to the players, thus invalidating game's learning objectives. These constraints are directly injected in the cognitive process of the agent. Hence, the agent will determine a dynamic preference preorder, according to its preferential profile, over allowed conservation levels.

Explaining final decision; In order to favor the learning cycle, park manager agent must be able to explain its final decision to the players. We can consider that the players could eventually argue about its decision; the agent should then defend its purposes using some kind of argumentative reasoning. Even if such cases will be explored in future work, it is our concern to conceive a cognitive architecture which provides a good basis for managing these situations.

6.2 Architecture Overview

Let us now present an architecture overview of the park manager agent. As depicted in Figure 5, agent's architecture is structured in two phases. We believe that sequential decision-making mechanisms can model complex cognitive behaviors along with enhanced explanation capabilities.

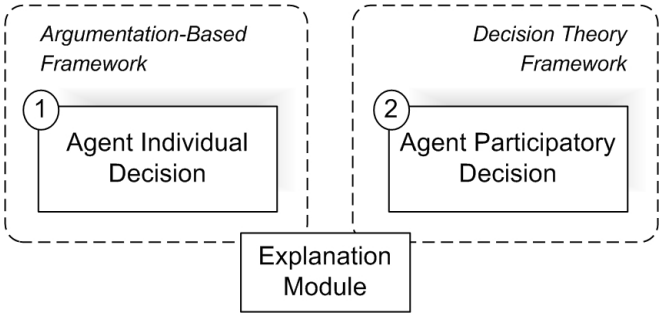


Fig. 5. Park Manager Agent 2-Steps decision process.

The first decision step concerns agent's individual decision-making process: the agent deliberates about the types of conservation for each landscape unit. Broadly speaking, park manager agent builds its preference preorder over allowed levels of conservation. An argumentation-based framework is implemented to support the decision making. The next step of our approach consists of taking account of players' preferences. The result of the execution is the modified park manager decision, called agent participatory decision, according to stakeholder's preferences.

6.3 Agent Individual Decision

Recently, argumentation has been gaining increasing attention in the multi-agent community. Autonomous and social agents need to deliberate under complex preference policies, related to the environment in which they evolve. Generally, social interactions bring new information to the agents. Hence, preference policies need to be dynamic in order to take account of newly acquired knowledge. Dung's work [10] proposes formal proof that argumentation systems can handle epistemic reasoning under open-world assumptions, usually modeled by nonmonotonic logics. Argumentation thus becomes an established approach for reasoning with inconsistent knowledge, based on the construction and the interaction between arguments. Recently, some research has considered argumentation systems capabilities to model practical reasoning, aimed at reasoning about what to do [11–13]. It is worth noticing that argumentation can be used to select arguments that support available desires and intentions. Consistent knowledge can generate conflicting desires. An agent should evaluate pros and cons before pursuing any desire. Indeed, argumentative deliberation provides a mean for choosing or discarding a desire as an intention.

We could argue that open-world assumptions don't hold in our context. Agent's knowledge base isn't updated during execution, since it's not directly exposed to social interactions. Knowledge base and inference rules consistency-checking methods are, therefore, not necessary. However, one key aspect here is

to conceive an agent capable of explaining its policy making choices; our concern is to create favorable conditions for an effective and, thus closed, learning cycle. We believe that argumentation "tracking" represents an effective choice for accurate explanations. Conflicts between arguments are reported, following agent's reasoning cycle, thus enhancing user comprehension.

From this starting position, we have developed an artificial agent on the basis of Rahwan and Amgoud's work [12]. The key idea is to use argumentation system to select the desires the agent is going to pursue: natural park stakes and dynamics are considered in order to define objectives for which to aim. Hence, decision-making process applies to actions, i.e. conservation levels, which best satisfy selected objectives. In order to deal with arguments and knowledge representation, we use first-order logic. Various inference rules were formulated with the objective of providing various types of reasoning capability. For example, a simple inference rule for generating desires from beliefs, i.e. natural park stakes, is:

$$Fire \rightarrow Avoid\_Fires,\ 4$$

where *Fire* (fire danger in the park) is a belief in agents knowledge base and *Avoid_Fires* is the desire that is generated from the belief. The value *4* represents the intensity of the generated desire.

Examples of rules for selecting actions, i.e. level of conservation, from desires are:

$$Primitive \rightarrow Avoid\_Fires,\ 0.4$$

$$Intangible \rightarrow Avoid\_Fires,\ 0.8$$

where *Primitive*, *Intangible* represent the levels of conservation and the values *0.4*, *0.8* represent their utility in order to satisfy the corresponding desire.

6.4 Agent Participatory Decision

Despite participatory ideals, a whole spectrum of park managers, from autocratic to fully democratic ones, can be measured, depending on how more participatory and democratic decision-making is operationalized. We propose a method, fitted into the social-choice framework, in which participatory attitude is a model parameter.

In a real case scenario, a decision-maker would examine each stakeholder's preferences in order to reach the compromise that best reflects its participatory attitude. Our idea is to represent this behavior by weighting each player's vote according to manager's point of view.

This concept is illustrated in Figure 6. The process is structured in two phases. Firstly, manager agent injects its own preferences into players' choices by means of an influence function describing agent's participatory attitude. Stronger influence translates into more autocratic managers. Secondly, modified players' choices are synthesized, using an aggregation function, i.e. Condorcet voting method. The result of the execution will be the agent participatory decision.

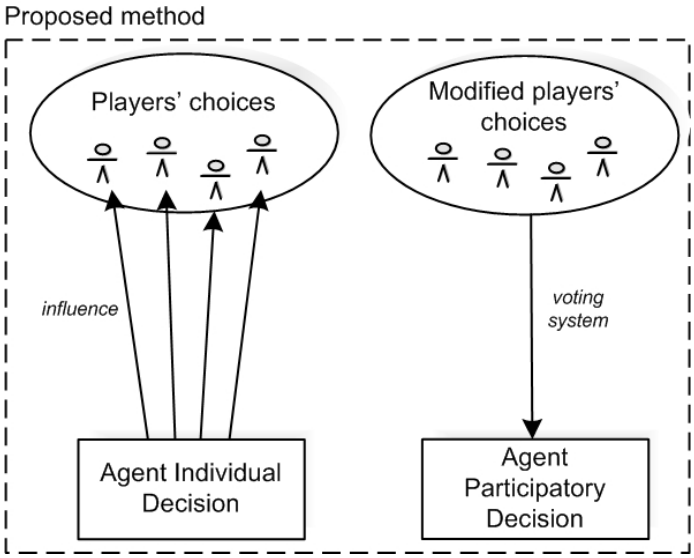


Fig. 6. Park Manager Agent Participatory Decision

Example. Let the following be players' choices, where $\succ$ is a preference relation (a $\succ$ b means "a is preferred to b"):

$$player_1 = Intangible \succ Primitive \succ Extensive, \mathbf{v}_1 = (3, 2, 1)$$

$$player_2 = Extensive \succ Primitive \succ Intangible, \mathbf{v}_2 = (1, 2, 3)$$

$$player_3 = Primitive \succ Extensive \succ Intangible, \mathbf{v}_3 = (1, 3, 2)$$

Let Manager individual decision be:

$$manager_{ind} = Extensive \succ Primitive \succ Intangible, \mathbf{v}_M = (1, 2, 3)$$

Players' choices are converted into numeric vectors specifying the candidates' rank for each vote. Let the following be the influence function:

$$\odot(x, y) = \begin{cases} x & \text{if } x = y \\ x * 1/|x - y| & \text{otherwise} \end{cases}$$

Modified players' vectors will be:

$$\begin{aligned} \mathbf{mv}_1 &= \langle \odot(\mathbf{v}_1(1), \mathbf{v}_M(1)), \odot(\mathbf{v}_1(2), \mathbf{v}_M(2)), \odot(\mathbf{v}_1(3), \mathbf{v}_M(3)) \rangle \\ &= (1.5, 2, 0.5) \\ \mathbf{mv}_2 &= (1, 2, 3) \\ \mathbf{mv}_3 &= (1, 3, 2) \end{aligned}$$

In order to find the manager participative decision, we apply the Choquet integral $\mathcal{C}_\mu$ [14] choosing a symmetric capacity measure $\mu(S) = |S|^2/|\mathcal{A}|^2$, where $\mathcal{A}$ is the candidates set.

$$\mathcal{C}_\mu(Intangible) = 1.05,\ \mathcal{C}_\mu(Primitive) = 2.12,\ \mathcal{C}_\mu(Extensive) = 1.27$$

The result of the execution will then be:

$$manager_{part} = Primitive \succ Extensive \succ Intangible$$

Further details about architecture formal background and implementation are reported in [15].

6.5 Examples of Results

Presented manager agent architecture and its first implementation were tested over different scenarios. Simulations conducted in laboratory have been validated by team experts. However, results from further tests are expected. In particular, open simulations with various participants are planned for Spring 2009. We report hereafter an example of explanation for managers decision over a landscape unit. Let manager individual decision be the following:

$$manager_{ind} = Intangible \succ Recuperation$$

Arguments for *Intangible* are:

$$Endangered\_species\ \&\ Tropical\_forest \rightarrow Maximal\_protection$$

$$Intangible \rightarrow Maximal\_protection$$

Arguments for *Recuperation* are:

$$Fire\ \&\ Agricultural\_activities \rightarrow Recover\_deteriorated\_zone$$

$$Recuperation \rightarrow Recover\_deteriorated\_zone$$

6.6 Implementation framework

The architecture presented in this paper is implemented in Jason multi-agent platform [8]. Besides interpreting the original AgentSpeak(L) language, thus disposing of logic programming capabilities, Jason also features extensibility by user-defined internal actions, written in Java. Hence, it has been possible to easily implement aggregation methods.

7 Conclusion

In this paper, we have presented the SimParc project, a role-playing serious game aimed at computer-based support for participatory management of protected areas. The lack of human resources implied in RPG gaming process acts as a constraint to the fulfillment of pedagogical and epistemic objectives. In order to guarantee an effective learning cycle, park manager role must be played by a domain expert. Required expertise obviously narrows game's autonomy and limits its context of application. Our solution to this problem is to insert artificial agents into the game. In this paper, we focused on the architecture to implement park manager cognitive decision rationale. The decison-making agent can justify its behavior and generate a participatory decision. Our argumentation system is based on [11, 12]: conflicts between arguments can be reported thus enhancing user comprehension. Moreover, we presented a decision theory framework responsible for generating a participatory decision. To the best of our knowledge and belief, this issue has not yet been adressed in the literature. The final integration of the complete SimParc prototype is under completion and we will soon (Spring 2009) start to test it by organizing game sessions with expert players. Besides the project specific objectives, we also plan to study the possible generality of our prototype for other types of human-based social simulations. In the current architecture of the artificial park manager, only static information about the park and about the votes by players are considered. We are considering exploring how to introduce dynamicity in the decision model, taking into account the dynamics of negotiation among players (the evolution of decisions by players during negotiation).

Acknowledgments We thank the other current members of the project: Vinícius Sebba-Patto, Diana Adamatti, Altair Sancho and Davis Sansolo for their current participation; and also: Ivan Bursztyn and Paul Guyot for their past participation. This research is funded by: the ARCUS Program (French Ministry of Foreign Affairs, Région Ile-de-France and Brazil) and the MCT/CNPq/CT-INFO Grandes Desafíos Program (Project No 550865/2007-1, Brazil). Some additional individual support is provided by French Ministry of Research (France), Alban (Europe), CAPES and CNPq (Brazil) PhD fellowship programs.

References

1. Briot, J.P., Vasconcelos, E., Adamatti, D., Sebba, V., Irving, M., Barbosa, S., Furtado, V., Lucena, C.: A computer-based support for participatory management of protected areas: The simparc project. XXVIIIth Congress of Computation Brazilian Society (CSBC/SEMISH'08), Belem, Brazil (July 2008) 1–15
2. Adamatti, D., Sichman, J., Coelho, H.: Virtual players: From manual to semi-autonomous RPG. In: AI, Simulation and Planning in High Autonomy Systems (AIS) and Conceptual Modeling and Simulation (CMS), International Modeling and Simulation Multiconference 2007 (IMSM07) (2007) Buenos Aires, Argentina.

3. Briot, J.P., Guyot, P., Irving, M.: Participatory simulation for collective management of protected areas for biodiversity conservation and social inclusion. In Fernando Barros, Claudia Frydman, N.G., Ziegler, B., eds.: AIS-CMS'07 International Modeling and Simulation Multiconference (IMSM'07), The Society for Modeling Simulation International (SCS) (2007) 183–188 isbn: 978-2-9520712-6-0.
4. Irving, M.: Áreas Protegidas e Inclusão Social: Construindo Novos Significados. Rio de Janeiro: Aquarius (2006)
5. Barreteau, O.: The joint use of role-playing games and models regarding negotiation processes: Characterization of associations. Journal of Artificial Societies and Social Simulation (2003)
6. Guyot, P., Honiden, S.: Agent-based participatory simulations: Merging multi-agent systems and role-playing games. Journal of Artificial Societies and Social Simulation **9**(4) (2006) 8
7. Rao, A., Georgeff, M.: Modelling rational agents within a bdi-architecture. Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (1991) 473–484
8. Bordini, R.H., Wooldridge, M., Hübner, J.F.: Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology). John Wiley & Sons (2007)
9. Vasconcelos, E., Briot, J.P., Barbosa, S., Furtado, V., Irving, M.: A user interface to support dialogue and negotiation in participatory simulations. In Proceedings of 9th International Workshop on Multi-Agent-Based Simulation (MABS' 2008), Estoril, Portugal (May 2008) 47–58
10. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence (1995) 321–357
11. Moraitis, P.: Un modèle de raisonnement pour agents autonomes fondé sur l'argumentation. Proc. Journées Nationales de Modèles de Raisonnement (JNMR'03) (2003)
12. Rahwan, I., Amgoud, L.: An argumentation based approach for practical reasoning. In: AAMAS'06: Proceedings of the fifth international joint conference on autonomous agents and multiagent systems, New York, NY, USA, ACM (2006) 347–354
13. Amgoud, L., Kaci, S., Sabatier, U.P.: On the generation of bipolar goals in argumentation-based negotiation. In: 1st International Workshop on Argumentation in Multi-Agent Systems, Springer (2004) 192–207
14. Choquet, G.: Theory of capacities. Annales de l'Institut Fourier **5** (1953) 131–295
15. Sordoni, A.: Conception et implantation d'un agent artificiel dans le cadre du projet simparc. Master's thesis, EDITE, Université Pierre et Marie Curie (Paris 6), Paris, France (2008)

Two Case Studies for Jazzyk BSM

Michael Köster, Peter Novák, David Mainzer and Bernd Fuhrmann

Department of Informatics, Clausthal University of Technology
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany

Abstract. Recently, we introduced *Behavioural State Machines* (*BSM*), a novel programming framework for development of cognitive agents with *Jazzyk*, its associated programming language and interpreter. The *Jazzyk BSM* framework draws a strict distinction between *knowledge representation* and *behavioural* aspects of an agent program. *Jazzyk BSM* thus enables synergistic exploitation of heterogeneous knowledge representation technologies in a single agent, as well as offers a transparent way for embedding cognitive agents in various simulated or physical environments. This makes it a particularly suitable platform for development of simulated, as well as physically embodied cognitive agents, such as virtual agents, or non-player characters for computer games.
In this paper we report on *Jazzbot* and *Urbibot* projects, two case-studies we developed using the *Jazzyk BSM* framework in simulated environments provided by a first person shooter computer game and a physical reality simulator for mobile robotics respectively. We describe the underlying technological infrastructure of the two agent applications and provide a brief account of experiences and lessons we learned during the development.

1 Introduction

One of the long-term aims of Artificial Intelligence is to enable development of intelligent cognitive agents. I.e., such which internally model their environment, their own mental attitudes, reason about them and subsequently base their decisions regarding their future actions upon these models. Even though AI research provides a plethora of approaches for solving partial problems on the way towards this aim, we only rarely encounter approaches enabling integration of the various developed technologies. The field of agent oriented programming, and in consequence multi-agent systems programming, offers a sound theoretical basis allowing synergistic exploitation of heterogeneous AI technologies in a single agent system.

Therefore in our recent work, we introduced the theoretical framework of *Behavioural State Machines* with its associated agent oriented programming language *Jazzyk* [11,12]. *Jazzyk BSM* provides a simple, theoretically sound language for modular agent programming based on a generic computational model for reactive systems. It draws a strict distinction between the knowledge representation (KR) and behavioural aspects of an agent program and thus enables exploiting heterogeneous KR technologies in a single agent system.

To provide a proof-of-concept, as well as to further nurture our research towards a methodology of development with *Jazzyk BSM* (cf. [14] and [13]), we developed two case study applications *Jazzbot* and *Urbibot*. *Jazzbot* is a virtual bot in the simulated 3D environment of an open source first person shooter computer game *Nexuiz*. Its task is to explore a virtual building, search for certain objects in it and subsequently deliver them to the base. At the same time, *Jazzbot* is supposed to differentiate between other players present in the building and seek safety upon being attacked by an enemy player. When the danger disappears, it should return back to the activity interrupted by the attack.

Urbibot, on the other hand, was developed as a step towards programming mobile robots. It is an agent program steering a model of customized *e-Puck*, a small two-wheeled mobile robot in an environment provided by the physical robotic simulator *Webots*. Similarly to *Jazzbot*, *Urbibot* explores its environment in order to find red poles present in it. It tries to bump into each of them, while trying to avoid patrol robots policing the environment. Upon encounter with such a patrol robot, *Urbibot* runs away to finally return to the previously interrupted activity when safe again.

Both agents feature a BDI inspired architecture. While interacting with two different types of virtual bodies, a character in the game and an interface to robot hardware sensors and actuators respectively, both implementations exploit the power of non-monotonic reasoning for representation and reasoning about their beliefs and goals. We employ an interpreted object oriented programming language to enable efficient representation of and reasoning about topological structure of the environment.

After a brief introduction to the framework of *Behavioural State Machines* and its associated programming language *Jazzyk* in Section 2, Sections 3 and 4 describe respectively *Jazzbot* and *Urbibot* agents in a closer detail. Subsequently, Section 5 provides a description of the underlying technological infrastructure used in the implemented agents. Finally, a discussion of our experiences and lessons learned from the development of *Jazzbot* and *Urbibot* agents, together with an outlook to the ongoing and future work wraps up the paper in Section 6.

2 Jazzyk BSM

In [12] we introduced the framework of *Behavioural State Machines* (*BSM*). *BSM* framework draws a clear distinction between the *knowledge representation* and *behavioural* layers within an agent. It thus provides a programming system that clearly separates the programming concerns of *how to represent an agent's knowledge* about, for example, its environment and *how to encode its behaviours*. In the core of the framework is a *generic reactive computational model* inspired by Gurevich's *Abstract State Machines* [3], enabling for efficient structuring of the program code. This section briefly introduces the *BSM* framework. For the complete formal description of the *BSM* framework, see [12].

2.1 Syntax

BSM agents are collections of one or more so-called *knowledge representation modules* (KR modules), typically denoted by $\mathcal{M}$, each representing a part of the agent's knowledge base. KR modules may be used to represent and maintain various mental attitudes of an agent, such as knowledge about its environment, or its goals, intentions, obligations, etc. Transitions between states of a *BSM* result from applying so-called *mental state transformers* (*mst*), typically denoted by τ. Various types of mst's determine the behaviour that an agent can generate. A *BSM agent* consists of a set of KR modules $\mathcal{M}_1, \ldots, \mathcal{M}_n$ and a mental state transformer $\mathcal{P}$, i.e., $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$; the mst $\mathcal{P}$ is also called an *agent program*.

The notion of a KR module is an abstraction of a partial knowledge base of an agent. In turn, its states are to be treated as theories (i.e., sets of sentences) expressed in the KR language of the module. Formally, a KR module $\mathcal{M}_i = (\mathcal{S}_i, \mathcal{L}_i, \mathcal{Q}_i, \mathcal{U}_i)$ is characterized by a knowledge representation language $\mathcal{L}_i$, a set of states $\mathcal{S}_i \subseteq 2^{\mathcal{L}_i}$, a set of query operators $\mathcal{Q}_i$ and a set of update operators $\mathcal{U}_i$. A query operator $\models \in \mathcal{Q}_i$ is a mapping $\models : \mathcal{S}_i \times \mathcal{L}_i \rightarrow \{\top, \bot\}$. Similarly an update operator $\oplus \in \mathcal{U}_i$ is a mapping $\oplus : \mathcal{S}_i \times \mathcal{L}_i \rightarrow \mathcal{S}_i$.

Queries, typically denoted by φ, can be seen as operators of type $\models : \mathcal{S}_i \rightarrow \{\top, \bot\}$. A primitive query $\varphi = (\models\phi)$ consists of a query operator $\models \in \mathcal{Q}_i$ and a formula $\phi \in \mathcal{L}_i$ of the same KR module $\mathcal{M}_i$. Complex queries can be composed by means of conjunction $\wedge$, disjunction $\vee$ and negation $\neg$.

Mental state transformers enable transitions from one state to another. A primitive mst $\oslash\psi$, typically denoted by ρ and constructed from an update operator $\oslash \in \mathcal{U}_i$ and a formula $\psi \in \mathcal{L}_i$, refers to an update on the state of the corresponding KR module. Conditional mst's are of the form $\varphi \longrightarrow \tau$, where φ is a query and τ is a mst. Such a conditional mst makes the application of τ depend on the evaluation of φ. Syntactic constructs for combining mst's are: non-deterministic choice | and sequence $\circ$.

Definition 1 (mental state transformer). *Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be KR modules of the form $\mathcal{M}_i = (\mathcal{S}_i, \mathcal{L}_i, \mathcal{Q}_i, \mathcal{U}_i)$. The set of* mental state transformers *is defined as below:*

- **skip** *is a* primitive *mst,*
- *if $\oslash \in \mathcal{U}_i$ and $\psi \in \mathcal{L}_i$, then $\oslash\psi$ is a* primitive *mst,*
- *if φ is a query, and τ is a mst, then $\varphi \longrightarrow \tau$ is a* conditional *mst,*
- *if τ and τ' are mst's, then $\tau|\tau'$ and $\tau \circ \tau'$ are mst's (*choice*, and* sequence *respectively).*

Even though it is a vital feature of the *BSM* theoretical framework, for simplicity we omit the treatment of variables in the definitions of query and update formulae above. For a full fledged description of the BSM framework consult [12].

2.2 Semantics

The *yields* calculus, summarised below after [12], specifies an update associated with executing a mental state transformer in a single step of the language inter-

preter. It formally defines the meaning of the state transformation induced by executing an mst in a state, i.e., a mental state transition.

Formally, a *mental state* σ of a *BSM* $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \tau)$ is a tuple $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ of KR module states $\sigma_1 \in \mathcal{S}_1, \ldots, \sigma_n \in \mathcal{S}_n$, corresponding to $\mathcal{M}_1, \ldots, \mathcal{M}_n$ respectively. $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ denotes the space of all mental states over $\mathcal{A}$. A mental state can be modified by applying primitive mst's on it and query formulae can be evaluated against it. The semantic notion of truth of a query is defined through the satisfaction relation $\models$. A primitive query $\vDash\phi$ holds in a mental state $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ (written $\sigma \models (\vDash\phi)$) iff $\vDash(\phi, \sigma_i)$, otherwise we have $\sigma \not\models (\vDash\phi)$. Given the usual meaning of Boolean operators, it is straightforward to extend the query evaluation to compound query formulae. Note that evaluation of a query does not change the mental state σ.

For an mst $\oslash\psi$, we use $(\oslash, \psi)$ to denote its semantic counterpart, i.e., the corresponding *update* (state transformation). Sequential application of updates is denoted by $\bullet$, i.e., $\rho_1 \bullet \rho_2$ is an update resulting from applying ρ_1 first and then applying ρ_2. The application of an update to a mental state is defined formally below.

Definition 2 (applying an update). *The result of applying an update $\rho = (\oslash, \psi)$ to a state $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ of a* BSM *$\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$, denoted by $s \bigoplus \rho$, is a new state $\sigma' = \langle \sigma_1, \ldots, \sigma_i', \ldots, \sigma_n \rangle$, where $\sigma_i' = \sigma_i \oslash \psi$ and σ_i, $\oslash$, and ψ correspond to one and the same $\mathcal{M}_i$ of $\mathcal{A}$. Applying the empty update* **skip** *on the state σ does not change the state, i.e., $\sigma \bigoplus \mathbf{skip} = \sigma$.*

Inductively, the result of applying a sequence of updates $\rho_1 \bullet \rho_2$ is a new state $\sigma'' = \sigma' \bigoplus \rho_2$, where $\sigma' = \sigma \bigoplus \rho_1$. $\sigma \overset{\rho_1 \bullet \rho_2}{\rightarrow} \sigma'' = \sigma \overset{\rho_1}{\rightarrow} \sigma' \overset{\rho_2}{\rightarrow} \sigma''$ denotes the corresponding compound transition.

The meaning of a mental state transformer in state σ, formally defined by the *yields* predicate below, is the update set it yields in that mental state.

Definition 3 (yields calculus). *A mental state transformer τ yields an* update *ρ in a state σ, iff $yields(\tau, \sigma, \rho)$ is derivable in the following calculus:*

$$\frac{\top}{yields(\mathbf{skip}, \sigma, \mathbf{skip})} \quad \frac{\top}{yields(\oslash\psi, \sigma, (\oslash, \psi))} \quad \textit{(primitive)}$$

$$\frac{yields(\tau, \sigma, \rho), \sigma \models \phi}{yields(\phi \longrightarrow \tau, \sigma, \rho)} \quad \frac{yields(\tau, \sigma, \rho), \sigma \not\models \phi}{yields(\phi \longrightarrow \tau, \sigma, \mathbf{skip})} \quad \textit{(conditional)}$$

$$\frac{yields(\tau_1, \sigma, \rho_1),\ yields(\tau_2, \sigma, \rho_2)}{yields(\tau_1 | \tau_2, \sigma, \rho_1),\ yields(\tau_1 | \tau_2, \sigma, \rho_2)} \quad \textit{(choice)}$$

$$\frac{yields(\tau_1, \sigma, \rho_1),\ yields(\tau_2, \sigma \bigoplus \rho_1, \rho_2)}{yields(\tau_1 \circ \tau_2, \sigma, \rho_1 \bullet \rho_2)} \quad \textit{(sequence)}$$

We say that τ yields an update set *ν in a state σ iff $\nu = \{\rho | yields(\tau, \sigma, \rho)\}$.*

The mst **skip** yields the update **skip**. Similarly, a primitive update mst $\oslash\psi$ yields the corresponding update $(\oslash, \psi)$. In the case the condition ϕ of a conditional mst $\phi \longrightarrow \tau$ is satisfied in the current mental state, the calculus yields one of the updates corresponding to the right hand side mst τ, otherwise the no-operation

skip update is yielded. A non-deterministic choice mst yields an update corresponding to either of its members and finally a sequential mst yields a sequence of updates corresponding to the first mst of the sequence and an update yielded by the second member of the sequence in a state resulting from application of the first update to the current mental state.

The following definition articulates the denotational semantics of the notion of mental state transformer as an encoding of a function mapping mental states of a *BSM* to updates, i.e., transitions between them.

Definition 4 (mst functional semantics). *Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be KR modules. A mental state transformer τ encodes a function $\mathfrak{f}_\tau : \sigma \mapsto \{\rho | yields(\tau, \sigma, \rho)\}$ over the space of mental states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle \in S_1 \times \cdots S_n$.*

Subsequently, the semantics of a *BSM* agent is defined as a set of traces in the induced transition system enabled by the *BSM* agent program.

Definition 5 (BSM semantics). *A* BSM $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ *can make a step from state σ to a state σ', iff $\sigma' = \sigma \bigoplus \rho$, s.t. $\rho \in \mathfrak{f}_\mathcal{P}(\sigma)$. We also say, that $\mathcal{A}$ induces a (possibly compound) transition $\sigma \xrightarrow{\rho} \sigma'$.*

A possibly infinite sequence of states $\sigma_1, \ldots, \sigma_i, \ldots$ is a run of BSM $\mathcal{A}$, iff for each $i \geq 1$, $\mathcal{A}$ induces a transition $\sigma_i \rightarrow \sigma_{i+1}$.

The semantics of an agent system characterized by a BSM $\mathcal{A}$, is a set of all runs of $\mathcal{A}$.

Additionally, we require the non-deterministic choice of a *BSM* interpreter to fulfil the *weak fairness condition*, similar to that in [9], for all the induced runs.

Condition 1 (weak fairness condition) *A computation run is weakly fair iff it is not the case that an update is always yielded from some point in time on but is never selected for execution.*

2.3 Jazzyk

Jazzyk is an interpreter of the *Jazzyk* programming language implementing the computational model of the *BSM* framework. The syntax of the *Jazzyk* language is an instantiation of the abstract mathematical syntax of the *BSM* theoretical framework. when ϕ then τ construct encodes a conditional *mst* $\phi \longrightarrow \tau$. Symbols ; and , stand for choice | and sequence $\circ$ operators respectively. To facilitate operator precedence, mental state transformers can be grouped into compound structures, blocks, using curly braces {...}.

To better support source code modularity and re-usability, *Jazzyk* interpreter integrates GNU M4[1], a state-of-the-art macro preprocessor. Macros are a powerful tool for structuring and modularizing and encapsulating the source code and writing code templates. Before feeding the *Jazzyk* agent program to the language interpreter, first all the macros are expanded. Listing 1 depicts a *Jazzyk* code snippet. For further details on the *Jazzyk* programming language and the macro preprocessor integration with *Jazzyk* interpreter, consult [12].

[1] `http://www.gnu.org/software/m4/`

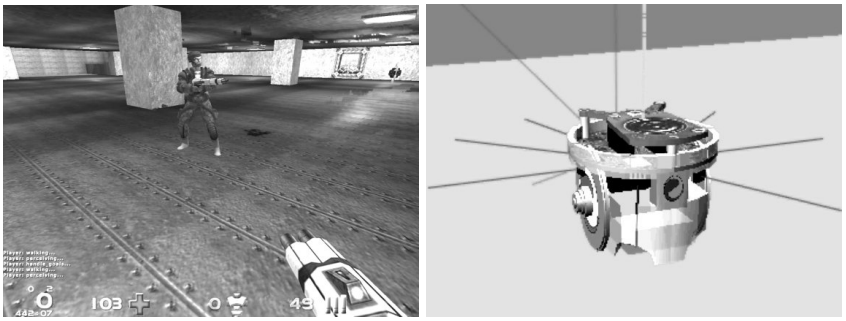


Fig. 1. Screenshots of the *Jazzbot* and *Urbibot* agents.

3 Jazzbot

Jazzbot is a virtual agent embodied in a simulated 3D environment of the first-person shooter computer game *Nexuiz*[2]. It is a goal-driven BDI inspired cognitive agent developed with the *Jazzyk* language. The *Nexuiz* death-match game takes place in a virtual building containing various objects (e.g., weapons, flags or armor kits), is capable of simulating diverse terrains like solid floor, or liquid and provides a basic means for inter-player interaction. Because of its accessibility (*Nexuiz* is published under the open source GNU GPL licence), we chose the *Nexuiz* game server as the simulator for *Jazzbot* case-study, the first larger proof-of-concept application for the *Jazzyk BSM* framework. Figure 1 left depicts a screenshot of the *Jazzbot* agent acting in the simulated environment. Demonstration videos and source code can be found on the project website[3].

Jazzbot's behaviour is implemented as a *Jazzyk* program. In the experimental scenario, the bot searches for a particular item in the environment, which it then picks up and delivers to the base point. While during the search phase the agent tries to always move to unexplored segments of the environment, when it tries to deliver the item, it exploits a path planning algorithm to compute the shortest path to the base point. Hence, during the search phase, in every step the bot randomly selects a direction to move to a previously unexplored part of the building and in the case there is none such, it returns to the nearest waypoint from which an unexplored direction exists. The behaviour for environment exploration is interrupted, whenever *Jazzbot* feels under attack, i.e., an enemy player attempts to shoot at it. Upon that it triggers emergency behaviours, such as running away from the danger. After the sense of emergency fades away, it returns back to its previously performed goals of item search, or delivery.

The *Jazzbot*'s control cycle consists of three steps that are executed sequentially. Firstly, the bot reads its sensors (perception), then if necessary it deliberates about its goals, (goal commitment strategies implementation) and finally it

[2] `http://www.alientrap.org/nexuiz/`
[3] `http://jazzyk.sourceforge.net/`

Listing 1 Code snippet from the *Jazzbot* agent.

```
define('ACT','{
  /* The bot searches for an item, only when it does not have it */
  when ⊨goals [{ task(search(X)) }] and not ⊨beliefs [{ hold(X) }] then SEARCH('X');
  /* When a searched item is found, it picks it */
  when ⊨goals [{ task(pick(X)) }] and ⊨beliefs [{ see(X) }] then PICK('X') ;
  /* When the bot finally holds the item, it deliver it */
  when ⊨goals [{ task(deliver(X)) }] and ⊨beliefs [{ hold(X) }] then DELIVER('X') ;
  /* Simple behaviour triggers without guard conditions */
  when ⊨goals [{ task(wander) }] then WALK ;
  when ⊨goals [{ task(safety) }] then RUN_AWAY ;
  when ⊨goals [{ task(communicate) }] then SOCIALIZE
}')
```

selects an action according to its actual goals and beliefs (act). Listing 1 provides an example code implementing selection of goal oriented behaviours, realized as parametrized macros, triggered by *Jazzbot*'s goals. While the bot simply triggers behaviours for walking around, danger aversion and social behaviour, execution of behaviours finally leading to getting an item are guarded by belief conditions.

The Figure 2 provides an overview of the *Jazzbot*'s architecture. The agent features a belief base consisting of two KR modules for representation of agent's actual beliefs and storing the map of the environment, a goal base encoding interrelationships between various agent's declarative, performance and maintenance goals and finally the module interfacing the bot with the simulated environment.

JzNexuiz KR module (cf. Subsection 5.4), the *Jazzbot*'s interface to the environment, the body, provides the bot with capabilities for sensing and acting in the virtual world. The bot can move forward, backward, it can turn, or shoot. Additionally, the *Jazzbot* is equiped with several sensors: GPS, sonar, 3D compass and an object recognition sensor. The module communicates over the network with the *Nexuiz* game server and thus provides an interface of a pure client side *Nexuiz* bot, i.e., the bot can access only a subset of the perceptual information a human player would have available.

The *Jazzbot*'s *belief base* is composed of two modules: *JzASP* (cf. Subsection 5.1) and *JzRuby* (cf. Subsection 5.2). While the first one integrates an *Answer Set Programming* [2] (*ASP*) solver *Smodels* [18] and contains a logic program reflecting agent's beliefs about itself, the environment, objects in it and other players, the second, based on an interpreted object oriented programming language *Ruby*, stores the map of the agent's environment.

The *Jazzbot*'s *goal base* is again an *ASP* logic program representing agent's current goals and their interdependencies. Goals can be either of a declarative (*goals-to-be*), or performative nature (*goals-to-do*, or *tasks*). In *Jazzbot* agent implementation, each *goal-to-do* activates one, or more *tasks*, which in turn trigger, one or more corresponding behaviours the agent is supposed to execute. On the ground of holding certain beliefs, the agent is also allowed to adopt new, or drop goals which are either satisfied, irrelevant, or subjectively recognized as impossible to achieve. The agent thus implements goal *commitment strategies*. We explore the details of the programming methodology employed in the *Jazzbot*

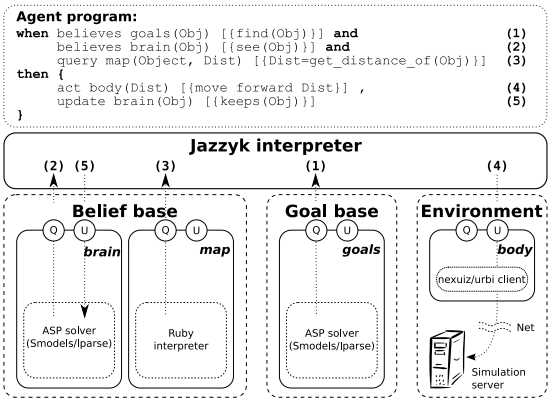


Fig. 2. Internal architecture of *Jazzbot* and *Urbibot* agents.

project in [14]. The Section 5 below provides details on the implementation of the *Jazzbot*'s KR modules.

4 Urbibot

Urbibot [4] is the second case-study developed as a step towards applications of the *Jazzyk BSM* framework in the mobile robotics domain. It is a robot exploring a maze where it searches for red poles and then tries to kick them down while at the same time avoiding patrols policing the space. *Urbibot* is embodied as an *e-Puck*[4], a small educational mobile robot simulated in *Webots*[5] [10], a robotics oriented physical world simulator. The robot is steered using *URBI*[6], a highly flexible and modular robotic programming platform based on event-based programming model. The main motivation for using *URBI* is the direct transferability of the developed agent program from simulator to the real robot.

Similarly to the *Jazzbot*, the overall agent design is inspired by the BDI architecture and reuses parts of the code developed for *Jazzbot*. In turn, except for using *JzASP* KR module to represent agent's beliefs about itself, *Urbibot* features similar agent architecture as the one depicted in the Figure 2 for the *Jazzbot* agent. *Urbibot*'s beliefs comprise exclusively information about the map. The interface with the simulator environment is provided by the *JzUrbi* KR module (see Subsection 5.3).

As already noted above, *Urbibot*'s behaviour is similar to that of *Jazzbot* agent. However instead of controlling the agent's body with rather discrete commands, such as `move forward`, or `turn left`, *Urbibot*'s *URBI* allows a more sophisticated control by directly accessing the robot's actuators, which in the case of the *e-Puck* robot, are only it's two wheels. The robot features a mounted

[4] `http://www.e-puck.org/`
[5] `http://www.cyberbotics.com/`
[6] `http://www.gostai.com/`

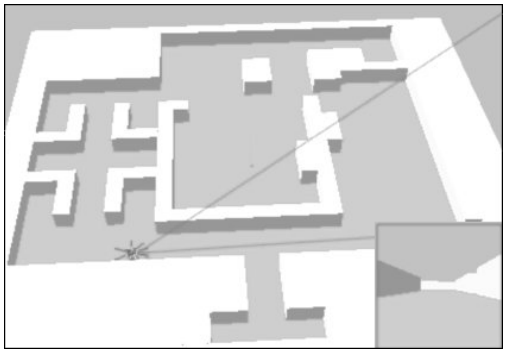


Fig. 3. *Urbibot* exploring the simulated environment. The lower right corner provides the current snapshot of the *Urbibot*'s camera perception.

camera, a directional distance sensor and an additional GPS sensor (our customization of the original *e-Puck* robot). In the *JzRuby* module, the robot analyzes the camera image stream and by joining it with the output of the distance and GPS sensors it constructs a 2D map of the environment. Upon encountering a patrol robot, *Urbibot* calculates an approximation of the space the patrol robot can see, and subsequently tries to navigate out of this area as quickly as possible. Again, the details on the implementation of the *Urbibot*'s KR modules can be found later in the Section 5. Figures 1 (right) and 3 depicts a screenshot of the *Urbibot* agent and the maze environment with the *Urbibot* acting in it. Furthermore, demonstration videos and source code are provided in the corresponding section of the *Jazzyk* project website[3].

5 Modules

The *Jazzyk Software Development Kit (Jazzyk SDK)* provides a C++ interface from which each KR module plug-in has to be derived. Basically, the class defines five methods: `initialize`, `finalize`, `cycle`, `query` and `update`. It is possible to define multiple `query` and `update` methods corresponding to KR module's query and update operators. These methods then define the plug-in's interface to the *Jazzyk* interpreter. While initialize, finalize and cycle are mainly used for initialization, shutdown and maintenance of the module the query and update provide means for modification of the stored knowledge base.

Below we describe KR module's we employed in development of the *Jazzbot* and *Urbibot* case studies. We introduce two modules facilitating agent's knowledge representation *JzASP* and *JzRuby* followed by description of two modules, *JzUrbi* and *JzNexuiz*, interfacing the agents with their respective environments.

5.1 JzASP

In [5] we presented the *JzASP* module. It integrates *Lparse* [17] and *Smodels* [18], an *Answer Set Programming* grounder and solver respectively. The stored knowledge base thus consists of a logic program in the syntax of *AnsProlog** [2] (*Prolog* style syntax) and can be accessed by two query methods sure_believes and poss_believes and two update methods add and del allowing for retrieval and modification of the stored knowledge base. Internally, the *JzASP* module processes the program by passing it to the *Lparse* library and subsequently let's *Smodels* solver to compute the program's answer sets.

The two query methods sure_believes and poss_believes check whether the query formula, an *AnsProlog** term, is contained in all the computed answer sets, or there exists at least a single answer set containing it respectively. Before the query formula is processed by the module, all the free variables occurring in it are substituted by their valuations and subsequently, the query method attempts matching the remaining free variables with a term from a computed answer set.

The update interface methods add and del provide a means to assert, or retract a clause (a fact, or a rule) to/from the stored knowledge base. The variable substitution treatment is similar to that in processing query formulae.

While the *Jazzbot* agent employs the *JzASP* for both, reasoning about its beliefs regarding its environment, other agents and its own body state, as well as to represent and reason about its goals, the *Urbibot* agent employs the module only to treat its goal base. Using the power of non-monotonic reasoning, in particular the default negation, to reason about agent's goals turned out to be advantageous and led to an elegant encoding of interrelations between various goals. We elaborate more on the technique used in [14].

5.2 JzRuby

The *JzRuby* module, detailed description in [4], integrates the interpreted object oriented scripting language *Ruby*[7]. The KR module interface methods for initialization and finalization as well as the query and update routines are able to process plain *Ruby* programs as argument formulae (query/update). *Jazzyk* variables are treated as global variables in the *Ruby* interpreter's memory space. Query invocations of the single query method return the truth value of the code invocation within the *Ruby* interpreter, i.e., provided the code execution yields a value other than 0, the KR module returns $\top$ and $\bot$ otherwise. The single update method of the KR module simply executes the provided update formula, a plain *Ruby* code chunk.

To represent the *Jazzbot*'s information about the topology of its environment, the agent uses a *circle-based waypoint graph* (*CWG*) [16] to generate the map of its environment. *CWG*'s are an improved version of *waypoint graphs*, extended with a radius for each waypoint. The radius is determined by the distance between the avatar and the nearest obstacle. This technique ensures, especially in

[7] `http://www.ruby-lang.org/`

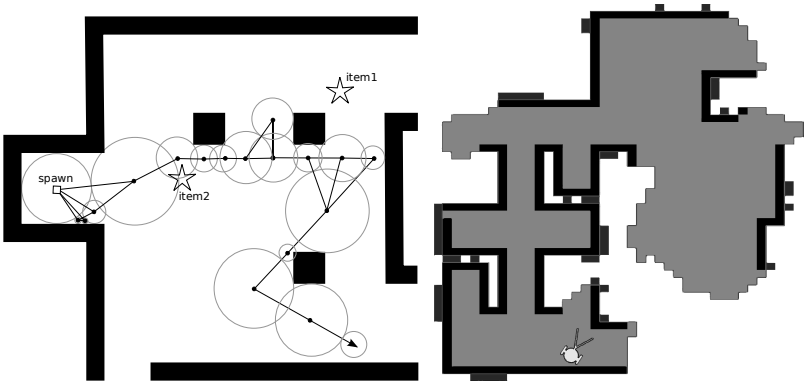


Fig. 4. Environment maps representation in *Jazzbot* and *Urbibot*.

big rooms or open spaces, a smaller number of nodes and connections within the graph what in turn speeds up the path search algorithm. Figure 4 (left) shows a graphical representation of the *CWG* for a sample walk of the *Jazzbot* agent from the spawn point to the point marked by the arrow.

Additionally, each waypoint stores a list of objects present within its range as well as about walls touching it and information about unexplored directions, i.e., such in which there's no connection to another waypoint, nor a wall. By employing a breadth-first graph search algorithm, the agent can compute the shortest path to a particular object, or a position.

The *CWG* graph is constructed by the agent so that in each step it determines whether its current absolute position corresponds to some known waypoint, and if not, it turns around in 60° steps and by checking its distance sensor, it determines the nearest obstacle around. Subsequently, the newly added waypoint is incorporated into the *CWG* by connecting it to all the other waypoints with which it overlaps and all the perceived objects together with all the directions in which the agent can see a wall are stored with it.

While similarly to the *Jazzbot*, the *Urbibot* uses the *JzRuby* KR module for representing its environment too, the map representation approach differs. *Urbibot* represents the map of its environment as a 2D grid, where in each cell it stores an information about its content, such as `unknown`, `free`, `wall`, `patrol`, `avoid` or `pole`. A new row or column is added to the grid when the robot reaches the edge of the known region. While the agent continuously adds new information to cells, the map becomes more and more precise. Furthermore, the *Urbibot* employs the A^* path planning algorithm to compute the shortest path between the current position and a position to go, be it an unexplored cell, the nearest safe cell a patrol robot cannot see, or a cell containing a pole which it tries to kick down. Also, the *Urbibot* uses the *JzRuby* module to algorithmically process the sensory input from its mounted camera and the distance sensors.

5.3 JzUrbi

In order to interface a *Jazzyk* program with the robot's body, the *JzUrbi* KR module [4] integrates the *URBI* programming language interpreter. It connects over TCP/IP to an *URBI* server on the simulator's side, or with the *URBI* robot controller that controls the robot's body.

The single query method query provide the agent program with the sensor information from the body. *Jazzyk* variables are treated the same way as in *JzRuby*, i.e., as global variables of the underlying *URBI* interpreter. Similarly, the single update method update simply sends the provided update formula, an *URBI* program, to the *URBI* server.

The *Urbibot* connects over the *JzUrbi* module with the *URBI* server running in the *Webots* simulator and thereby steers the *e-Puck* robot, extended with a GPS sensor. In the particular case of the *Urbibot*, the sensory input accesses the following sensors: camera, distance sensor, GPS, touch-sensor and a light-sensor. Together with the update interface steering the *Urbibot*'s two wheels, these two methods provide the basic interface for *e-Puck*'s control.

5.4 JzNexuiz

Finally, the *JzNexuiz* KR module, invented in [8], facilitates *Jazzbot*'s interaction with the *Nexuiz* game environment. By the means of a single query interface method sense and a single update method act, it enables control of the *Jazzbot*'s avatar body in the virtual building. The query method provides access to several sensors of GPS, sonar, 3D compass and object recognition. The update method allows issuing commands to the avatar's body, such as move, turn, jump, use, attack or say.

Technically, the module connects over TCP/IP with a *Nexuiz* server and thus provides an interface of a pure client side *Nexuiz* bot. The consequence of this setup is that the *Jazzbot* agent can access only a strict subset of the perceptual information a human player would have. The *Jazzbot* plug-in integrates a stripped down and customized *Nexuiz* client. In turn, the bot's actions are implemented as the corresponding key strokes of a virtual player.

Figure 5 depicts the syntax accepted by the plug-in's query and update interface methods sense and act. Each query formula starts with the name of the accessed virtual sensor device followed by the corresponding arguments being either constants, or variables facilitating retrieval of information from the environment. Similarly, the update formulas consist of the action to be executed by the avatar followed by a list of arguments specifying the command parameters.

The truth values of query formula evaluation depends on the sensory input retrieved from the environment. In the case the query evaluates to true ($\top$), the additional information about e.g., the distance of an obstacle, or the reading of the body health sensor, is stored in provided free variables.

```
nex_query  ::= sensor (constant | variable)+
nex_update ::= action (constant | variable)+
sensor     ::= sen_const | variable
sen_const  ::= 'body' | 'liquid' | 'ground' | 'gps' | 'compass' |
               'sonar' | 'map' | 'eye' | 'listen'
action     ::= act_const | variable
act_const  ::= 'move' | 'turn' | 'jump' | 'use' | 'attack' | 'say'
```

Fig. 5. *JzNexuiz* EBNF.

6 Experiences and Conclusion

The two case-studies described in this paper served us most importantly as a vehicle to nurture and pragmatically drive our research towards a methodology for using an agent oriented programming language exploiting strengths of heterogeneous KR technologies in a single cognitive agent system. For further details concerning the methodology consult [14]. As an important side effect, we collected experiences with programming BDI inspired virtual cognitive agents for computer games and simulated environments, as well.

As in the long run we aim at development of autonomous robots, in both cases the virtual agents had to be running autonomously and independently from the simulator of the environment. This choice had a strong impact on the design of the agents w.r.t. the action execution model and the model of perception. In both described applications, the agents are remotely connecting to the simulated environment in which they execute actions in an asynchronous manner, i.e., they can only indirectly observe the effects (success/failure) of their actions through later perceptions. As far as the model of perception is concerned, unlike other game bots, *Jazzbot* is a pure client side bot, i.e., the amount of information it can perceive is a strict subset of the information provided to the game client used by human players. Hence, the *Jazzbot* agent cannot take advantage of additional information, such as the global topology of the environment, or information about objects in distant parts of the environment, which are accessible to the majority of other bots available for first-person shooter games. In the case of *Urbibot*, the simulator provides only perceptions accessible to the models of robot's sensors. In our case these are most importantly a camera, a directional distance sensor and global positioning, hence the available information is, similarly to *Jazzbot*, only local, incomplete and noisy.

As both implemented agents are running independently from the simulation engine and execute their actions in an asynchronous manner, their efficiency is only loosely coupled to the simulation platform speed. In our experiments, the speed of agent's reactions was reasonable w.r.t. task the bots were supposed to execute. However, especially in the case of *Jazzbot*, due to deficiencies on the side of sensors, such as missing camera rendering the complete scene the bot can see, *Jazzbot* in its present incarnation cannot match the reaction speed of advanced human players in a peer-2-peer match.

Since, the agents store their internal state in the application domain specific KR modules, the control model of *Jazzyk BSM* framework results in agents which can instantly change the focus of their attention w.r.t. an observed change of the context in the environment. The goal orientedness of agent's behaviours emerges from the coupling between behaviour triggers and agent's attitudes modeled in its components [14]. This turned out to be of a particular advantage when a quick reaction to interruptions, such as an encounter of an enemy agent, or a patrol, was needed. On the other hand, because of the open plug-in architecture of the *Jazzyk BSM* framework, we were able to quickly prototype and experiment with various approaches to knowledge representation and reasoning, as well as various models of interaction with the environment.

Our research project follows the spirit of [7], where Laird and van Lent argue that approaches for programming intelligent agents should be tested in realistic and sophisticated environments of modern computer games. *Jazzbot* project thus follows in footsteps of their *SOAR QuakeBot* [6].

Another relevant project, *Gamebots* [1], provides a general purpose interface to a first-person shooter game *Unreal Tournament*[8]. *Gamebots*' approach is however server side, i.e., the virtual agent is provided with much more information than a human player has, what was not in the spirit of our aim to emulate mobile robots in a virtual environment. Why we did not pick the *Gamebots* framework for our project was also the fact, that it is specific to the commercially available game *Unreal Tournament* and since 2002, the project does not seem to be further maintained.

To our knowledge, our work on the *Jazzbot* and *Urbibot* case-studies is novel in the sense that it seems to be the first efficient application of non-monotonic reasoning framework of *ASP* in a highly dynamic domain of simulated robotics, or a first-person shooter computer game. Even though, to our knowledge the first attempt by Provetti et al. [15] uses *ASP* for planning and action selection in the context of the *Quake 3 Arena*[9] game, authors note that their bot could not recalculate its plans rapidly enough since each computation required up to 7 seconds in a standard setup [15]. Thus, in comparison to *Jazzbot* or *Urbibot*, their agent was capable to react to events occurring in the environment only to a lesser extend, because both their action selection and planning was in *ASP*.

Similarly, to our knowledge, the transparent integration of various KR technologies such as a declarative, logic based technology for representing agent's beliefs and goals, an object oriented language for storing the topological information about the environment together with a generic reactive control model of the agent program in the *Jazzyk BSM* framework is unique. In consequence, the *Jazzyk BSM* framework shows a lot of potential for further experimentation with synergies of exploiting various AI technologies in cognitive agent systems, especially in the attractive domain of virtual agents and autonomous non-player characters, for computer games. Yet, we believe, more experimentation is needed

[8] `http://www.unrealtournament3.com/`
[9] `http://www.idsoftware.com/`

to explore the limits and deficiencies of our approach in the domain of simulated, as well as physical reality embodied robotics.

References

1. R. Adobbati, A.N. Marshall, A. Scholer, S. Tejada, G.A. Kaminka, S. Schaffer, and C. Sollitto. Gamebots: A 3D Virtual World Test-Bed For Multi-Agent Research. In *Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, 2001.
2. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge, 2003.
3. Egon Börger and Robert F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, 2003.
4. Bernd Fuhrmann. Implementierung eines URBI- und Rubymoduls für Jazzyk zur Entwicklung von Robotern. Master's thesis, to appear.
5. Michael Köster. Implementierung eines autonomen Agenten in einer simulierten 3D-Umgebung - Wissensrepräsentation. Master's thesis, 2008.
6. J.E. Laird. It knows what you're going to do: adding anticipation to a Quakebot. In *Proceedings of the fifth international conference on Autonomous agents*, pages 385–392. ACM New York, NY, USA, 2001.
7. John E. Laird and Michael van Lent. Human-level AI's killer application: Interactive computer games. *AI Magazine*, 22(2):15–26, 2001.
8. David Mainzer. Implementierung eines autonomen Agenten in einer simulierten 3D-Umgebung - Interaktion mit der Umwelt. Master's thesis, 2008.
9. Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.
10. Olivier Michel. Webots: Symbiosis between virtual and real mobile robots. In *Virtual Worlds*, volume 1434 of *Lecture Notes in Computer Science*, pages 254–263. Springer Berlin / Heidelberg, 1998.
11. Peter Novák. Behavioural State Machines: programming modular agents. In *AAAI 2008 Spring Symposium: Architectures for Intelligent Theory-Based Agents, AITA'08*, pages 49–54, March 26-28 2008.
12. Peter Novák. Jazzyk: A programming language for hybrid agents with heterogeneous knowledge representations. In *Proceedings of the Sixth International Workshop on Programming Multi-Agent Systems*, pages 143–158, May 2008.
13. Peter Novák and Wojciech Jamroga. Code patterns for agent-oriented programming. In *AAMAS*, 2009, to appear.
14. Peter Novák and Michael Köster. Designing goal-oriented reactive behaviours. In *Proceedings of the 6th International Cognitive Robotics Workshop, CogRob 2008, July 21-22 in Patras, Greece*, pages 24–31, July 2008.
15. Luca Padovani and Alessandro Provetti. Qsmodels: Asp planning in interactive gaming environment. In José Júlio Alferes and João Alexandre Leite, editors, *JELIA*, volume 3229 of *Lecture Notes in Computer Science*, pages 689–692. Springer, 2004.
16. S. Rabin. *AI Game Programming Wisdom 2*. Charles River Media, 2004.
17. Tommi Syrjänen. Lparse 1.0 User's Manual. *University of Helsinki, Finland*, 2000.
18. Tommi Syrjänen and Ilkka Niemelä. The Smodels System. In *LPNMR*, pages 434–438, 2001.

Adaptive serious games using agent organizations.

Joost Westra[1], Hado van Hasselt[1], Frank Dignum[1], and Virginia Dignum[1]

Universiteit Utrecht

Abstract. Increasing complexity in serious games and the need to reuse and adapt games to different purposes and different user needs, requires distributed development approaches. The use of software agents has been advocated as a means to deal with the complexity of serious games. Current approaches to dynamic adjustability in games make it possible for different elements to adjust to the player. However, these approaches most use centralized control, which becomes impractical if the complexity and the number of adaptable elements increase. The serious games we are investigating are constructed using complex and independent subtasks that influence each other. In this paper, we propose a model for game adaptation that is guided by three main concerns: the trainee, the game objectives and the agents. In particular we focus on how the adaptation engine determines tasks to be adapted and how agents respond to such requests and modify their plans accordingly.

1 Introduction

Increasing complexity of software games, in particular of serious games [12, 13], and the need to reuse and adapt games to different purposes and different user needs, requires distributed development approaches. As games become more and more sophisticated in terms of graphical and technological capabilities, higher demands are also put on their content and believability. The use of software agents has been advocated as a means to deal with the complexity of serious games [8]. In these often many different characters performing complicated tasks interact with each others and with the trainee. In the case of the fire command training game, one could think of a situation where characters playing the role of fire agents need to be replaced by soldiers (for instance, when the disaster increases to a very complex situation). It would be very useful if this could be done without rewriting the complete strategy. Serious games are applications developed with game technology and game design principles for non-entertainment purposes, including games used for educational, persuasive, political, or health purposes. The goal of a serious game is to teach certain pre-specified tasks to the trainee of the game (the trainee). 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 does the gaming experience contributes to the learning goals of the trainee [3].

Believability is a main driver of game development. The search for enhanced believability has increasingly led game developers to exploit agent technology in games [8]. In particular, dynamic response, multiple conflicting goals, team work and cognitive models are all agent research issues that are relevant for game developers. Furthermore, serious games need to be suitable for many different people. Currently this adjustment is either done externally by experts that need to guide the adaptation, or the game provides a number of predefined levels based on (learner) stereotypes. While the use of expert guidance to adaptation usually results in quite effective training, such experts are rare, not available during the training session, and expensive. The use of predefined adaptation levels may lead to less optimal adaptation in the case trainees do not fit well with the stereotypes. Furthermore, the enhancement of the learning experiences requires to (automatically) adjust the game online.

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 fast as possible. Third, the behavior of the game must remain believable. In this paper we will mainly concentrate on the third aspect while trying to achieve the first two. 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 in serious games. This poses an extra burden on the believability of the game, namely coherence of long-term behavior [9].

The realization of the tasks of the trainee require the coordination of the actions of many different characters. For example, a serious game for training a fire commander includes scenarios aiming at learning how to make sure the victims in a burning building are saved. Adaptability to learning objectives implies these characters to show a spectrum of behavior. The victims could be more or less mobile or they could be located in simple or difficult locations. There may be bystanders that could obstruct the medics. The police could autonomously control the bystanders or could only act if ordered by the fire commander. Also the behavior of the medical personnel affects the difficulty of the task. As becomes clear from this example the difficulty of the trainee's task is very much dependent on the behavior of all the characters in the scenario, resulting in many variations of the same global task.

Performance of each subtask can not be measured separately because all behaviors influence each other. If all characters are allowed to adapt to the trainee without coordination, situations will occur where all adapt simultaneously resulting in unwanted scenarios for the trainee. For example, all victims become less mobile, all police agents become less autonomous and bystanders provide higher obstruction to medics. Consequently, adaptation should be coordinated according to the learning objectives while maintaining a coherent storyline. In previous work [15] 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, we introduce GAM, the model for game adaptation. The background and motivation for this model are discussed in section 3. In section 4 we describe the task selection mechanism based on user and game model, and in section 5 the consequences of the adaptation model in terms of agent architecture are described. Extensions and conclusions are discussed in the last section.

2 Game Adaptation Model

Systems of learning agents are usually very unpredictable. This is not a problem for applications like simulations or optimizations where only the end result matters. But for games, where the agents are adapting to the trainee during the game and thus the adaptation must have a direct beneficial influence, the system needs to be a lot more predictable. Therefore, adapting the game to the trainee in complex learning applications requires both learning capabilities and decentralized control. In order to guarantee the successful flow of the game and the fulfillment of the learning objectives, the system also needs to be able to describe and maintain global objectives and rules.

Adaptation in games is guided by three main concerns: the trainee, the game objectives and the agents. The trainee, its capabilities, learning objectives and learning style are central to the adaptation. The purpose of the game is to provide a suitable environment for the trainee, so the primary requirement for adaptation is the need to determine trainee's initial state, objectives and style. As important is the game setup. In order to ensure believability and therefore contribute to the learning experience of the trainee, adaptation should maintain coherence of the storyline and integrate seamlessly to the scene where the trainee is situated. In our approach, agents provide the behavior for different game elements. Agents can be behind a game character, but also control some other (non living) elements, such as the strength of the fire and the quality of obstacles in the fire commander training example given above. Each agent pursues its own goals, and should be able to adapt its behavior by providing different plans for a goal or a range of alternative goals. GAM (Game Adaptation Model) includes the elements described above as depicted in figure 1.

Based on the current state of the trainee and on the current moment in the game storyline, the adaptation engine should be able to determine what type of behavior should be requested from the agents. Based on this request, each agent can determine its own possibilities for adaptation (or a range of possibilities) that should fit with the agent's own goals and its own perception of its role in the storyline. Based on the proposals by the agents, the adaptation engine will determine the overall adaptation and request the chosen agents to change their behavior accordingly. Finally, the resulting game scene is presented to the trainee.

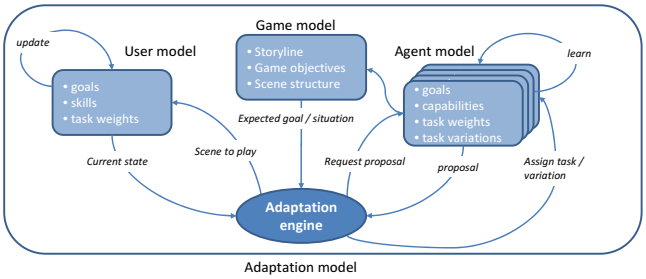


Fig. 1. The Game Adaptation Model

In training applications it usually is possible to make an estimation of the difficulty of a certain task. This can be done using domain expert knowledge that associates different task implementations with a specific difficulty level. These estimates can be updated in an offline learning phase and will get more accurate if more users have used the application. Having this information in advance significantly speeds up the adaptation because the algorithm not only needs to learn the strength relations between the different task implementations but is even able to directly select appropriate implementations for a given trainee skill level. The coordination of the difficulty of the tasks of the different agents is done using the organizational description of the storyline as given in the Game Model. If the trainee is performing above expectation and all the agents decide to increase the difficulty at the same time the application will probably be too difficult the next time. The adaptation engine will determine how much agents are allowed to adapt.

3 Background

As described in the previous section, we advocate to bring 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.

3.1 Adaptation in games

Even though many commercial games do not use any adaptation [10], already some research has been done on adaptation in games. However, most of this research focuses on adaptation of certain simple quantitative elements in the game. For example better aiming by opponents or adding more or a stronger type of opponents.

Most commercial computer games that have varying degrees of difficulty do not use online adaptation. They have a number of preset difficulty levels that

need to be selected in the beginning of the game. Offline learning could be used for these predefined levels but usually these are just scripted by hand. Current research on online adaptation in games is based on a centralized approach [14, 6]. 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 limit period 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.

Research has been done on using reinforcement learning in combination with adaptation to the user [14, 2]. Most of these algorithms rely on learning relatively simple (sub-)tasks. 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 (sub-)tasks 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. However, even when adding more enemies the algorithm should already take into account that they can only be added in unexplored areas and do not influence the progress of the game.

3.2 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. This is not a problem for applications like simulations or optimizations where only the end result matters. But for (serious) games, where the agents are adapting during the game and thus the adaptation directly has to have a beneficial influence, the system needs to be a lot more predictable. Systems with this type of characteristics have been successfully modelled using agent organization frameworks such as OperA [5]. 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 [15].

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 (the boxes in the figures are separate 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. 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. 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.

3.3 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 to game because not all information is known before the game 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 [4] 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 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 learning agents is available, in which multiple equivalent 2apl plans are available that are suitable for different skill levels [7].

4 Game Adaptation Engine

As discussed in the previous sections, current work on game adaptation does not fulfill the requirements of complex distributed serious game applications. We propose an adaptation approach that uses expert knowledge for the adaptation but is more flexible than current direct adaptation methods.
Important to keep in mind is that the task that needs to be executed by the trainee follows from the behavior of the agents. Each agent is responsible for adapting its own subtask(s) while making sure that believability is preserved. Think of the example where the trainee is a fire commander that needs to secure victims inside a burning building. Example of a subtask is controlling the fire, which can easily be increased in difficulty by letting the fire spread and grow more or less rapidly. However it would be unrealistic if the fire suddenly doubles in size without a gradual increase. In the same way, many more agents' subtasks influence the performance of the trainee. The bystanders could be obstructing the medical personal more if time progresses but it would be unbelievable if the number of bystander suddenly doubles. A typical example of subtasks directly interacting with each other occurs when there is a police agent that has the ability to influence the bystanders. These are however subtasks from the training perspective. The trainee needs to give orders to the police man if he is not performing well while dealing with the bystanders is a different task.

4.1 Task selection

The goal of the game engine as depicted in figure 1 is to select the best tasks that suit the needs of the user. Speed of adaptation is important in games to ensure a continuous flow, therefore, to make adaptation as fast as possible we use a flexible form of direct adaptation in the online phase. This means that all the different possible behavior variations for all agents are implemented a priori and stay fixed. The adaptation task is then to select a subset from this set of possible behaviors that is most suitable task for the user at a given moment. It should be noted that the tasks to be performed by the trainee are dependent on the game environment but even more on the possibilities provided by the (adaptive) agents. These agents can perform different plans based on task preference. For example, a task could be made easier if a cooperative agent is autonomously

performing a large portion of the task without requiring lots of input from the user.

Assuming an user model indicating the user aims, capabilities and learning objectives, the game engine will select a task that is most suitable for the trainee but also preserves the storyline and gameflow. We assume that the storyline is defined by an agent organization framework, such as OperA. At the beginning of the game, a default user model is used, which may be not a very accurate representation of the trainee but this model will be adapted according to the trainee's performance. On the other hand, each agent has it own preferences and these preferences are likely to conflict with each other. We have chosen to use a kind of combinatorial auction [11] to select the best suitable plans for all the agents.

The main objective of the adaptation engine is to choose the most optimal task for the trainee, given the user model, the game flow and the capabilities of the agents. The task of the trainee is created by combining the subtasks of the agents. The agents representing the different adaptable elements will make different possible variants of their subtasks available that can be executed in the next time step. The actual selection is thus dependent on the required task for the trainee and on the preferences of the agents.

Figure 1 shows a schematic overview of the whole task selection process. This is optimized on the skills of the trainee but also most fitting with the preferences of the agents. The preferences of the agents are different because they have as a goal to stay as consistent and believable as possible and to follow their own preferred ordering of tasks. The agent uses information from the Game Model where the ordering of tasks can be specified. During this task selection we also use the restrictions from the Game Model to select a fitting task according to the organizational requirements. An example of this is that in the organization it is specified that one agent always goes left and one agent goes right. In the case that one agent specified no preference while the other agent prefers to go right, the agent without a preference should select the variations where it goes left.

The resulting "selected task" is a combination of all the plans that will be executed by the agents. This task has a certain combined difficulty for all the separate learning tasks. After the task is completed this information can be used to update the user model. The update of the user model is also dependent on the performance of the user.

4.2 Formalization

In this section we will formalize some important aspects of a training simulation. Of each of these aspects, we will discuss how to obtain the values of these.

Definition 1. ***(Training)***
The complete training consist of a partially ordered set of tasks:

(D1) $M = (\{t_T^i\}, P_c)$

with P_c the partial ordering.

For a given task T,

(D2) $V_T = \{t_T^1, ..., t_T^n\}$

are the possible variations of T.

Each variation t_T^i is formed as a partially ordered set of subtasks:

(D3) $t_T^i = (X_T^i \subseteq S, P_T^i)$

The partial ordering P_x is selected from a number of predefined partial orderings $V_P = \{P_1, \ldots, P_n\}$ belonging to each task T.

S consists of all possible atomic subtasks:

(D4) $S = \{s_1, \ldots, s_x\}$

Where each subtask corresponds to an optional agent plan.

Subtask belong to different skill categories that need to be learned by the trainee. Skills correspond to a certain user task type such as *extinguishing a fire* or *giving team orders*.

Definition 2. ***(Difficulty)***
The difficulty for a variation is determined separately for each skill l. The difficulty for a skill is only dependent on the subtasks that correspond to that skill:

(D5) $d(t_T^i, l) = g(\{hd(s) | s \in X_T^i \wedge L(s) = l\})$

Function g determines the total difficulty for a skill category from the difficulty of the separate subtasks that belongs to the corresponding skill category. Currently function g uses the average difficulty but this can be improved in the future. Function hd determines the difficulty of the separate tasks, this depends on the selected subtask execution and are defined by a human expert. These values vary between 0 and 1. These values indicate the expected needed skill of a trainee to perfectly complete the task. For instance, on a task with a value of 0.5 on some property, a trainee with a skill level of 0.2 is expected to have difficulty completing the task, while a trainee with a skill of 0.6 should be able to perform the task perfectly. The assumption is that tasks with a skill level a little higher than the skill level of a user for certain properties can be expected to be challenging enough to be valuable for training, but not too hard to complete.

Definition 3. ***(Weights)***
The weights for each skill category are dependent on the partial ordering P_x:

(D6) $wgt(t_T^i, l) = hw(P_T^i, l)$

The possible partial ordering and the corresponding weight for each skill category are predefined by a human expert (function hw). These values indicate how much each skill is needed for the task. These weights will be used to determine how much to update each skill level of a trainee that performed the corresponding task. We require each weights value to be a real valued number $0 \leq hw(P_T^i, l) \leq 1$. There are two equivalent ways to establish these weight variables in a useful way. In the first case, each weight corresponds to the amount of impact that each property has on the training as a whole. For instance, one way to determine such weights is to use the expected amount of time needed for a certain subtask corresponding to a certain skill. Note that this is an orthogonal concept to difficulty, because subtasks that make up the largest part of a task are not necessarily also the hardest parts of that task. In the second case, a perhaps easier way to handle these weights variables is to separate the impact on learning as a whole from the make up of a certain task. Then, the weights of a task could be required to sum to one over all skills. The interpretation of a weight then is the part of that skill that is needed to perform this task. Such a tuple may be somewhat easier to construct, for instance by human experts. However, an indication of the size of the task compared to the training as a whole is needed, that should be stored in a different number V.

A reinforcement function is needed to evaluate the performance of the trainee. Such a function should be defined for each task, such that it can give feedback about the level of performance of the trainee on that task. The output of a reinforcement function is a number r, that can range from 0 to 1, where the interpretation is that when a trainee receives a reinforcement of $r = 1$ the task was performed perfectly. Conversely, a reinforcement of $r = 0$ indicates that the trainee failed to complete the task at all. We only consider reinforcement functions that are constructed by human experts, although for most domains some metrics such as the time needed to complete a task compared to the average time needed to complete the task by similar users could be used as reinforcement.

Definition 4. ***(User Model)***
This model is a tuple of size l:

(D7) $U = (u_1, u_2, \ldots, u_l)$

where each element indicates the expected skill level of the user on each corresponding skill category.

This model is hard to obtain a priori, since it would require a screening of each user on all the relevant skills. Therefore, in the next section we will discuss ways to update this model automatically.

The game organization model describes which combinations of subtasks are allowed at any state of the game. Each agent is able to provide some of the subtasks at a certain difficulty level. Furthermore, the user model will indicate the desired difficulty level for each skill category for the user task, $d_{U,p}$. Then, the objective of the adaptation engine is to determine a ordered set of subtasks that is allowed according to the game model, possible to be executed by a set of agents, and which difficulty is $\forall l : dfclt(t_T^i, l) = d_{U,l}$.

4.3 Updating User Models

As mentioned in the previous section, a user model is a tuple of size l containing estimates of skill levels for the user. For instance, these could be instantiated with a value of 0 for each skill, meaning we do not expect the user to be able to perform any skill to a desired level yet. For instance, these could be instantiated with a value of 0 for each skill, meaning we do not expect the trainee to be able to perform any skill to a desired level yet. We now allow a trainee to try to perform some scenario. This scenario will consist of a number of tasks that can potentially be divided further into subtasks. For each task we assume the difficulty $D = (d_1, \ldots, d_l)$ (using $d_i = dfclt(t^i_T, i)$) and weights $W = (w_1, \ldots, w_l)$ (using $w_i = wgt(t^i_T, i)$) are calculated and stored in tuples and that a reinforcement function exist. After a trainee completes any subtask for which each of these aspects is defined, we can update the user model as follows:

$$\forall u_i \in U: \quad u_i = (1 - \alpha_i w_i)u_i + \alpha_i w_i r \max(d_i, u_i) \ , \tag{1}$$

where $0 \leq \alpha_i \leq 1$ is a step size learning parameter. First assume that $u_i < d_i$. Since we required that w_i is between 0 and 1, the effect of this update is as follows: if the task was performed perfectly, $r = 1$ and u_i will get updated towards d_i with a step size dependent on α_i and w_i. If, on the other hand, the task failed completely, the update results in an update of u_i towards zero. It can be easily verified that for skills that are not needed for the task the trainee skill is not updated, since w_i is then equal to zero. Typically, we will want α_i to be reasonably high in order to learn quickly from the received reinforcements. It is however possible to set α_i to zero, for instance when the main goal of a certain task is to only update specific parts of the user model.

When $u_i > d_i$, the trainee is assumed to be able to perform a task with difficulty d_i perfectly. If that is indeed the case, $r = 1$ and the trainee is updated towards u_i. This implies the user model is not updated. However, if the trainee is not perfect and $r < 1$, the trainee will get a negative update that is dependent on how low the reinforcement in fact is. This update should ensure that the skill levels in the user model will converge to the actual skill of the user. We note that the expected user skill u_i will get updated positively every time that $rd_i > u_i$. This implies that for difficult tasks, a trainee with low skill does not need to perform perfectly to increase the expected skill. In general, we want users to increase their skill levels, so we want them to perform tasks where the expected reward is higher than u_i/d_i. If we are correct in estimating these expected reinforcements, we can easily determine which tasks are fruitful options for training. This brings us to the following important point: estimating the expected reinforcements.

5 Agent perspective

5.1 Action selection

Agents are modelled according to the BDI architecture, which allows for plan and goal selection. In particular, we use a modified version of 2APL that provides

learning capabilities to the agents [7]. It should be noted that in our approach, the individual agents are not responsible for the learning rate of the trainee. That is taken care of by the adaptation engine. However, the goals of the agents are explicitly determined to facilitate the trainee's objectives. Furthermore, we assume that all agents aim at being as helpful as possible and rather act in a scene, to support the trainee, than they are idle. This means that the agent is primarily responsible for fulfilling its own goals. This means that all tasks performed by the agents contribute in some way to the learning of the trainee, but agents are not directly responsible for finding the right combination of plans or to coordinate their tasks with the other agents.

Usually, the agent will propose to the adaptation engine multiple different plans that it would like to execute, according to its goals. We make a distinction between different types of plan sets that the agent can propose. At the highest level we have plans that are different because they are designed to fulfill different goals. Next, we have plans that are in a different class because they are designed to fulfill the same goal but use different action types to achieve this. For example, an agent could have multiple actions for satisfying the goal to extinguish a fire. It can achieve this by using water, or oxygen deplete the fire by using explosives. And finally we have multiple plans that use the same action type but have a different execution of this action that is similar but are created to propose different difficulty levels. For example, there could be multiple implementations of using a water hose to extinguish a fire with different degrees of effectiveness

Based on the information on the required difficulty of the subtask set, provided by the adaptation engine, an agent will generate one or more fitting proposals. The agent has two conflicting concerns when proposing actions to the adaptation engine. One the one hand, the agent has certain preferences related to keeping its behavior as consistent and believable as possible. On the other end should the agent also propose actions that allow for a suitable combination for the trainee. Suggesting suitable plans is not only dependent on the desired difficulty but also on information of the current environment and of past occurrences. For example, the agent can observe that another agent is already performing a certain task and the agents knows that only one agent can perform this task at the same time. The numbers of plans proposed by the agents have a big influence on the whole system. In the extreme case that each agent only proposes one option, the adaptation engine can only select which agents participate in the scene not what behavior they perform and which variation is used. This allows for little possible adaptation to the user and a higher chance that no valid combinations can be made. An advantage of proposing very few options is that the behavior of the agent is more believable because the agents propose actions that are best fitting with their own preferences (staying as believable as possible). In the other extreme where the agents propose all valid actions, there are a lot more possible combinations, allowing multiple valid solutions and more possibilities to adapt to the user. A disadvantage is that the agents could end up performing actions that do not fit their storyline very well. The agents are able to give preferences on the actions they propose but it depends on the adapta-

tion engine to which degree these preferences are used. The optimal solution will lie somewhere between these extremes. Agents should provide enough plans to allow for proper adaptation to trainee but not propose unrealistic plans, while the adaptation engine should keep the preferences of the agent into account.

The adaptation engine not only selects the best matching combination of plans but also is able to influence plans proposed by the agents to achieve a better combination for the trainee. If the organization detects the trainee would be best served if more agents suggest plans related to a certain action type at a certain time, it can request this from the agents. Remember that the agents already propose all reasonable options so this request should not be used to try to let the agents propose different plans without a special reason. One reason could be that there is no valid combination possible with the current bids, but this should not happen frequently. A situation that will occur more often is that agents are performing actions that continue for longer periods of time but can be terminated upon request. If this is not the case, the organization could have the agent switch plans with an active plan that should not be terminated at that time. Or if the organization always waits for the agents to finish its active plan before assigning a new plan, some plans that very easily could be switched might not be executed because the agent is performing a simple plan that takes a very long time (possible infinite) to complete.

Both the timing of switching plans as well as the decision to comply with a request from the adaptation engine are the responsibility of the agents. In some cases the agent is able to perform certain plans (and thus can be requested by the organization) that at that time would not fit the storyline of the game very well. Even though the expected combined difficulty of the task might be less suitable for the trainee if the agent does not comply to the request, it could still be a wise decision if the storyline of the game is better preserved.
We assume that when the default internal reasoning of the agents suggests preferences on actions, the behavior of the agent is more natural. This provides an advantage when compared with top-down orchestration of all the plans.

Definition 5. ***(Agent Proposal)***
An agent proposal is defined as a set of actions with their corresponding preference:

(D5) $Prop = \{(s, pref(s)) | x_l \leq hd(s) \leq y_l \wedge s \in S_{ok}\}$

With x_l and y_l defining the boundaries of the difficulty and S_{ok} defines the actions that are legal in regards to the storyline.

5.2 Learning agents

Agents are able to adapt their behavior in order to optimize achieving its own goals. On the level of individual adaptation, it is preferable that the agents do not learn completely new plans, as learning new plans is slow in online adaptation. Another disadvantage of learning new plans is that it requires exploration which might actually degrade performance. This leaves the agent with the ability to

learn which of its predesigned behaviors to select. The performance measure for this task is separate from the adaptation model. The agent still needs to explore different actions but all the available actions are viable options.

Three different methods can be used when doing this online adaptation. The most simple approach is to predefine expected outcomes for each action type. If the performance is worse than the predefined expected outcome then the agent should try a different action type. For tasks which can have the same feedback parameter for the different actions, reinforcement learning can be used to optimize the agents performance.[1] Agents can use reasoning for selecting different action types. For example, if the agent knows that a truck is blocking the road than it could better execute a different action where access to the road is not needed.

6 Conclusion

In this paper we discussed online adaptation in serious games. The basis for the adaptation lays with the use of learning agents. In order to coordinate the adaption of the agents we use an organizational framework that specifies the limitations of the adaptation in each context. We have shown how GAM, 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 GAM uses a user model, the preferences of the agents and uses the guidelines from the organization model.

We argued that an agent based approach for adapting complex tasks is more practical than a centralized approach. It is much more natural if 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 does not only need to be flexible, the designer also must be able to define the storyline and put certain restrictions on the combined behavior of the agents. Agent organization frameworks are very suitable for creating a flexible system that still must follow a certain progression and behave according to certain norms. Hence the use of OperA for the specification of the organization of the agents in the game.

The proposed model for game adaptation selects tasks that are most suitable for the trainee while following the specification of the game model and keeping the preferences of the separate agents into account. The combination of adaptations selected at each moment is done through a kind of auction mechanism

[1] It is beyond the scope of the article to discuss the implementation of the adaptation using reinforcement learning. As we discussed in the related work section we have already done some experiments with online learning using an extended version of 2apl. The learning will go very quickly because of the limited number possible action type.

that provides a balance between local optimization of the task and believability of the agent and overall difficulty of the situation for the trainee.

The next steps will be the actual testing of a complete serious game on firefighting with actual firefighters in the Netherlands and a professional simulation environment provided by VSTEP using the Quest3D engine. This is set up to take place in the near future.

References

1. G. Andrade, G. Ramalho, A. S. Gomes, and V. Corruble. Dynamic game balancing: An evaluation of user satisfaction. In J. E. Laird and J. Schaeffer, editors, *AIIDE*, pages 3–8. The AAAI Press, 2006.
2. G. Andrade, G. Ramalho, H. Santana, and V. Corruble. Extending Reinforcement Learning to Provide Dynamic Game Balancing. *Reasoning, Representation, and Learning in Computer Games*, 2005.
3. J. Brusk, T. Lager, A. Hjalmarsson, and P. Wik. Deal: dialogue management in scxml for believable game characters. In *Future Play '07: Proceedings of the 2007 conference on Future Play*, pages 137–144, New York, NY, USA, 2007. ACM.
4. M. Dastani and Meyer, J.-J. Ch. A practical agent programming language. *Proceedings of ProMAS 2007*, 2008.
5. V. Dignum. A Model for Organizational Interaction: based on Agents, founded in Logic. *SIKS Dissertation, series.*
6. R. Hunicke and V. Chapman. AI for Dynamic Difficulty Adjustment in Games. *Proceedings of the Challenges in Game AI Workshop, Nineteenth National Conference on Artificial Intelligence (AAAI'04).*
7. E. Kok. Learning Effective Rule Selection in 2APL Agents . Technical report, Master thesis, Utrecht University".
8. M. Lees, B. Logan, and G. Theodoropoulos. Agents, Computer Games and HLA. *International Journal of Simulation Systems, Science and Technology.*
9. D. Moffat. Personality parameters and programs. In *Creating Personalities for Synthetic Actors, Towards Autonomous Personality Agents*, pages 120–165, London, UK, 1997. Springer-Verlag.
10. S. Rabin. *AI Game Programming Wisdom.* Charles River Media, 2002.
11. T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.
12. N. Schurr, J. Marecki, J. P. Lewis, M. Tambe, and P. Scerri. The DEFACTO system: Training tool for incident commanders. In M. M. Veloso and S. Kambhampati, editors, *AAAI*, pages 1555–1562. AAAI Press / The MIT Press, 2005.
13. B. Silverman, G. Bharathy, K. O'Brien, and J. Cornwell. 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.
14. P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma. Adaptive game AI with dynamic scripting. 2006.
15. J. Westra, F. Dignum, and V. Dignum. Modeling agent adaptation in games. *Proceedings of OAMAS 2008*, 2008.

Intelligent Agent Modeling as Serious Game

D. W. F. van Krevelen

Section Systems Engineering — Faculty of Technology, Policy and Management
Delft University of Technology, Jaffalaan 5, 2628 BX Delft, The Netherlands
d.w.f.vankrevelen@tudelft.nl
http://www.sk.tbm.tudelft.nl/

Abstract. This paper proposes a new approach to serious gaming simulations designed for both educational and inquiry purposes. The idea is to enable users to construct, learn about and exchange strategies for their delegate agents which they can test in various simulated scenarios of complex multi-actor systems such as infrastructures or markets to receive automated feedback. Besides being a multi-agent simulation platform with intelligent tutoring, we propose users construct the behaviors for their delegate agents at their own level of expertise, simply selecting from a predefined set of behaviors, shaping desired behavior as it evolves in user-defined train scenarios, enacting example behavior from which general behavior models are induced, or advising exactly what the behavior model should look like. We motivate why we believe this requires a discrete event system specification that supports complex behavior modeling, evolutionary optimization, and visual programming. The envisioned platform is a low-threshold interface for knowledge exchange and discovery where learning takes place between users, from user to agent and vice versa.

1 Introduction

According to edutainment researcher Egenfeldt-Nielsen today, after some decades of educational games, we are witnessing the third generation educational use of computer games: *serious games* [1]. At the 2008 Game Developers Conference, serious game proponents Sawyer and Peters presented a comprehensive taxonomy that shows how different sectors, including public sectors like government and NGO's, defense, health care and education but also private sectors like marketing & communications, corporate and industry, are beginning to embrace 'serious games' for diverse purposes, ranging from training, education and research to advertisement, production and design [2]. These human-in-the-loop simulation games, designed either to collaboratively educate players about particular phenomena or to study human behavior within the simulated environments, have become an important medium for many applications and the body of related research is growing rapidly. This paper proposes a new game genre where users are supported at varying difficulty levels in modeling behaviors for delegate agents and we see several application areas.

Programming microworlds are a kind of serious game introduced by Seymour Papert [3] for primary and high school education where players (students) learn about mathematics, physics or programming by constructing agent behaviors in what he called Turtle Geometry. Simply put, we are interested in extending these with evolution, standardized modeling frameworks and automated tutoring feedback. We believe that such advanced programming microworlds hold great potential in complex, less formal domains for higher education and other uses as well which we categorized in educational, scientific and private/public applications.

In *educational* settings, users can design and exchange agents that are able to make many more decisions than the players could themselves. Since the game simply executes the user-defined behavior models and does not need to wait for user input on each decision, the game environment complexity can increase much further as players gain experience using a technique called *dynamic difficulty adjustment* applied in games. Also, since the behavior is already formalized in a model, the game environment can evaluate the behavior much more effectively to determine whether learning goals are being achieved and provide suitable challenges or hints as is done in intelligent tutoring systems.

In *scientific* settings, user-defined behavior models provide much richer game data to study than a traditional action history does, since the reasoning behind the actions is also available. One could also track changes in the underlying behavior model, so it becomes easier observe whether and when learning occurs. Given their formal structure, it becomes possible to use statistical methods to generalize behavioral patterns from the user-defined models stored from many sessions and apply these as human representative agents in other simulations without any human-in-the-loop.

Finally, in *private/public* settings an easy-to-use behavior modeling language would open up the realm of multi-agent simulation to a large audience of users besides computer scientists and students. Non-expert employees of corporations and governments could compare and experiment with new strategies in their particular domain, since the language allows them to understand how the agents behave. Similarly, individual consumers would be able to define exactly how they would like their negotiator agent to represent them in for instance electronic markets or legal courts.

In this paper we make some propositions for realizing the intelligent agent modeling game concept. First we review some of the existing theory related to intelligent agent modeling as serious game. We then describe the basic ideas of our game concept in more detail. Finally we conclude with a summary and discuss future developments.

2 Related Theory

This paper identifies some important challenges in implementing the agent modeling game concept. Creating a programming environment for agent behaviors that suits any type of domain requires some generalized behavior modeling and

simulation formalism that can deal with common procedural commands as well as domain specific concepts, while remaining suitable for evolving new rule sets. Another open question is whether and how such a formalism for agent simulation should adopt the beliefs-desires-intentions approach commonly applied in multi-agent systems. Furthermore, in order for non-experts to understand and manipulate the agent models these must be human-readable, something the evolved neural nets in the NERO game are not. Finally, to keep the usability threshold for non computer experts as low as possible some visual programming approach is likely to be more suitable than text-based interfaces.

2.1 Programming Microworlds

Almost every game mentioned in Sawyer and Peters' taxonomy [2] is created for the purpose of education or instruction. In these games, players learn about a particular rule system or environment, perhaps even without being aware, by exploring the behavior of its concepts through trial and error or what-if simulation, possibly facilitated by discussions with teachers or fellow students. One way to gain understanding on how the modeled environment behaves is to construct an agent that has to perform some task. For example, already in the 1970's Seymour Papert introduced Logo[1] as a *programming microworld* for children where they can program their Turtle agent to draw complex geometric figures [3]. In this constructionist teaching approach, kids not only learn new ways of tackling problems, that mistakes are essential for debugging your thinking, or that there are often many solutions, but they also gain a new, more personal perspective on the subject matter. Besides the formal domains with well-defined laws such as geometry and physics, also less 'formal' knowledge has been modeled since then thanks to knowledge engineering techniques such as those described by Schreiber et al. [4]. For instance, Virtual Leader[2] [5] helps the player to develop leadership skills and the multi-player Global Supply Chain Game[3] [6] educates groups of students about the complexity involved in managing a supply network.

Today's authoring environments or programming microworlds however seem to be targeted at primary and high school children only as they offer only well-defined rule systems such as introductory computer programming or animation in Boxer[4] [7], Alice[5] [8], ToonTalk[6] [9], Stagecast Creator[7] [10], Scratch[8] [11], Etoys[9] [12] and Kodu[10]. We are interested in creating microworlds for educating

[1] http://el.media.mit.edu/logo-foundation/
[2] http://www.simulearn.net/
[3] http://www.gscg.org/
[4] http://www.soe.berkeley.edu/boxer/
[5] http://www.alice.org/
[6] http://www.toontalk.com/
[7] http://www.stagecast.com/
[8] http://scratch.mit.edu/
[9] http://www.squeakland.org/
[10] http://research.microsoft.com/en-us/projects/kodu/

about more complex domains where uncertainty plays an important role, for instance constructing successful strategic behaviors for multi-actor infrastructures and markets.

Another, more recent application of programming microworlds is to use them not to validate the simulated system, but as a *research method*, either to study or to capture the richness of human behavior in complex systems. By examining how players (subjects) behave, researchers can potentially replicate their performance in agent-based simulations that would otherwise contain only simple heuristic rules and assumptions. For instance, the Trust and Tracing game is used to study cultural differences in trade by playing the game with players of different backgrounds and the Mango Chain Game studies the searching and bargaining sources of transaction costs [13]. The next step could be to use such games as interfaces to elicit or capture expert knowledge on how to behave in complex multi-actor systems.

Proposition 1. *Agent modeling games should accommodate both education (knowledge transfer to player) and inquiry (knowledge gain by expert/scientist).*

2.2 Agent-Based Models

From the social studies such as economy and biology eventually came computational models to describe whole societies and the effects of their interactions on particular global measures such as the distribution of wealth or health. As computation becomes cheaper and more powerful, simulations are moving from differential equations assuming homogeneous populations towards heterogeneous models with much more complexity. According to Van Dyke Parunak et al., "*agent-based modelling is most appropriate for domains characterised by a high degree of localisation and distribution and dominated by discrete decision.*"

Agent-based modeling is an approach that has gained great popularity in social and economic sciences to study and educate about phenomena emerging from the interactions of large heterogeneous populations [14]. The recent establishment of the Open Agent Based Modeling Consortium[11] [15] mirrors the field's growing popularity. Well-known examples of such agent-based models (ABMs) include John Conway's *Game of Life*, Wolfram's class of cellular automata, Epstein and Axtell's *SugarScape* [16], and today topics range from the evolution of epidemics, traffic jams to the self-organization of ants, sensor networks and unmanned aerial vehicles.

Recent years have seen the appearance of increasingly powerful agent-based modeling and simulation engines such as StarLogo[12] [17], NetLogo[13] [18, 19], Mason[14] [20] and RePast[15] [21]. Unfortunately, these engines have little or no

[11] http://www.openabm.org/
[12] http://education.mit.edu/starlogo/
[13] http://ccl.northwestern.edu/netlogo/
[14] http://cs.gmu.edu/~eclab/projects/mason/
[15] http://repast.sourceforge.net/

specification of the simulation formalisms that describe the underlying assumptions. Existing formalisms could clearly separate models from their reference or source systems as well as from the simulator as Zeigler et al. propose in their theory of modeling and simulation [22]. In fact, the authors argue that the discrete event system specification (DEVS) formalism is the most general type which can be applied to cellular automata simulations as implemented in aforementioned simulation engines (which is possible if one considers the cells to be agents). For example, Müller recently presented a formal semantics for event-based multi-agent simulation [23].

Dolk proposes to "apply model management design principles to design a *generalized agent-based modeling environment* (GAME) with ABMS language(s) and accompanying interfaces that support the entire spectrum of potential users: programmer, analyst, decision-maker, and also cross-fertilizes with existing OR/MS modeling paradigms." He mentions a caveat: "Model management was successful in capturing structural dimension of models, *not* the dynamic aspects prevalent in simulation" [24, p.6]. By adopting a single simulation formalism that separates a model from its simulator, the following issues could be resolved:

- *reusability*, the simulation engine can be applied to new models and vice versa;
- *maintenance*, the engine and/or model can be maintained separately;
- *consistency*, the models can be proven to have certain properties before running any simulation; and
- *validity*, the model can be validated and verified on other simulators.

Proposition 2. *Agent modeling games should follow a general simulation formalism to ensure validity of models across different simulation environments.*

2.3 Multi-Agent Systems

From the fields of Distributed Artificial Intelligence and Cognitive Psychology many frameworks have emerged that support the design and validation of complex organizations with (multiple) intelligent agents to respectively mimic or study their (collective) intelligence. Wooldridge and Jennings define these agents as processes implemented on a computer that have *autonomy* (they control their own actions); *social ability* (they interact with other agents through some kind of 'language'); *reactivity* (they can perceive their environment and respond to it); and *pro-activity* (they are able to undertake goal-directed actions) [25]. The Multi-Agent Systems (MAS) paradigm is already being adopted as a new framework for software development to extend the service-oriented approach for developing robust scalable software systems. Thanks to agent middleware or platforms such as JADE[16] [26] or AgentScape[17] [27], autonomous and mobile agents can go out on the Internet to interact on their owner's behalf.

[16] http://sharon.cselt.it/projects/jade/
[17] http://www.iids.org/research/aos

Today entire organizations are modeled and validated thanks to human-readable reasoning frameworks such as Beliefs-Desires-Intentions or BDI paradigm such as AgentSpeak [28], Jason[18], DESIRE [29, 30], 2APL[19] [31, 32] and GOAL [33]. These reasoning paradigms are also suited to support task delegation to intelligent agents in critical situations. For example, commanders could delegate reconnaissance tasks to a group of BDI agents (e.g. unmanned aerial vehicles) and keep updated on their goals and plans as they proceed. This behavioral transparency makes the delegate agents dependable parts of a cooperative man-machine team capable of completing complex tasks.

Proposition 3. *Agent modeling games should employ standard human-readable reasoning methods such as Beliefs-Desires-Intentions to increase generalizability.*

2.4 Evolutionary Algorithms

Evolutionary algorithms are population-based algorithms that apply principles of evolution are known to be successful in finding innovative solutions in a large variety of problems ranging from design and art to scheduling and control. Especially in complex, poorly understood, nonlinear, multi-objective domains, the simple mechanisms of evolution, variation and selection, have proven to be powerful optimization tools [34].

There are different dialects in evolutionary algorithms, including genetic algorithms and symbolic regression [35]. Genetic algorithms in particular have been combined with other machine learning paradigms to solve (agent) control problems traditionally approached with machine learning algorithms like Q-learning and Sarsa. For instance, when genetic algorithms combine with neural networks we get *neuroevolution* algorithms which combine the innovative power of evolution with the pattern recognition power of the neural nets [36]. Alternatively when genetic algorithms are combined with production rules, one gets different variants of *learning classifier systems* which may be less accurate than neuroevolution algorithms but produce control models that are much more human-readable [37].

Proposition 4. *Agent modeling games should be enabled to employ evolutionary machine learning algorithms to optimize behavior models.*

2.5 Visual Programming

The agent modeling environments mentioned earlier, Alice [8], ToonTalk [9], Stagecast Creator [10], Scratch [11] and Etoys [12] have one thing in common: each provides a visual programming language that children can understand so they can create their own games and animations. Most are based on the LogoBlocks research developed at MIT, later evolved into OpenBlocks [38] as used

[18] http://jason.sourceforge.net/
[19] http://www.cs.uu.nl/2apl/

in StarLogo TNG[20] [39, 40]. These are powerful languages that make efficient use of shapes and colors to represent command structures, procedures and data types. However, the programming blocks usually map directly onto traditional programming commands. An interesting addition to such languages would be the learning mechanisms that agents may use to autonomously learn and improve their own behavior.

Proposition 5. *Agent modeling games should provide users with a visual programming techniques to model their agent's behavior and learning approach.*

3 Agent Modeling Game Concept

We conceive our agent modeling game shown in Fig. 1 to extend around a *simulator* which takes as input some *test scenario* and several *behavior models*, one for each actor-agent in the scenario, to generate an *interaction trace* as output to visualize and analyze. Typically, an *expert* designs the test scenario, one or more *players* receive tutoring as they develop the behavior model for their actor-agent, and a *scientist* collects evidence. The actual interactions differ slightly depending on whether players select, shape, guide or advise the behavior model for their actor-agent.

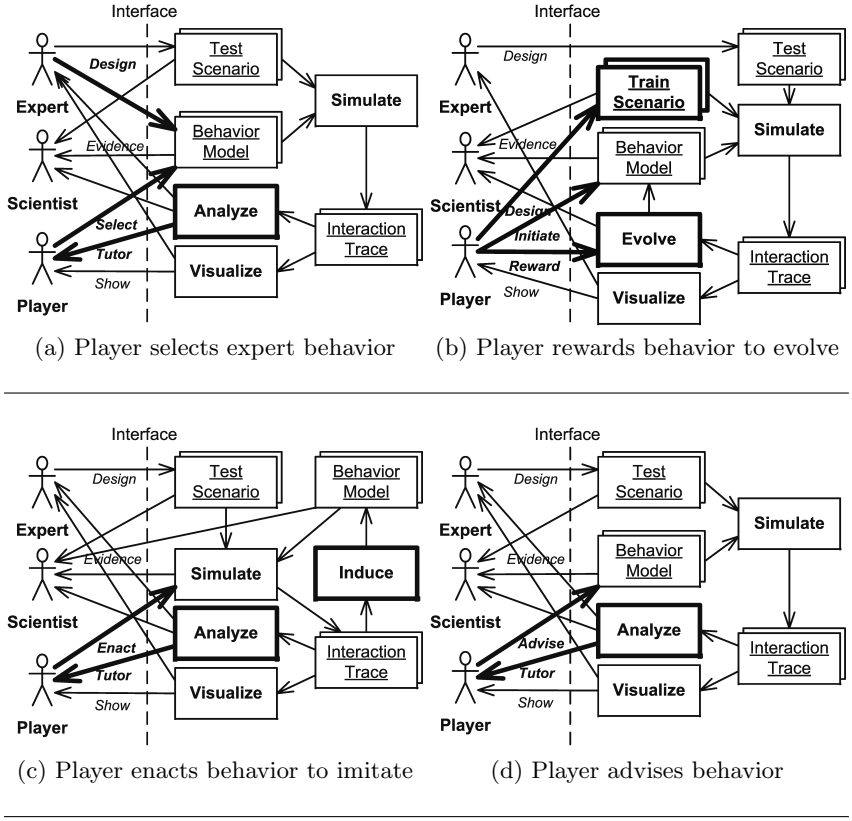
3.1 Selecting, shaping, guiding or advising your agent

The simplest way for a user to define his agent is to select from a set of predefined behaviors and possibly tweak some parameters (see Fig. 1a). This approach is like switching on the auto-pilot to see what happens and can be useful to find out how effective several by-the-book strategies actually are.

What makes our game concept a new genre in *serious* games is an important extension inspired in particular by Stanley and Miikkulainen's Neuro-Evolving Robotic Operatives[21] or NERO game [41]. In their novel kind of game, the user's objective is to train an army by designing increasingly complex learning environments in a *sand box*. Here intelligent agents evolve to optimize a user-defined reward function, before being combined into an army containing various specialties that will battle another army in a more complex environment. Such coaching by designing increasingly complex challenges is known in biology as *shaping* [42]. In stead of telling each soldier exactly what to do or programming their behavior to the letter, users become drill sergeants and think about how best to train their agents so they can be trusted to execute the desired strategy. We will investigate applying this method to the serious games (see Fig. 1b).

Besides shaping behaviors through simple reward knobs, alternative types of agent modeling could also be included in the end-user agent modeling game concept. In particular, we consider *imitation* (see Fig. 1c) and *advise giving* (see Fig. 1d) because they are more comprehensive thus suitable for more experienced

[20] http://education.mit.edu/starlogo-tng/
[21] http://www.nerogame.org/



(a) Player selects expert behavior (b) Player rewards behavior to evolve

(c) Player enacts behavior to imitate (d) Player advises behavior

Fig. 1. Serious game concept with four agent behavior modeling approaches.

users. Furthermore, they have already been tried and tested with games. For instance, Togelius applies imitation in car racing and helicopter control games [43], Bryant applies policy induction (imitation) to demonstrating user examples in turn-based strategy games [44] and Yong et al. extend the NERO game by incorporating user advise given in the form of rules into the behavior models [45]. These approaches together form a gradual learning path towards actually programming the agent behavior directly, which diSessa noted is difficult especially for real people who are not computer scientists [46]. As players are able to choose from several methods for modeling ones agent to match their current skill level, the game may presumably bring them into a state of *flow* as described by Csikszentmihalyi [47] where they loose track of time and become eager to learn.

3.2 Main Components

First we look at the components of an agent modeling game based on our current concept: the *users* (expert, scientist and player), the *information* (test and train

scenarios, behavior models and interaction traces) and the *processes* (simulate, analyze, visualize, evolve and induce). We also suggest some approaches which we consider useful for realizing each component.

Users We included three user types in the agent modeling game concept: the expert, the scientist and the player.

The *expert* typically designs and validates the test scenarios and some default behavior models that together can be run by the simulator engine. He designs these simulations (scenarios + behavior models) either to inquire about the effectiveness of new strategies or to educate players who in turn use the behavior models and scenarios as starting points for designing their own strategies that can deal with the test scenarios.

The *scientist* is basically just an observer, interested only in collecting anything put into the system by the experts and players such as the scenarios, behavior models, interaction traces and other simulation data. The scientist uses this data for inquiry for instance about the choices various users make or to see whether the player has learned something or whether the expert discovered something new.

The *player* can also modify and test scenarios and behavior models but is assumed to have less domain knowledge and will start from models that are already present in the system, either from experts or other players. The player may modify these models through techniques described earlier such as selecting, shaping, imitation or advising, whatever method best challenges the player's current skills. This may presumably bring players into a state of *flow* [47]. Players may be tutored by the system through hints based on some automated analysis and generalization of all models already stored in the system. By comparing these to the player's current behavior model the system may provide hints that are targeted at the player's current level of understanding, thus functioning as an intelligent tutoring system.

Selecting from predefined behaviors and advising new ones should be fairly simple to implement. We currently look at various human readable evolutionary approaches that could support the shaping of new behavior, including neuro-evolution extended with reverse engineering, genetic programming, and classifier systems. For the imitation process we also look at existing techniques that could fit our system specification.

Information To model a domain it is wise to use a well-defined ontology. Besides conceptual consistency, such an ontology also provides some advantages for implementation. For instance, one could imagine using the ontology as a formal specification of the objects and functions that must be present in the visual programming language that is offered to the end-users to model their environments (scenarios) and intelligent agents (behaviors). Perhaps it will be possible to automate the process of creating visual representations for each ontological concept. In this way game designers would only have to describe the ontology and never bother with any coding to adjust the visual programming interface.

Several *test and train scenarios* can be stored and shared within the system. As with the sandbox in the NERO game [41], users must have the ability to design environments in which to shape their delegate agents' behavior, increasing complexity stepwise.

The *behavior models* contain procedural information or rule sets that describe an actor-agent's behavior. In cases where existing or evolved behaviors do not suffice, users may wish to fine tune the behavior models by hand.

The *interaction traces* of each components input and output in the system will be the basis from which all learning takes place, both for the users as well as for the machine learning algorithm that may optimize behavior models. Of course, such traces easily become very complex and presenting them to the end-user in a way that facilitates learning effectively is likely an important and difficult problem. Perhaps even domain-specific preferences exist for presenting the game flow data to the user, so a large variety of graphs and other visualizations must be available if a game platform is to support many different types of games.

We currently have a basic ontology for describing socio-technical multi-actor systems for agent-based simulation. We intend to extend this to include procedural commands such that the ontology can support a visual programming language like OpenBlocks [38] as used in StarLogo TNG [39, 40], adapted to our domain of interest. Once the ontology holds these programming concepts we hope to automate the generation of the visual programming language using various colors and shapes. Including Beliefs-Desires-Intentions as a standardized reasoning formalisms based on has been done in the DEVS-based Global Supply Chain Game [6] but we need to evaluate this work further.

Processes There are five processes part of the agent modeling game concept: simulation, analysis, visualization, evolution and induction.

The *simulator* forms the core of the framework it must be robust and suited to host many different types of simulation environments. Not only must users be able to operate the simulator from the editor, an optimizer agent which executes the agent learning behavior must be able to run the simulator to explore variations through trial-and-error which users can exploit later. We are planning on implementing the general DEVS formalism using the Distributed Simulation Object Library[22] (DSOL) framework [48].

The *analyzer* serves as an intelligent tutoring system for the player, as an educational monitoring system for the expert and as an observational tool for the scientist. For the tutoring function we will investigate relevant literature on intelligent tutoring systems further so that we can perhaps apply case-based reasoning and provide users with hints that match their current level of expertise. Given that the scenarios will easily become very complex, the difficulty will lie in determining the user's current skill level and which existing behavior models have tackled problems which the user is unable to solve.

The *visualizer* must be able to show how the behavior models were responsible for the activities shown in the interaction traces, otherwise learning or

[22] http://www.simulation.tudelft.nl/

inquiry will be very hard for the users. We are thinking of lighting up parts of the behavior model which cause the behavior as it occurs. This way users will be able to determine what part of the model is responsible for the delegate agent's successes or failures.

The *evolution* process requires a reward function to be parameterized by the user in order to decide which agent behaviors are deemed successful for reproduction. Of course the reward functions themselves will be very different across various domains. They are also very important in determining the outcome of the evolutionary process, so design of this function must be done carefully. The user also provides training scenarios to challenge the agents with an increasing level of difficulty in order to optimize the evaluations from the reward function. Scenario editing should be very simple and the visual programming language should be able to deal with this as well.

The *induction* process requires some example behavior enacted by the user and tries to generalize these into some rule set useful in new, unseen scenarios. This may be one of the most difficult processes from a machine learning point of view. However its value is rather great for inquiry purposes using models of actual human behavior.

4 Conclusion

This paper presented a new serious game concept that revolves around end-user modeling of intelligent agents for knowledge transfer (education) and discovery (inquiry). To realize such a concept we made a number of propositions that are based on combining existing solutions and frameworks from various research areas. We suggest the adoption of a single simulation formalism such as DEVS, the adoption of a human-readable reasoning method such as BDI, the integration of human-readable reasoning methods with evolutionary algorithms such as classifier systems, and finally the adoption of a visual programming language like OpenBlocks generated from a shared ontology.

Simulation platforms that support end-user modeling of intelligent agents may not only prove to be powerful inquiry methods but also methods especially suited in education, for as the Latin proverb says: *docendo discimus* (by teaching we learn). Leelawong already applied the learning-by-teaching approach to agents using knowledge models only [49] whereas Thomaz and Breazeal applied reinforcement learning for their teachable agents [50]. However, we also identified some important gaps to bridge before we can realize our own agent modeling game concept which adds complexity.

Concerning our agent modeling game concept there remain some important questions that need answering before we know whether this is a successful approach for gaining and transferring knowledge. For instance, can users actually apply shaping, imitation and advise giving, do they want to, and what kind of results can we expect in terms of validity and performance? In future work we hope to answer these questions.

Acknowledgements This research is funded in part by the Next Generation Infrastructures Foundation[23] and in part by the Delft University of Technology. The author wishes to thank Alexander Verbraeck, Pieter Bots, Sebastiaan Meijer, Michele Fumarola and the anonymous reviewers for their insightful comments.

References

1. Egenfeldt-Nielsen, S.: Third generation educational use of computer games. Journal of Educational Multimedia and Hypermedia **16**(3) (July 2007) 263–281
2. Sawyer, B., Smith, P.: Serious games taxonomy (February 19 2008) Presented at the Game Developers Conference, San Francisco, CA, USA.
3. Papert, S.A.: Mindstorms: Children, Computers, and Powerful Ideas. Basic Books, Inc., New York, NY, USA (1980)
4. Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., de Velde, W.V., Wielinga, B.: Knowledge Engineering and Management: The CommonKADS Methodology. (17 December 1999)
5. Aldrich, C.: Simulations and the Future of Learning: An Innovative (and Perhaps Revolutionary) Approach to e-Learning. Pfeiffer, San Francisco, CA, USA (September 2004)
6. Corsi, T., Boyson, S., Verbraeck, A., van Houten, S.P.A., Han, C., MacDonald, J.: The real-time global supply chain game: New educational tool for developing supply chain management professionals. Transportation Journal **45**(3) (2006) 61–73
7. diSessa, A.A., Abelson, H.: Boxer: a reconstructible computational medium. Communications of the ACM **29**(9) (1986) 859–868
8. Pausch, R.: Alice: Rapid prototyping for virtual reality. IEEE Computer Graphics and Applications **15**(3) (1995) 8–11
9. Kahn, K.: Toontalk^{TM}—an animated programming environment for children. Journal of Visual Languages & Computing **7**(2) (June 1996) 197–217
10. Smith, D.C., Cypher, A., Tesler, L.: Programming by example: novice programming comes of age. Communications of the ACM **43**(3) (March 2000) 75–81
11. Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., Resnick, M.: Scratch: a sneak preview. In: C5'04: Proceedings. Second International Conference on Creating, Connecting and Collaborating through Computing. (2004)
12. Kay, A.: Interview with Alan Kay. Computers in Entertainment **1**(1) (Oct 2003) 8–8
13. Meijer, S.A.: The organisation of transactions – Studying supply networks using gaming simulation. PhD thesis, Wageningen Universiteit, Wagningen, The Netherlands (4 March 2009)
14. Schut, M.C.: Scientific handbook for simulation of collective intelligence. http://sci.collectivae.net/ (2007)
15. Janssen, M.A., Alessa, L.N., Barton, M., Bergin, S., Lee, A.: Towards a community framework for agent-based modelling. Journal of Artificial Societies and Social Simulation **11**(2) (31 March 2008) 6
16. Epstein, J., Axtell, R.: Growing Artificial Societies: social science from the bottom up. MIT Press, Cambridge, USA (1996)

[23] http://www.nginfra.org/

17. Resnick, M.: Decentralized Modeling and Decentralized Thinking. In: Modeling and Simulation in Precollege Science and Mathematics, New York, NY, USA (1999) 114–137
18. Wilensky, U.: NetLogo (1999) http://ccl.northwestern.edu/netlogo/.
19. Wilensky, U.: Modeling nature's emergent patterns with multi-agent languages. In: Proceedings of EuroLogo 2001, Linz, Austria (2001)
20. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: MASON: A multiagent simulation environment. Simulation **81**(7) (Jul. 2005) 517–527 http://www.cs.gmu.edu/ eclab/projects/mason/.
21. North, M.J., Collier, N.T., Vos, J.R.: Experiences creating three implementations of the repast agent modeling toolkit. ACM Transactions on Modeling and Computer Simulation **16**(1) (2006) 1–25
22. Zeigler, B.P., Praehofer, H., Kim, T.G.: Theory of Modeling and Simulation—Integrating Discrete Event and Continuous Complex Dynamic Systems. 2nd edn. Academic Press, Inc., San Diego, CA, USA (2000)
23. Müller, J.P.: Towards a formal semantics of event-based multi-agent simulations. In: MABS'08: Proceedings of the 9th International Workshop on Multi-Agent-Based Simulation at AAMAS 2008, Estoril, Portugal (May 12-13 2008)
24. Dolk, D.R.: Model management for agent-based modelling and simulation. In: IFIP WG 7.6 Workshop on Modelling and decision support for network-based services, Warsaw, Poland (1-3 September 2008)
25. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. Knowledge Engineering Review **10**(2) (1995) 115–152
26. Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with JADE. In Castelfranchi, C., Lespérance, Y., eds.: ATAL 2000: Proceedings of the 7th International Workshop on Intelligent Agents: Agent Theories, Architectures and Languages. Volume 1986., Boston, MA, USA (7-9 July 2001) 42–47
27. Brazier, F.M.T., Mobach, D.G.A., Overeinder, B.J., van Splunter, S., van Steen, M., Wijngaards, N.J.E.: Agentscape: Middleware, resource management, and services. In: SANE 2002: Proceedings of the 3rd International SANE Conference. (2002) 403–404
28. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: MAAMAW '96: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world: agents breaking away, Eindhoven, The Netherlands, Springer-Verlag New York, Inc. (1996) 42–55
29. van Langevelde, I., Philipsen, A., Treur, J.: Formal specification of compositional architectures. In: ECAI '92: Proceedings of the 10th European conference on Artificial intelligence, Vienna, Austria, John Wiley & Sons, Inc. (1992) 272–276
30. Brazier, F.M.T., Dunin-keplicz, B.M., Jennings, N.R.: DESIRE: Modelling multi-agent systems in a compositional formal framework. International Journal of Cooperative Information Systems **6** (1997) 67–94
31. Dastani, M.: 2APL: a practical agent programming language. Autonomous Agents and Multi-Agent Systems **16**(3) (June 2008) 214–248
32. van Riemsdijk, M.B., Dastani, M., Winikoff, M.: Goals in agent systems: a unifying framework. In Padgham, L., Parkes, D., Müller, J., Parsons, S., eds.: AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems, Estoril, Portugal (12-16 May 2008) 713–720
33. Hindriks, K.: Modules as policy-based intentions: Modular agent programming in GOAL. Number 4908 in Lecture Notes in Artificial Intelligence, Honolulu, HI, USA, Springer-Verlag (15 May 2007) 156–171

34. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer-Verlag, Berlin, Germany (2003)
35. Keijzer, M.: Symbolic regression. [51] 2895–2905
36. Miikkulainen, R., Stanley, K.O.: Evolving neural networks. [51] 2829–2848
37. Lanzi, P.L.: Learning classifier systems: then and now. Evolutionary Intelligence **1**(1) (March 2008) 63–82
38. Roque, R.V.: OpenBlocks: An extendable framework for graphical block programming systems. Master's thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, MA, USA (May 2007)
39. Klopfer, E., Yoon, S.: Developing games and simulations for today and tomorrow's tech savvy youth. TechTrends **49**(3) (2004) 33–41
40. Wang, K., McCaffrey, C., Wendel, D., Klopfer, E.: 3d game design with programming blocks in StarLogo TNG. In: ICLS '06: Proceedings of the 7th international conference on Learning sciences, International Society of the Learning Sciences (2006) 1008–1009
41. Stanley, K.O., Bryant, B.D., Miikkulainen, R.P.: Real-time neuro-evolution in the NERO video game. IEEE Trans. Evolutionary Computation **9**(6) (2005) 653–668
42. Hilgard, E.R., Bower, G.H.: Theories of Learning. Fourth edn. Prentice-Hall, Englewood Cliffs, NJ, USA (1975)
43. Togelius, J.: Optimization, Imitation and Innovation: Computational Intelligence and Games. PhD thesis, University of Essex, Essex, England, UK (September 2007)
44. Bryant, B.D.: Evolving visibly intelligent behavior for embedded game agents. PhD thesis, University of Texas, Austin, TX, USA (2006)
45. Yong, C.H., Stanley, K.O., Miikkulainen, R., Karpov, I.V.: Incorporating advice into neuroevolution of adaptive agents. In Laird, J., Schaeffer, J., eds.: AIIDE'06: Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference, Marina del Rey, CA, USA (20-23 June 2006)
46. diSessa, A.A.: Twenty reasons why your should use Boxer (instead of Logo). In Turcsányi-Szabó, M., ed.: Learning & Exploring with Logo: Proceedings of the Sixth European Logo Conference, Budapest, Hungary (1997) 7–27
47. Csikszentmihalyi, M.: Flow: The Psychology of Optimal Experience. Harper Collins Publishers, New York (1990)
48. Jacobs, P.H.M.: The DSOL simulation suite. PhD thesis, Delft University of Technology, Delft, The Netherlands (15 November 2005)
49. Leelawong, K.: Using the Learning-by-Teaching Paradigm to Design Intelligent Learning Environments. PhD thesis, Vanderbilt University, Nashville, TN, USA (August 2005)
50. Thomaz, A.L., Breazeal, C.: Teachable robots: Understanding human teaching behavior to build more effective robot learners. Artificial Intelligence **172**(6-7) (April 2008) 716–737
51. Keijzer, M., Antoniol, G., Bates Congdon, C., Deb, K., Doerr, B., Hansen, N., Holmes, J.H., Hornby, G.S., Howard, D., Kennedy, J., Kumar, S., Lobo, F.G., Miller, J.F., Moore, J., Neumann, F., Pelikan, M., Pollack, J., Sastry, K., Stanley, K., Stoica, A., Talbi, E.G., Wegener, I., eds.: GECCO'08: Proc. 10^{th} Genetic and Evolutionary Computation Conf., Atlanta, GA, USA, ACM Press (11-16 Jul. 2008)

The MMOG Layer: MMOG based on MAS

G. Aranda, C. Carrascosa, V. Botti

Universidad Politécnica de Valencia
Departamento de Sistemas Informáticos y Computación
Camino de Vera, s/n – 46022 Valencia – Spain
{garanda, carrasco, vbotti}@dsic.upv.es

Abstract. Massively Multiplayer Online Games present a new and exciting domain for service-oriented agent computing as the mechanics of these virtual worlds get more and more complex. Due to the eminently distributed nature of these game systems and their growing necessity of modern AI techniques, it is time to introduce design methods that take advantage of the power of Multi-Agent Systems, Agent Organizations and Electronic Institutions in order to face the challenges of designing a modern Massively Multiplayer Online Game. This article follows previous pieces of work in the line of Multi-Agent Systems and Massively Multiplayer Online Games research into a common ontology to represent the information that agents store and share and towards the use of the THOMAS service architecture and the SPADE agent platform to address these challenges. The main focus of the article is the so called *MMOG Layer*: a software layer which is independent of the environment simulation and the human-interface devices. An example implementation of such layer is also set out.

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[15, 9].

A typical MMOG (such as World of Warcraft™[1] or Age of Conan™[2], two successful games of this kind) gathers a large community of concurrent players in a persistent virtual game world. These players take part in a common game which has no foreseeable end (i.e. the game is never completed or finished) and, along with some synthetic characters and creatures driven by AI, they populate the virtual world and create an online society with its own rules, landscape, economy and idiosyncrasy.

[1] http://www.blizzard.com
[2] http://www.ageofconan.com

MMOGs are similar in nature to established online virtual communities, like Second Life™[3] or There™[4], as both have three-dimensional virtual worlds inhabited by the users' online personas and both have a virtual environment in which the activities of the users take place. However, virtual communities usually spot an open policy and an enforcement of freedom among its members. They do not completely lack rules or norms, but in general terms, users are left to do as they please and there are no short or long term goals for the users' personas to accomplish.

On the other hand, a MMOG also presents some degree of freedom to its players, but it is normally built around a set of rules, logics, conventions and other game mechanics that the game system itself enforces onto the players and that, together, represent the actual way of playing that particular MMOG and set what the game is about and how it becomes an enjoyable and challenging experience for the players. Besides, MMOGs have a tradition of integrating the development, evolution and upgrade of the players' *alter egos* into the game mechanics, making it a *de facto* mandatory long-term game goal for most players.

The present paper focuses in presenting one of the different layers of the architecture *MMOGboMAS* (MMOG based on Multi-Agent Systems) detailing, not only the architecture (ontology and agent taxonomy), but also a prototype applied to a concrete game. The layer presented is the **MMOG layer**, where in an abstraction level that is independent from not only the way the user is going to access the game, but also from the virtual environment, the game rules and mechanics are defined and dealed with.

In this way, the ontology defined is a common basic ontology that the agents that form the game can use to express semantic knowledge in common terms. Also this ontology is used as the base foundation for other, more specific, ontologies that can be built to represent knowledge for a given game or game genre. An example of this later kind of ontology is also provided. The language chosen to represent these ontologies is OWL-DL the standard content language for the semantic web, sanctioned by the W3C[5].

On the other hand, the agents pertaining to the *MMOG layer* will allow the creation of virtual organizations and make use of complex services. So, this layer has been designed and implemented according to the specifications of the THOMAS architecture[6][7]. This architecture is an extension of the classical FIPA platform [12] to open systems dealing with dynamic virtual organizations and services ala SOA.

To implement the prototype, the SPADE[10] MAS platform has been used due to its python language-based feature, so that it is very easy to develop quick and functional agent prototypes.

[3] http://www.secondlife.com
[4] http://www.there.com
[5] http://www.w3.org
[6] http://www.fipa.org/docs/THOMASarchitecture.pdf
[7] http://www.dsic.upv.es/users/ia/sma/tools/Thomas/archivos/arquitectura.pdf

In section 2 the concept of *MMOG based on MAS* architecture is provided. In section 3 the *MMOG Layer* is introduced. In section 4 a specific example deriving from the MMOG ontology is presented. Finally, in section 5 some conclusions are presented along with some hints at future works.

2 MMOG based on MAS Architecture (MMOGboMAS)

This piece of work follows in the footsteps of previous research efforts like [1] and [2] in which games in general and MMOG in particular are researched as natural scenarios for agents and MAS. 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:

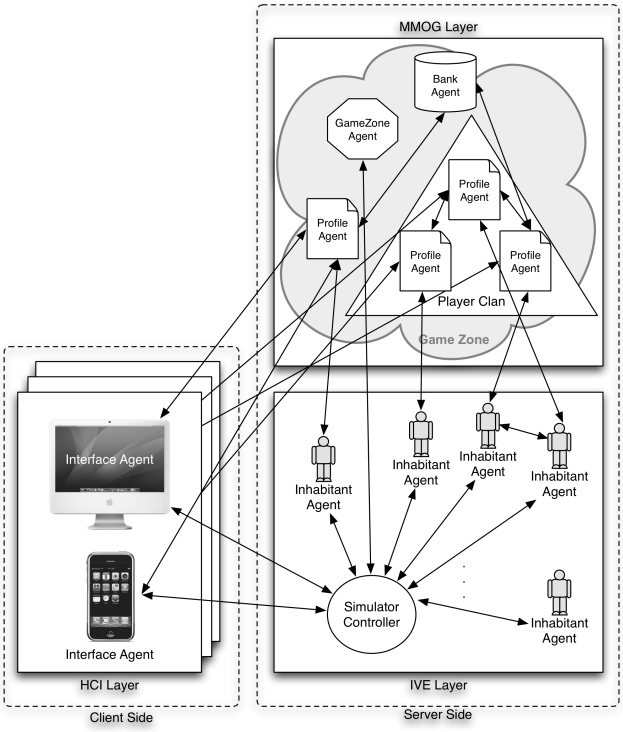


Fig. 1. A MMOG Multi-Agent System

– **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, ...). On a computer, for instance, this software does not differ technically from other types of game software (i.e. offline games), and it is normally built around a game engine, giving it agent capabilities. Due to this layered design, the user is able to interact with the same game in different ways according to the device it is using, that is, the *Interface Agent* it interacts with. Its desirable features are efficiency and good usability. This agent is an *external* agent (out of the direct control of the platform) because it is executed in the **client** machine, that is, the players computer. It is a light-weighted agent that provides the user an interface to the MMOG. The InterfaceAgent is capable of triggering a set of actions that output information messages towards the *MMOG Layer* (in fact, to the *ProfileAgent*). A user may have many InterfaceAgents installed: one on a computer, another on a mobile phone, etc. But since all of them represent the same user, all of them end up communicating with the same *ProfileAgent* (representation of the user in the *MMOG Layer*).

- **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. The IVE layer falls outside the scope of this article, but it is thoroughly described in [4], presenting two kind of agents: *Simulation Controller Agents* and *Inhabitant Agents*. The first ones are in charge of controlling the environment (or pieces of it) and communicating with the graphical viewers along with the agents inhabiting such environment, that is, the *Inhabitant Agents*.
- **MMOG Layer:** It is a complex subsystem where all the game logics and mechanics are implemented and must be solved at run-time. It is independent of the IVE layer. It implements 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 must facilitate game server scalability. In this line of research, this subsystem is seen as the core of a MAS and requires, at least, one agent platform as its foundation.

The main scope of this paper lies within the **MMOG Layer** that will be detailed in the next section.

3 The MMOG Layer

3.1 Agent Taxonomy

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

Game Concept	Agent Concept	Technology
Players Clan	Agent Organization	THOMAS
Players Clan: Lord	Agent Organization Supervisor	
Players Clan: Objective	Agent Organization Goal	
Players Clan: Loot Share	Agent Organization Reward Policy	
Game Zone	Electronic Institution	AMELI
Game Zone Hub	Federation of Electronic Institutions	
Client Software	Interface Agent *(not in its entirety)*	SPADE
Player Character	AvatarAgent	
Non-Player Character	NPCAvatarAgent	
Game World	Intelligent Virtual Environment	OSG

Table 1. Translation between game concepts and MAS concepts

Institutions[11, 6] and Agent Organizations[13, 7, 3] 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, role models, etc. A (rough) translation table between game and agent concepts can be seen in table 1.

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. A ProfileAgent is executed server-side, that is, inside the agent platform (which makes it an *internal* agent). 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. It is a persona of the player inside the virtual world and it is a very active type of agent. Since it is an internal agent of the game and it is designed by the game developers, it will strictly follow the game rules, much like a *Governor* agent does not break the rules of an Electronic Institution. It is related to the IVE's Inhabitant Agent (see figure 1).
- **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. It is also related to the IVE's Inhabitant Agent.
- **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 1).

– **BankAgent**: an agent that stores information persistently for other agents and helps maintain the consistency and reliability of the system.

As summarized in table 1, this approach makes use of Agent Organizations and Electronic Institutions too. Agent Organizations are used to express what is known as player Clans or Guilds, i.e. player associations, which may or may not be permanent, and that bond players which have something in common. This commonness is usually a goal or set of goals within the game world, e.g. in a medieval fantasy game like World of Warcraft™ a clan may be formed to build an army to participate in a big-scale battle against opposing armies. In agent terms, these clans are really Agent Organizations within the agent platform. Each organization has its supervisors (which double as *Clan Lords* in the game world), its members (which are AvatarAgents), its goals and its reward policies. The later two are expressed within game terms using an ontology (more on this later in section 3.2).

Electronic Institutions, however, come handy for another task: modeling the game zones. A game zone is a segment of the virtual game world with its own name, theme and uniqueness. For example, in a medieval fantasy game which has a big game world representing an island, a game zone could be a castle called *'Hidden Palace'* and the dungeon below it. Game zones like that work essentially in a very similar way as Electronic Institutions do: there are clear entry and exit points, interconnected rooms, requirements for entering certain rooms, crowd limits, AI-controlled agents that play different roles, etc. In fact, there is a separate branch of the Electronic Institutions called the '3D Electronic Institutions'[6] that cover precisely this kind of virtual places. The whole game world is envisioned as a set of interconnected (or *federated*) Electronic Institutions. The main institution is called the *'Free Zone'*, an Electronic Institution with relaxed constraints and where the Player Clans may be formed.

3.2 A common Ontology for MMOG agents

One issue left partially unsolved in previous works, and that this paper addresses, is the need for a common ontology for all these agents to be able to share information in a consistent way (i.e. they need to speak the same content language). It is clear that every game or game genre has its own mechanics, rules and players, but, in essence, all MMOGs share some common concepts, like player avatars. These common concepts can be expressed using a common set of conventions, a common knowledge base. Then, in order to express specific concepts for a given game or genre, custom ontologies that derive from the original one can be described. This common ontology is called **MMOG Ontology**. Besides providing a good starting point for expressing knowledge in MMOGs, the use of a common ontology (or set of ontologies) allows for some interesting developments: first, some parts of the MAS (or even the whole system) can be reused or cloned from game to game with little to no change needed; second, it would be possible to interchange or carry over knowledge between different MMOGs, such as player information, statistics, avatars, etc; third, it settles a

common way of working on a MMOG that transcends the current project and helps developer to establish work methodologies; and fourth, it allows for rapid game prototype creation using already available tools for developing ontologies.

The **MMOG Ontology** has been defined using the Web Ontology Language (OWL). More precisely, it is expressed using the OWL-DL (Description Logics) sublanguage for two reasons: one, some of the features of this sublanguage, like defining *one-to-many*-like relational properties, where needed; and two, the full set of the OWL language (called OWL-Full) is not deemed as being computable [5]. Even so, it is a quite light ontology, as it is composed of some base classes that have *datatype*[8] and *object* properties. A summary table of the ontology is presented in table 2. The ontology uses the *'mmog:'* prefix and it presents the following classes: **mmog: Avatar**, **mmog: Clan**, **mmog: GameBeacon**, **mmog: GameItem**, **mmog: GameZone**, **mmog: Goal**, **mmog: Profile**, **mmog: Requirement** and **mmog: RewardPolicy**. Let's review each one of them.

Ontology Class	Property	Type	Inverse Property
mmog:Avatar	atBeacon	*(single mmog:GameBeacon)*	–
mmog:Avatar	hasName	*(single string)*	–
mmog:Avatar	hasOwner	*(single mmog:Profile)*	*owns*
mmog:Avatar	isLordOf	*(multiple mmog:Clan)*	*hasLord*
mmog:Avatar	isMemberOf	*(multiple mmog:Clan)*	*hasMember*
mmog:Clan	hasGoal	*(multiple mmog:Goal)*	–
mmog:Clan	hasLord	*(multiple mmog:Avatar)*	*isLordOf*
mmog:Clan	hasMember	*(multiple mmog:Avatar)*	*isMemberOf*
mmog:Clan	hasName	*(single string)*	–
mmog:Clan	hasRewardPolicy	*(multiple mmog:RewardPolicy)*	–
mmog:Clan	maxMembers	*(single int)*	–
mmog:GameBeacon	hasName	*(single string)*	–
mmog:GameBeacon	placedAtZone	*(single mmog:GameZone)*	*containsBeacon*
mmog:GameItem	hasID	*(single string)*	–
mmog:GameItem	hasName	*(single string)*	–
mmog:GameZone	containsBeacon	*(multiple mmog:GameBeacon)*	*placedAtZone*
mmog:GameZone	hasName	*(single string)*	–
mmog:GameZone	hasRequirement	*(multiple mmog:Requirement)*	–
mmog:Goal	hasCondition	*(multiple owl:Thing)*	–
mmog:Profile	createdWhen	*(single date)*	–
mmog:Profile	hasContact	*(multiple mmog:Profile)*	*hasContact*
mmog:Profile	hasPlayerName	*(single string)*	–
mmog:Profile	owns	*(multiple mmog:Avatar)*	*hasOwner*
mmog:Profile	playsRole	*(single owl:oneOf {'Spectator' 'Player' 'GameMaster'}*	–
mmog:Requirement	atBeacon	*(single mmog:GameBeacon)*	–
mmog:Requirement	hasGoal	*(single mmog:Goal)*	–
mmog:Requirement	playsRole	*(single owl:oneOf {'Spectator' 'Player' 'GameMaster'}*	–
mmog:Requirement	requireClan	*(single boolean)*	–
mmog:Requirement	requireClanMax	*(single int)*	–
mmog:Requirement	requireClanMembers	*(single int)*	–
mmog:RewardPolicy	hasAction	*(multiple owl:Thing)*	–

Table 2. Summary table of the MMOG ontology

[8] A property whose value is a basic data type such integer, float, string, boolean, etc.

- **mmog:Avatar**: This is the generic class to represent an avatar within the game. It does not matter if its a player-controlled avatar (i.e. played by an AvatarAgent) or an AI-controlled avatar (played by a NPCAvatarAgent). Avatars are uniquely identified by their *hasName* property which ties each of them to a unique name string identifier within the game system.
- **mmog:Clan**: This class represents a Player Clan that the system implements by means of an Agent Organization. It has properties referring its members, lords, goals, reward policies and constraints.
- **mmog:GameBeacon**: A *beacon* is a sort of *'named point'* of the virtual world. A marked position inside a mmog:GameZone which is referred by name. For example, the starting point or one of the exits.
- **mmog:GameItem**: An inanimate object within the game world uniquely identified by a code string. For example, in an adventure game, the items that the players virtually carry in their inventories are instances of a subclass of mmog:GameItem.
- **mmog:GameZone**: A class to represent a game zone. As stated earlier, a game zone is a segment of the virtual game world with its own name, theme and uniqueness. It holds one or more requirements to be entered.
- **mmog:Goal**: A collection of conditions expressed within valid semantic terms of the game. When all the conditions are met, then the goal is reached. For example, a condition could be the equivalent expression of *'Place object A in beacon XYZ'* or *'Avatar B is a member of Clan C'*.
- **mmog:Profile**: The profile of a player within the game community: The player's screen name along with all the information the system wants to keep from the player, the avatars the player controls, which role the player plays within the game system and the contact list for this player (i.e. the *'friends list'*).
- **mmog:Requirement**: A requirement to enter a given GameZone. It can specify terms regarding player clans attributes, goals or player role.
- **mmog:RewardPolicy**: A reward policy to apply once a goal has been achieved. It holds a collection of actions defined within game terms. Actions will normally be specified using terms from derived ontologies of specific games.

And so, this ontology serves as a foundation to build other, more specific ontologies to express the knowledge of a particular game or type of game. In the next section, an example of these kind of ontologies can be found.

4 Prototyping the MMOG layer

Analysing the feasibility of the proposal by means of a prototype based on a concrete game, and making use of different agent platforms to allow to manage the different services, virtual organizations and agents involved in this kind of layer.

4.1 Technology Used: THOMAS: Open Systems, Virtual Organizations & Services

As it has been stated before and can be seen in table 1, there are different software technologies used in this implementation of an example of a MMOG layer: to manage Agent Organizations, the THOMAS platform has been used; to implement agents, the SPADE platform has been used for its high prototyping speed; and so on.

The THOMAS abstract architecture is an approach to extend the classic FIPA platform to be used in open systems giving support for dynamic virtual organizations and allowing to express services offered and demanded for the entities in the system *ala* SOA (in fact, every component of the architecture is offering his functionalities as services). Figure 2 shows the THOMAS architecture, which main components are:

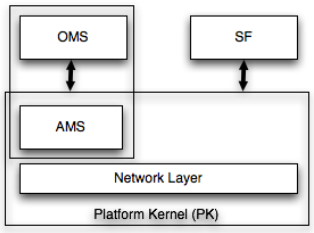


Fig. 2. THOMAS Architecture

- *Platform Kernel*: Component in charge of the communication and internal agent management. It is composed by the *Network Layer* and *AMS* that appears in the classic FIPA architecture.
- *OMS (Organization Management Service)*: This component controls the life cycle of the virtual organizations in the system.
- *SF (Service Facilitator)*: This is an improvement of the FIPA architecture's DF, so that it allows not only to search and publish services, but also to compose, discover and matchmaking, using SOA standards and techniques.

4.2 An Example: RacingGame

Let's imagine an online racing game where players can assume the roles of drivers of different types of vehicles (Cars, Motorbikes and Trucks) that compete in online races (e.g. a simplified vision of the genre of games like Mini-Racing Online^{TM}[9] or Test Drive Unlimited^{TM}[10]). Players can also participate in an in-game virtual marketplace that allows them to buy and sell parts to upgrade

[9] http://www.miniracingonline.com
[10] http://www.testdriveunlimited.com

their vehicles. There are basically two kinds of environments within the game: roads (where the racing takes place) and shops (where the selling and buying takes place). Besides, racers can find two types of objects on the road: gas cans (which replenish their gas tanks) and obstacles (which deal damage to a vehicle if it crashes into them). The developers of the game also wish to incorporate some sort of *'Top Racers Ranking'* or *'Leaderboard'*, and so, they need to collect some data from the races, such as how many races has a given player taken part in and how many of them did that player win.

With that game design in mind, a custom ontology called **'RacingGame'** built on top of the *MMOG ontology* can be defined. Let's start by refining the *mmog:GameItem*. Three different kinds of items have been identified: parts, gas cans and obstacles, and such, three distinct subclasses of *mmog:GameItem* can be created to represent them: *VehiclePart*, *GasCan* and *RoadObstacle*. Let's start with the later two. GasCan needs an obvious datatype property called *hasFuel* that indicates how many fuel units are contained in the can. It also needs a property to indicate where is it placed. Looking at the *MMOG ontology*, the *placedAtZone* property does just that, and so, it is extended to cover the domain of GasCan too. The RoadObstacle class is very similar, but instead of the *hasFuel* property, it has one called *dealsDamage* which indicates how much damage is dealt to a vehicle if it crashes with the obstacle. The VehiclePart class is going to be the superclass of every subtype of part, so it can have the common properties of every part: *hasPrice*, which is a datatype property indicating how much does the part cost, and *canApplyTo*, a multiple property which indicates that the part can be applied (i.e. is *'compatible'*) to a certain subclass of VehicleAvatar (more on that later). The subclasses of VehiclePart are BodyWork, Engine and Wheels, each one representing one of the main parts of a vehicle and each one having some properties identifying its features and advantages and some other aspects, like the color of the BodyWork (see table 3).

Subclass	**Property**	**Type**
BodyWork	hasArmour	*(single owl:oneOf* {1 2 3 4 5})
BodyWork	hasColour	*(single owl:oneOf* {'Red' 'Black' 'White'})
BodyWork	hasWeight	*(single float)*
Engine	gasConsumption	*(single float)*
Engine	hasPower	*(single int)*
Wheels	hasGrip	*(single owl:oneOf* {1 2 3 4 5})

Table 3. Subclasses of VehiclePart

It is clear that the avatars in this game are going to be the vehicles that race, and that all of them have common properties that can be reflected using OWL properties. Let's subclass mmog:Avatar creating the new class **VehicleAvatar** that represents an in-game vehicle. Each vehicle is composed by the union of three parts: a bodywork, an engine and the wheels, and so, the VehicleAvatar class has three object properties called *hasBodywork*, *hasEngine* and *hasWheels* that represent this fact. It also has a numeric property called *maxSpeed* that

represents the maximum speed that a vehicle can reach based on the properties of the parts that conform it. VehicleAvatar also has a series of subclasses starting with **BikeAvatar**, **CarAvatar** and **TruckAvatar**, each one representing one type of vehicle that the player can control. These classes refine the type of vehicle, but do not add any significant property (see figure 3).

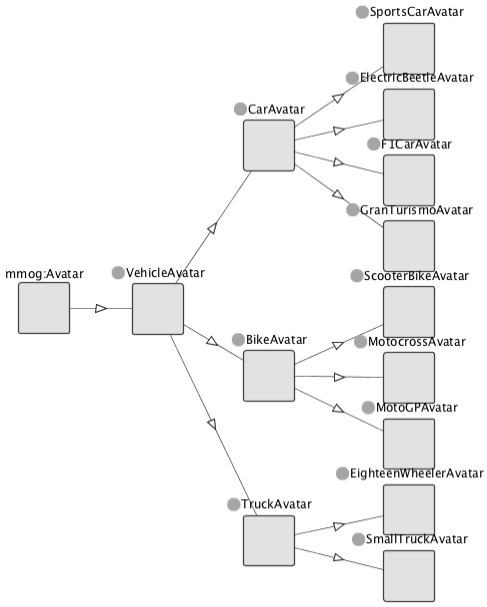


Fig. 3. Diagram of the mmog:Avatar subclasses in the RacingGame ontology

Regarding zones, two kinds of *mmog:GameZones* are defined: the **CarPartsShop** and the **Road**. The CarPartsShop is the place where player can buy and sell car parts to upgrade their vehicles. It is envisioned to be implemented with an Electronic Institution that resembles those used in other market types (like the one used in the Fishmarket^{TM}system[8]). The other zone is the Road, which represents the virtual courses where the races take place. For this zone, two classes of *mmog:GameBeacon* have been defined to mark the start and end points of a race: **RaceStartPoint** and **RaceEndPoint**.

Last, the *mmog:Profile* must be refined in order to represent all the specific data for the races, and such, the **RacerProfile** class is created. It holds an object property called *favoritePlace* which points to the game zone where the player's avatars have spent more time since the moment the player joined the game community (represented by the property *mmog:createdWhen*). It also has a couple of numeric properties to represent the number of races the player has

been involved in and how many of them did he actually win: *numberOfRaces* and *numberOfWins*.

4.3 Agents developed

The SPADE platform has been chosen for it allows prototyping of system agents in a very short amount of time. It can be seen as a *Rapid Application Development* Agent Platform, due in great deal to the use of python as its main programming language and the fact that it is fully FIPA-compliant. This latter compliance with the FIPA standards, allows to easily set up communications between a SPADE agent and THOMAS, as, in its current version, THOMAS is supported by another agent platform supporting FIPA communication.

To illustrate the interplay of these technologies, an example implementation of the Player Clans has been built on top of THOMAS' Agent Organizations. Following the *RacingGame* example, the 'RacingMatches' application facilitates the connection between players interested in becoming part of or creating a brand new race (or clan) and the race management system (Agent Organizations). In this particular example, races are managed inside a THOMAS unit called *RaceManagement* which provides search, membership and registration services to the players. This example shows how a VehicleAvatar agent can set up a new race and how other agents can join the race. To set up a race, a VehicleAgent must provide an implementation of one of these pre-defined services: *"BeginnersRace"*, *"AdvancedRace"*, *"DifficultRace"* or *"VeryHardRace"*. It must play the role of *"Organizer"* (or *Provider*) of the race and the other participants play the role of *"Challengers"* (or *Consumers* of the service).

Register in THOMAS: Any VehicleAvatar agent that wants to use THOMAS services, must first register in the THOMAS service platform. This is accomplished by means of sending an *AcquireRole("Virtual", "Member")* message command to the THOMAS OMS. This message implies that the agent wishes to acquire the role 'Member' in the 'Virtual' organization, which is the default unit where all system agents are. Once the OMS checks that the agent can become part of the unit, it adds the agent to the unit (See figure 4).

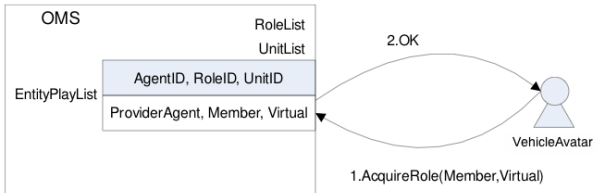


Fig. 4. VehicleAvatar agent joining THOMAS

Register a new service: After registering into a THOMAS platform, an agent can make use of the services offered by the OMS and SF. One of these default services is to look for other services with a similar profile to the one the agent is interested in. In this example, the service is called *BeginnersRace*. So, a VehicleAvatar agent would send the SF a command message like *SearchService(BeginnersRace)*, and the SF then would return the list of registered services with a similar profile and a ranking value for each one of them. This ranking value marks the degree of *affinity* between the given service and the actual result. In the example, the SF may return: *((BeginnersRace,5), (AdvancedRace,4), (DificultRace,3), (VeryHardRace,2))*, which means that there are already similar services (races) in the system and one of them is exactly what the agent is looking for. The VehicleAvatar agent wishes to obtain more semantically detailed information about the *BeginnersRace* service, so it sends a message like *GetProfile ("BeginnersRaceProfile")* to the SF. The SF answers with a URI where the profile is described, and then the VehicleAvatar agent can analyse the profile. This profile, as any semantic document within the THOMAS framework, is written in the OWL-S[14] language, and as such, can be understood by a software agent that can parse and extract information from these kind of documents. The VehicleAvatar can read the service profile and then know that, according to the profile, an agent must adopt a role in order to register a new *process* (a new implementation) of the service. In this example, the *BeginnersRaceProfile* demands the role of *Organizer*, and such, the VehicleAvatar agent can assume that role via a message like: *AcquireRole("Organizer", "BeginnersRace")* .

Implement the new service: Once registered as a provider of a new service, a VehicleAgent must provide a process description (i.e. a valid implementation) of the service it successfully described via the profile. This process description must be accessible through an URI and it is provided by the VehicleAgent to the THOMAS platform with a message like: *RegisterProcess ("BeginnersRaceProfile", "http://.../ BeginnersRaceProcess.owl")* . Normally, the OWL-S documents describing processes are placed in the THOMAS platform, which serves them via a web server (i.e. Apache Tomcat), but SPADE agents can easily serve web documents themselves using platform facilities, and so, a VehicleAgent can host its own process document (see figure 5).

Client connection: Of course a race is no fun if only the organizer of the event races. So, other VehicleAvatar agents may join the race. To do so, the first step is to join THOMAS as can be seen a few lines above. Then, VehicleAgents can use a *SerchService* kind of message to discover the *"BeginnersRace"* service, and a *GetProfile* message to obtain the profile. After that, they can make use of a message like *GetProcess(BeginnersRaceProfile)* to obtain the process information and know that they must acquire the *Challenger* role through an *AcquireRole("Challenger", "BeginnersRace")* message. And finally, they can make actual use of the service, that is, join an active race, by sending the right command message to the *Organizer* VehicleAgent. In this particular example, such

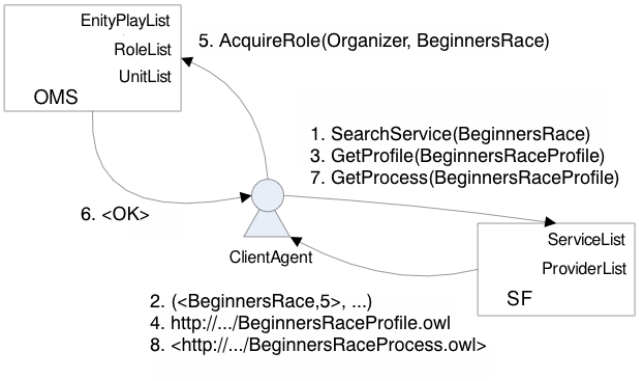


Fig. 5. Register and implement a new service

message includes a service call with the following properties as parameters: *VehicleAvatar:hasName* and *Profile:hasPlayerName*.

5 Conclusions and Future Work

This work has continued the research line of **MMOG Multi-Agent Systems** by providing a starting point for building and exchanging semantic content between agents of these kind of systems. The existence of a basic ontology (a common starting point) to represent game-related knowledge is needed in order to achieve greater long-term goals like standardization, inter-systems communication or define a methodology to build these kind of system from the ground up. In this work, such common ontology has been defined using the standard OWL-DL semantic language, as well as a small example of a more refined ontology for a specific game type. Both of them can be downloaded from the web: *http://gti-ia.dsic.upv.es/ontologies/* and are ready for deployment and use.

An interplay between two previously unrelated agents and services platforms, THOMAS and SPADE, has been created in order to allow agents of the **MMOG Layer** to provide and consume services specified in the OWL-S semantic language, which is derived from the OWL language used in the common ontology.

Moreover, the path has been open to properly define and specify the agent interactions and protocols that conform the whole communication scheme of the system, as now there is an ontology for these agents to use and to express their semantic knowledge and an organizational platform to support and enforce services, norms and agent organizations.

6 Acknowledgements

This work has been partially funded by TIN2006-14630-C03-01, PROMETEO/2008/051 and GVPRE/2008/070 projects and CONSOLIDER-INGENIO 2010 under grant CSD2007-00022.

References

1. G. Aranda, C. Carrascosa, and V. Botti. Intelligent agents in serious games. In *Fifth European Workshop On Multi-Agent Systems (EUMAS 2007)*. Association Tunnisienne dIntelligence Artificielle, 2007.
2. G. Aranda, C. Carrascosa, and V. Botti. Characterizing Massively Multiplayer Online Games as Multi-Agent Systems. *E. Corchado, A. Abraham, and W. Pedrycz (Eds.): Hybrid Artificial Intelligence Systems (HAIS 2008), LNAI 5271*, pages 507–514, 2008.
3. E. Argente, J. Palanca, G. Aranda, V. Julian, V. Botti, A. Garcia-Fornes, and A. Espinosa. Supporting agent organizations. In *CEEMAS'07*, 2007.
4. A. Barella, C. Carrascosa, and V. Botti. Agent architectures for intelligent virtual environments. In *2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 532–535. IEEE, 2007.
5. S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, L. Stein, et al. OWL Web Ontology Language Reference. *W3C Recommendation*, 10:2006–01, 2004.
6. A. Bogdanovych, H. Berger, S. Simoff, and C. Sierra. Narrowing the Gap between Humans and Agents in E-commerce: 3D Electronic Institutions. *K. Bauknecht, BP oll, and H. Werthner, editors, E-Commerce and Web Technologies, Proceedings of the 6th International Conference, EC-Web*, pages 128–137, 2005.
7. N. Criado, E. Argente, V. Julian, and V. Botti. Organizational services for spade agent platform. In *IWPAAMS07*, volume 1, pages 31–40. Universidad de Salamanca, 2007.
8. G. Cuni, M. Esteva, P. Garcia, E. Puertas, C. Sierra, and T. Solchaga. MASFIT: Multi-Agent System for FIsh Trading. *ECAI*, 16:710, 2004.
9. N. Ducheneaut, N. Yee, E. Nickell, and R. Moore. " Alone together?": exploring the social dynamics of massively multiplayer online games. *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 407–416, 2006.
10. M. Escrivà, J. Palanca, G. Aranda, A. García-Fornes, V. Julian, and V. Botti. A jabber-based multi-agent system platform. In *Proc. of AAMAS06*, pages 1282–1284, 2006.
11. M. Esteva, B. Rosell, J. Rodriguez-Aguilar, and J. Arcos. AMELI: An Agent-Based Middleware for Electronic Institutions. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 236–243, 2004.
12. FIPA. Abstract architecture specification. Technical Report SC00001L, 2002.
13. E. Garcia, E. Argente, and A. Giret. Issues for organizational multiagent systems development. In *Sixth International Workshop From Agent Theory to Agent Implementation (AT2AI-6)*, 2008.
14. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, et al. OWL-S: Semantic Markup for Web Services. *W3C Member Submission*, 22, 2004.
15. M. Matskin. Scalable Agent-Based Simulation of Players in Massively Multiplayer Online Games. *Eighth Scandinavian Conference on Artificial Intelligence: SCAI'03*, 2003.

A Teamwork Infrastructure for Computer Games with Real-Time Requirements

Ivan Medeiros Monteiro and Luis Otavio Alvares

Universidade Federal do Rio Grande do Sul, Instituto de Informática
Av. Bento Goncalves, 9500, Porto Alegre - RS, Brazil
Phone: +55 51 3316 6843, Fax: +55 51 3316 7308
{immonteiro, alvares}@inf.ufrgs.br

Abstract. Although there are many works on teamwork, the development of agent teams for complex environments still imposes many challenges, especially if these environments have real-time requirements. Many tools have been developed, but there is no silver bullet, and the most general tools have serious problems with the real-time requirements. This paper introduces a new proxy-based tool, based on *Joint Intentions*, to help agents to be a teammate in partially observable, dynamic and stochastic environments with real-time requirements. Validation experiments with the computer game Unreal Tournament 2004 have demonstrated impressive results.

1 Introduction

In computer games most effort has been done to improve graphic aspects, but very little has been done considering the behavioral aspects, in order to reduce the distance to the human behavior [3]. The situation of team-based games is even worse, because bots[1] usually do not assume coherent roles inside the team. Thus, although there is progress on individual behavior, not much has been done for social behavior in computer games.

Teamwork[2] has emerged as the paradigm for coordinating cooperative agents in dynamic environments and it has been shown to be capable of leading to flexible and robust behavior [29]. However, there is no general solution to build agent teams and each new domain may have new requirements. It still remains a challenge for teamwork applications in which the domain is highly dynamic and requires a short response time.

Teamwork has been applied to many domains, such as rescue disaster [19], military combat [7] [9], robocup soccer [10] [15], and collaboration between humans and agents [20]. Flexible teamwork, promoted by the explicit team model, that defines commitments and responsibilities for teammate, allows a robust coordination, keeping coherence even with individuals faults and unpredictable changes in the environment [26].

Some theories have been proposed to formalize the behavior of a group of agents to act as a team. Two of these theories have important results for real problems. They are *Joint*

[1] A character controlled by an artificial agent.
[2] A cooperative effort realized by teammates to achieve a goal.

Intentions [14] and *Shared Plans* [5]. Both are described through a logic formalism, but with different approaches. *Joint Intentions* focus on a team's joint mental state, while *Shared Plans* focus on the agent's intentions towards its collaborator's action or towards a group's joint action.

Tools as *STEAM* [30] and *Machinetta* [26] had good results in many complex environments [7] [31] [15] [23] [20] [2] [25] [33]. The natural way would be to use some of these tools to improve teamwork on bot's development in computer games. However, practical issues have pointed out for the development of a new specialized tool. *STEAM* was developed using an old version of *SOAR* [13], that made changes in its language since then. *Machinetta*, that is a successor of *STEAM*, has shown many limitations for the game domain.

In order to overcome these limitations, this work introduces *TWProxy*, a new tool that enables agents to perform teamwork in highly dynamic environments. *TWProxy* uses many of the good ideas of *Machinetta*, avoiding some of its limitations and adding new features.

The remainder of the paper is organized as follows: Section 2 shows the main related works, Section 3 describes the features of *TWProxy*, Section 4 presents experiments and evaluation, and Section 5 concludes the paper.

2 Related Works and Main Contributions

Theoretical works in agent teamwork [14] [5] [6] define some features about team behavior. They describe what the agents need to share in order to perform a teamwork, like goals to achieve, plans to perform together, and knowledge about the environment. The agents need to share their intentions to perform a plan in order to achieve a common goal, the teammates need to be aware about their capabilities and how to fill roles to perform the high level team plan. They must be able to monitor both, their own progress and the group's progress to achieve the goals of the team. Some systems have been developed using the teamwork idea as teams with human collaboration [23], teams for disaster situation [19], and teams for industry production line [8].

Joint Intentions [14] focus on teammates' joint mental states. A team jointly intends a team action if team members are jointly committed to completing such team action, while mutually believing that they were doing it [30]. *Joint Intentions* assume that the agents are modeled in a dynamic multi-agent environment, without complete or correct beliefs about the world or other agents. They have changeable goals and their actions may fail. Thus, the teammate jointly commits to make public the knowledge when a goal is achieved, impossible to be reached, or irrelevant.

In contrast with *Joint Intentions*, the concept of *SharedPlans* [5] relies on a novel intentional attitude, *intending-that*. An individual agent's *intending-that* is directed toward its collaborator's action or toward a group's joint action. It is defined through a set of axioms that guide an individual to take actions, including communication actions, that either enable or facilitate its teammates to perform the assigned tasks [30].

Based on these theories, important teamwork infrastructures have been developed, like *GRATE** [8], *COLLAGEN* [21], *STEAM* [30], TEAMCORE [20] *RETSINA* [28], *MONAD* [32], and *Machinetta* [26]. *STEAM* was developed to provide flexibility and re-usability of agent coordination through the use of a general model of teamwork. Such model exploits the autonomous reasoning about coordination and communication. Previous works have failed either in fulfilling responsibilities or in discovering unexpected opportunities, once they have used precomputed coordination plans, which are inflexible. Indeed, they have been developed for a specific domain, what hampers the reuse. *STEAM* is specially based on *Joint Intentions* in order to make the basic building blocks of teamwork, and it uses *Shared Plans* to avoid an agent to undo the actions of another agent.

MONAD defines a multi-agent control architecture based on *STEAM* teamwork model to automatically coordinate the execution of multi-agent tasks. *MONAD* provides the designer with tools to specify different protocols and the situations in which they should be applied. It shows how different teams can be easily configured, leading the teams to differ in performance. In spite of *MONAD* be applied to Unreal Tournament[3], its experiments just evaluate how the team configuration can affect the team performance, but this does not evaluate how efficient a team with this tool can be.

Machinetta is a proxy-based integration infrastructure where there is a symbiotic relationship between proxies and team members. The proxies provide teamwork models reusable for heterogeneous entities. *Machinetta* provides each agent with a proxy and the proxies act together in a team. The *Machinetta* project was build over previous works like *STEAM* and *TEAMCORE proxies* [20], adding features like adjustable autonomy reasoning, a novel role allocation algorithm, and the representation of coordination tasks as roles. The pair agent-proxy execute Team-Oriented Programs(TOP) [24], that are abstract team plans that provide high-level description of the activities to be performed. Such programs specify joint plans of the team, but do not contain all of the required details of coordination and communication. Team plans are reactively instantiated and they allocate roles to capable agents.

Although there are many tools with similar proposals, none of them focus on the response time for answering the requirements of real-time. In addition, many of these tools show some practical limitation. For instance, *STEAM* was developed using an old version of *SOAR* [13]. *Machinetta* has shown a high response time for highly dynamic domains with real-time requirements. It also does not reuse the terminated plans, being necessary to create extra plans and increasing the memory usage. MONAD is no longer available.

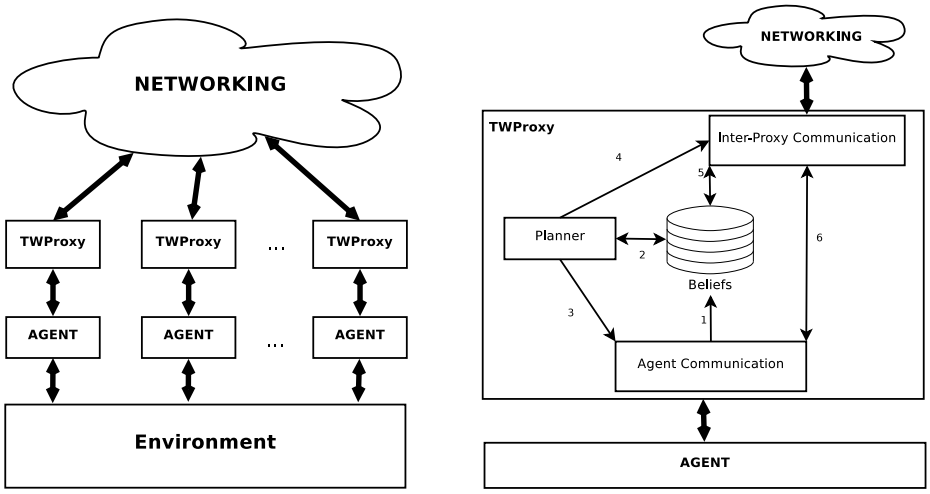
In this paper we present *TWProxy* as a solution to the teamwork problem with real-time requirements. It is based on the *Joint Intentions* formalism and is inspired on *Machinetta*. We add new important features, including: **(i)** a simple and extensible language to describe plans and beliefs, **(ii)** maintenance of consistency through atomic communication, **(iii)** reuse of terminated plans, and **(iv)** plans that are invariant to the number of agents.

[3] A computer game that provides highly dynamic environment

3 TWProxy

In order to meet the requirements for the development of teams of agents in highly dynamic environments with real-time requirements, we introduce *TWProxy*, a new lightweight and efficient infrastructure to provide coordination to teams of heterogeneous agents.

In our approach, summarized in Figure 1(a), each agent is associated with a proxy that performs the communication and the multi-agent planning. The team coordination is achieved using role allocation. Each proxy knows the capabilities of the other teammates. When plan preconditions are achieved, roles are assigned according to this plan. When the plan postconditions are achieved, each teammate releases its role. The proxy does not tell how to execute the role, it just deliberates about what to do. Because of the high-level coordination, a flexible teamwork is possible, leaving the details about role execution with the agent.



(a) Teamwork model based on proxy

(b) TWProxy organization

Fig. 1: TWProxy model.

Figure 1(b) shows the *TWProxy* internal organization. It is composed by the planner, the set of beliefs, the interface for communication with the agent, the module for inter-proxy communication and interaction between these elements. The modules of communication are easily extensible to meet new requirements of organization or integration of *TWProxy* with new types of agents. The planner and the set of beliefs are driven by the description language of plans and beliefs, as shown in listing 1. In the following we describe the internal interactions of *TWProxy*, represented by arrows in Figure 1(b):

1. A new belief or an update of beliefs, that was perceived by the agent and that will update the set of beliefs.

2. Usage of the set of beliefs by the planner, that uses stored beliefs to check the condition to either activate or deactivate plans.
3. Deliberation on allocation of tasks to the involved agent.
4. Resolution on allocation of tasks to other teammates.
5. Exchange of beliefs between proxies.
6. Channel used to share information between agents through *TWProxy*. This channel is not used in the coordination model, it exists to meet the specific requirements for communication that are not represented as beliefs of team.

3.1 Plans and Beliefs.

The initial agent beliefs and its plans are stored in a structured file specified with the language shown in listing 1. This file contains agent beliefs about capabilities, team composition and the specific domain. It also describes high-level team plans.

Listing 1: Grammar of the language to define the plans and initial beliefs

```
         <blf> ::= <block> | <block> <blf>
       <block> ::= "belief" <belief_block> |
                   "plan" <plan_block>
 <belief_block> ::= <id> "{" <inside_belief> "}"
<inside_belief> ::= <attr_value> |
                   <attr_value> <inside_belief>
   <attr_value> ::= <id> ":" <value> ";"
        <value> ::= <id> | <id> "," <values> |
                   <number> | <number> "," <value> |
                   "true" | "false"
   <plan_block> ::= <id> "{" <inside_plan> "}"
  <inside_plan> ::= "roles" ":" <role_values> ";"
                   <pre_block> <post_block>
  <role_values> ::= <id> | <id> "," <role_values> |
                   <id> "+" | <id> "+" <role_values>
    <pre_block> ::= "precondition" "{" <expr> "}"
   <post_block> ::= "postcondition" "{" <expr> "}"
         <expr> ::= "(" <expr> ")" |
                    <id> "." <id> <comp> <number> |
                    <id> "." <id> <comp> <id> "." <id> |
                    <id> "." <id> <comp> "true" |
                    <id> "." <id> <comp> "false" |
                    <expr> "|" <expr> |
                    <expr> "&" <expr>
          <id> ::= "[a-zA-Z][a-zA-Z_0-9]*"
      <number> ::= "[+-]?[0-9]+|[+-]?[0-9]+\.[0-9]+"
        <comp> ::= "==" | "!=" | "<" | "<=" | ">" | ">="
```

Teams of agent-proxy pairs execute Team-Oriented Programs (TOPs) [24]. These TOPs specify the joint plans of the team and their inter-dependencies, but they do not contain all the required details of coordination and communication.

Listing 2: An example about capability belief

```
belief capability_agent001_CTFProtectTheBase {
        type: capability;
        rapId: agent001;
        roleId: CTFProtectTheBase;
        ability: 76;
}
```

Listing 2,3 and 4 show examples of different parts of the beliefs file for the *Capture The Flag* domain. Listing 2 defines that the agent *agent001* can perform the role *CTFProtectTheBase* and that his level of ability for this role is 76.

Listing 3: An example about domain specific belief

```
belief flag_enemy{
        type: flag;
        owner: enemy;
        have: false;
}
belief flag_friend{
        type: flag;
        owner: friend;
        have: true;
}
```

The domain specific belief is also described in the beliefs file, as shown in listing 3. The belief contents is flexible enough to describe new properties, because the new attributes are handled dynamically.

Listing 4: A plan example

```
plan p1{
        roles: CTFProtectTheBase, CTFCaptureTheFlag+;
        precondition{
                ( flag_friend.have == true ) &
                ( flag_enemy.have == false )
        }
        postcondition {
                ( flag_friend.have == false ) |
                ( flag_enemy.have == true )
        }
}
```

Listing 4 shows an example of a plan for the *Capture The Flag* context. When a team has its flag and it does not have the enemy flag, plan *p1* is activated launching a role allocation process. This plan specifies two roles to be allocated, *CTFProtectTheBase* and *CTFCaptureTheFlag*, where the second is flexible for more than one allocation, which is indicated by the symbol ”+” at the end of the role’s name. If a team has four agents available, one performs *CTFProtectTheBase*, while the other three execute *CTFCaptureTheFlag*.

An agent does not need to share every belief with its team, because just the beliefs that appear in the plan definitions are relevant for the team. Therefore, the proxy only shares the beliefs that appear in the plan definitions.

3.2 Communication

The module Inter-Proxy Communication uses two kinds of communication, the atomic broadcast [4] and unicast. Each kind of communication is separated in different channels. The atomic broadcast is used to share team beliefs, keeping the distributed blackboard consistent, and unicast is used in the role allocation process.

In order to solve the problem of atomic communication, *TWProxy* has a simple atomic broadcast layer that can be extended to meet requirements of other kinds of organization. This kind of communication is used when a new team belief is available. In other words, when a teammate perceives a new situation that affects the team, he shares it, keeping the knowledge of the team consistent. However, the teammate does not need to share his individual knowledge, decreasing the number of messages exchanged.

The unicast channel is used to perform a simplified form of Contract Net Protocol[27], in order to achieve role allocation. When the team leader perceives that the precondition of a plan was achieved, he asks the other agents about their ability, and after receiving the answers he allocates the roles for the necessary number of agents.

3.3 Planner

The *TWProxy* planner uses team leader to launch a new plan. The team leader may be either statically pre-fixed or dynamically defined at run-time. A simple way to dynamically choose a team leader is giving the leadership position to the first member that requests it, what is easy to do using atomic communication. The use of team leader avoids conflict problems in role allocation, saving time to react to the environment changes. The team leader has updated team beliefs and it can launch a plan consistently, because everyone is committed to share information relevant to the team .

In order to achieve the real-time requirements, the process of role allocation is not thoroughly distributed. After the team leader receives information about the ability of the teammates, it decides by its own about the allocation of the roles. This is the main limitation of *TWProxy*, but it is also the main issue that increases the performance of role allocation. Only a few messages are needed to update the team leader with the team ability, thus the process is optimized to achieve the best performance.

The planner is like a rule-based system. Every plan has rules to both activate it (the preconditions) and to stop it (the postconditions). These rules are checked using the current knowledge. Roles are assigned in the activation of a plan and the involved agents release their roles in the plan stopping.

With *TWProxy* the team leader can use two algorithms to perform the role allocation: an optimal algorithm with complexity $O(n^3)$ or a new heuristic algorithm with complexity $O(n^2 log(n^2))$. The optimal algorithm is the Hungarian or Kuhn-Munkres algorithm, a classical solution to the assignment problem, originally proposed by H. W. Kuhn in 1955 [12] and refined by J. Munkres in 1957 [18]. The heuristic algorithm is the *Normalized Relative Weight (NRW)*, proposed in this work and presented in Algorithm 1.

In the role allocation problem, each agent $a_i \in A$, is defined by its capability to perform roles, being $R = r_1, ..., r_k$ the set of roles. The capability of an agent a_i to perform a role r_j, is quantitatively given by $C(a_i, r_j)$. The matrix M defines the allocation of roles to team members, where $m_{i,j}$ is the value for the ith row and jth column, and $m_{i,j} = 1$ if a_i performs r_j, otherwise $m_{i,j} = 0$.

Thus, the goal in this problem is to maximize the function

$$f(M) = \sum_{a_i \in A} \sum_{r_j \in R} C(a_i, r_j) * m_{i,j}$$

such that

$$\forall i \sum_{1 \leq j \leq k} m_{i,j} = 1$$

The *NRW* algorithm is a modification of the greedy strategy. It considers that each agent has a vector with all the capabilities, and the algorithm uses just a sub-vector of these capabilities related to the roles that should be performed. After that, *NRW* normalizes each subvector from the agent. Thus, each capability may be compared fairly. Then, the algorithm can use a greedy strategy to find a solution.

Algorithm 1 Normalized Relative Weight

Require: C is a subset of all capabilities, and it has just roles that will be assigned.
  for $i = 0$ to $nAgents$ **do**
    $norma \Leftarrow 0$
    for $j = 0$ to $nRoles$ **do**
      $norma \Leftarrow norma + C(i * nRoles + j).cap^2$
    end for
    $norma = sqrt(norma)$
    for $j = 1$ to $nRoles$ **do**
      $C(i * nRoles + j).cap \Leftarrow C(i * nRoles + j).cap/norma$
      $C(i * nRoles + j).agent \Leftarrow i$
      $C(i * nRoles + j).role \Leftarrow j$
    end for
  end for
  $C \Leftarrow sortByCapability(C)$
  for $k = 0$ to $nRoles * nAgents$ **do**
    $aIdx \Leftarrow C(k).agent$
    $rIdx \Leftarrow C(k).role$
    if $Allocated(rIdx) =$ **false** and $HasRole(aIdx) =$ **false then**
      $Allocated(rIdx) \Leftarrow$ **true**
      $HasRole(aIdx) \Leftarrow$ **true**
      $Allocation(aIdx) \Leftarrow rIdx$
    end if
  end for

3.4 Agent Interface

Each individual agent is attached to a *TWProxy* forming a social agent. The communication between the individual agent and the *TWProxy* is defined by a flexible interface, allowing interaction with several kinds of agents, and extention for new domains.

The interface between proxy and agent is defined by a base class that must be extended to implement this communication. Such interface is based on message exchange and it can be adjusted to each kind of agent. In a new domain, the interface with the agent is the main modification needed. Thus, it is possible to build a team, to a new domain, by just developing the interface between agent and proxy, and the team program. The individual agent does not need to have social commitment, because *TWProxy* does this for it.

4 Experiments and Evaluation

In this section we present three kinds of experiments to show the main contributions of *TWProxy* . The first one evaluates the role allocation process, comparing the Hungarian algorithm with the heuristic proposed here. The second measures the period of teamwork inconsistency, evaluating the elapsed time between an important change in the environment and the team adoption of a new strategy. The third kind of experiment evaluates the efficiency of bots using *TWProxy* in matches.

4.1 Role Allocation Efficiency

The first experiment about the efficiency of the role allocation intends to explain why to use an heuristic algorithm if there exists an optimal one that runs in polynomial time. The answer is simple, the heuristic algorithm scales better than the optimal one. Considering one second as the maximum time to wait for role allocation, the optimal algorithm scaled up around 230 agents while the heuristic scaled up around 1200 agents. For this reason, *TWProxy* uses by default the Hungarian algorithm when the number of agents is lower than a given threshold, otherwise the proxy uses the heuristic method.

An important issue when dealing with heuristic algorithms is to know how good the heuristic is. A statistical analysis sounds a good solution. Thus, many role allocations were simulated to evaluate the difference between the optimal and the heuristic algorithm. The result of 999 allocations with the number of agents ranging from 3 to 100 is shown in Figure 2 representing the mean of percentage of optimum allocation and its standard deviation. By increasing the number of agents, the algorithm tends to 98% of the optimum while the standard deviation decreases.

4.2 Evaluating the Period of Teamwork Inconsistency

In highly dynamic environments, a great concern in teamwork is to avoid teamwork inconsistency. In the *Capture The Flag* domain [4], a teamwork inconsistency may be the following: given a team composed of agents *A*, *B* and *C*, if either *A* or *B* are trying to recover a flag that has been recovered by *C*, then the team is in an inconsistent state. Every team in a dynamic environment will stay in an inconsistent state during some period of time. Therefore, the goal is to minimize inconsistency states.

[4] In the game type *Capture The Flag*, one team must score flag captures by taking the enemy flag from the enemy base and returning it to their own flag. If the flag carrier is killed, the flag drops to the ground for anyone to pick up. If the friend flag is taken, it must be returned (by touching it after it is dropped) before the team can score a flag capture.

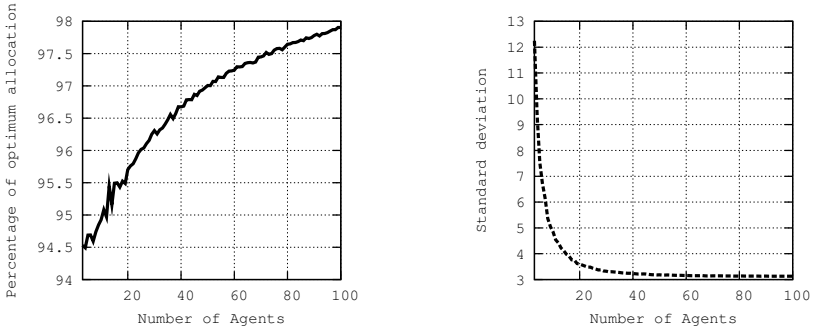


Fig. 2: Means of percentage of the maximum value and standard deviation.

The experiment about Period of Teamwork Inconsistency(PTI) intended to evaluate the ability of *TWProxy* to react in (soft) real-time to environment changes. In order to compare its performance, *Machinetta* was exposed to the same situation. A group of five agents were updating their beliefs about the flag status, and the time period between the environment change and the role allocation to the whole team has been computed.

The sequence of environment changes was the same for *Machinetta* and *TWProxy*. It was composed of 25 updates, representing a whole match. Each proxy played 100 matches, in order to get a general behavior of PTI. This experiment was performed on a single machine. For this reason, the network latency did not appear in the results.

The results shown in Table 1 describe the mean(μ) and the standard deviation(σ) of the experiments set. The measure unit is milliseconds and the time represents the total time between a new received belief and the modification of the strategy. *TWProxy*, in this experiment, achieved a mean of $77.91ms$, while *Machinetta* got $12303.34ms$. In other words, *TWProxy* was more than one hundred times faster than *Machinetta* to reach a consistent state. The standard deviation shows how stable *TWProxy* execution was in comparison to *Machinetta*.

Table 1: Mean and standard deviation of the PTI experiment.

	Elapsed time(ms)	
TWProxy	$\mu = 77.91$	$\sigma = 12.95$
Machinetta	$\mu = 12303.34$	$\sigma = 6459.54$

The performance and the stability of *TWProxy* over *Machinetta* was the main result of this experiment. Besides, *Machinetta* also presented a high usage of memory, more than 100MB per proxy, making its usage difficult in some domains like modern games. *TWProxy* in these experiments did not use more than 1.6MB of memory. The key of these results relies on the kind of application for which *Machinetta* and *TWProxy* were designed. While *TWProxy* design was concerned to achieve real-time requirements, *Machinetta* was originally developed to allow humans to participate in teams with robots and artificial agents. Several points may explain this efficiency, including: the usage of C++ instead of Java; the maintenance of the knowledge of team for each proxy, speed-

ing up the access to beliefs; optimization in local access to the beliefs; and mainly no conflicts to solve, because the communication with atomic broadcast guarantees the total order of messages and the usage of team leader avoids the role allocation conflict.

4.3 Teamwork Efficiency

To evaluate the teamwork efficiency of *TWProxy* we used the well known game *Unreal Tournament 2004 (UT2004)*, a multiplayer *First-Person Shooter*(FPS) game. The classical team-based game type *Capture The Flag* was chosen to evaluate the team coherence, because it is easy to measure the results and simple to identify the team behavior. The *GameBOTs* [1] [11] project is used to communicate the developed bot with the game.

The UT2004 game provides an environment with the following characteristics [22]:

- **partially observable** - the agent, using its perception, cannot access all relevant environment states;
- **stochastic** - the next state of the environment is not completely determined by the current state and actions selected by the agents;
- **nonepisodic** - the quality of bot's actions depends on its previous actions;
- **dynamic** - the environment state can change between the agent's perception and agent's actions;
- **continuous** - the number of environment states is unlimited;
- **multi-agent** - there is cooperative and competitive interaction between agents.

In our experiments, the agents have been developed using the framework IAF [17], that implements a hybrid agent architecture [16]. Such agents communicate with the game controlling a player. For these experiments every agent has the same capabilities. Thus, just the teamwork is different. In order to compare *TWProxy* with another teamwork solution, *Machinetta* is used, because of the large number of successful applications in dynamic environments [23] [20] [24] [25].

The team that plays *Capture The Flag* using *TWProxy* has just four plans, that are activated when the state of a flag changes. Such plans generate the following team behavior:

1. when a team has its own flag but it has not the enemy's flag, someone needs to protect the base while others go to capture the enemy's flag;
2. when a team has both, its own and the enemy's flag, all agents must go back to their base;
3. when a team does not have its own flag but it has the enemy's flag, the agent that has the enemy's flag must go back to its base and the others should recover the team's flag;
4. when the team has no flags, every agent must search any flag.

The matches experiments were composed of 30 matches with 15 minutes and teams were composed of 5 players (except in experiments against humans, when 4 players were used). The following teams participated in the matching experiments: TWproxy-Team, a team composed of agents developed with *IAF* and using *TWProxy*; UTBots-Average, the *Capture The Flag* team from UT2004 in *Average* difficulty level (default level); UTBots-Experienced, the *Capture The Flag* team from UT2004, in *Experienced* difficulty level; Machinetta-Team, a team composed of agents developed with *IAF* and using *Machinetta*; and Human-Team, a team composed of humans with different levels of skill.

The Tables 2, 3, 4, 5 and 6 summarize the results of matches, showing mean (μ) and standard deviation (σ) of the team score and the sum of individual scores. The team score represents how many flags were successfully captured and the sum of individual scores indicates the teammate behavior, computed by the number of killed enemies and captured flags.

TWProxy-Team vs UTBots-Average *TWProxy-Team* had no difficulties to win *UTBots-Average*, winning 93.33% of the battles, as seen in Table 2. Being *UTBots-Average* a centralized approach that accesses the full state of the game to decide what to do, the result of this experiment represents the possibility to use distributed artificial intelligence in domains like modern games. The immediate advantage of the distributed approach is the possibility of new gameplay, despite increasing the complexity to develop bots.

Table 2: *TWProxy-Team* versus *UTBots-Average*.

	TWProxy-Team		**UTBots-Average**	
Wins	93.33%		0%	
Team Score	$\mu = 2.42$	$\sigma = 1.45$	$\mu = 0.03$	$\sigma = 0.18$
Sum of Individual Score	$\mu = 230.97$	$\sigma = 31.11$	$\mu = 100.77$	$\sigma = 20.27$

TWProxy-Team vs UTBots-Experienced In this experiment, *TWProxy-Team* also had no difficulties to win *UTBots-Experienced*, as shown in Table 3. Although the score of *UTBots-Experienced* was better than *UTBots-Average*, in comparison with the previous experiment, *TWProxy-Team* won 76.67% of the battles.

Table 3: *TWProxy-Team* versus *UTBots-Experienced*.

	TWProxy-Team		**UTBots-Experienced**	
Wins	76.67%		0%	
Team Score	$\mu = 1.37$	$\sigma = 1.24$	$\mu = 0$	$\sigma = 0$
Sum of Individual Score	$\mu = 227.03$	$\sigma = 27.32$	$\mu = 117.87$	$\sigma = 27.72$

Machinetta-Team vs UTBots-Average Using the same agents of *TWProxy-Team*, but with a different proxy (*Machinetta* instead of *TWProxy*), the performance of *Machinetta-Team* was worse than the performance of *TWProxy-Team* in the same situation. Table 4 shows the results of these experiments, where *Machinetta-Team* won only 20% of the

battles. Thus, it is possible to see how a tool of coordination can influence the team performance.

Table 4: *Machinetta-Team* versus *UTBots-Average*.

	Machinetta-Team		**UTBots-Average**	
Wins	20%		0%	
Team Score	$\mu = 0.27$	$\sigma = 0.64$	$\mu = 0.03$	$\sigma = 0.18$
Sum of Individual Score	$\mu = 185.47$	$\sigma = 48.28$	$\mu = 73.87$	$\sigma = 31.84$

TWProxy-Team vs Machinetta-Team In these experiments, as shown in Table 5, *Machinetta-Team* did not score on *TWProxy-Team*, while *TWProxy-Team* scored on average 0.6 points per match. In general, *TWProxy-Team* won 43.33% of the battles, while *Machinetta-Team* did not win any one. Although the agents were the same in both teams, *TWProxy* shows advantages over *Machinetta* for this domain, that is a highly dynamic environment with real-time requirements.

Table 5: *TWProxy-Team* versus *Machinetta-Team*.

	TWProxy-Team		**Machinetta-Team**	
Wins	43.33%		0%	
Team Score	$\mu = 0.6$	$\sigma = 0.81$	$\mu = 0$	$\sigma = 0$
Sum of Individual Score	$\mu = 217.27$	$\sigma = 57.15$	$\mu = 176.07$	$\sigma = 29.61$

TWProxy-Team vs Human-Team This last experiment aims to evaluate the team coherence, putting *TWProxy-Team* against *Human-Team*. Usually, teams of humans have the ability to explore some coherence fault in a team of bots, what justifies this experiment. Thus, a group of 25 people, with different skills in FPS[5], participated in the team of humans. Table 6 shows how disputed these battles were. *TWProxy-Team* won 20% of the matches and *Human-Team* won 26.67%. The individual score indicates that the human players were better than bots, however, no incoherence was found in the team of bots. The main advantage for *Human-Team* was the evolving of strategy, because the strategy of *TWProxy-Team* did not evolve. Nevertheless, the results were so close, with a little advantage for *Human-Team*.

Table 6: *TWProxy-Team* versus *Human-Team*.

	TWProxy-Team		**Human-Team**	
Wins	20%		26.67%	
Team Score	$\mu = 0.23$	$\sigma = 0.43$	$\mu = 0.5$	$\sigma = 0.97$
Sum of Individual Score	$\mu = 108.73$	$\sigma = 25.02$	$\mu = 154.2$	$\sigma = 39.23$

[5] First Person Shooter

5 Conclusions

This work introduces *TWProxy*, a new tool based on a team-proxy approach to coordinate groups of agents. Experiments presented here have shown that it is possible to use the teamwork paradigm for highly dynamic environments with real-time requirements. These also confirms the flexibility of high-level planning and the reusability of the proxy approach.

TWProxy shows new important features to develop teams in complex environments. It provides two partially distributed algorithms to perform role allocation, including a new efficient heuristic algorithm. It also provides a simple and extensible language to describe plans and beliefs. The plans can be reusable and the number of agents performing a plan can be variable. A distributed blackboard with team beliefs is available to the agents, and the use of *TWProxy* for new domains just requires to write the agent's interface and the TOP.

Based on experiments, *TWProxy* has shown advantages compared to *Machinetta*, being more stable and efficient. These experiments evaluated the following aspects: the quality of the role allocation performed by *TWProxy*; the period of teamwork inconsistency, where *TWProxy* has much better response time than *Machinetta* in a simulated environment; and the efficiency of teamwork, where it was possible to see *TWProxy* keeping the team coherence in matches against *UTBots*, *Machinetta-Team* and *Human-Team*. The comparison between *TWProxy* and *Machinetta* was also important to show how the teamwork efficiency may change according to the used coordination tool.

As future works, we intend to implement support to new organization of agents. We also intend to develop a new layer of fault-tolerant communication, trying to minimize the delay inserted in the treatment of lost messages. New games beyond the *Capture The Flag* will also be investigated.

Acknowledgements

This work was partially supported by the Brazilian agency CNPq. We would like to thank Vania Bogorny and the anonymous reviewers for their constructive comments.

References

1. G. K. Andrew Scholer. Gamebots, 2000. Disponvel em: http://www.planetunreal.com/gamebots. Last access on September 2008.
2. H. Chalupsky, Y. Gil, C. Knoblock, K. Lerman, J. Oh, D. Pynadath, T. Russ, and M. Tambe. Electric elves: Applying agent technology to support human organizations. In *Proceedings of IAAI*, 2001.
3. P. B. de Byl. *Programming Believable Characters for Computer Games*. Charles Development Series, 2004.
4. X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, 2004.
5. B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

6. B. J. Grosz and S. Kraus. The evolution of sharedplans. In *Foundations and Theories of Rational Agency*, pages 227–262. Kluwer Academic Publishers, 1999.
7. R. Hill, J. Chen, J. Gratch, P. Rosenbloom, and M. Tambe. Intelligent agents for the synthetic battlefield: A company of rotary wing aircraft. In *Innovative Applications of Artificial Intelligence (IAAI-97)*, 1997.
8. N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.
9. R. M. Jones, J. E. Laird, P. E. Nielsen, K. J. Coulter, P. G. Kenny, and F. V. Koss. Automated intelligent pilots for combat flight simulation. *AI Magazine*, 20(1):27–41, 1999.
10. G. A. Kaminka. The robocup-98 teamwork evaluation session: A preliminary report. In *RoboCup*, pages 345–356, 1999.
11. G. A. Kaminka, M. M. Veloso, S. Schaffer, C. Sollitto, R. Adobbati, A. N. Marshall, A. Scholer, and S. Tejada. Gamebots: a flexible test bed for multiagent team research. *Commun. ACM*, 45(1):43–45, 2002.
12. H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.
13. J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: an architecture for general intelligence. *Artif. Intell.*, 33(1):1–64, 1987.
14. H. J. Levesque, P. R. Cohen, and J. H. T. Nunes. On acting together. In *Proc. of AAAI-90*, pages 94–99, Boston, MA, 1990.
15. S. Marsella, M. Tambe, J. Adibi, Y. Al-Onaizan, G. A. Kaminka, and I. Muslea. Experiences acquired in the design of robocup teams: A comparison of two fielded teams. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):115–129, 2001.
16. I. M. Monteiro. Uma arquitetura modular para o desenvolvimento de agentes cognitivos em jogos de primeira e terceira pessoa. In *Anais do IV Workshop Brasileiro de Jogos e Entretenimento Digital*, pages 219–229, 2005.
17. I. M. Monteiro and D. A. dos Santos. Um framework para o desenvolvimento de agentes cognitivos em jogos de primeira pessoa. In *VI Brazilian Symposium on Computer Games and Digital Entertainment - Computing Track*, pages 107–115, 2007.
18. J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.
19. R. Nair, T. Ito, M. Tambe, and S. Marsella. Robocup-rescue: A proposal and preliminary experiences. In *Proceedings of RoboCup-Rescue Workshop, Fourth International Conference on Multi-Agent Systems(ICMAS-2000)*, 2000.
20. D. V. Pynadath and M. Tambe. An automated teamwork infrastructure for heterogeneous software agents and humans. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):71–100, 2003.
21. C. Rich and C. L. Sidner. COLLAGEN: When agents collaborate with people. In W. L. Johnson and B. Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 284–291, New York, 5–8, 1997. ACM Press.
22. S. Russel and P. Norving. *Artificial Intelligence - A Modern Approach*. Prentice Hall, 2002.
23. P. Scerri, D. Pynadath, L. Johnson, R. Schurr, M. Si, and M. Tambe. A prototype infrastructure for distributed robot-agent-person teams. In *The Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.
24. P. Scerri, D. Pynadath, N. Schurr, A. Farinelli, S. Gandhe, and M. Tambe. Team oriented programming and proxy agents: The next generation. In *Proceedings of 1st international workshop on Programming Multiagent Systems*, 2004.
25. P. Scerri, D. Pynadath, and M. Tambe. Towards adjustable autonomy for the real world. *Journal of Artificial Intelligence Research*, 17, 2003.

26. N. Schurr, S. Okamoto, R. T. Maheswaran, P. Scerri, and M. Tambe. Evolution of a teamwork model. In *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, pages 307–327. Cambridge University Press, 2005.
27. R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1981.
28. K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The RETSINA MAS infrastructure. Technical Report CMU-RI-TR-01-05, Robotics Institute Technical Report, Carnegie Mellon, 2001.
29. K. Sycara and G. Sukthankar. Literature review of teamwork models. Technical Report CMU-RI-TR-06-50, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, November 2006.
30. M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.
31. M. Tambe and W. Zhang. Towards flexible teamwork in persistent teams: Extended report. *Autonomous Agents and Multi-Agent Systems*, 3(2):159–183, 2000.
32. T. Vu, J. Go, G. Kaminka, M. Veloso, and B. Browning. Monad: A flexible architecture for multi-agent control. In *In Proceedings of the AAMAS03*, pages 449–456, 2003.
33. J. Yen, J. Yin, T. R. Ioerger, M. S. Miller, D. Xu, and R. A. Volz. CAST: Collaborative agents for simulating teamwork. In *IJCAI*, pages 1135–1144, 2001.

NonKin Village: A Training Game for Learning Cultural Terrain and Sustainable Counter-Insurgent Operations

Barry G. Silverman, PhD, David Pietrocola, Nathan Weyer, Deepthi Chandrasekaran, Ransom Weaver

Ackoff Collaboratory for Advancement of the Systems Approach (ACASA)
University of Pennsylvania, Philadelphia. PA 19104-6315
basil@seas.upenn.edu

Robert Might, PhD, Innovative Management Concepts, Inc., Dulles, VA 20166

Abstract. This article describes a virtual village we are currently assembling. Called NonKin Village, this is a gameworld that brings life to factional agents in sort of an emergent SimCity. It supports street level interaction and dialog with agents to learn their issues, needs, grievances, and alignments and to try to assist them in countering the agenda of an insurgent faction aimed at ending the rule of law. The player can attempt tactical DIME actions and observe PMESII effects unfold, but watch out! It's easy to go wrong in this foreign culture, to undertake operations with spurious side-effects. The player's goal is to learn enough about the foreign culture and their situation so as to be successful at influencing the world and facilitating a new dynamic to take effect, that of equitable and self-sustaining institutions.

1 Introduction

This article describes a training game, called NonKin Village, where the player (s) interacts with other virtual or real followers and leaders of contending factions at a local village level. These factions offer a corrupt sim-city type of world where one must convince various 'crime' families to convert to legit operation. NonKin is used to simulate insurgent operations in the village. The insurgent leader uses recruits to carry out missions. The player (s) has constrained resources, and must use them judiciously to try and influence the world via an array of Diplomatic, Intelligence, Military, and Economic (DIME) actions. The outcomes are presented as a set of intended and unintended Political, Military, Economic, Social, Informational, and Infrastructure (PMESII) effects.

The goal is to push the player through the three stages of COIN theory: survey the social landscape, make friends/coopt the agenda, and foster self-sustaining institutions so the player can safely depart. The player learns to use the given resources judiciously and in a culturally sensitive way to achieve desired outcomes. All agents in the game are conversational and are able to explain their internal states, group grievances, relations/alignments, fears, and wants. The agents carry out daily life functions in the

village in order to satisfy their internal needs for sleep, sustenance, company/belonging, maintaining relationships, prayer, etc. The village has places of employment, infrastructure, government and market institutions, and the leaders (agents) manage the economic and other institutional resources of their factions.

1.2 Theories of Insurgency/Terrorism

To begin, one can readily envision an insurgency existing in a world where a number of factions or clans range across the spectrum from those desiring the rule- of-law to those interested in chaos and regime change for any of a variety of reasons (ethno-political grievance, greed, crime, etc.). This is depicted across the top of Figure 1. Indeed, in the Maoist theory of armed struggle, the preparatory stage of an insurgency is characterized by actions that seek to affect separate factions of the population of the nations or regions they are trying to influence, causing different factions to iterate (dynamically) through several states ranging from animosity and paralyzing fear to sympathy and membership in the insurgent movement (Griffith, 1961).

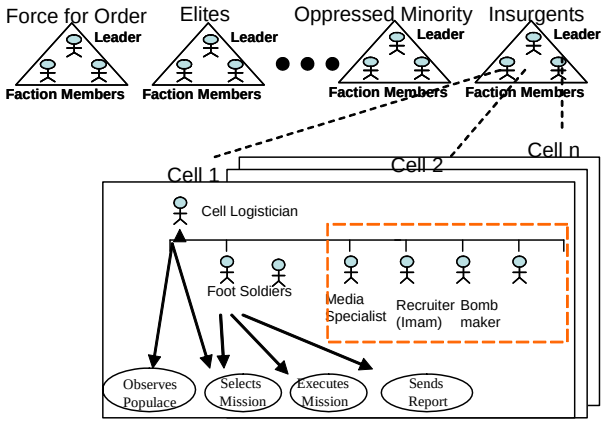


Figure 1 – Factions and Agents in an Insurgent Area

Ideally one would like to realistically simulate such behaviors for the purposes of being able to train against it and analyze what influences it in a given area of operation. To train/analyze how to co-opt the agenda of an insurgency and mobilize the populace toward the rule of law one needs a set of simulated factions and insurgent agents readily adapted to any given region. Since members of a given populace will be at varying degrees of support for and participation with each side, this implies that the aim of counter insurgency is not solely to destroy groups at the enemy end of the spectrum, but also to progressively shift individuals and groups closer to the friendly end. In fact, since

insurgent cells are often hidden amongst supporting members of the population (bottom of Figure 1), a focus strictly on reaction to insurgent attacks can be counter-productive. It will leave the agenda in their hands, cause collateral damage to potentially woo-able factions, and make the force for order seem to have no successful agenda of its own. Instead one must encourage the force for order to use a 'full spectrum' of approaches to help diagnose the source of grievance, attempt to ameliorate the root causes, and build up whatever services and institutions that are lacking and potentially also causing discontent: eg., see [1-3, 6-8, 13].

2 Socio-Cognitive Agents

We have been assembling an architecture able to support the authoring of games in this milieu. This section overviews the main components of the architecture. Sun [1] and Zacharias et al. [2] provide a useful survey of the respective fields of social agents and cognitive agents and show that there are very few meso-level environments that straddle both topics to provide socio-cognitive architectures. Let us therefore, briefly review its major layers – social and cognitive.

2.1 FactionSim –The Social System Layer [11]

FactionSim is an environment that captures a globally recurring socio-cultural "game" that focuses upon inter-group competition for control of resources (e.g, Security/Economic/Political Tanks). It is a tool that allows conflict scenarios to be established in which the factional leader and follower agents all run autonomously and are free to employ their micro-decision making as the situation requires. This environment facilitates the codification of alternative theories of factional interaction and the evaluation of policy alternatives. A faction has roles for one or more leaders; various ministers influenced by the leaders and in charge of institutions that tap the resources and provide services; and several named and/or archetypical sets of followers (e.g., core, fringe, and hierarchies of sub-groups). FactionSim uses PMFserv (see below) to fill these leader-minister-follower roles. Factions can be clustered into super-groups ahead of time. They also may dynamically form or break alliances by autonomous agent decision making. Analysts can interact with the FactionSim and attempt to employ a set of DIME actions to influence outcomes and PMESII effects. FactionSim has recently been extended to handle specific follower groups (military, bureaucracy, elites, religious groups, etc.) and a module was added that contains economic institutions that manage the allocation and distribution of public and some private goods. It also supports various types of economies (i.e., tribal, developmental, black market, etc.) and various regime types ranging from authoritarian to democracies with election processes. Third party economic models can also be used to supplant this set.

2.2 Profiling Cognitive Agents [10]

PMFserv is a COTS (Commercial Off The Shelf) human behavior emulator that drives agents in simulated gameworlds. This software was developed over the past ten years at the University of Pennsylvania as an architecture to synthesize many best available models and best practice theories of human behavior modeling. PMFserv agents are unscripted, but use their micro-decision making, to react to actions as they unfold and to plan out responses. A performance moderator function (PMF) is a micro-model covering how human performance (e.g., perception, memory, or decision-making) might vary as a function of a single factor (e.g., event stress, time pressure, grievance, and so on.). PMFserv synthesizes dozens of best available PMFs within a unifying mind-body framework and thereby offers a family of models where micro-decisions lead to the emergence of macro-behaviors within an individual. For each agent, PMFserv operates its perception and runs its physiology and personality/value system to determine coping style, emotions and related stressors, grievances, tension buildup, impact of rumors and speech acts, and various mobilization and collective and individual action decisions to carry out the resulting and emergent behaviors. None of these PMFs are "home-grown"; instead they are culled from the literature of the behavioral sciences. Users can turn on or off different PMFs to focus on particular aspects of interest. When profiling an individual, various personality and cultural profiling instruments are utilized with GUI sliders and web interviews to elicit the parameter estimates from a country, leader, or area expert.

2.3 World Markups

In addition to managing agents, PMFserv also manages objects (representing both agents and non-agents, such as a car, location, a business, etc), including when and how they may be perceived and acted on by agents. PMFserv implements affordance theory, meaning that each object applies perception rules to determine how it should be perceived by each perceiving agent. Objects then reveal the actions (and the potential results of performing those actions) afforded to the agent. For example, an object representing a car might afford a driving action which can result in moving from one location to another. A business might afford running it, working there, purchasing goods, and/or attacking and damaging it. These object affordance markups permit the PMFserv agents to perceive and reason about the world around them. In NonKin a great many objects are being pre-encoded in its libraries so that training scenario developers need not fill in the markups, but only need to link them to structures, areas, organizations, etc. of that town or region.

2.4 Abstract Objects: Event, Interaction, Transgression

Abstract objects, such as plans, obligations, and speech acts, are represented in the same way as concrete objects. Indeed, we make use of this representational form to store historical grievances and new social transgressions as they occur in the simulation. By “social transgression” we mean an offense an agent can commit against social rules. We utilize a simple, yet comprehensive taxonomy of the types of transgressions possible [5]. All transgressions have a transgressor, a set of victims, and a set of effects. Effects are the

direct effects of the offending action, not the emotional effects on observers. Those are handled internally by the PMFserv agents. Beyond these basic properties, our transgression objects keep track of some relations with the transgressor, relations with observers, properties of the effects, and relations between the transgressor and observers. The transgression objects also keep track of atonement actions emanating from transgressors and their agents (e.g., compensation, apology, etc.) and forgiveness actions by the victims. At any event, NonKin has an interface for the scenario training developer to add the historical transgressions of an area, while others that agents and players commit as the training unfolds get dynamically generated and autonomously managed.

Two other types of objects bear mentioning. One can build up complex interaction objects (scripts/templates) that might contain multiple scenes and roles. These are stored in the Village and triggered as needed by AESOP or by events. Event objects keep a record of what has happened (good and bad) and who did what. Both interaction and event objects contain know-how that can be explained to the player by many agents in a story (e.g., how to atone for each type of transgression event, how to reach a COIN or SSTR milestone, etc), though they might offer their own perspective as well.

2.5 AESOP and Conversational Abilities [9]

A final feature to cover about PMFserv is that the agents are conversational. That is, a player may interrogate them about the parameter settings of any of their tanks, goals, standards, preferences, social relations, group dynamics, decision making history, opinions about other agents and the actions they have done, opinions about FactionSim institutions and organizations, how they feel about transgressions and transgressors, whether atonement is possible and how forgivable is the grievance, etc. PMFserv includes a narrative engine (AESOP, [9]) able to tell stories about what an agent knows. In this way, the PMFserv agents can give qualitative explanations when they talk. PMFserv agents do not parse natural language queries, but instead expose a large list of things you can talk to them about. These lists are dynamic and hierarchical so that the conversations are multi-stepped. Further, the agents include a “familiarity” scale (1-6) so they will reveal more things they will discuss with the player, the more familiar and trustworthy the player is to them. In terms of trustworthiness, the agents apply their full set of cognitions, emotions, social relation PMFs, etc. to the player and the player’s group. So this means that they will grow closer to the player the more the player does good deeds for them. A player can do good deeds in the village in general, or via a structured dialog language make “social contracts” with specific agents that spawn objects the agents keep track of. Since PMFserv agents keep track of commitments and promises the player makes to them, it is equally possible to build trust with them and/or to violate expectations and cause their enmity.

3 NonKin Overview

FactionSim is abstracted from geo-political considerations and, further, it makes no commitment to populating the factional leaders and followers in any sense in real locations, occupations, or other roles. We have a country sim generator that does this for state and international actor modeling. NonKin is the name of our generator intended to bring FactionSim into focus for human terrain in tactical regions (See Figure 2).
Specifically, NonKin is a region scenario generator meant for use to implement villages, towns, and city neighborhoods, including connectivity of these areas to higher level institutions and assets. It is a SimCity genre of game engine. It is a portable/plugin and role playing game generator that is being outfitted with a reusable and extensible library of mock characters, institutions, factions, militias, and so on who carry out daily life and various economic, political, familial, and security activities in a culturally accurate way. Factions and institutions/organizations and roles are defined atop FactionSim, while agents are driven by the PMFserv engine.

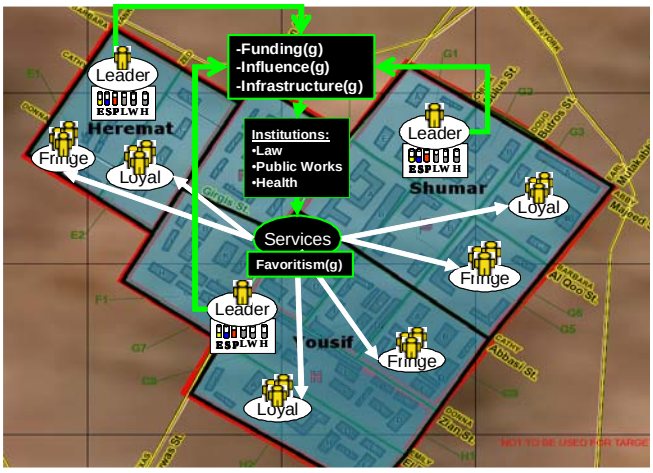
Shown in Figure 2 is a test scenario currently being implemented for the USMC. This is the hypothetical town of Hamariyah occupied by 200 individuals. The USMC folks from 29 Palms generated Hamariyah and descriptions of the town history, its 200 residents, 7 tribal groups, families, jobs, institutions, inter-factional grievances, and so on. This is a paper-based description, though some of it was provided in csv files that we recast into spreadsheet workbooks that were then read by the PMFserv model constructor. It is a plan for a mock town of role players that they have since deployed at 29 Palms.

The high level goals of NonKin include:

- Easy to use scenario generator -- profile actual personalities, cultures, group norms, historical grievances, institutions, infrastructure, etc. (based on turning on/off toggles from libraries of reusable agents, organizations, and other types of objects)

- Easy to use training content generator that offers a library of default training sets for the tasks, tactics, and procedures relevant to each of the three phases of the COIN theory as espoused in FM 3-24 (See [13] but NOTE: since the TTPs are unspecified by USMC doctrine, the defaults are examples taken from interviewing the 29 Palms observer/controllers or Coyotes. The default cases are editable and extensible).
  - Conduct security operations, patrols, checkpoints, etc.
  - Interact in different areas of town, in order to inventory and befriend the population.
    Discover who lives where and the residents' grievances, hopes, and stabilization and reconstruction needs. Also, find out intel about bad guys.
  - Visit often, become familiar to the residents, observe the residents going about their daily
    routines, and learn to detect anomalous and suspicious behaviors before bad things actually happen.

a. FactionSim layer of the village



b. PMFserv residents in daily life

Figure 2 – Overview of NonKin Village's Social and Cognitive Layers as well as the Player Dialog Window

- Make reconstruction commitments and further attempt to befriend factions and coopt the agenda away from the simulated insurgent faction's leaders and followers. Try out interventions of the DIME-FIL type and observe PMESII effects spread through communities.
- Attempt to understand what is needed community-wide and in the institutional organizations that provide services and resources so that a self-sustaining peace can take sway and withdrawal is possible. Try to run some transition and NonKin agent training so the local leaders and followers can assume running of their own services.

- Commit mistakes and errors in this simulated world so you don't need to commit them in the real world that it simulates. Encounter multiple cases and situations for the TTPs of each of the COIN phases.
- Receive missions, during-play coaching, and after action reviews from PMFserv-profiled agents.

We have spent nearly two years building up these diverse capabilities and have working prototypes of many of them. The first year was devoted to COIN phase I (inventory the human and cultural terrain), while the current year is aimed at COIN phase II (coopt the agenda, make friends in the village). A series of video clips demo these capabilities.

4 Conclusions and Next Steps

This article has described a virtual village we are currently assembling for the US Marine Corp. Called NonKin Village, this is a gameworld that brings life to agents of all factions in sort of a SimCity style of play although it supports street level interaction and dialog with agents to learn their issues, needs, grievances, and alignments and to try to assist them in countering the agenda of the insurgent faction

On the technology side, this research thus far concerned how to merge three successful tools from our lab -- a cognitive sim emulator (PMFserv), a social layer (FactionSim), and a story engine (AESOP). The goal was to develop a prototype synthesis that preserves the strengths of each tool. The theories contained in the default version of NonKin straddle individual psycho-physiological ones (stress, emotion, sacred values, etc.) as well as social ones (belonging, mobilization and grievance, group membership, motivational congruence). One can edit these starting theories with the internal editors of FactionSim and PMFserv. As such, NonKin serves as a successful proof-of-existence test for a socio-cognitive agent architecture.

As Figure 2 portrayed, this exercise was successful in producing a first prototype synthesis, a proof of concept that a fuller featured synthesis could be accomplished. Today, one can play NonKin Village and not realize that previously non-communicative technologies are now interoperable in the background.

To support NonKin, we successfully implemented first versions of several features that are now open for further research as we spiral toward the next version. In fact, there are a host of further research directions that a project like NonKin opens up. We mention but a few of these here.

An important goal of our research is to instantiate sufficient content in the models, objects, and markups of the NonKin generator so that it will be largely a matter of turning things off when trying to set up NonKin for a new village or region of the world. Our approach for doing that relies on filling the sim with social science theories about descriptive and profiling methods. Thus the PMFserv holds hundreds of models whose parameters have been tested and shown to pertain in applications for North and South Asia, Africa, the Mid-East, and so on. In addition, we are evolving object taxonomies for the range of objects mentioned in this paper – transgressions, events, interactions, etc. – that will support migrating the village to new settings. There is much to be done, but ultimately, we will add editors that permit non-programmers to readily turn off features not needed and to adjust others to the setting of interest. We have achieved this style of editing before with our Athena's Prism world diplomacy game and believe it is repeatable here.

In the expectation that NonKin will ultimately be fielded in remote locations, we have teamed with a company (IMC Corp) that is capable of supporting NonKin applications and users around the world. Also, they have begun looking into ways to automatically instantiate a village game directly from intelligence and event databases. If successful, this would turn NonKin into a recreation of actual rather than mock villages. That would make it useful for tactical purposes, a topic perhaps best reserved for a different paper.

To support the NonKin trainee, we also added a rucksack where the player keeps a journal and can do analysis on what has transpired. This is also where the player can find copies of his current orders, dialogs-to-date held with all characters, after action reports, and feedback on his training thus far relative to pedagogical objectives for the village. A Colonel agent gives orders and a feedback agent exists – both of whom the player can converse with. Each of these player support elements are items of continual refinement.

The training is meant to unfold in three tiers that we are adding to constantly as well:

- Casual Interaction & Cultural Flash Cards – This is a short and simple method of interacting with a given villager in a given context, It is useful for learning cultural norms about interacting with women, children, head of household, etc. Skills learned here will help in Lane training.
- Interaction-Oriented LANES – Lanes are specific tasks such a searching a house, staffing a checkpoint, detaining a suspect, and/or talking to a tribal leader, among others. They are isolated tasks and one gets to rehearse and hear feedback on each of them separately.
- Grievance-Oriented FEX Game -- This is a full village scenario. If the player does nothing, the factional leader and follower agents use their micro-decision making to act on their rivalries, grievances, and resource control concerns. If you mis-manage the situation, various factions and members might be drawn to the insurgent faction's side. Alternatively, through the mechanics of the story engine,

the user should instead be able to dialog and interact with each character to try and influence the world so that a new dynamic takes effect, that of cooperation. Attempt tactical DIME actions and observe PMESII effects unfold.

As a final note, we have just begun a collaboration with a 3-D videogame authoring group who has an immersive world where the player interacts with individual agents to learn task- relevant foreign languages, gestures and basic matters of courtesy -- Alelo TLT's Tactical Language Training System [4]. The NonKin project hopes to benefit from the use of their services to provide its village interaction, cultural training, and COIN operations in an immersive 3-D rehearsal and training environment. This should enhance the immersive nature of the training experiences and help players to more rapidly transfer lessons to the real world.

Acknowledgement. The research on the NonKin was supported by ONR/HPTE and USMC/TECOM. Some of the FactionSim related work was supported by AFOSR. The transition is being supported by the DOD/HSCB program. None of these agencies are responsible for the views expressed here.

References

1. Anon: The U.S. Army/Marine Corps Counterinsurgency Field Manual, FM3-24, (2007).
2. Chiarelli, P.W., Michaelis, P.R.: Requirement for Full-Spectrum Operations. Military Review July-August (2005)
3. Griffith, S.B. (Translation): Mao Tse-Tung On Guerrilla Warfare. Praeger, New York: (1961)
4. Johnson, W.L.: Serious use of a serious game for language learning. In: R. Luckin et al. (Eds.), *Artificial Intelligence in Education*, 67-74. Amsterdam: IOS Press. (2007)
5. Knight, K., et al.: Transgression and Atonement, in V. Dignum (ed), AAAI Workshop Proc., Chicago (2008)
6. Kilcullen, D.: Twenty-Eight Articles: Fundamentals of Company-Level Counterinsurgency. ISPERE – Joint Information Operations Center. (2004)
7. Nagl, J.: Learning to Eat Soup with a Knife. University Chicago Press, Chicago (2002)
8. Petraeus, D.H.: Observations from Soldiering in Iraq. Military Review. (2006).
9. Silverman, BG, Johns, M, Weaver, R., Mosley, J.: Authoring Edutainment Stories for Online Players (AESOP). In: Internat'l Conference on Virtual Storytelling, Toulouse, FR: Springer, (2003)
10. Silverman, B. G., Bharathy, G., O'Brien, K.: Human Behavior Models for Agents in Simulators and Games: Part II – Gamebot Engineering with PMFserv. Presence, 15, 2, 163-185 (2006b)

11. Silverman, B.G., Bharathy, G.K., Nye, B., Eidelson: Modeling Factions for 'Effects Based Operations': Part I – Leader and Follower Behaviors. Journal Computational &Mathematical Organization Theory (2007a)
12. Sun, R.: Cognition and Multi-Agent Action. Cambridge Univ. Press, Cambridge (2004)
13. USMC (2006), *Small Unit Leaders Guide to Counter Insurgency Operations (COIN)*, Quantico: USMC.
14. Zacharias, G.L., MacMillan, J., and Van Hemel, S.B. (Eds.): Behavior Modeling and Simulation: From Individuals to Societies. National Academies Press, Washington. (2008)

Intelligent NPCs for Educational Role Play Game

Mei Yii Lim[1], João Dias[2], Ruth Aylett[1], and Ana Paiva[2]

[1] School of Mathematical and Computer Sciences,
Heriot Watt University,
Edinburgh, EH14 4AS, Scotland
{myl, ruth}@macs.hw.ac.uk
[2] INESC-ID, IST, Taguspark,
Av. Prof. Dr. Cavaco Silva,
2744-016 Porto Salvo, Portugal
{joao.dias, ana.paiva}@gaips.inesc-id.pt

Abstract. Video games in general and educational role play games in particular would increase in believability if Non Player Characters reacted appropriately to the player's actions. Realistic and responsive feedback from game characters is important to increase engagement and enjoyment in players. In this paper, we discuss the modelling of autonomous characters based on a biologically-inspired theory of human action regulation taking into account perception, motivation, emotions, memory, learning and planning. These agents populate an educational Role Playing Game, ORIENT (Overcoming Refugee Integration with Empathic Novel Technology) dealing with the cultural-awareness problem for children aged 13 to 14.

1 Introduction

Non Player Characters (NPCs) vary in importance and may play roles of bystanders, allies or competitors to the player in the fictional world of computer games. These NPCs' behaviour is usually scripted and automatic, triggered by certain actions of or dialogue with the player. This method is cumbersome and produces gameplay that is repetitive and thus unnatural. In more advanced Computer Role Playing Games (CRPGs), NPC behaviour can be more complex and players' choices may affect the course of the game, as well as the conversation (eg. Fallout3[3]). However, true dialogues with NPCs are still a problem. In most CRPGs, the same dialogue option chosen by the player will usually receive the same reply from the NPC.

It is beneficial for NPCs to be believable and 'real' so that the player will enjoy interacting with them. Characters that are able to express their feelings and can react emotionally to events are more life-like. NPCs capable of dynamically reacting to player actions in reasonable and realistic ways are therefore very

[3] http://fallout.bethsoft.com/eng/home/home.php

desirable. Natural interaction between NPCs and the player is very important because it transforms the challenge of the game from a technical one to an interpersonal one, and thus may increase both the enjoyment and the engagement of players. However, up-to-date, real autonomous agents that are capable of improvisational actions, appear to be able to 'think', and have desires, motivations and goals of their own, are still rare in games.

2 Educational Role Playing Game

Researchers pointed out that play is a primary socialization and learning mechanism common to all human cultures and many animal species. 'Lions do not learn to hunt through direct instruction but through modeling and play.' [1]. Games are effective because learning takes place within a meaningful context where what must be learned is directly related to the environment in which learning and demonstration take place.

How can cultural studies be made exciting? Perhaps through an educational role play game. For instance, one in which the student is a space command member who must master the patterns of behaviour of an alien culture and pass as their friend within a digitally simulated world. The students will have interesting missions to keep them motivated and engaged. This approach shifts the students' cognitive effort from reading about educational content to hands-on experience of achieving compelling goals. Members of a team can cooperate with each other to solve the team's conflicts with other agents, whether a player from another team or an NPC. Such an opponent must be perceivable as endowed with a personality if the player is to be able to suspend disbelief in the way engagement with the storyworld requires.

In ORIENT[4], our game world is designed in just such a way. It is an interactive computer assisted role-playing game where three players act as visitors to a foreign planet that is inhabited by an alien culture. In order to save the planet from an imminent catastrophe, the players have to cooperate with the alien inhabitants, which can only be achieved by integrating themselves into the culture. Since the game incorporates a social setting, each NPC must be able to establish social relationships with other NPCs and the players to ensure successful collaboration. ORIENT characters must be able to recognise cultural differences and use this information to adapt to other cultures dynamically. The ability to empathise, that is, to detect the internal states of others and to share their experience, is vital to the formation of long-term relationships. Since enhancement of integration in a cultural group relies both on the understanding of the internal states of the persons involved and their affective engagement, both cognitive [2] and affective [3] empathy are relevant. Additionally, previous experience is crucial in maintaining long-term social relationships, which means a requirement for an autobiographic memory [4] is inevitable. Through an ability to retrieve previous experiences from its autobiographic memory, an NPC

[4] http://www.e-circus.org/

will be able to know how to react sensibly to a similar future situation. Thus, ORIENT provides a good case study for modelling NPCs with adaptive and improvisational capabilities, that possess autobiographical memory, individual personality and show empathy.

3 Related Work

Much recent work has been carried out on developing agents with autonomous capabilities. Some of this work focuses on physiological aspects while some focuses instead on cognitive aspects of human action regulation. Examples of existing physiological architectures are those by Cañamero [5], Velásquez [6] and Blumberg [7]. Cañamero's architecture relies on both motivations and emotions to perform behaviour selection for an autonomous creature. Velásquez developed a comprehensive architecture of emotion based on Izard's four systems model [8], focusing on the neural mechanism underlying emotional processing. Blumberg developed an animated dog, Silas that has a simple mechanism of action-selection and learning combining the perspective of ethology and classical animation. A more recent implementation of the model is AlphaWolf [9], capturing a subset of the social behaviour of wild wolves. These architectures are useful for developing agents that have only existential needs but are too low level for characters which require planning and storytelling capabilities as in ORIENT. Another problem of these architectures is that the resulting agents do not show emotional responses to novel situations because all behaviours are hard-coded.

On the cognitive end, the OCC cognitive theory of emotions [10] is one of the most used emotion appraisal model in current emotion synthesis systems. The authors view emotions as valenced reactions that result from three types of subjective appraisals: the appraisal of the desirability of events with respect to the agent's goals, the appraisal of the praiseworthiness of the actions of the agent or another agent with respect to a set of standards for behaviour, and the appraisal of the appealingness of objects with respect to the attitudes of the agent. Numerous implementations of the theory aimed at producing agents with a broad set of capabilities, including goal-directed and reactive behaviour, emotional state and social knowledge exist, beginning with the Affective Reasoner architecture [11], the Em component [12] of the Hap architecture [13], EMA [14], FAtiMA (FearNot! Affective Mind Architecture) [15] and many more. On the other hand, most deliberative agent architectures are based on the BDI (Beliefs, Desires, Intentions) model [16]. The ways BDI agents take their decisions, and the reason why they discard some options to focus on others, however, are questions yet to be answered. These problems are associated with the BDI architecture itself and not with a particular instantiation. Furthermore, BDI agents do not learn from errors and experience.

In order to create purely autonomous agents, we argue that a hybrid architecture combining both physiological and cognitive aspects is required. Some examples of this type of architecture are those by Sloman [17], Jones [18], Oliveira [19] and Dörner [20]. The agent cognitive processes should result from lower-level

physiological processing and the outcome of cognitive processes should influence the agent's bodily states, producing complex behaviours that can be termed emotional. Damasio [21] provides neurological support for the idea that there is no 'pure reason' in the healthy human brain but emotions are vital for healthy rational human thinking and behaviour which means both cognitive and physiological systems are essential parts of intelligent agents.

4 ORIENT Agent Mind

4.1 Inspiration

The ORIENT agent mind (i.e. the program that controls NPCs' behaviour) is built upon FAtiMA [15] architecture applied in FearNot!v2.0. FAtiMA was an extension of a BDI architecture, hence, faced the problem of ambiguity in its decision making processes common in any BDI architecture. It has a reactive and a deliberative appraisal layer. The reactive appraisal process matches events with a set of predefined emotional reaction rules while the deliberative appraisal layer generates emotions by looking at the state of current intentions, more concretely whether an intention was achieved or failed, or the likelihood of success or failure. After the appraisal phase, both reactive and deliberative components perform practical reasoning. The reactive layer uses simple and fast action rules that trigger action tendencies. On the other hand, the deliberative layer uses the strength of emotional appraisal that relies on importance of success and failure of goals for intention selection. A goal is activated only if its start conditions are satisfied. Each goal also contains success and failure conditions.

The main reason for choosing FAtiMA is that it incorporates the OCC theory [10], has a continous planner [22] that is capable of partial order planning and includes both problem-focused and emotion-focused coping [23] in plan execution. The OCC theory models empathy easily because it takes into consideration appraisals of events regarding the consequences for others. It is - as far as we know - the only model that provides a formal description of non-parallel affective empathic outcomes (i.e. emotions that take a bad relationship between one agent and another into account, e.g., gloating and resentment). Moreover, since the OCC model includes emotions that concern behavioural standards and social relationships based on like/dislike, praiseworthiness and desirability for others, it allows appraisal processes that take into consideration cultural and social aspects, important for ORIENT agents.

However, the number of empathic emotional outcomes described in OCC: happy-for, resentment, gloating and pity is limited. Moreover, FAtiMA does not take the physiological aspects of emotion into account. Another problem with FAtiMA is the tedious authoring process of the character's goals, emotional reactions, actions and effects, and action tendencies so that the final behaviour of the characters is as intended. Having these values scripted reduces the dynamism of some of the core aspects modeled, resulting in agents that are not adaptive and do not learn from experience.

To address these constraints, we considered the PSI theory [20], a psychologically-founded theory that incorporates all the basic components of human action regulation: perception, motivation, cognition, memory, learning and emotions. It allows for modelling autonomous agents that adapt their internal representations to a dynamic environment. A few successes of the PSI model in replicating human behaviour in complex tasks can be found in [20, 24]. A PSI agent does not require any executive structure that conducts behaviour, rather, processes are self-regulatory and run in parallel driven by needs. Memory functions as a central basis for coordination.

Emotions within the PSI theory are conceptualised as specific modulations of cognitive and motivational processes enabling a wide range of empathic emotional effects. These modulations are realised by *emotional parameters*. *Arousal* is the preparedness for perception and reaction; *resolution level* determines the accuracy of cognitive processes; and *selection threshold* prevents oscillating between behaviours by giving the current intention priority. Different combinations of these parameter values lead to different physiological changes that resemble emotional experiences in biological agents. For example, if an event leads to a drop in the character's certainty, then its *arousal* level increases causing a decrease in the *resolution level*. In such situation, a quick reaction is required hence forbidding time-consuming search. The character will concentrate on the task in order to recover the deviated need(s) and hence may choose to carry out the first action that it found feasible. The character may be diagnosed as experiencing anxiety. Therefore, depending on the cognitive resources and the motivational state of the agent in a given situation, these parameters are adjusted, resulting in more or less careful or forceful ways of acting, as well as more or less deliberate cognitive processing.

Since, FAtiMA already includes perception, cognition, memory and emotions, we added the PSI motivational and learning components into the existing architecture. The motivational system serves as a quick adaptation mechanism of the agent to a specific situation and may lead to a change of belief about another agent as shown in [25], important for conflict resolution among ORIENT characters. PSI's other advantage over FAtiMA is that it does not require much authoring except initialising the agents with some prior knowledge. PSI agents' differences in behaviour will then correspond to different life-experiences that lead to different learned associations. Thus, PSI permits more flexibility both in authoring and the characters' behaviour than FAtiMA. Unfortunately, this also means lack of control over the characters' behaviour which is a problem because characters in ORIENT need to behave in certain ways so that the educational goal can be reached. According to [26], good educational games are games where narrative events situate the activity, constraining actions, provoking thought and sparking emotional responses. By making the NPCs react in certain ways, the player's ability to access information or manipulate the world is limited. This forces the player to evaluate the relative value of information and to devise appropriate goals and strategies to resolve complex authentic problems and help

them to develop an experiential understanding of what might be otherwise an abstract principle.

Combining FAtiMA and PSI, the problems of psychological plausibility and control are addressed, neither of which can be solved by either architecture alone. Cultural and social aspects of interaction can be modelled using FAtiMA while PSI provides an adaptive mechanism for action regulation, fulfilling the requirements of ORIENT characters. Author are free to decide how much information they want to provide the characters to start with and leave the rest for the characters to learn. The degree of desirability (or undesirability) of an action or event is proportionate to the degree of positive (or negative) changes that an action or event brings to the agent's drives. This desirability value can be used to automatically generate emotions according to the OCC model, removing some of the need to write predefined domain-specific emotional reaction rules. This means that the reactive layer in FAtiMA may be omitted.

4.2 FAtiMA-PSI Architecture

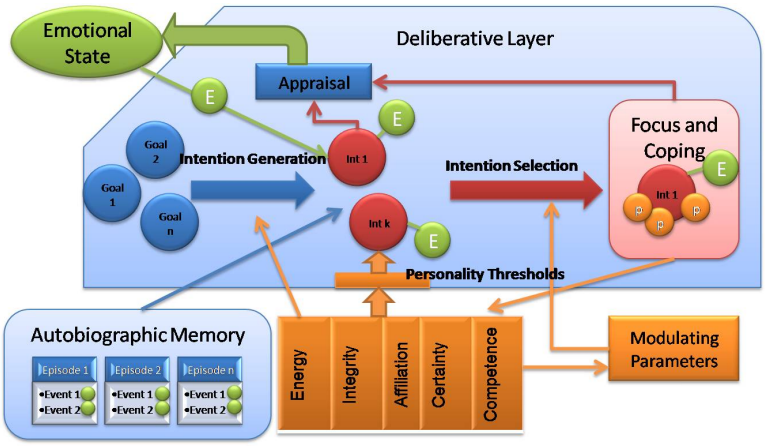


Fig. 1. FAtiMA-PSI architecture

In the ORIENT agent mind architecture shown in Figure 1, goals are driven by needs. A motivational system as in PSI provides the character with a basis for selective attention, critical for learning and memory processes, hence increases its adaptive prowess. Five basic drives from PSI are modeled in ORIENT including Energy, Integrity, Affiliation, Certainty and Competence. These needs can emerge over time or can be activated by events happening in the environment. Energy represents an overall need to preserve the existence of the agent (food +

water). As the agent carries out actions, it consumes energy which means that eventually, it will have to rest or perform actions to regain energy. Integrity represents well being, i.e. the agent avoids pain or physical damage while affiliation is useful for social relationships. On the other hand, certainty and competence influence cognitive processes. It is assumed that the scales for all drives are comparable, ranging from 0 to 10 where 0 means complete deprivation while 10 means complete satisfaction. An agent's aim is to maintain these drives at the highest level possible at all time in order to function properly.

The motivational system also allows the creation of agents with personality. Each drive has a specific weight ranging from 0 to 1 that underlines its importance to an agent. The strength of a drive (*Strength(d)*) depends on its current strength plus the amount of deviation from the set point (effect of goal/action) and the specific weight of the drive. For example, if agent A is a friendly character, affiliation would be an important factor in its social relations, say weight 0.8 while a hostile agent B would have a low importance for affiliation, say weight 0.3. Now, if both agents have a current affiliation value of 2 and if the deviation from set point is -4, agent A's strength for affiliation would be -1.2 (2+(-4*0.8)) while agent B's strength for affiliation would be 0.8 (2+(-4*0.3)) based on Equation 1. The higher the strength of a drive, the lower the agent's need is for that particular drive. In this case, agent A will work harder to satisfy its need for affiliation than agent B. So, by assigning different weights for different needs to different agents, characters with different personalities can be produced.

$$Strength(d) = Strength(d) + (Deviation(d) * Weight(d)) \quad (1)$$

A goal is define by the following attributes:

- Id: the goal identifier or name
- Preconditions: a list of conditions that determine when the goal becomes active
- SuccessConditions: a list of conditions used to determine if the goal is successful
- FailureConditions: a list of conditions that determine the goal failure
- EffectsOnDrives: specifies the effects that the goal will have on the agent's drives if the goal succeeds

Each goal contains information about its expected contribution to energy, integrity and affiliation, that is, how much the drives may be deviated from or satisfied if the goal is performed. Likewise, events or actions also include contributions to drives. Based on this information, the importance of goals to each character at a particular time instance can be determined, allowing the character to give priority to goals that satisfy its needs under different circumstances. This is an advantage over the previous FAtiMA architecture where a goal's importance is pre-authored which mean that whenever a goal activation condition becomes true, the goal is always created with the same importance of success and failure, independently of the situation that originated the goal. This causes

a problem in deciding which goal should be selected when there are several conflicting goals. The effects of needs are also useful in the appraisal phase to create emotional impact that will be stored in the autobiographic memory and guide the agent's further actions. Since each agent has a different personality, the effect of an event may differ from one agent to another, which in turn affects their emotional and behavioural responses. Thus, needs can be considered both the source of behaviour and feedback from the effect of behaviour, a fundamental aspect necessary for learning agents.

As for certainty and competence, no explicit specification of contributions to these is necessary because they are cognitive needs and their values can be calculated automatically as described below. Whenever an expected event fails to turn up or an unknown object appears, the agent's certainty drops. Thus, uncertainty represents the extent to which knowledge about a given fact/action is not accurate or not known. We model uncertainty using error prediction with an Exponential Moving Average where the weighting factors decreases exponentially resulting in the lastest data being the most important. ORIENT characters continuously make predictions about the probability of success of their goals. These predictions are then compared with the actual outcomes. The difference between these two values is the *ObservedError*. Based on past observed errors, we can estimate the current error, that is, the current uncertainty using Equation 2. α represents the rate past observations lose importance and t is the time step for the character's mind cycle.

$$Uncertainty(t) = \alpha * ObservedError(t-1) + (1-\alpha) * Uncertainty(t-1) \quad (2)$$

Hence, gaining certainty is not about avoiding uncertain goals but trying out these goals. This is because in order to achieve certainty, the character has to reduce the estimation error, and the goals which have high error estimations are goals that contribute more to certainty. Certainty is achieved by exploration of new strategies or actions, which leads to the construction of more complete hypotheses. If trial and error is too dangerous, developments in the environment are observed in order to collect more information. By doing so, the character can change its behaviour dynamically. Please note that the character does not learn by forming new goals because this will lead to a lack of control over its behaviour. Instead, it learns by trying out different actions from a pre-specified set of actions and remembering which actions helped it to tackle a situation best. This information is stored in its autobiographic memory and serves as an indicator to the success probability of satisfying a specific need in future. Since certainty depends on the amount of unkown information relating to a goal, the more an agent encounters the same type of situation, the higher its certainty is regarding the situation.

Competence represents the efficiency of an agent in reaching its goals and fulfilling its demands. Success increases competence while failure decreases it. The agent's autobiographic memory provides a history of previous interactions, which records the agent's experience in a task (the number of successes in performing

a goal) useful for the calculation of goal competence (likelihood of success in performing a goal, Equation 3). Since no distinction is made in calculating competence between achieving an important goal or a less important one, one can assume that all goals have the same contribution to the success rate. If the agent cannot remember previous activations of the goal, then it ignores the likelihood of success and increases the goal's contribution to certainty.

$$Comp(g) = NoOfSuccesses(g)/NoOfTries(g) \tag{3}$$

$$OverallComp = NoOfSuccesses/NoOfGoalsPerformed \tag{4}$$

The autobiographic memory also stores information about the agent's overall performance (the number of successes so far taking into consideration all goals performed) useful for the calculation of overall competence (Equation 4). The expected competence (Equation 5) of the agent will then be a sum of its overall competence and its competence in performing a current goal. A low competence level indicates that the agent should avoid taking risks and choose options that have worked well in the past. A high competence means that the agent can actively seek difficulties by experimenting with new courses of action that are less likely to succeed. Together, competence and certainty direct the agent towards explorative behavior; depending on its abilities and the difficulty of mastering the environment, it will actively seek novelty or avoid complexity. During this learning process, the agent also remembers any specific expressed emotions by other agents in particular situations. It continuously updates and adapts this information enabling empathic engagement in future interactions.

$$ExpComp(g) = OverallComp + Comp(g) \tag{5}$$

At the start of an interaction, each agent has a set of initial values for needs. Based on the level of its current needs, the agent generates intentions, that is, it activates goal(s) that are relevant to the perceived circumstances. A need may have several goals that satisfy it (e.g. I can gain affiliation by making a new friend or socialising with an old friend) and a goal can also affect more than one need (e.g. eating food offered by another agent satisfies the need for energy as well as affiliation). So, when determining a goal's strength (Equation 6), all drives that it satisfies are taken into account. A goal that satisfies more drives will have a higher strength than those that satisfy less.

$$Strength(g) = \sum Strength(d) \tag{6}$$

For a particular need, the more a goal reduces its deviation, the more important that goal is (e.g. eating a full carbohydrate meal when you're starving satisfies you better than eating a vegetarian salad). By looking at the contribution of the goal to overall needs and to a particular need, goals that satisfy the same need can be compared so that success rate in tackling the current circumstances can be maximised. So, the utility value of a goal can be determined taking into consideration overall goal strength on needs (*Strength(g)*),

contribution of the goal to a particular need (*ExpCont(g, d)*) and the expected competence (*ExpComp(g)*) of the agent. Additionally, the urgency of a goal is taken into account. *Urgency(g)* gives importance to goals that should become active immediately, usually goals that satisfy the most current deviated need(s).

$$EU(g) = (1+goalUrgency(g))*ExpComp(g)*Strength(g)*ExpCont(g,d) \quad (7)$$

On each cycle, goals are checked to see if any has become active by testing the goal's preconditions. Once a goal becomes active, a new intention to achieve the goal is created and added to the intention structure. The intention represents the agent's commitment to achieve the goal and stores all plans created for it. Since there can be more than one intention activated at any particular time instance, the character must choose one of them to continue deliberation (and planning). Applying PSI, the selection of goal in ORIENT is performed based on the *selection threshold* value. The current active intention is selected based on a winner takes all approach, that is, the goal with the highest expected utility value is chosen. An unselected goal can be activated if its strength exceeds the value of the current active intention multiply by the *selection threshold*. So, if the *selection threshold* is high, it is less likely for another goal to be activated, hence, allowing the agent to concentrate on its current active intention. After an intention is selected, the agent proceeds to generate plan(s) to achieve it.

When a plan is brought into consideration by the reasoning process, it generates and updates OCC prospect based emotions such as:

- Hope: Hope to achieve the intention. The emotion intensity is determined from the goal's importance of success and the plan's probability of success.
- Fear: Fear of not being able to achieve the intention. The emotion intensity is determined from the goal's importance of failure and the plan's probability of failing.

All active goals are then checked to determine if the goal succeeds or fails. If the planner is unable to make a plan, more prospect based emotions will be generated, such as Satisfaction, Disappointment, Relief and Fears-Confirmed. In order to cope with different circumstances, ORIENT characters perform two types of coping: problem-focused coping and emotion-focused coping as in FAtiMA. Problem-focused coping focuses on acting on the environment to tackle a situation. It involves planning a set of actions that achieve a desired result and executing those actions. On the other hand, emotion-focused coping works by changing the agent's interpretation of circumstances, that is, lowering strong negative emotions for example, by lowering the importance of goals, a coping strategy used often by people when problem focused coping has low chances of success. These coping strategies are triggered by emotions and personality of the characters. For instance, a fearful character has a higher chance to drop an uncertain goal than a hopeful character.

5 Conclusion and Future Work

In this paper, we discussed our effort in developing autonomous NPCs for an educational role play game. In ORIENT, characters behaviour is regulated by a biologically-inpired architecture of human action regulation. The new addition of the motivational system onto FAtiMA provides ORIENT characters with a basis for selective attention, critical for learning and memory processes. Intentions are selected based on strength of activated needs, urgency and success probability addressing the BDI architecture ambiguity in decision making. The resulting agents learn through trial and error, allowing more efficient adaptation and empathic engagement in different social circumstances. The successful linking of body and mind is consistent with that of humans' and hence, should produce characters with behaviours that seem plausible to a human. The software which is written in Java has been made available at the open source portal SourceForge[5] and is reusable in autonomous agents applications.

Acknowledgements

This work was partially supported by European Commission (EC) and is currently funded by the eCIRCUS project IST-4-027656-STP with university partners Heriot-Watt, Hertfordshire, Sunderland, Warwick, Bamberg, Augsburg, Wuerzburg plus INESC-ID and Interagens. The authors are solely responsible for the content of this publication. It does not represent the opinion of the EC, and the EC is not responsible for any use that might be made of data appearing therein.

References

[1] Eck, R.V.: Digital game-based learning: It's not just the digital natives who are restless. EDUCAUSE Review **41**(2) (2006) 16–30
[2] Hogan, R.: Development of an empathy scale. Journal of Consulting and Clinical Psychology (35) (1977) 307–316
[3] Hoffman, M.L.: Empathy, its development and prosocial implications. Nebraska Symposium on Motivation **25** (1977) 169–217
[4] Ho, W.C., Dautenhahn, K., Nehaniv, C.L.: Computational memory architectures for autobiographic agents interacting in a complex virtual environment: A working model. Connection Science **20**(1) (2008) 21–65
[5] Cañamero, D.: A hormonal model of emotions for behavior control. In: VUB AI-Lab Memo 97-06, Vrije Universiteit Brussel, Belgium (1997)
[6] Velásquez, J.D.: Modeling emotions and other motivations in synthetic agents. In: Proceeding AAAI 97, AAAI Press and The MIT Press (1997) 10–15
[7] Blumberg, B.: Old Tricks, New Dogs: Ethology and Interactive Creatures. PhD thesis, Massachusetts Institute of Technology, MIT, Cambridge, MA (1996)
[8] Izard, C.E.: Four systems for emotion activation: Cognitive and noncognitive processes. Psychological Review **100**(1) (1993) 68–90

[5] http://sourceforge.net/projects/orient-ecircus

[9] Tomlinson, B., Blumberg, B.: Alphawolf: Social learning, emotion and development in autonomous virtual agents. First GSFC/JPL Workshop on Radical Agent Concepts (Oct, 04 2002)
[10] Ortony, A., Clore, G., Collins, A.: The cognitive structure of emotions. Cambridge University Press, Cambridge, UK (1988)
[11] Elliot, C.D.: The Affective Reasoner: A process model of emotions in an multi-agent system. PhD thesis, Northwestern University, Illinios (1992)
[12] Reilly, W.S., Bates, J.: Building emotional agents. Technical Report CMU-CS-91-143, School of Computer Science, Carnegie Mellon University (1992)
[13] Loyall, A.B., Bates, J.: Hap: A reactive adaptive architecture for agents. Technical Report CMU-CS-91-147, School of Computer Science, Carnegie Mellon University (1991)
[14] Gratch, J., Marsella, S.: Evaluating a computational model of emotion. Journal of Autonomous Agents and Multiagent Systems (Special issue on the best of AAMAS 2004) **11**(1) (2004) 23–43
[15] Dias, J., Paiva, A.: Feeling and reasoning: A computational model for emotional agents. In: 12th Portuguese Conference on Artificial Intelligence (EPIA 2005), Portugal, Springer (2005) 127–140
[16] Bratman, M.E.: Intention, Plans and Practical Reasoning. Harvard University Press, Cambridge, Massachusetts (1987)
[17] Sloman, A.: Varieties of affect and the cogaff architecture schema. In: Symposium on Emotion, Cognition and Affective Computing, AISB 01 Convention, University of York, United Kingdom (Mar, 21-24 2001)
[18] Randolph M. Jones, Amy E. Henninger, E.C.: Interfacing emotional behavior moderators with intelligent synthetic forces. In: Proceeding of the 11th CGF-BR Conference, Orlando, FL (May, 7 2002)
[19] Oliveira, E., Sarmento, L.: Emotional advantage for adaptability and autonomy. In: AAMAS, Melbourne, Australia, ACM (2003) 305–312
[20] Dörner, D.: The mathematics of emotions. In Frank Detje, D.D., Schaub, H., eds.: Proceedings of the Fifth International Conference on Cognitive Modeling, Bamberg, Germany (Apr, 10–12 2003) 75–79
[21] Damasio, A.: Descartes' Error: Emotion, Reason and the Human Brain. Gosset/Putnam Press, New York (1994)
[22] Aylett, R., Dias, J., Paiva, A.: An affectively driven planner for synthetic characters. In: International Conference on Automated Planning and Scheduling (ICAPS2006), UK (2006)
[23] Marsella, S., Johnson, B., LaBore, C.: Interactive pedagogical drama. In: Fourth International Conference on Autonomous Agents (AAMAS), Bologna, Italy, ACM Press (2002) 301–308
[24] Dörner, D., Gerdes, J., Mayer, M., Misra, S.: A simulation of cognitive and emotional effects of overcrowding. In: Proceedings of the 7th International Conference on Cognitive Modeling, Trieste, Italy (Apr, 5–8 2006)
[25] Lim, M.Y.: Emotions, Behaviour and Belief Regulation in An Intelligent Guide with Attitude. PhD thesis, School of Mathematical and Computer Sciences, Heriot-Watt University, Ediburgh, Edinburgh (2007)
[26] Squire, K.: Design principles of next generation digital gaming for education. Educational Technology **43**(5) (2003) 17–33 The Games-To-Teach Team at MIT.

Architecture for Affective Social Games

Derek J. Sollenberger and Munindar P. Singh

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
Phone: 1-717-860-1809
Fax: 1-919-515-7896
{djsollen,singh}@ncsu.edu

Abstract. The importance of affect in delivering engaging experiences in entertainment and education is well recognized. We introduce the Koko architecture, which describes a service-oriented middleware that reduces the burden of incorporating affect into games and other entertainment applications. Koko provides a representation for affect, thereby enabling developers to concentrate on the functional and creative aspects of their applications. The Koko architecture makes three key contributions: (1) improving developer productivity by creating a reusable and extensible environment; (2) yielding an enhanced user experience by enabling independently developed applications to collaborate and provide a more coherent user experience than currently possible; (3) enabling affective communication in multiplayer and social games.

1 Introduction

Games that incorporate reasoning about their player's affective state are gaining increasing attention. Such games have been prototyped in military training [6] and educational [10] settings. However, current techniques for building affect-aware applications are limited, and the maintenance and use of affect is in essence handcrafted in each application.

We take as our point of departure the results of modeling affect based on appraisal theory. A fundamental concept of appraisal theory is that the environment of the agent is essential to determining the agent's affective state. As such, appraisal theory yields models of affect that are tied to a particular domain with a defined context. Therefore, each new problem domain requires a new affect model instance. A current and common practice has been to copy and edit a previous application (and, occasionally, to build from scratch) to meet the specifications of a new domain. This approach may be reasonable for research proofs-of-concept, but is not suitable for developing production applications.

Additionally, in order to more accurately predict the user's affective state, many affective applications use physical sensors to provide additional information about the user and the user's environment. The number and variety of sensors continues to increase and they are now available via a variety of public services (e.g., weather and time services) and personal commercial devices (e.g., galvanic skin response units).

Current approaches require each affective application to interface with these sensors directly. This is not only tedious, but also nontrivial as the application must be adjusted whenever the set of available sensors changes.

To address these issues we propose a service-oriented architecture, called Koko, that compliments existing gaming engines by enabling the prediction of a gamer's affective state. When compared to existing approaches the benefits of our architecture are an increase in developer productivity, an enhanced user experience, and the enabling of affective social applications. Koko is not another model of emotions but a middleware from which existing (and future) models of affect can operate within. It provides the means for both game developers and affect model designers to construct their respective software independently, while giving them access to new features that were before impossible. Further, Koko is intended to be used by affective models and applications that seek to recognize emotion in a human user. While it is possible to use Koko to model the emotions of non-playing characters, many benefits, such as using physical sensors, most naturally apply when human users are involved.

The Koko architecture is realized as a service-oriented middleware which runs independently from the game engine. The primary reason for this separation becomes more apparent when we discuss the social and multiplayer aspects of Koko. Such an approach is consistent with existing techniques for multiplayer access to a central game server.

1.1 Benefits

Koko concentrates on providing three core benefits to affective game developers. In the remainder of this section, we elaborate on these benefits.

Developer Productivity. Koko separates the responsibility of developing an application from that of creating and maintaining the affect model. In Koko, the application logic and the affect model are treated as separate entities. By creating this separation we can in many cases completely shield one entity from the other and provide standardized interfaces at the points of interaction.

Additionally, Koko avoids code duplication by identifying and separating modules for accessing affect models and various sensors, and then absorbs those modules into the middleware. For example, by extracting the interfaces for physical sensors into the middleware, Koko enables each sensor to be leveraged through a common interface. Further, since the sensors are independent of the affect models and applications, a sensor that is used by one model or application can be used by any other model or application without the need for additional code.

Quality of User Experience. Abstracting affect models into Koko serendipitously serves another important purpose. It enables an enhanced user experience by providing data to both the application and its affect model that was previously unattainable, resulting in richer applications and more comprehensive models.

With current techniques it is simply not possible for separate applications to share affective information for their common users. This is because each application is independent of and thereby unaware of other applications in use by that user. By contrast, Koko-based applications can share common affective data through Koko. This reduces each application's overhead of initializing the user's affective state for each session, as

well as providing insight into a user's affective state that would normally be outside the application's scope.

Such cross-application sharing of a user's affective information improves the value of each application. As a use case, consider a student using both an education and an entertainment application. The education application can proceed with easier or harder questions depending on the user's affective state even if the state was changed by participation in some unrelated game.

Affective Social Applications. In addition to enabling cross-application sharing of affective data, the Koko architecture enables cross-user sharing of affective data. The concept of cross-user sharing fits naturally into the realm of social and multiplayer games. Through Koko, a user may view the affective state of other members in their social circle or multiplayer party. Further, that information can potentially be used to better model the inquiring user's affective state.

1.2 Contributions and Significance

Any software architecture is motivated by improvements in features such as modularity and maintainability: you can typically achieve the same functionality through more complex or less elegant means [14]. Of course, an improved architecture facilitates accessing new functionality: in our case, the sharing of affective information and the design of social applications.

Additionally, architectural improvements can have scientific significance. For example, separating knowledge bases from problem solving not only improved developer productivity, but also led to greater clarity and improvements in the concepts of knowledge representation and problem solving. Koko takes similar steps with respect to affect, especially by supporting cross-application and cross-user affective interactions.

1.3 Paper Organization

The remainder of this paper is arranged as follows. The background section reviews appraisal theory models. The architecture section then provides detailed description of the components that compose Koko. Finally, the evaluation section demonstrates the merits of the Koko architecture.

2 Background

Smith and Lazarus' [15] cognitive-motivational-emotive model, the baseline for current appraisal models (see Figure 1), conceptualizes emotion in two stages: appraisal and coping. *Appraisal* refers to how an individual interprets or relates to the surrounding physical and social environment. An appraisal occurs whenever an event changes the environment as interpreted by the individual. The appraisal evaluates the change with respect to the individual's goals, resulting in changes to the individual's emotional state as well as physiological responses to the event. *Coping* is the consequent action of the individual to reconcile and maintain the environment based on past tendencies, current emotions, desired emotions, and physiological responses [8].

Koko focuses on a section of the appraisal theory process (denoted by the dashed box), because Koko is intended to model emotions in human subjects. As a result, the

other sections of the process are either difficult to model or outside Koko's control. For instance, the coping section of the process is outside Koko's control as it is an action that must be taken by the user.

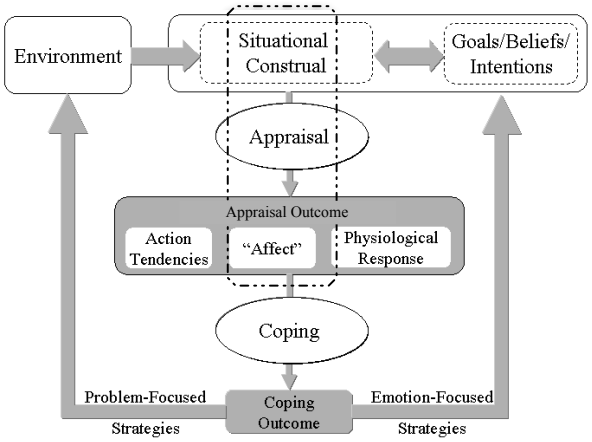


Fig. 1. Appraisal Theory [7]

A situational construal combines the environment (facts about the world) and the internal state of the user (goals and beliefs) and produces the user's perception of the world, which then drives the appraisal and provides an appraisal outcome. This appraisal outcome is made up of multiple facets, but the central result is "Affect" or current emotions. For practical purposes, "Affect" can be interpreted as a set of discrete states with an associated intensity. For instance, the result of an appraisal could be that you are happy with an intensity of α as well as proud with an intensity of β. Unfortunately, selecting the set of states to use within a model is not an easy task as there is no one agreed upon set of states that covers the entire affective space.

Next, we look at three existing appraisal theory approaches for modeling emotion. The first is a classic approach which provides the foundation for the set of affective states used within Koko. The final two are contemporary approaches to modeling emotion with the primary distinction among them being that EMA focuses on modeling emotions of non-playing characters while CARE concentrates on measuring human emotional states.

OCC. Ortony, Clore, and Collins [12] introduced the so-called OCC model, which consists of 22 types of emotions that result from a subject's reaction to an event, action, or object. The OCC model is effectively realized computationally, thus enabling simulations and real time computations. Further, the OCC's set of emotions have turned out to cover a broad portion of the affective space. Elliot [2] expanded the set of emotions provided by the OCC to a total of 24 emotions. Koko employs this set of 24 emotions as its baseline affective states.

EMA. The EMotion and Adaptation (EMA) model leverages SOAR [11] to extend Smith and Lazarus' model for applications involving non-playing characters [7]. EMA

monitors a character's environment and triggers an appraisal when an event occurs. It then draws a set of conclusions or *appraisal frames*, which reflect the character's perception of the event. EMA then assigns emotions and emotional intensities to each appraisal frame, and passes it to the coping mechanism, which decides the best actions to take based on the character's goals [5].

CARE. The Companion-Assisted Reactive Empathizer (CARE) supports an on-screen character that expresses empathy with a user based on the affective state of the user [9]. The affective state is retrieved in real-time from a pre-configured static affect model, which maps the application's current context to one of six affective states. The character's empathic response is based on these states.

CARE populates its affective model offline. First, a user interacts with the application in a controlled setting where the user's activity is recorded along with periodic responses from the user or a third party about the user's current affective state. Next, the recorded data is compiled into a predictive data structure, such as a decision tree, which is then loaded into the application. Using this pre-configured model the application can thus predict the user's affective state during actual usage.

3 Architecture

Existing affective applications tend to be monolithic where the affective model, external sensors, and application logic are all tightly coupled. As in any other software engineering endeavor, monolithic designs yield poor reusability and maintainability. Similar observations have led to other advances in software architecture [14]. Most pertinently, the idea of a knowledge base being separate from a solution strategy is motivated on such grounds.

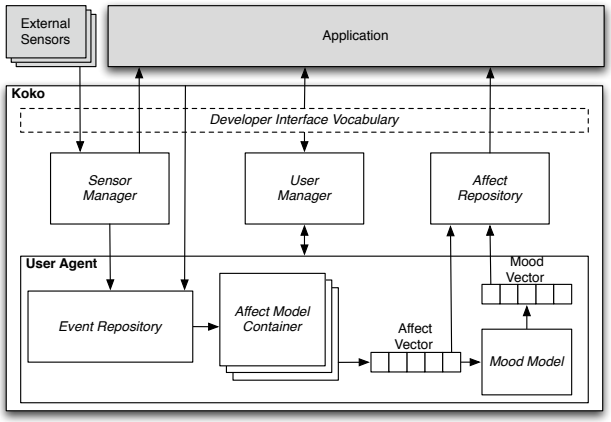


Fig. 2. Koko architectural overview

Figure 2 shows Koko's general architecture using arrows to represent data flow. The following portion of this section provide details on each of Koko's major components. Then after the groundwork of the architecture has been explained we elaborate on the formal interface definitions for the remainder of the section.

3.1 Main Components and their Usage

Koko hosts an active computational entity or *agent* for each user. In particular, there is one agent per human, who may use multiple Koko-based applications. Each agent has access to global resources such as sensors and messaging but operates autonomously with respect to other agents.

The Affect Model Container This container manages the affect model(s) for each agent. Each application specifies exactly one affect model, whose instance is then managed by the container. As Figure 3 shows, an application's affect model is specified in terms of the affective states as well as application and sensor events, which are defined in the application's configuration (described below) at runtime.

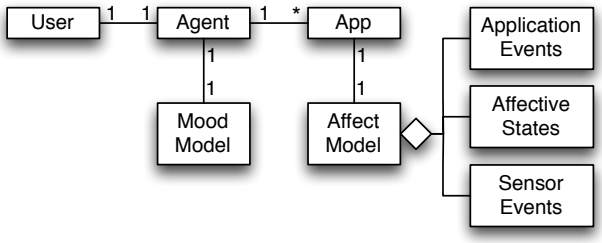


Fig. 3. Main Koko entities

Configuring the affect model at runtime enables us to maintain a domain-independent architecture with domain-dependent affect model instances. Further, in cases such as CARE, we can construct domain-independent models (generic data structures) and provide the domain context at runtime (object instances). This is the approach Koko has taken by constructing a set of generic affect models that it provides to applications. These affect models follow CARE's supervised machine learning approach of modeling affect by populating predictive data structures with affective knowledge. These affect models are built, executed, and maintained by the container using the Weka toolkit [16]. The two standard affect models that Koko provides use Naive Bayes and decision trees as their underlying data structures.

To accommodate models with drastically different representations and mechanisms, Koko encapsulates them via a simple interface. The interface takes input from the user's physical and application environment and produces an *affect vector*. The resulting *affect vector* contains a set of elements, where each element corresponds to an affective state. The affective state is selected from an ontology that is defined and maintained via the developer interface vocabulary. Using this ontology, each application developer selects the emotions to be modeled for their particular application. For each selected emotion, the vector includes a quantitative measurement of the emotion's intensity. The intensity is a real number ranging from 0 (no emotion) to 10 (extreme emotion).

Mood Model Following EMA, we take an *emotion* as the outcome of one or more specific events and a *mood* as a longer lasting aggregation of the emotions for a specific user. An agent's mood model maintains the user's mood across all applications registered to that user.

Koko's model for mood is simplistic as it takes in affect vectors and produces a *mood vector*, which includes an entry for each emotion that Koko is modeling for that user. Each entry represents the aggregate intensity of the emotion from all affect models associated with that user. Consequently, if Koko is modeling more than one application for a given user, then the user's mood is a cross-application measurement of the user's emotional state.

To ensure that a user's mood is up to date with respect to recent events, we introduce a *mood decay formula* [13] as a way to reduce the contribution of past events. This formula reduces the effect that a given emotion has on the user's mood over time. We further augment our mood model with the concept of *mood intensity* from the work of Dias and Paiva [1]. The mood intensity sums all positive and negative emotions that make up the user's current mood which is used to determine the degree to which a positive or negative emotion impacts a user. For example, if a user has a positive mood intensity then a slightly negative event may be perceived as neutral, but if the event were to recur the mood intensity would continue to degrade, thereby amplifying the effect of the negative event on the user's mood over time.

Affect Repository The affect repository is the gateway through which all affective data flows through the system. The repository stores both affect vectors (application specific) and mood vectors (user specific). These vectors are made available to both external applications as well as the affect and mood models of the agents. This does not mean all information is available to a requester, as Koko implements security policies for each user (see user manager). An entity can request information from the repository but the only vectors returned are those they have the permission to access.

The vectors within the repository can be retrieved in one of two ways. The first retrieval method is through a direct synchronous query that is similar to a SQL select statement. The second method allows the requester to register a listener which is notified when data is inserted into the repository that matches the restrictions provided by the listener. This second method allows for entities to have an efficient means of receiving updates without proactively querying and placing unnecessary burden on the repository.

Event Repository The event repository is nearly identical to the affect repository with respect to storage, retrieval, and security. Instead of storing vectors of emotion, the event repository stores information about the user's environment. This environmental information is comprised of two parts: information supplied by the application and information supplied by sensors. In either case, the format of the data varies across applications and sensors as no two applications or sensors can be expected to have the same environment. To support such flexibility we characterize the data in discrete units called *events*, which are defined on a per application or sensor basis. Every event belongs to an ontology whose structure is defined in the developer interface.

Sensor Manager Information about a user's physical state (e.g., heart rate and perspiration) as well as information about the environment (e.g., ambient light, temperature, and noise) can be valuable in estimating the user's emotional state. Obtaining that data is a programming challenge because it involves dealing with a variety of potentially arcane sensors. Accordingly, Koko provides a unified interface for such sensors in the form of the sensor manager. This yields key advantages. First, a single sensor can be

made available to more than one application. Second, sensors can be upgraded transparently to the application and affect model. Third, the overhead of adding sensors is amortized over multiple applications.

The manager requires a plugin for each type of sensor. The plugin is responsible for gathering the sensor's output and processing it. During the processing stage the raw output of the sensor is translated into a sensor event whose elements belong to the event ontology in the developer interface vocabulary. This standardized sensor event is then provided as input to the appropriate affect models.

User Manager The user manager keeps track of the agents within Koko and maintains the system's security policies. The manager also stores some information provided by the user on behalf of the agent. This includes information such as which other agents have access to their affective data and which aspects of that data they are eligible to see. It is the user manager's responsibility to aggregate that information with the system's security policies and provide the resulting security restrictions to the sensor manager and the affect repository. Privacy policies are crucial for such applications, but their details lie outside the scope of this paper.

Developer Interface Vocabulary Koko provides a vocabulary through which the application developer interacts with Koko. The vocabulary consists of two ontologies, one for describing affective states and another for describing the environment. The ontologies are encoded in OWL (Web Ontology Language). The ontologies are designed to be expandable to ensure that they can meet the needs of new models and applications.

The *emotion ontology* describes the structure of an affective state and provides a set of affective states that adhere to that structure. Koko's emotion ontology provides by default are the 24 emotional states proposed by Elliot [2]. Those emotions include states such as *joy*, *hope*, *fear*, and *disappointment*.

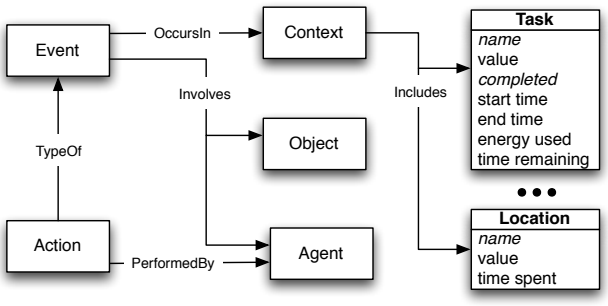


Fig. 4. Event ontology example

The *event ontology* can be conceptualized in two parts, event definitions and events. An event definition is used by applications and sensors to inform Koko of the type of data that they will be sending. The event definition is constructed by selecting terms from the ontology that apply to the application, resulting in a potentially unique subset of the original ontology. Using the definition as a template an application or sensor generates an event that conforms to the definition. This event then represents the state

of the application or sensor at a given moment. When the event arrives at the affect model it can then be decomposed using the agreed upon event definition.

Koko comes preloaded with an event ontology (partially shown in Figure 4) that supports common contextual elements such as time, location, and interaction with application objects.

Consider an example of a user seeing a snake. To describe this for Koko you would create an event *seeing*, which involves an object *snake*. The context is often extremely important. For example, the user's emotional response could be quite different depending on whether the location was in a *zoo* or the user's *home*. Therefore, the application developer should identify and describe the appropriate events (including objects) and context (here, the user's location).

3.2 The Architecture Formally

Now that we have laid the groundwork, we describe the Koko architecture in more formal terms from the perspective of an application developer. The description is divided into two segments, with the first describing the runtime interface and the second describing the configuration interface. Our motivation in presenting these interfaces conceptually and formally is to make the architecture *open* in the sense of specifying the interfaces declaratively and leaving the components to be implemented to satisfy those interfaces.

Application Runtime Interface The application runtime interface is broken into two discrete units, namely, *event processing* and *querying*. Before we look at each unit individually, it is important to note that the contents of the described events and affect vectors are dependent on the application's initial configuration, which is discussed at the end of this section.

Application Event Processing. The express purpose of the application event interface is to provide Koko with information regarding the application's environment. During configuration a developer defines the application's environment via the event ontology in the developer interface. Using the ontology, snapshots of the application's environment are then encoded as events which are passed into Koko for processing. The formal description of this interaction is defined as follows.

$$\mathsf{userID} \times \mathsf{applicationID} \times \mathsf{applicationEvent} \mapsto \bot \tag{1}$$

Upon receipt Koko stores the event in the agent's event repository, where it is available for retrieval by the appropriate affect model. This data combined with the additional data provided by external sensors provides the affect model with a complete picture of the user's environment.

Application Queries. Applications are able to query for and retrieve two types of vectors from Koko. The first is an application-specific affect vector and the second is a user-specific mood vector, both of which are modeled using the developer interface's emotion ontology. The difference between the two vectors is that the entries in the affect vector are dependent upon the set of emotions chosen by the application when it is

configured, while the mood vector's entries are an aggregation of all emotions modeled for a particular user.

When the environment changes, via application or sensor events, the principles of appraisal theory dictate that an appraisal be performed and a new affect vector computed. The resulting vector is then stored in the affect repository. The affect repository exposes that vector to an application via two interfaces. Formally,

$$\mathsf{userID} \times \mathsf{applicationID} \mapsto \mathsf{affectVector} \tag{2}$$

$$\mathsf{userID} \times \mathsf{applicationID} \times \mathsf{timeRange} \mapsto \mathsf{affectVector(s)} \tag{3}$$

Additionally, an application can pass in a contemplated event and retrieve an affect vector based on the current state of the model. The provided event is not stored in the event repository and does not update the state of the model. This enables the application to compare potential events and select the one expected to elicit the best emotional response. The interface is formally defined as follows.

$$\mathsf{userID} \times \mathsf{applicationID} \times \mathsf{predictedEvent} \mapsto \mathsf{affectVector} \tag{4}$$

Mood vectors, unlike affect vectors, aggregate emotions across applications. As such, a user's mood is relevant across all applications. Suppose a user is playing a game that is frustrating them and the game's affect model recognizes this. The user's other affect-enabled applications can benefit from the knowledge that the user is frustrated even if they can't infer that it is from a particular game. Such mood sharing is natural in Koko because it maintains the user's mood and can supply it to any application. The following formalizes the above mechanism for retrieving the mood vector.

$$\mathsf{userID} \mapsto \mathsf{moodVector} \tag{5}$$

Application Configuration Interface Properly configuring an application is key because its inputs and outputs are vital to all of the application interfaces within Koko. In order to perform the configuration the developer must gather key pieces of information and then supply that information Koko using the following interface:

$$\mathsf{affectiveStates} \times \mathsf{eventDefinitions} \times \mathsf{sensorIDs} \times \mathsf{modelID} \mapsto \mathsf{applicationID} \tag{6}$$

The *affectiveStates* are the set of states (drawn from the emotion ontology) that the application wishes to model. The *eventDefinitions* describe the structure (created using the event ontology) of all necessary application events. The developer can encode the entire application state using the ontology, but this is often not practical for large applications. Therefore, the developer must select the details about the application's environment that are relevant to the emotions they are attempting to model. For example, the time the user has spent on a current task will most likely effect their emotional status, where as the time until the application needs to garbage collect its data structures is most likely irrelevant. The *sensorIDs* and *modelID* both have trivial explanations. Koko maintains a listing of both the available sensors and affect models, which are accessible by their unique identifiers. The developer must simply select the appropriate sensors and affect model and record their identifiers.

Furthermore, Koko enables affect models to perform online, supervised learning by classifying events via a set of emotions. Applications can query the user directly for their emotional state and then subsequently pass that information to Koko. In general, many applications have well-defined places where they can measure the user's responses in a natural manner, thereby enabling online learning of affective state. Applications that do exercise the learning interface benefit from improved accuracy in the affect model. The formal definition of this interface is as follows. Notice there is no output because this interface is used only to update Koko's data structures not to retrieve information.

$$\mathsf{userID} \times \mathsf{applicationEvent} \times \mathsf{emotionClassifier} \mapsto \bot \tag{7}$$

4 Evaluation

Our evaluation mirrors our claimed contributions, namely, the Koko architecture. First, we evaluate the contributions of the architecture. Subsequently, we demonstrate the usefulness of the architecture with case studies of both single and multiplayer games.

4.1 Architecture Evaluation

A software architecture is motivated not by functionality but by so-called "ilities" or nonfunctional properties [3]. The properties of interest here—reusability, extensibility, and maintainability—pertain to gains in developer productivity over the existing monolithic approach. In addition, by separating and encapsulating affect models, Koko enables sharing affective data among applications, thereby enhancing user experience. Thus we consider the following criteria.

Reusability. Koko promotes the reuse of affect models and sensors. By abstracting sensors via a standard interface, Koko shields model designers from the details of how to access various sensors and concentrate instead on the output they produce. Likewise, application developers can use any installed affect model.

Maintainability. Koko facilitates maintenance by separating the application, affect model, and sensors. Koko supports upgrading the models and sensors without changes to the components that use them. For example, if a more accurate sensor has been released you could simply unregister the old sensor and register the new sensor using the old sensor's identifier. Any model using that sensor would begin to receive more accurate data. Likewise, a new affect model may replace an older model in a manner that is transparent to all applications using the original model.

Extensibility. Koko specifies generic interfaces for applications to interact with affect models and sensors as well as providing a set of implemented sensors and models. New sensors and models can be readily installed as long as they respect the specified interfaces.

User Experience. Koko promotes sharing at two levels: *cross-application* or intra-agent and *cross-user* or inter-agent communication. For a social or multi-player application Koko enables users or rather their agents to exchange affective states. Koko also provides a basis for applications—even those authored by different developers—to share information about a common user. An application may query for the mood of its user. Thus, when the mood of a user changes due to an application or otherwise, this change becomes accessible to all applications.

4.2 Case Studies

In this section we present two case studies that demonstrate Koko in operation. The first study is the creation of a new social application, called booST, which illustrates the social and multiplayer aspects of Koko. The second study is the re-creation of the affect-enabled Treasure Hunt game, which allows us to illustrate the differences between the Koko and CARE architectures.

booST The subject of our first case study is a social, physical health application with affective capabilities, called booST. To operate, booST requires a mobile phone running Google's Android mobile operating system that is equipped with a GPS sensor. Notice, that while booST does take advantage of the fact that the sensor and application run on the same device this is not a restriction that is imposed by Koko.

The purpose of booST is to promote positive physical behavior in young adults by enhancing a social network with affective capabilities and interactive activities. As such, booST utilizes the OpenSocial platform to provide support for typical social functions such as maintaining a profile, managing a social circle, and sending and receiving messages. Where booST departs from traditional social applications is in the use of *energy levels* and *emotional status*.

Each user is assigned an *energy level* that is computed using simple heuristics from data retrieved from the GPS. Additionally, each user is assigned an *emotional status* generated from the affect vectors retrieved from Koko. The emotional status is represented as a number ranging from 1 to 10. The higher the number the happier the individual is reported to be. A user's energy level and emotional status are made available to both the user and members of the user's social circle.

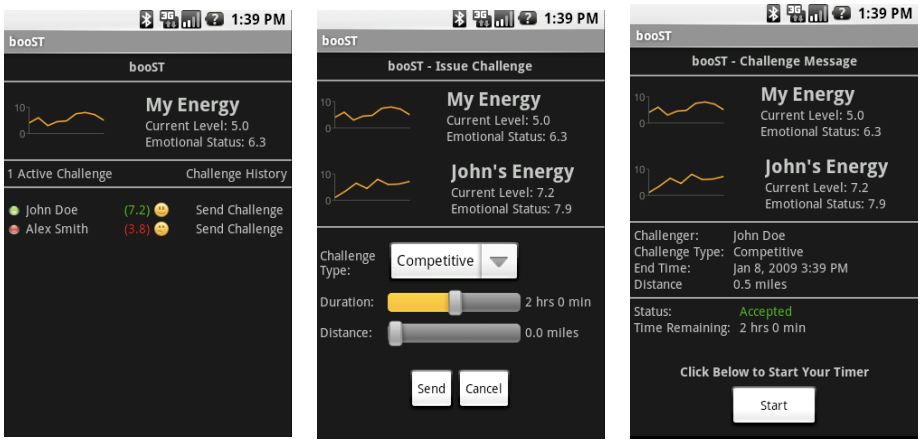


Fig. 5. booST buddy list and activities screenshots

To promote positive physical behavior, booST supports interactive physical activities among members of a user's social circle. The activities are classified as either competitive or cooperative. Both types of activities use the GPS to determine the user's

progress toward achieving the activities goal. The difference between a competitive activity and a cooperative activity is that in a competitive activity the user to first reach the goal is the winner, whereas in a cooperative activity both parties must reach the goal in order for them to win.

Koko's primary role in booST is to maintain the affect model that is used to generate the user's emotional status. The application provides the data about its environment, which in the case of booST is the user's social interactions and their participation in the booST activities. Koko passes this information to the appropriate user agent who processes the data and returns the appropriate emotional status. Further, Koko enables the exchange of affective state between members of a user's social circle. This interaction can been seen in Figure 5 in the emoticons next to the name of a buddy. The affective data shared among members of a social circle is also used to provide additional information to the affect model. For instance, if all the members of a user's social circle are sad then their state will have an effect on the user's emotional status.

Treasure Hunt We showed above how a new application, such as booST, can be built using Koko. Now we show how Koko can be used to retrofit an existing affective application, resulting in a more efficient and flexible application.

Treasure Hunt (TH) is a educational game which demonstrates the CARE model [9] wherein the user controls a character to carry out some pedagogical tasks in a virtual world. TH appraises the emotional state of the user based on a combination of (1) application-specific information such as location in the virtual world, user's objective, and time spent on the task and (2) data from physiological sensors providing information on the user's heart rate and skin conductivity. TH conducts an appraisal every time the state changes and produces one of six perceived emotional states is output.

To reconstruct TH using Koko, we first abstract out the sensors. The corresponding sensor code is eliminated from TH code-base because the sensors are not needed by the application logic. After the sensors have been abstracted we select the six emotional states from the emotion ontology that match those already used in TH. The final step is to encode the structure of the application's state information into application events. This basic format of the state information is already defined as the original TH logs the information for its internal affect model.

Both CARE and Koko models can be thought of as having three states, as outlined in Table 1. The difference between the two architectures is how they choose to perform those steps. For example, in CARE's version of TH the environmental data and emotional classifiers are written to files. The data is then processed offline and the resulting affect model is injected into the application allowing TH to produce the emotional probabilities.

Table 1. Three states of CARE models

#	Description
1	Gather environmental data and emotional classifiers
2	Perform supervised ML techniques on the data and classifiers
3	Given environmental data produce emotional probabilities

Koko improves on the CARE architecture by eliminating the need for the application to keep record of its environmental data and also by performing the learning online. As a result, the Koko-based TH can move fluidly from State 1 to State 3 and back again. For example, if TH is using Koko then it can smoothly transition from providing Koko with learning data, to querying for emotional probabilities, and return to providing learning data. In CARE this sequence of transitions while possible, results in an application restart to inject the new affect model.

This improvement can be compared to an iterative software engineering approach versus the more rigid waterfall approach. CARE corresponds to the waterfall approach, in that it makes the assumption that you have all the information required to complete a stage of the process at the time you enter that stage. If the data in a previous stage changes you can return to a previous state but at a high cost. Koko follows a more iterative approach by removing the assumption that all the information will be available and ensuring that transitions to a previous state incur no additional cost. As a result, Koko yields a more efficient architecture than CARE.

5 Discussion

This paper shows how software architecture principles can be used to specify a middleware for social affective computing. If affective computing is to have the practical impact that many hope it will, advances in software architecture are crucial. Further, the vocabulary of events and context attributes introduced here can form the basis of a standard approach for building and hosting affective applications.

Game Integration. Due to the social and multiplayer nature of Koko, it cannot be contained within a traditional gaming engine. However, Koko can interoperate with gaming engines in a loosely coupled manner. To incorporate Koko into an existing game engine API, the engine can simply provide a façade (wrapper) around the Koko API. The façade is responsible for maintaining a connection to the Koko service and marshalling or unmarshalling objects from the engine's data structures to those supported by the Koko. Currently, Koko has service endpoints that support the communication of data structures encoded as Java objects, XML documents, and JSON objects.

Affect Modeling. Existing appraisal theory applications are developed in a monolithic manner [2] that tightly couples application and model. As a notable exception, EMA provides a domain-independent framework that separates the model from the application. Whereas EMA focuses on modeling virtual characters in a specific application, Koko models human emotion in a manner that can cross application boundaries.

We adopt appraisal theory due to the growing number of applications developed using that theory. Our approach can also be applied to other theories such as Affective Dimensions [13], whose models have inputs and outputs similar to that of an appraisal model. Likewise, we have adopted Elliot's set of emotions because of its pervasiveness throughout the affective community. Its selection does not imply that Koko is bound to any particular emotion ontology. Therefore, as the field of affective computing progresses and more well-suited ontologies are developed, they can be incorporated into the architecture.

Enhanced Social Networking. Human interactions rely upon social intelligence [4]. Social intelligence keys not only on words written or spoken, but also on emotional cues provided by the sender. Koko provides a means to build social applications that can naturally convey such emotional cues, which existing online social network tools mostly disregard. For example, an advanced version of booST could use affective data to create an avatar of the sender and have that avatar exhibit emotions consistent with the sender's affective state.

Future Work. Koko opens up promising areas for future research. In particular, we would like to further study the challenges of sharing affective data between applications and users. In particular we are interested in exploring the types of communication that can occur between affective agents.

Demonstration. Both Koko and booST have been implemented and will be demonstrated at AAMAS 2009. The demonstration will feature a detailed look into the application interfaces of Koko architecture as well as a live presentation of booST.

References

1. J. Dias and A. Paiva. Feeling and reasoning: A computational model for emotional characters. In *Progress in Artificial Intelligence, LNCS*, 3808/2005:127–140. 2005.
2. C. Elliott. *The Affective Reasoner: A Process Model of Emotions in a Multi-agent System*. PhD, Northwestern, 1992.
3. R. E. Filman, S. Barrett, D. D. Lee, and T. Linden. Inserting ilities by controlling communications. *Communications of the ACM*, 45(1):116–122, 2002.
4. D. Goleman. *Social Intelligence: The New Science of Human Relationships*. Bantam Books, 2006.
5. J. Gratch, W. Mao, and S. Marsella. *Modeling Social Emotions and Social Attributions*. Cambridge University Press, 2006.
6. J. Gratch and S. Marsella. Fight the way you train: The role and limits of emotions in training for combat. *Brown Journal of World Affairs*, Vol X (1), Summer/Fall:63–76, 2003.
7. J. Gratch and S. Marsella. A domain-independent framework for modeling emotion. *Journal of Cognitive Systems Research*, 5-4:269–306, 2004.
8. R. S. Lazarus. *Emotion and Adaptation*. Oxford University Press, 1991.
9. S. McQuiggan, S. Lee, and J. Lester. Predicting user physiological response for interactive environments: An inductive approach. *AIIDE*, 2006.
10. S. McQuiggan and J. Lester. Modeling and evaluating empathy in embodied companion agents. *International Journal of Human-Computer Studies*, 65(4), 2007.
11. A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.
12. A. Ortony, G. L. Clore, and A. Collins. *The Cognitive Structure of Emotions*. Cambridge University Press, 1988.
13. R. W. Picard. *Affective Computing*. MIT Press, 1997.
14. M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, 1996.
15. C. Smith and R. Lazarus. Emotion and adaptation. In L. A. Pervin and O. P. John, editors, *Handbook of Personality: Theory and Research*, pages 609–637. Guilford, 1990.
16. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

Pogamut 3 Can Assist Developers in Building AI for Their Videogame Agents

Jakub Gemrot, Rudolf Kadlec, Michal Bída, Ondřej Burkert, Radek Píbil, Jan Havlíček, Lukáš Zemčák, Juraj Šimlovič, Radim Vansa, Michal Štolba, Cyril Brom

Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 2/25, Prague, Czech Republic
{jakub.gemrot, rudolf.kadlec}@gmail.com, brom@ksvi.mff.cuni.cz

Abstract. Many research projects oriented on control mechanisms of virtual agents in videogames have emerged in last years. However, this boost has not been accompanied with the emergence of toolkits supporting development of these projects, which slows down the progress of the field. Here, we present Pogamut 3, an open source platform for rapid development of behaviour of virtual agents embodied in a 3D environment of the Unreal Tournament 2004 videogame. Pogamut 3 is tailored to support research as well as educational projects.

Keywords: Virtual agents, behaviour, control mechanisms, Unreal Tournament, 3D environment, agent development toolkit.

1 Introduction

The development of control mechanisms of videogame agents (that is, their "artificial intelligence") is hard by itself. The life of developers is typically complicated further by the fact that they have to solve many tedious technical issues that are out of their main scope. For instance, they have to integrate their agent with a particular 3D environment, build at least a simple debugging tool or write a code for low-level movement of the agent. This list continues for pages and developers address its items again and again, a waste of energy.

For last three years, we have been developing the Pogamut 3 toolkit, which provides general solutions for many of these issues, allowing developers to focus on their main goals. Importantly, Pogamut 3 is designed not only for advanced researchers, but also for newcomers. Pogamut 3 can help them to build their first virtual agents, a feature which makes it applicable as a toolkit for training in university and high-school courses [5].

The purpose of this paper is to review main features of Pogamut 3, its architecture, and work in progress, providing supplementary information for the demonstration of Pogamut 3 at the AAMAS-09 Workshop on Agents for Games and Simulations.

2 Features of Pogamut 3

The agent development cycle can be conceived as having five stages: (1) inventing the agent, (2) implementing the agent, (3) debugging the implemented agent, (4) tuning the parameters of the agent, and (5) validating the agent through series of experiments. Individual features of Pogamut 3 were purposely designed to provide the support during the latter four stages.

The features of Pogamut 3 includes:

1) a binding to the virtual world of the Unreal Tournament 2004 videogame (UT2004) [7],
2) an integrated development environment (IDE) with a support for debugging,
3) a library with sensory-motor primitives, path-finding algorithms, and a support for shooting behaviour and weapon handling,
4) a connection to the POSH [6], which is a reactive planner for controlling behaviour of agents, and a visual editor of POSH plans (the editor is not included in the current release yet),
5) a support for running experiments, including distributed experiments run on a GRID.

Integral part of the Pogamut 3 toolkit is the homepage [3] where one can download the installer, see tutorial videos, learn more about the platform, and get support on forums.

3 Details of the Features and Technicalities

The Pogamut 3 toolkit integrates five main components: *UT2004*, *GameBots2004*, the *GaviaLib library*, the *Pogamut agent*, and the *IDE* (Fig. 1). These components implement the abovementioned features, enabling a user to create, debug, and evaluate his or her agents conveniently.

UT2004 is a well known action videogame with so-called “partly opened” code. In particular, the game features UnrealScript, a native scripting language developed by the authors of UT2004, in which a substantial part of the game is programmed: almost everything except of the graphical and the physical engine. The code in UnrealScript can be modified by users. Additionally, the game features a lot of pre-built objects and maps, and also a level editor. These two points allow users to create new game extensions and blend them with the original game content. In the context of the Pogamut 3 toolkit, the game provides the virtual environment for running agents.

One can implement agents directly in UnrealScript; however, it is often advantageous, both for educational as well as experimental reasons, to develop the agents using another, external, mechanism. This means that one has to create a binding to the UT2004. Several years ago, Adobbati et al. developed a generic binding called GameBots [1] for this purpose, which were used by many from then. The original GameBots worked with UT99. GameBots2004 is our extension of the original GameBots.

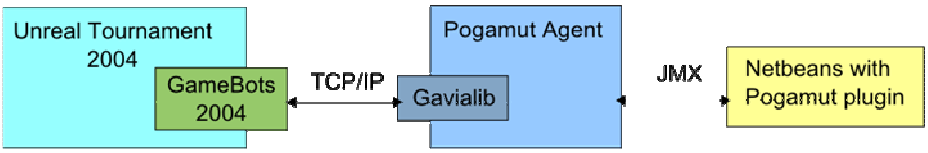


Fig. 1. Pogamut architecture.

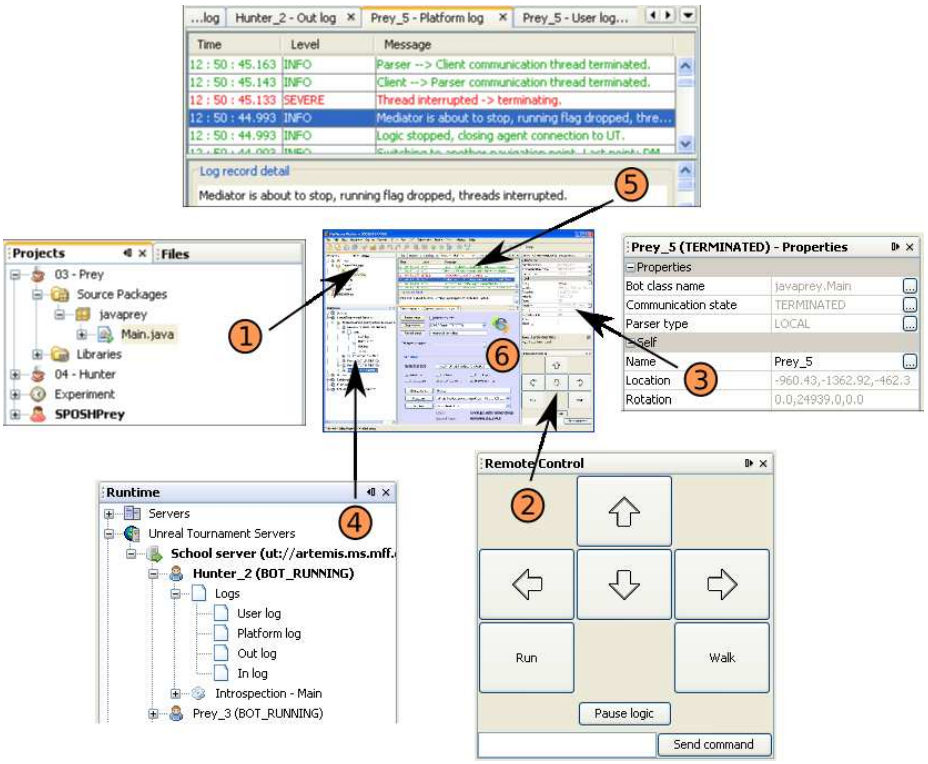


Fig. 2. Pogamut 3 NetBeans plugin (6) and its parts (1-5). The parts are described in the text.

GameBots2004 exports information about the game environment through TCP/IP text based protocol allowing a user to connect to the UT in the client-server manner. This means that the user implements an agent's control mechanisms in the client and uses GameBots2004 as the server. Note, that GameBots2004 can be used separately without other components of Pogamut 3.

The GaviaLib library is a general purpose Java library for connecting agents to almost any virtual environment. Only mild assumptions have been imposed upon virtual environments; in a nutshell, an environment has to work with objects and be able to provide information in an event-based manner. GaviaLib handles communication with the environment, manages object identities, notifies the agent of

important object events (e.g. object appeared/disappeared/was changed) and supports out of the box remote control of agents through JMX [11], which is a standard Java protocol for remote instrumentation of programs. The GaviaLib's layered architecture makes it possible to use different means of communication with the virtual world. A GaviaLib agent can be connected through plain text TCP/IP protocols like GameBots2004 are, through CORBA or, in the case of Java environments, it can reside in the same Java Virtual Machine as the world simulator and communicate directly through method calls. Presently, the only binding provided with GaviaLib is the UT2004 binding, which employs Gamebots2004, as detailed below.

The Pogamut agent is a set of Java classes and interfaces built on top of the GaviaLib library. It adds UT2004 specific features, such as UT2004's sensory-motor primitives, the navigation using the UT2004 system of way-points and auxiliary classes providing information about the game rules. Developers can program UT2004 agents using the classes of the Pogamut agent; they can implement an agent's control mechanism directly in Java or use the reactive planner POSH. POSH has been developed as an independent action selection planner by Joanna Bryson [6] and Pogamut 3 exploits it as a plug-in. This planner enforces the design that clearly separates low level actions and sensory primitives (coded in Java or Python) from a high level hierarchical plan of an agent's behaviour.

The IDE is developed as a NetBeans plugin [12] (Fig. 2). It can communicate with running agents via JMX. The IDE offers a designer many features, such as empty agent project templates, example agents (e.g. hunter and prey agents or a khepera-like agent) (Fig. 2, Window 1), management of UT2004 servers, list of running agents (Fig. 2, Window 4), introspection of variables of agents and a quick access to their general properties (Fig. 2, Window 3), the step-by-step debugging, log viewers (Fig. 2, Window 5), and the manual controller of positions of agents (Fig. 2, Window 2). However, the Pogamut agents can be developed without this plug-in in any other Java IDE.

In general, the components of the Pogamut 3 toolkit can be used *per partes*. As already said, one can use GameBots2004 independently as well as parts of the GaviaLib library or the IDE.

4 Conclusion and Prospect Work in Progress

The Pogamut 3 toolkit provides many general solutions for videogame agents developers. The important facet of Pogamut 3 is that it has been designed not only for advanced researchers, but also for newcomers. Its applicability for beginners has been proved by using it as a toolkit for training in a university and a high-school course on programming virtual agents [5].

The current release of Pogamut has also several limitations. Its integration with UT2004 and the support provided by GaviaLib classes make it most suitable for gaming AI projects; this suits some, but is unsuitable for others. We are now extending Pogamut 3 with features that can be utilised in the context of virtual storytelling and computational cognitive psychology. Most notably, Pogamut 3 is being extended by a storytelling manager, an emotion model based on the ALMA

model [8], a support for gestures coded in BML [13], and the ACT-R cognitive architecture [2] for controlling agents that will become an alternative to the POSH planner or the plain Java. We are also working on an educational scenario that will utilise these technologies.

To prove the flexibility of the GaviaLib library, we would like to connect it to a new environment. We are considering the Defcon game [10] or Virtual Battle Space 2 (VBS2) [4]. The latter features the industry standard HLA interface (IEEE 1516) [9]. The creation of a general GaviaLib–HLA bridge would make it possible to connect many more environments with a minimal effort.

Acknowledgments. This work was partially supported by the project CZ.2.17/3.1.00/31162 that is financed by the European Social Fund, the Budget of the Czech Republic and the Municipality of Prague. It was also partially supported by the Program "Information Society" under project 1ET100300517, by the grant 201/09/H057, and by the research project MSM0021620838 of the Ministry of Education of the Czech Republic.

References

1. Adobbati, R., Marshall, A. N., Scholer, A., and Tejada, S.: Gamebots: A 3d virtual world test-bed for multi-agent research. In: Proc. of the 2nd Int. Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, Canada (2001)
2. Anderson, J. R.: How can the human mind occur in the physical universe? Oxford University Press (2007)
3. "Artificial Minds for Intelligent Systems" Research Group: The Pogamut platform. Charles University in Prague. http://artemis.ms.mff.cuni.cz/pogamut [4.3.2009]
4. Bohemia Interactive: Virtual Battle Space 2. http://virtualbattlespace.vbs2.com/ [4.3.2009]
5. Brom, C., Gemrot, J., Burkert, O., Kadlec, R., Bída, M.: 3D Immersion in Virtual Agents Education. In: Proc. of 1st Joint Int. Conference on Interactive Digital Storytelling ICIDS 2008, LNCS 5334, Erfurt, Germany, Berlin: Springer-Verlag. (2008) 59-70
6. Bryson, J. J.: Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents. PhD thesis, Massachusetts Institute of Technology (2001)
7. Epic Games: Unreal Tournament 2004. (2004) URL: http://www.unreal.com [4.3.2009]
8. Gebhard, P.: ALMA – A Layered Model of Affect. In: Proc. of the 4th Int. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05), Utrecht. (2005) 29– 36
9. IEEE: IEEE 1516, High Level Architecture (HLA). http://www.ieee.org (2001)
10. Introversion: Defcon. http://www.introversion.co.uk/defcon/ [4.3.2009]
11. Sun Microsystems: JMX. http://java.sun.com/javase/6/docs/technotes/guides/jmx/ [4.3.2009]
12. Sun Microsystems, Netbeans IDE. http://www.netbeans.org [4.3.2009]
13. Vilhjalmsson, H., Cantelmo, N, Cassell, J., Chafai, N., Kipp, M., Kopp, S., et al.: The Behavior Markup Language: Recent Developments and Challenges. In: Proc. of 7th Conference on Intelligent Virtual Agents (IVA 07), LNAI 4722, Springer-Verlag (2007) 99-111

Agent-Based Aircraft Control Strategies in a Simulated Environment

Daniel Castro Silva[1], Ricardo Silva[1], Luís Paulo Reis[1], and Eugénio Oliveira[1]

FEUP-DEI / LIACC, Rua Dr. Roberto Frias s/n 4200-465 Porto, Portugal,
(dcs, ee99208, lpreis, eco)@fe.up.pt

Abstract. In the recent years, game engines have been increasingly used as a basis for agent-based applications and multiagent systems, alongside traditional, more dedicated, simulation platforms. In this paper, we test and compare different control strategies for specified high-level maneuvers in aircraft within a gaming simulation environment, each aircraft represented by an independent agent, focusing on communication necessities and maneuver effectiveness. An overview of the simulation environment's capabilities, with focus on the structured experiences system, shows the potentials of this platform as a basis for the more comprehensive goals of our project, allowing for the definition and execution of cooperative missions, such as surveillance and search & rescue operations.

1 Introduction

Simulation tools are widely used in several fields of research, providing researchers with the means to develop their work in a cost- and time-effective manner. These and other advantages of simulation environments have led to a significant growth in their general use by the scientific community, as well as in several business areas, supporting decision-making processes, planning, training, and many other tasks. Alongside professional simulation tools, designed specifically to the research purpose they are used in, game engines and gaming tools are being adopted more and more by the scientific community as serious tools in their work. Serious gaming explores the use of games and game-like applications in training or learning, or using them as decision support tools, to improve business processes. Not only the entertainment organizations use of these new applications, but also governmental and military organizations, mainly for training purposes, be it in the leadership and management areas, or in a more technical approach, using primarily simulation tools [1].

One field of research where simulator tools are extensively used is aviation. In fact, simulators used in this area range from multi-million dollar full hardware and software simulators used to train professional pilots to freely available flight simulators, which have been used mainly for entertainment purposes and by aviation enthusiasts. As an example, one of the world's most famous simulators, NASA's VMS (Vertical Motion Simulator), is a full simulator capable of six degrees-of-freedom movements, housed in a ten-story building in Ames

Research Center, California [2][3]. This simulator is capable of providing different simulation experiences, thanks to the ICAB (Interchangeable Cab) feature, which allows a modular interior of the simulation cabin, presenting the pilot with different cockpit interfaces. An impressive 1300 variables can be retrieved from the simulator, allowing for thorough data analysis of simulation sessions. This simulator has been used for pilot and astronaut training but also in cooperation with other entities [4]. On the other end, flight simulators such as FlightGear (an open-source flight simulator) are available for download at no cost or at a reduced price. Some of these low-cost flight simulators have, in the latest years, achieved a high level of realism, both in the simulation of aircraft systems, kinematics and weather conditions, and also in the visualization of all these aspects with a realistic and detailed terrain. Some of them have been certified by the FAA (Federal Aviation Administration) for pilot training, when used in conjunction with the proper hardware simulator. For instance, the X-Plane simulator is used by some simulator companies, such as Simtrain as the software basis for simulation [5].

In the recent years, simulators in several areas have evolved into more complex systems, and in many cases a multi-agent system approach has been adopted. In these systems, the participating entities are represented by independent agents, which in turn communicate with each other, coordinating information and actions. In the aeronautical area, this approach has also been adopted, usually with each agent representing one aircraft (or even several agents representing the several systems of one single aircraft, if a more detailed simulation of a specific system is required). Such systems are used to develop new strategies to solve problems such as collision avoidance among UAVs (Unmanned Aerial Vehicles) [6], formation flight or fault tolerance [7], among others. These systems should also address the interaction with human beings, allowing for a real-time awareness of the situation and the possibility to define missions, or even to override decisions made by the agents [8].

The authors' vision is to make use of such a system in order to develop strategies to use in cooperative missions, such as surveillance and search & rescue scenarios. This paper focuses on control strategies for aircraft within the simulation environment, and their effectiveness in terms of both communication necessities and maneuver smoothness and accuracy. Three distinct strategies were tested to control different aircraft, and the experimental results were analyzed as to assess which of the methods is most suited to use in the simulator. As to provide with a more comprehensive context on the project, the following paragraphs summarize the steps that were previously taken.

The main goals of the overall project are to use a simulation platform as a basis for the design and cooperative execution of joint missions. These missions are to be performed by a team of heterogeneous vehicles, including planes, helicopters, land vehicles, and submersible vehicles, capable of communicating with one another, in order to coordinate their actions. The applications of such a system are diverse, including surveillance (forest surveillance that provides an early fire detection system, coastal and border patrol, in order to detect and track

illegal activities, such as smuggling, urban observation that would detect dense traffic patterns, preventing larger traffic jams, and many other applications), reconnaissance and target tracking (especially useful in military operations and law enforcement activities, to provide real-time valuable information about enemy movements, or to follow a fugitive until apprehended by competent entities), aiding in search & rescue operations, and many other applications.

The project's envisioned architecture incorporates the chosen simulator, a number of external agents, each representing one vehicle within the team, as well as some valuable peripheral services, such as a monitoring application, capable of real-time vehicle tracking, as briefly mentioned in the results section, simulation logging that enables mission replay and further analysis, and an interface with external modules, for interaction with real vehicles.

1.1 Simulation Engine

Previous work on this project included a thorough study of simulation platforms, which resulted in the choice of Microsoft Flight Simulator X (FSX) as the platform to be used. This was based on the platform's admirable graphical features, simulation of aircraft dynamics and interaction with the environment, possibility to inject failures in certain aircraft systems and sensors, excellent application programming interface documentation, with hundreds of simulation variables that can be read and written to, hundreds of events that can be sent to the simulator, and many other features (which allow an external application to interact with the simulator in real-time), as well as the available mission system, with its outstanding possibilities.

FSX's programming interface, SimConnect, provides a flexible, powerful and robust client-server communications protocol that allows asynchronous access to hundreds of simulation variables and events. Some of these variables can only be read, but nearly two hundreds can also be written to, several hundred events can be sent to the simulator, and there are several dozens of functions to deal with the weather system, missions, in-game menus, AI objects, communication and data access and manipulation, among other useful features.

Structured experiences, or missions, as they are commonly known, are a feature of FSX that allows regular users to have a different, interactive experience of flight, with measurable goals. FSX includes numerous missions, ranging from tutorials that teach the user how to fly an aircraft, to racing missions, simulating the Red Bull Air Race environment, as well as several transport or rescue operations, among others. The mission system allows for the definition of objects, areas, triggers and actions that can be linked to work together in an orchestrated manner, producing diverse, realistic and complex missions [9].

Microsoft has also already recognized the advantages of simulation in various business areas and the potential of these structured experiences, and is commercializing the engine behind FSX as a new enterprise-oriented product, called ESP [10]. There are numerous applications of this technology, and several high-profile companies have already signed partnerships with Microsoft for the use of ESP, such as FlightSafety International, SAIC, Lockheed Martin and Northrop

Grumman [10] [11]. These companies, as well as others that use this product, will be able to use it to train staff (not only piloting skills, but also management skills for airline companies and airports, for instance), learning, research or decision support - by simulating different scenarios, the most favorable one can be identified, problems and design flaws can be corrected, business processes can be improved.

1.2 Autopilot Systems

Although autopilot systems are usually associated with aircraft, they can be used in any kind of vehicle, including boats, cars, or even missiles. In this paper, however, the focus is on aircraft autopilot systems. There is a variety of autopilots commercially available for small unmanned aircraft, usually comprised of light-weight hardware to connect to the aircraft, and a control station, also often including some software to interact with the system. These autopilots range from simple one-axis controllers to full three-axis systems, including redundant processors, sensor diagnostics and failure tolerance, medium- to long-range communication system and many other useful features when dealing with payloads. Integration and testing of autopilot systems with the aircraft has to be carefully executed, as to calibrate the system to the aircraft in question [12]. Commercial aircraft also feature autopilot systems, which can control the aircraft from takeoff to land without human intervention (these systems are, however, usually only used during the leveled part of the flight, and during landing, when visibility conditions are below certain limits). These full autopilot systems use redundant computers to ensure that control decisions are correct, and provide a more stable flight than human pilots would, lowering fuel consumption at the same time.

The rest of this paper is organized as follows. In the next section, the three approaches that were compared are explained in more detail. Section three presents the settings for the experiments that were conducted and section four presents the results that were attained, comparing the approaches. Finally, in the last section, some conclusions are withdrawn and lines of future work are presented.

2 Control Strategies

In this section, the three devised control strategies are presented and briefly compared. The first approach consists on direct manipulation of the aircraft controls, such as aileron and elevators through a PID controller. The two other approaches use auto-pilot systems. The second one makes use of the auto-pilot system available to actual pilots, delegating the handling of aircraft controls to the system. The third approach uses the AI auto-pilot, used by the simulator to perform automated flights. The differences between these methods and the specifics of each one are detailed in the following subsections.

2.1 PID Controller

The first approach to controlling an aircraft within the simulation environment is to recreate the actions of a pilot when attempting to follow a certain route. This was accomplished by direct manipulation of the main controls of the aircraft, namely the throttle, aileron and elevator. The throttle control is used to control the speed of the aircraft. The elevator control is used to control the pitch angle of the aircraft (sometimes referred to as angle of attack, pitch measures the nose up or down angle of an aircraft), and, when used in conjunction with the throttle control, it allows to control the altitude of the aircraft - in order to climb, the elevator must be set to a positive value, and the throttle should be set to full throttle; in order to descend, the elevator must be set to a negative value, and the throttle should be set to idle. The aileron control is used to control the bank angle of the aircraft (also called roll, it is the angle of rotation of the plane about its longitudinal axis). In order to make a turn, the aircraft rolls toward the inner part of the desired turn. By using these controls together, higher-level maneuvers are executed.

In order to handle these aircraft controls, a PID controller was used. A PID controller is a generic control loop feedback mechanism that attempts to correct the error between a measured variable and the desired value by calculating a corrective action that can adjust the process accordingly. It is a well-known controller from the control theory field, and has been used successfully in the industry for many years. Its numeric implementation consists on the evaluation of Equation (1), where e is the difference between the reference value and the feedback measured value, τ is the time in the past contributing to the integral response and K_p, K_i and K_d are the proportional, integral and derivative gains, respectively. The proportional term adjusts the output signal in direct proportion to the error, the integral term is proportional to both magnitude and duration of the error, and the derivative term measures the approximate rate of change of the error. Tuning the gain factors of the PID controller is an important step, to assure optimal values for the desired control response [13].

$$output\,(t) = k_p e\,(t) + k_i \int_0^t e\,(\tau)\,d\tau + k_d \frac{de}{dt} \tag{1}$$

For this approach, the control agent communicates with the simulated aircraft at a given rate, in order to send the current values, calculated with the PID controller. Ideally, the communication would be uninterrupted, since continuous adjustments to the aircraft controls should be made in order to maintain the desired flights settings. However, the simulator only sends and accepts values once per simulation cycle, and to send data at a higher rate would be a waste of processing time. Moreover, one has to account for communication latencies that make it difficult to receive data from a given simulation cycle and to send the corrective values that reach the simulator in time for the following cycle. There is a simulation variable that is modified by the control agent for each of the three aircraft controls that are manipulated by the controller, and other variables that can be read in order to assess the current position, speed and attitude of the

aircraft, used in the calculations as mentioned above. In order to make a banked turn with a given radius (for circling maneuvers, for instance), some additional information has to be taken into account. The radius of a banked turn is given by Equation (2), where v is the true airspeed of the aircraft, g is the acceleration due to gravity and θ is the bank angle of the aircraft.

$$R = \frac{v^2}{g \tan \theta} \tag{2}$$

Given Equation (2), one can determine the necessary bank angle to generate a banked turn with a given radius at a given speed.

2.2 Autopilot

In this approach, the autopilot features of aircraft are used. Autopilot systems assume various forms, ranging from simple wing-levelers to complete systems, capable of controlling an aircraft from takeoff to land. Most modern aircraft feature altitude, heading and speed controls, allowing the pilot to set the desired values for each feature. The altitude control manipulates the elevator and throttle (if auto-throttle is available) in order to reach and maintain a certain altitude. The heading control manipulates the aileron in order to make a banked turn until the desired heading is reached, and the speed control automatically adjusts the throttle to maintain the desired airspeed. This approach uses the available autopilot systems as a pilot would, to adjust the aircraft's course to the desired settings.

There is a straightforward limitation to this approach, which is the necessity for the presence of an autopilot system. In fact, many smaller, older aircraft do not possess an autopilot system, or it is only comprised of the already mentioned wing-levelers, capable only of maintaining a straight level flight (but not to change altitude, heading or speed). The authors, however, feel that this limitation does not impose a major problem, since there are several commercially available autopilot systems for small unmanned aircraft, such as the Piccolo autopilot system, from CloudCap Technology [14], which are usually used for the type of aircraft that are intended to be used in real-life experiments.

With this approach, there are limits that need to be taken into account when planning the maneuvering scenarios. For instance, the autopilot system limits for the bank angle. As previously seen, the bank of a plane influences the radius of the curve, and as such, a minimum radius for turns is calculated based on the aircraft cruise speed and maximum bank angle at the beginning of the simulation, and that minimum value is enforced on maneuvers that require the plane to move in a circular fashion.

Since these systems do not allow for a direct control of the bank angle (some systems have the capability to further limit the maximum bank angle to usually half of that value, to produce smoother flights), a circular path is a bit more complicated to achieve. For circular paths, several points along the turn were calculated, and used as a basis for heading determination. By providing the

aircraft with regular changes in the desired heading, a smooth circular path is achieved. The number of points in the path is directly proportional to the difference between the desired radius and the minimum radius - if the desired radius coincides with the minimum radius at the current aircraft speed, there is no need to use more than two alternating points to update the heading control. As with the PID approach, this one also requires frequent communication with the aircraft, in order to send the desired values for the autopilot system.

2.3 AI Autopilot

The third approach makes use of the AI autopilot within the simulator to control the aircraft. This AI autopilot can be used in every aircraft, independently of the existence of an autopilot system such as the one described in the previous approach. It is used by the simulator to guide the generated air traffic from departure to arrival airports.

As with the previous approach, this autopilot also features limitations, namely the bank angle. When an autopilot system is available, the maximum bank angle is used to calculate the minimum radius for a turn, just as in the previous approach. When no autopilot system is present, the default value of twenty-five degrees is used - this is the most common value for autopilot systems, and experimental activities have demonstrated that this is also the most usual value for this AI autopilot.

This approach has, however, a few more limitations. In this case, all navigation is based on waypoints, which specify not only the latitude/longitude/altitude coordinates of the desired point, but also the desired speed or throttle percentage to be applied. Having this limitation in mind, and similarly to the previous approach, several points have to be calculated and fed to the aircraft for circular paths. This approach, however, does not require frequent communication with the aircraft - all points of a maneuvering sequence can be sent at the same time, in a waypoint list structure. The exception to this behavior is the existence of loops in the path - if a portion of a path is to be repeated until some external event occurs (such as human intervention), the waypoints that represent that loop have to be sent at the beginning of that loop, with a flag indicating that after the last waypoint, it should return to the first; when the loop is to be broken, the remaining waypoint must then be sent, replacing the previous ones.

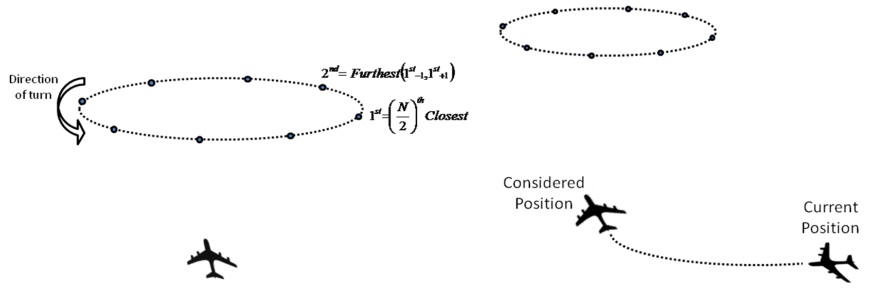
2.4 High-Level Maneuvering and Waypoint Computation

In order to compare these methodologies, some high-level maneuvers were considered, namely the 'go to point' instruction, the circle and the helix maneuvers. Additional instructions could be issued to the aircraft, including desired airspeed, heading or altitude and target vehicle interception. These additional instructions would force the aircraft to accelerate or decelerate to attain the desired airspeed, turn and maintain the desired heading or climb or descend in order to reach and maintain the given altitude. The target interception instruction provides the aircraft with the most probable point of interception with the target vehicle,

if possible (aircraft and target vehicle speed, distance and heading are used to determine if interception is possible), updating the interception point every few seconds. The authors believe the three first instructions to be sufficient to test the efficiency of vehicle control, since others can be achieved by using the first ones or similar method.

The 'go to point' instruction has three logical parameters: latitude, longitude and altitude of the point to be reached. The altitude is perceived as altitude above the mean sea level, and not altitude above ground level. The circle maneuver, also called loiter, is defined by the central point (latitude, longitude and altitude), the radius of the circle and the number of laps, if not to loop indefinitely. The helix is defined by a central line (latitude and altitude), the initial and final altitudes, the radius of the helix and the number of laps. As previously mentioned, the radius for the last two maneuvers is conditioned by the aircraft's maximum bank angle and airspeed.

In order to compute the points used by the two last maneuvers in the two approaches that use autopilot systems, some considerations were made. First, the number of points to consider is directly proportional to the difference between the intended radius and the aircraft minimum turn radius. Second, and most importantly, the order in which those points are presented can deeply influence the path of the aircraft. For a smoother transition, a tangential approach to the virtual circle is preferred, especially when the circle has a radius closer to the minimum turn radius. The following algorithm was used to determine the first point and the order of the remaining points: a list of points is determined for the given radius, ordered as to form a circle; the distances between the aircraft and the points are determined; the first point is the $(N/2)^{th}$ closest point to the aircraft, N being the number of points in the circle; the second point is the point further away from the aircraft, chosen among the two points that are adjoining the first point in the original list of points. The remaining points are determined according to the order in which they were initially determined. Fig. 1(a) illustrates the choice of the first and second points in a hypothetical 8-point circle and subsequent direction of turn.



(a) Waypoint Determination (b) Aircraft Position Adjustment

Fig. 1. Waypoint Determination and Aircraft Position Adjustment

The current aircraft position is used if the circle or helix maneuver is to be executed immediately; if the maneuver is to be queued after other maneuvers, the final point of the previous maneuvers is used. Also, the aircraft position to consider may be adjusted to a position where the aircraft's heading is towards the region of the desired maneuver, considering a turn with the minimum radius at current speed, as illustrated in Fig. 1(b).

3 Experimental Settings

In this section the experimental settings for the conducted experiments are presented. As already mentioned, FSX was the chosen simulator. Different experimental conditions were recreated for each control approach, with three main controllable variables - aircraft type, weather conditions and maneuver sequence.

Different aircraft types were used to assess how generic the approach was, or if it had limitations, namely regarding the size or type of aircraft. A total of three distinct aircraft were used in the experiments, which include a Piper J-3 Cub (a single-engine two-seater light aircraft, with a wingspan of 10.6 meters and a maximum speed of 74 knots), a Beechcraft Baron 58 (a twin-engine six-seater with a wingspan of 11.5 meters and a cruise speed of 200 knots) and a Bombardier Learjet 45 (a twin-engine-jet nine-seater with a wingspan of 14.6 meters and a cruise speed of 464 knots). Since this project is to be used with small to medium-sized aircraft, the authors felt no need to conduct the experiments with larger aircraft. Although the Piper J-3 Cub does not possess an auto-pilot system, the authors think it was important to include this aircraft, as to test the performance of the two other approaches on a smaller, slower plane.

Different weather conditions were used to test the control performance under adverse conditions, and how flexible the approach was to the existence of uncontrollable and unpredictable external factors. The experiments were conducted under two different weather conditions - fair weather and gray and rainy. The authors felt that harsher weather conditions would not be necessary, since the intended aircraft would not fly under more adverse weather conditions.

Different high-level maneuvering sequences were also tested, as to assess how the approach would handle maneuver transition and how smooth the final flight would be. Basic high-level components were used to compose the three maneuvering scenarios used in the experiments. These high-level instructions include a 'go to point' command, indicating that the aircraft should pass through a given latitude/longitude/altitude point; circle, indicating that the aircraft should circle a given point with a given radius, either one time or continuously; helix motion, either climbing or descending from one initial altitude to a final altitude though a series of turns around a central point.

All experiments begin with the selected aircraft stopped in the end of a runway (34R) of the selected airport - Seattle-Tacoma International - with idled engines, facing the runway. The first command of the maneuvering sequence is always a 'go to point' command, with a point directly in front of the aircraft, approximately 500 feet above the runway, in order to assure both a smooth

takeoff and that the aircraft attains its cruise speed. Slight variations in the scenarios were introduced for the three aircraft, due to their different cruising speeds. As already mentioned in the section above, the radius of a banked turn is proportional to the square of the velocity of the aircraft, and hence, a larger radius was used for faster aircraft in the circle and helix commands. As a result, other commands were also modified in terms of longitude/latitude, in order to accommodate for the larger radius of these commands, in an attempt to maintain the overall proportions between the several control points.

4 Results

In this section, the results that were attained through the experiments that were conducted, and as described in the previous section, are presented. For a more structured arrangement, the results are divided into two categories - communication necessities and maneuver effectiveness.

4.1 Communication Necessities

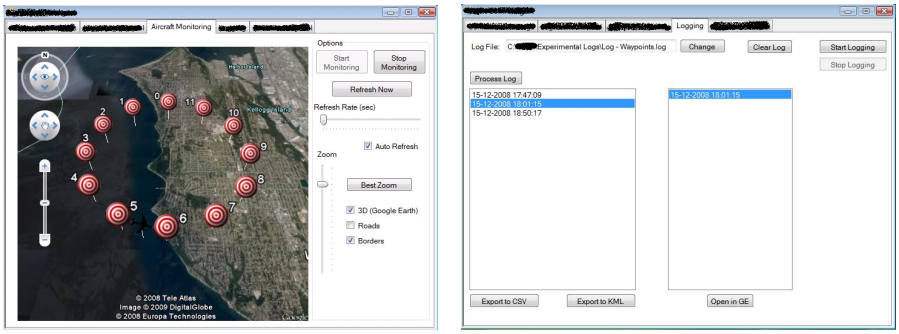
The first dimension that was analyzed is communication requirements. Although this is not a major issue when working in a simulated environment, the future use of actual vehicles connected to the platform requires this to be analyzed. In this subject, the third approach is far less demanding. While both the first and second approaches need to communicate with the aircraft at regular intervals, the third approach only communicates in the beginning of the experiment and, in the case of the third scenario, two other times, due to the existence of a user-controlled loop.

In a more detailed view, the first approach needs to send elevator, throttle and aileron data at regular intervals. In the experiments that were conducted, the values were sent every second. Although this was more than enough for the simulated experiments, it is believed that this kind of approach would require, in real live, more frequent adjustments to the aircraft controls to produce a stable flight. The second approach also sent data at regular intervals, even though a higher two-second interval was used. The data sent in this approach consists of heading, speed and altitude values for the autopilot knobs. With the third approach, and considering the mentioned exception of user-controlled loops, all data is sent before-hand. This data consists of a list of a waypoint-describing structure, containing the latitude, longitude and altitude of the point, desired speed or throttle and some flags, indicating how the waypoint should be interpreted.

4.2 Maneuver Effectiveness

Maneuver effectiveness was evaluated on a more subjective level, by analyzing and comparing the paths the aircrafts flew through. On a more immediate level, the experiment flights were accompanied in real-time, using both the simulator's

graphical interface and the developed monitoring tool. This tool uses collected aircraft data as well as information about the determined waypoints to display the current position and orientation of the aircraft, as well as the positions and order of the various waypoints. This was done using the Google Maps and Google Earth APIs, to render the desired icons on top of the two- or three-dimensional representation of the surrounding environment in a plugin using an embedded web browser. Fig. 2(a) shows the developed monitoring application, with the several waypoints that define a circle, and a visible aircraft between points five and six. In addition to this immediate and inaccurate visual inspection of aircraft



(a) Monitoring Tool (b) Logging Tool

Fig. 2. Monitoring and Logging Tools

data, information about aircraft position, speed and attitude was collected at regular intervals and stored in a log file. The developed logging application then converted this information into two formats: KML format (Keyhole Markup Language), and CSV format, as can be seen from Fig. 2(b).

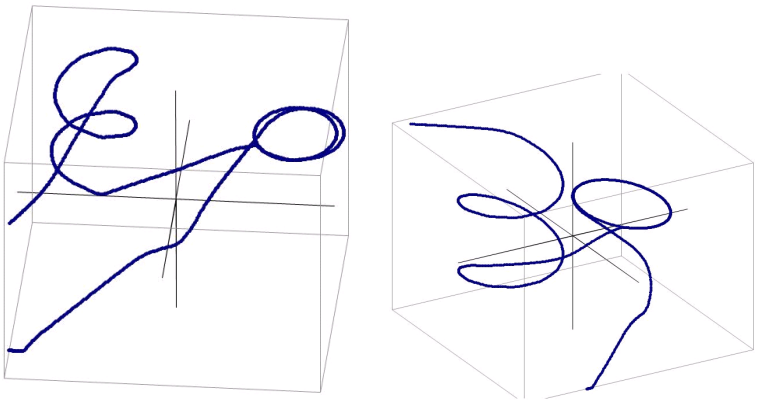
The first format, KML, allows for a visual inspection of the paths in an application such as Google Earth. Fig. 3(a) and Fig. 3(b) show the results of two experiments conducted for the first scenario, using two distinct aircraft.

It is clear by analyzing both Fig. 3(a) and Fig. 3(b) how a faster aircraft required a change in the location of the circle and helix centers, in order to accommodate the increase of the minimum turn radius, maintaining at the same time a similar overall proportion. The CSV format can be imported to an external application, such as Microsoft Excel, which was then used to generate three-dimensional graphs, as to facilitate in a further, more thorough, inspection of the flight paths without additional visual distractions.

Fig. 4(a) and Fig. 4(b) show the results from the two experiments above in three-dimensional graphs. A more detailed assessment of the efficiency of the control approaches can be made with these graphs, especially in the altitude dimension. As previously stated, experiments were conducted with three distinct aircraft, using three distinctive scenarios and under two different weather conditions. As to assess the influence of each of these three factors in the tested

(a) Flight Path A (b) Flight Path B

Fig. 3. Google Earth Preview of Flight Paths A and B



(a) 3D Graph of Flight Path A (b) 3D Graph of Flight Path B

Fig. 4. 3D Graphs of Flight Paths A and B

approaches, the results were evaluated by grouping experiments with two common variables and analyzing the results in respect to the third one.

Considering different aircraft, and as previously stated, the second approach could not be tested in the smallest aircraft, since there is no autopilot system available to the pilot. The first aircraft had a similar performance with both the first and third approaches. The second aircraft showed a slightly better performance when using the first and third approaches, compared to the second one. The third and largest aircraft had a comparable performance for all three approaches.

Regarding the three distinct scenarios, there were no significant differences among the three approaches. The three scenarios diverged in the complexity of the intended course, each scenario adding more maneuvers and decreasing the space between maneuvering areas. This caused the aircrafts to make more abrupt turns, but all approaches handled this without posing any problem.

In respect to the influence of weather conditions in maneuver effectiveness, they were similar with all three approaches. The resulting paths were a bit more unstable, presenting more variations in altitude and in some cases widening the radius of the circling maneuvers. In the cases, when this happened, the second and third approaches were more susceptible to erroneous maneuvering. In three cases, when using the third scenario, two with the second approach and one with the third approach, the aircraft performed a full circle in order to pass through a point it had previously missed. As for the first approach, the influence of deteriorating weather conditions was also felt in the form of course shifting. This was particularly visible with smaller aircraft, performing circle maneuvers, each lap slightly warped in the direction of the wind.

5 Conclusions and Future Work

In this section, some conclusions that can be withdrawn from the three tested approaches and the experimental results are presented. Also, some lines of future work are presented.

From a more theoretical point of view, one can draw some conclusions from the three approaches that were devised. The PID controller approach is a general method, which works with any aircraft that has the necessary controls (throttle, aileron and elevator). However, it requires an almost constant communication with the aircraft, in order to send the current values. Moreover, most aircraft now have some sort of auto-pilot system, which performs the same calculations as the PID controller, and, most probably, in a more effective manner, having the gains already tuned for the specific aircraft. The auto-pilot approach is not a general method, as can be seen from the chosen aircraft for the experimental activities - not all aircraft have auto-pilot systems. Moreover, it also requires an almost constant communication with the aircraft, just as the first approach. It does, however, a lot less calculations in the control agent side, leaving them to the auto-pilot system. The third approach is also a general approach, given that any aircraft can be used with the AI auto-pilot system. It is far less demanding in what concerns to communications requirements, since it only needs to communicate once. This is, however, a burst in communication, with far more data to be transmitted than any of the other methods. In addition to that, some substantial waypoint calculation has to be previously performed by the control agent.

From a more practical point of view, one has to consider the results that were attained through experimentation, as presented in section five. In respect to maneuver execution effectiveness, there are no significant differences among the three approaches. However, and although the second approach is viable when dealing with real aircraft, as already explained in section 2.2, it is not practical to use that approach in the simulated environment, since it would exclude some aircraft from being used. The use of different aircraft does not seem to influence the performance of the approaches, with the only visible impact being the increase of the turn radius caused by the increase of the aircraft speed. Also,

all approaches seem to be equivalently affected by deteriorating weather conditions, although the autopilot-based approaches are more susceptible to missing one waypoint. This can, however, be easily corrected if one increases the distance at which the aircraft is considered to have reached a waypoint when the weather conditions deteriorate. Regarding the communication requirements, one has to conclude that the third approach, based on the AI autopilot, is the best approach, since it does not require a constant communication with the aircraft, and therefore the possibility of error due to a failure in the communication with the aircraft is reduced to a minimum. The first two approaches, which have to communicate every second or once every two seconds, are more susceptible to errors caused by a communications malfunction, which could lead to an erratic maneuver performance, or even to more serious and unpredictable consequences, if the communication breakdown extents for a longer period of time. With the third approach, and even in the case of a complete communication crash, the aircraft would simply continue with the original flight plan and return to the base as originally intended to.

In spite of the project's overall success, some improvement opportunities have been identified. One possible improvement in maneuver definition and execution is the possibility to specify whether the circle and helix movements should be executed banking to the right or to the left. Currently, this is determined by the algorithm as described in section 2.4, resulting in some unpredictability. Another possible improvement would be to automate landing procedures, by automatically selecting the best runway for landing and determining a series of waypoints as to align the aircraft with the runway. Some additional factors that would increase the level of realism of the simulation could be achieved by fault injection, such as the introduction of noise in communication, as well as failures in some aircraft systems, sensors and actuators.

The next step of this project is to select an agent platform as to facilitate communication between agents, to assist in the execution of cooperative missions. When completed, the system will allow for the cooperative planning and execution of missions, such as surveillance of given areas, such as forests (to facilitate early fire detection) or coast lines (to detect and help prevent illegal activities) or even areas affected by pollution, to detect the source and probable progression of the polluting chemicals. The system can also be used to assist in military operations, such as search & rescue operations, covering a larger search area, or enemy surveillance operations.

As a final summary, one can say that the simulation environment shows promising capabilities, namely when referring to structured experiences; the aircraft control approaches were tested successfully, and the future of the project is promising, with many possible useful applications.

Acknowledgment

The authors would like to thank LIACC for providing with all the necessary equipment and for the excellent working conditions.

This work was supported by Project ACORD - Adaptative Coordination of Robotic Teams, FCT/PTDC/EIA/70695/2006

References

1. Stone, R.: Serious Gaming. Defense Management Journal, issue 31(December), 142–144 (2005)
2. Danek, G.L.: Vertical Motion Simulator Familiarization Guide. Technical Memorandum 103923. NASA Ames Research Center, Moffett Field, California, USA (May 1993)
3. National Aeronauticas and Space Administration: Vertical Motion Simulator. NASA Ames Aviation Systems Division. Available online at `http://www.aviationsystemsdivision.arc.nasa.gov/facilities/vms/index.shtml` (Consulted January 2009) (December 2008)
4. Tran, D., Hernandez, E.: Use of the Vertical Motion Simulator in Support of the American Airlines Flight 587 Accident Investigation. American Institute of Aeronautics and Astronautics Modeling and Simulation Technologies Conference and Exhibit (Aug), Providence, Rhode Island, USA (2004)
5. SimTrain, LLC: SimTrain. Available online at `http://www.simtrain.net/index.html` (Consulted January 2009) (2009)
6. Samek, J., Sislak, D. Volf, P., Pechoucek, M.: Multi-party Collision Avoidance Among Unmanned Aerial Vehicles IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'07), California, 2-5 November, 403–406 (2007)
7. Zhang, X., Xu, R. Kwan, C. Haynes, L. Yang, Y., Polycarpou, M.M.: Fault Tolerant Formation Flight Control of UAVs International Journal of Vehicle Autonomous Systems, Vol. 2, Numbers 3-4 (February), 217–235 (2005)
8. Schurr, N., Marecki, J. Tambe, M. Scerri, P. Kasinadhuni, N., Lewis, J.P.: The Future of Disaster Response: Humans Working with Multiagent Teams using DEFACTO American Association for Artificial Intelligence (AAAI) Spring Symposium on AI Technologies for Homeland Security (2005)
9. Stark, P.: FSX Mission Building, Parts 1-4 PC Pilot Magazine, Key Publishing Ltd. Issue 48 (Sep), 40-43; Issue 49 (Nov), 40-44; Issue 51 (Jan 2008), 40-44; Issue 52 (Mar 2008), 40-44. (2007, 2008)
10. Microsoft Corporation: Microsoft ESP - A New Era in Visual Simulation Available online at `http://www.microsoft.com/esp/` (Consulted January 2009) (2008)
11. Training & Simulation Journal: Lockheed Martin, FlightSafety to use Microsoft ESP platform Training & Simulation Journal Online (February 21). Available online at `http://www.tsjonline.com/story.php?F=3384514` (Consulted January 2009) (2008)
12. Erdos, D., Watkins, S.E.: UAV Autopilot Integration and Testing IEEE Region 5 Conference (April) 94–99 (2008)
13. Skogestad, S.: Simple Analytic Rules for Model Reduction and PID Controller Tuning Journal of Process Control, Vol. 13, Issue 4 (June), 291–309 (2003)
14. CloudCap Technology: A Brief History of Piccolo Development CloudCap Technology, Hood River, Oregon, USA, June (2008)