DYNAMIC PLANNING FOR AGENTS IN GAMES USING SOCIAL NORMS AND EMOTIONS

Palli R Þráinsson
Master of Science
Computer Science
April 2011
School of Computer Science
Reykjavík University

M.Sc. RESEARCH THESIS

ISSN 1670-8539

Dynamic Planning for Agents in Games using Social Norms and Emotions

by

Palli R Þráinsson

Research thesis submitted to the School of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science in **Computer Science**

April 2011

Research Thesis Committee:

Dr. Hannes Högni Vilhjálmsson, Supervisor
Associate Professor, Reykjavík University, Iceland

Dr. Ari Kristinn Jónsson
President, Reykjavík University, Iceland

Dr. Stacy Marsella
Associate Director for Social Simulation Research, Institute for Creative Technologies, USC, USA

Dynamic Planning for Agents in Games using Social Norms and Emotions

Palli R Þráinsson

April 2011

Abstract

As environments in games have become more and more realistic graphically, a new big challenge has emerged for gaming companies. Both the computer controlled agents and player controlled avatars now also need to act in a believable manner so that the illusion of reality created by exquisite graphics and physics is not broken. There is a requirement for agents to react to the environment around them as well as to act in a certain manner when introduced to social situations. For agents to be life-like they need to have basic human traits like emotions and the ability to make decisions. In this thesis we describe a way of tackling this challenge using a three pronged solution. We have incorporated a social norms model into the agents using social rules. These rules tell the agents how to act when engaged in social situations. Secondly we added an emotional model which affects the agents' emotional state and gives them the ability to vary their responses to situations in the environment. Both these models reside in an appraisal module that is based on emotional appraisal theory. The appraisal module will appraise how events triggered in the environment affect the agents both emotionally and socially and will give the agent instructions on how he might cope with that situation. To complete the cycle a planner will make the decisions on what can be done, what should be done and how it should be done. The resulting system provides the illusion that the agents are life-like individuals that can act differently to similar situations depending on what they think is important to them at that time.

Kvik áætlanagerð fyrir vitverur í leikjaumhverfi, sem byggir á félagslegum venjum og tilfinningum

Palli R Þráinsson

04 2011

Útdráttur

Með tímanum verða leikjaumhverfi alltaf raunverulegri í útliti. Þetta hefur skapað stórt áskorun fyrir tölvuleikja framleiðendur. Nú þurfa bæði tölvustýrðar vitverur og verur stjórnaðar af leikmönnum að haga sér á trúanlegan máta svo að ýmindaður raunveruleikinn sem stórkostleg grafík hefur búið til brotni ekki. Það er skilyrði að vitverur bregðist við umhvervinu sem þær eru í, ásamt því að haga sér á ákveðinn máta í félagslegum aðstæðum. Til að blása vitverur lífi þurfa þær að hafa grunn mannlega eiginleika eins og tilfinnigar og getu til að taka ákvarðanir. Í þessari ritgerð munum við lýsa hvernig hægt er að takast á við þessa áskorun með þrískiptri lausn. Við inleiddum líkan af félagslegum hegðumun í vitverurnar með notkun félagslegra reglna. Þessar reglur segja vitverunni hvernig hún á að bregðast við þegar hún aðili að félagslegum aðsætðum. Í öðru lagi þá bættum við við tilfinningalegu líkani sem hefur áhrif á tilfinningalegt ástand vitverunar og veitir henni möguleikan á að bregðast mismunandi við þeim aðstæðum sem hún lendir í í umhverfinu. Bæði þessi líkön eru hluti af mats einingu sem byggir á tilfinningarlegri mats kenningu. Mats einingin mun meta hvernig atburðir sem eiga sér stað í umhverfinu hafa áhrif á vitverur bæði tilfinningalega og félagslega og mun gefa vitverum leiðbeiningar um hvernig þær geta brugðist við því ástandi sem þær eru í. Til að loka hringnum mun ákvarðanna eining taka ákvarðanir um hvað er hægt að gera, hvað ætti að gera og hvernig það er gert. Þetta kerfi gefur þá ímynd að vitverurnar eru lifandi einstaklingar sen geta hagað sér á mismunandi hátt við svipaðar aðstæður eftir því hvað þeir líta á að sé þeim mikilvægt á þeim tíma.

Acknowledgements

This thesis is supported by the Humanoid Agents in Social Game Environments Grant of Excellence from The Icelandic Research Fund

I want to thank Dr. Hannes Högni Vilhjálmsson my supervisor for his patience and help, without it this thesis would probably still be ongoing and Arnkell Logi Péturson for all his work along side me on this project as well as the other members of the CADIA team.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In recent years, thanks to wide spread high speed broadband Internet connections, more and more people go online for entertainment as well as social interaction with other people. Huge gaming environments, whether they are a massive multiplayer online role playing game (MMORPG) like World of Warcraft[1] or a social environment like Second Life[2], are rapidly becoming one of the biggest form of mass entertainment. In these environments players interact with the world and other players as well as non-playing characters (NPC) by controlling their own human like avatar. Even though they look like real human beings there is something very important missing. They do not act or behave in a believable manner, were believable is how a user expects agents to behave depending on the situation or environment they are in. The avatars might look and dress different but in the end they all share the same limited behaviour and when game environments and avatars look as good as they currently do, a life-less carbon copy avatar sticks out like a sore thumb.

In fairness some games allow the players to give their avatar commands to look more believable, for example to smile, frown or wave goodbye, but a small set of actions does not help much in making the avatars look more human like. Lot of human behaviours also happen unconsciously and a player should not be taxed with having to micro manage behaviours that he is not even thinking about. Instead we can use AI to appraise a situation, come up with a plan to handle it and try not to be socially inept while performing it on behalf of the user. Game companies have tried to incorporate agents with emotions

[1] http://www.worldofwarcraft.com
[2] http://secondlife.com/

and personality into their games but that is almost always scripted or written for a specific scenario which becomes quite repetitive or out of place when agents do not consistently act in a believable manner in an interactive situation.

What is needed to make an avatar come to life is to use automation to create a convincing illusion of life, to make the avatar appear like they have a life of their own. To achieve this an avatar has to have what every human has: A personality as well as the ability to feel and think independently. Even though personality is what makes us different from other persons, we must balance that with the expressive behaviours that are inherited patterns or social norms of the society we live in. To create believable avatars they need to react to the game environment not only depending on their unique personality but also according to the social rules that govern the game world. An avatar must not only act correctly in social situations, he must also be able to expect that other agents respond appropriately to his social actions and appraise and respond to the situation if they do not. These social rules and behaviours provide important visual cues to what is going on and are very important tools for interpreting a social situation within the game.

Another big aspect of being human are emotions and the ability to display them. An avatar showing no emotional responses to events will seem lacking an inner life. Emotions are not only used for aesthetic purposes as emotions can affect people's reasoning process and therefore if they are missing, actions may seem out of character. Consistent behaviour of avatars is very important so that the player can relate to the avatar and even establish an emotional relationship with it. The player should be able to put him self in the avatars shoes and feel happy when the avatar is happy and that is not possible without a clear display of emotions even though they might be exaggerated for visual purposes. Cartoon animators have been using emotions for years to help people relate to things that they would not relate to otherwise like cars, ants and robots.

For the remainder of this thesis we will make no distinction between NPCs and avatars under partial control of players unless explicitly noted. Both kinds of characters suffer from lack of life-like emotional and social reaction to the environment. Let's assume that at all levels of control we are dealing with an *agent* with some autonomy of behaviour as explained in (Vilhjalmsson, 2004).

1.2 Humanoid Agents in Social Game Environments

This thesis is a part of a larger collaborative effort between CCP Games Inc.[3], a major Icelandic game developer, and the CADIA research laboratory at Reykjavik University[4], which is funded by a grant from the Icelandic Research Fund, called Humanoid Agents in Social Game Environments (HASGE).

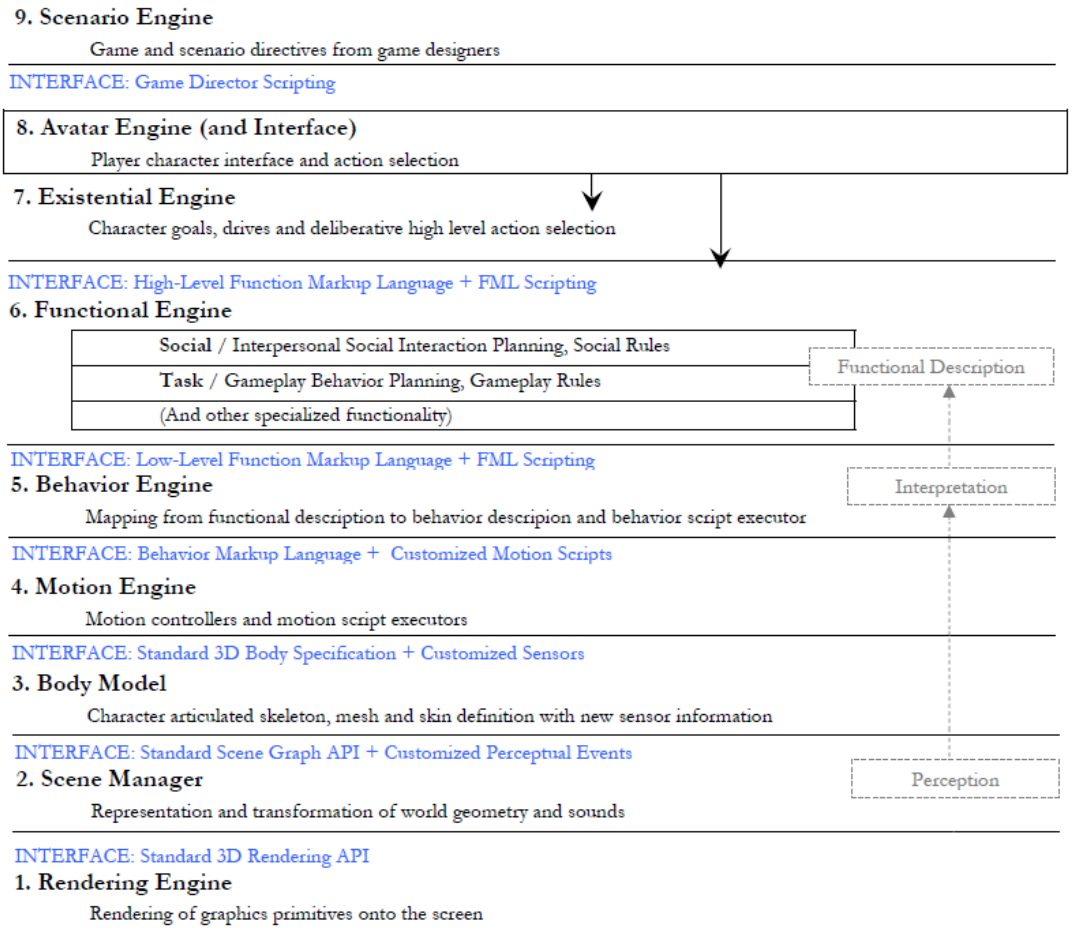

Figure 1.1: The layered architecture for intelligent animated characters as proposed in the HASGE project.

The overall goal of the HASGE project is to develop methods in AI and real-time character animation to render a fully interactive game environment where both NPCs and avatars exhibit a level of autonomy that makes them appear fully reactive to the social environment they are in. Doing this in both more believable and more engaging way than anything else that has been done before. The focus is on behaviours that underlie social interaction because it is the cornerstone of online community and yet also one of the least

[3] http://www.ccpgames.com/
[4] http://cadia.ru.is/

developed aspects of games today. Avatars are under the control of the player, but will also exhibit a level of autonomy that makes them appear fully reactive to the environment they are in. Mixed in with the player avatars there are fully computer controlled NPCs that have high level goals of their own and can interact with players to achieve them. Also other autonomous NPCs will populate the area, going about their business to bring the environment to life.

The realization of an intelligent animated character can be accomplished with a software framework where the first layers are responsible for getting moving images on the screen and the higher layers are responsible for making those images do something sensible. In a way, one could imagine the intelligence spread across all the layers, with a higher concentration at the top. The software framework proposed in the HASGE project is shown in figure 1.1.

The work presented by this thesis is two-fold, one part deals with introducing a continuous planning engine for a dynamic social gaming environment (supporting layers 5-7 in the image above) and the other introduces modelling of emotions and social norms for the agents that create relevant goals used by the planner (layer 6).

1.3 Problem Statement

CCP is working on introducing to their MMORPG Eve Online[5] the possibility for their players to dock at and enter space stations so the player can walk around and interact with other players and NPCs. For that new gaming experience to compare favorably to their existing game world they need their agents to behave in a believable manner.

For a behaviour to be reasonable, it must be oriented towards achieving the agent's goal while taking into account the dynamic environment. Note that the goals differ from one control level to another. At the highest level, in NPCs, these may be very high-level goals that require a long time to achieve, such as acquiring certain information or try to get people to perform quests to help them fulfill their goals. In contrast, at the lowest level the goals may simply be to reach a location or get some other agent's attention. For a agent to work towards achieving its goals, it must be able to plan out a series of steps and then execute those. This problem is made more complex by the constant creation of new goals, rapid changes in the environment and various dynamic constraints on possible steps. Furthermore, the problem cannot be adequately solved up-front, as the situations can vary too greatly to enumerate all possibilities or define a sensible strategy that works

[5] http://eveonline.com/

well in all different situations. Reasonable behavior and decision-making relies on solving the problem of being able to make goal-oriented decisions, executing those and adjusting them as needed in a dynamic environment, against a changing set of complex constraints. In addition, this needs to be done sufficiently fast so that agent behaviour does not slow or halt.

When an agent has a specific social goal in mind, such as saying "hello" to a friend, there are a number of social actions that have to be performed before the goal has a chance of being reached. The particular break-down of actions depends greatly on the social situation at hand, including both relatively static information such as social norms, to much more dynamic information such as when the planned recipient of a greeting is moving away. Games frequently deal with some level of planning, but most of the effort revolves around offensive tactics where the goal is the elimination of enemies. Very little has been done in helping agents in a game to be more effective planners socially. This thesis models a dynamic social environment, including possible actions and changing constraints, and feeds it to a planning engine for rapidly generating reasonably believable sequences of social behaviour.

Another thing that the agents must do is to be able to appraise the situations they find themselves in and act accordingly, and if they find themselves in a situations where things are not going according to plan they must be able to cope with it somehow. Here emotions play a large part since he may respond differently based on how the situation affects him emotionally. For example he might just accept the situation and drop the goal, try to recover from the situation by figuring out a new plan or even just enter a state of denial where he lessens the importance the situation has on his goal. While a lot of work has been done in the field of emotional modelling, as will be discussed later, most of it has been used in conversational situations between agents and humans, typically in an immobile environment where the player has limited control of the situation and not in highly dynamic massive multiplayer game environments.

This leaves us with the following question:

How can an agent in a massive multiplayer game environment be continuously guided by social norms and emotions to produce a natural sequence of behaviours and actions that are consistent with both societal and individual traits?

1.4 Outline of the Proposed Approach

For modelling emotions and social norms in NPCs and avatars, they need to be aware of their surroundings and more importantly the social situation they are engaged in and behave according to it. The proposed solution involves building on previous work done in CADIA Populus (Pedica & Vilhjálmsson, 2009), which introduced a virtual environment where agents are powered by territorial awareness and use reactive behaviours in social interactions. To fulfill the need of continuous planning for behaviours that go beyond the simple reactive behaviours of CADIA Populus, we integrate a Belief-Desire-Intention (BDI) system, called JADEX (Pokahr, Braubach, & Lamersdorf, 2005b), with a planning engine, into CADIA Populus. The BDI model is a two step process that first uses an agent's current situation (*beliefs*) to select which set of goals (*desires*) should be pursued and then how an agent builds a plan (*intentions*) by employing the means at his disposal to achieve those goals.

A model of social norms and emotions, heavily influenced by the work done in EMA (Marsella & Gratch, 2009) which provides a general framework of appraisal and coping as reasoning components for believable autonomous agents, will be used to appraise situations and events that occur in the virtual environment and subsequently provide goals for the agents so they can act in a manner consistent with the situation.

Even though a lot of work has been done in the fields touched on in this thesis, it is non-trivial to integrate them and get them to work in harmony in the dynamic environment so they do not provide contradicting behaviours and therefore damage the believability of the system. Another big challenge is the fact that the players still have a lot of control over what the avatar does and actions he might take could be considered faux pas.

To summarize:

Our goal is to create a system where agents must be able to personally appraise and react to situations around them, figure out what needs to be done to achieve their goals by creating plans and to be able to dynamically alter those plans if needed. The agents must also be able to follow the social rules of the scenario or context they are in and finally they must have emotions as they play a big part in creating unique personalities as well as influence decisions.

The architecture proposed in this thesis is based on a cycle (figure 1.2) where events that occur in the virtual environment, in this case CADIA Populus, trigger an appraisal module. The appraisal module uses its modelling of emotions and social norms to create suggestions in response to the events. The appraisal module's suggestions are then de-

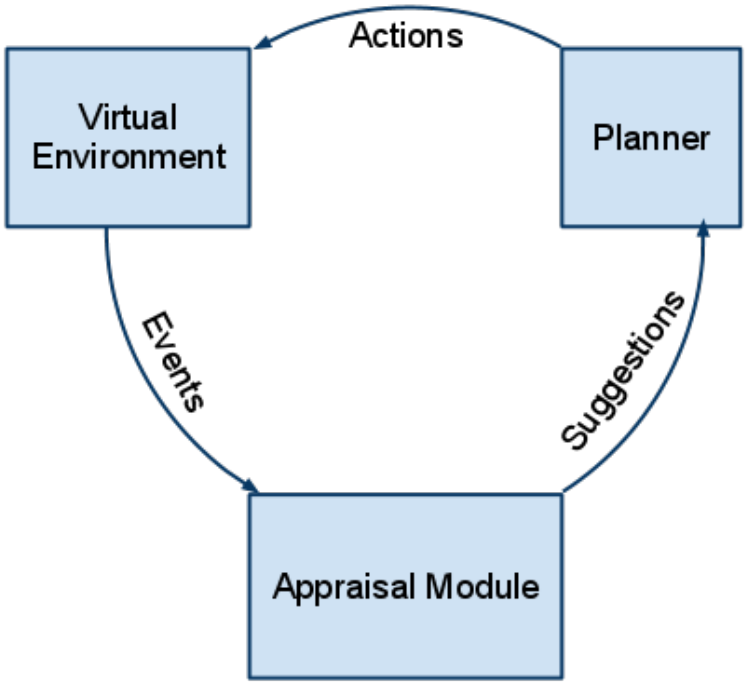

Figure 1.2: The event cycle for Social Planning Agents.

liberated over by a planner. The planner creates a plan containing the actions needed to respond appropriately to the event that started the process. Those actions are then played out in the virtual environment to complete the cycle. The architecture will be described in more details in later chapters.

Chapter 2

Related Work

The goal of this thesis is to create believable behaviours for agents in a social environment by adding social norms and emotions to Claudio Pedica's work, CADIA Populus (Pedica, 2009), as well as introducing a continuous planner to it. In this chapter earlier work will be reviewed which provides an important theoretical background. In the first section we present a brief overview of systems using behaviours controlled by social norms or culture and in the next section we will look into emotional models and systems using them. In the third section we review a few planning systems. All three sections will contain a subsection that reviews key aspects of the systems or models considered most influential to the work done in this thesis.

2.1 Culture and Social Norms

For agents to act believable in groups or in a crowd of people they have to be able to follow certain social norms as well as to perceive the people around them and respond to their behaviour accordingly. Social norms can vary from small details like how to take turn in a conversation to where and when to sit at a dinner party. Social norms can vary between cultures, a social act in one culture can be considered rude in others or even have opposite meaning, for example in Bulgaria nodding ones head is a show of disagreement while in most other European countries it is meant as a sign of agreement.

Social norms have been an interest to the Intelligent Virtual Agent (IVA) community for a while. Lot of the work has focused on behaviours in conversations like turn-taking and spontaneous gesturing (Thorisson, 1999; Cassell, Bickmore, Campbell, Vilhjalmsson, & Yan, 2001). In Thespian (Si, Marsella, & Pynadath, 2006) characters try to follow social

norms to have meaningful conversations, taking turns and keep the conversations flowing. Social norms are modelled as goals which the characters will try to achieve or follow and therefore the characters can violate those social norm goals if other more pressing goals interfere. As well as having automatically generated turn-taking behaviours BodyChat (Vilhjalmsson, 1997) introduced communicative signals like salutation or greetings where the avatar indicates he would like to engage in a conversation with another avatar. BodyChat also provided a setting so that if the person was not interested in having a chat the avatar would automatically start reactive avoidance behaviours if another avatar showed intentions to interact with him.

People have different personalities and different roles and even behave differently in certain situations and Demeanour proposes the use of multiple profiles for each autonomous agent (Gillies & Ballin, 2004). Each agent has a main profile, where all the main characteristics are defined, and three types of sub-profiles (person, role and situation) to handle different social situations. When a sub-profile is loaded it overrides the settings of other profiles.

A person's background or culture has a major influence on interactions between people, and as mentioned above different cultures have different social norms. Differences in cultures have been studied for a long time by anthropologists and behavioural scientists. Geert Hofstede's idea of five dimensions of culture (Hofstede, 2001) has been used to construct a culture agent architecture (Mascarenhas, Dias, Enz, & Paiva, 2009) in which agents behave according to their cultural context and status amongst the group.

2.1.1 CADIA Populus

CADIA Populus is a simulation platform for social game environments developed at the CADIA laboratory at Reykjavik University. CADIA Populus provides an environment with animated agents that can move around, using steering behaviours that can be activated and deactivated, and start chatting with other agents. The agents are also able to perceive their environment using low-level visual perception and high-level social perception, which provides information regarding the social environment like how many individuals are within *personal distance* according to Hall's Proxemics Theory (Hall, 1966).

The agents also have territorial awareness controlled by territorial behaviors (Pedica & Vilhjálmsson, 2009), for example when agents are in a conversation (figure 2.1) they will try to keep a personal distance, social equality, group cohesion, common attention and domain awareness. These behaviours generate motivational forces that reposition or reorient

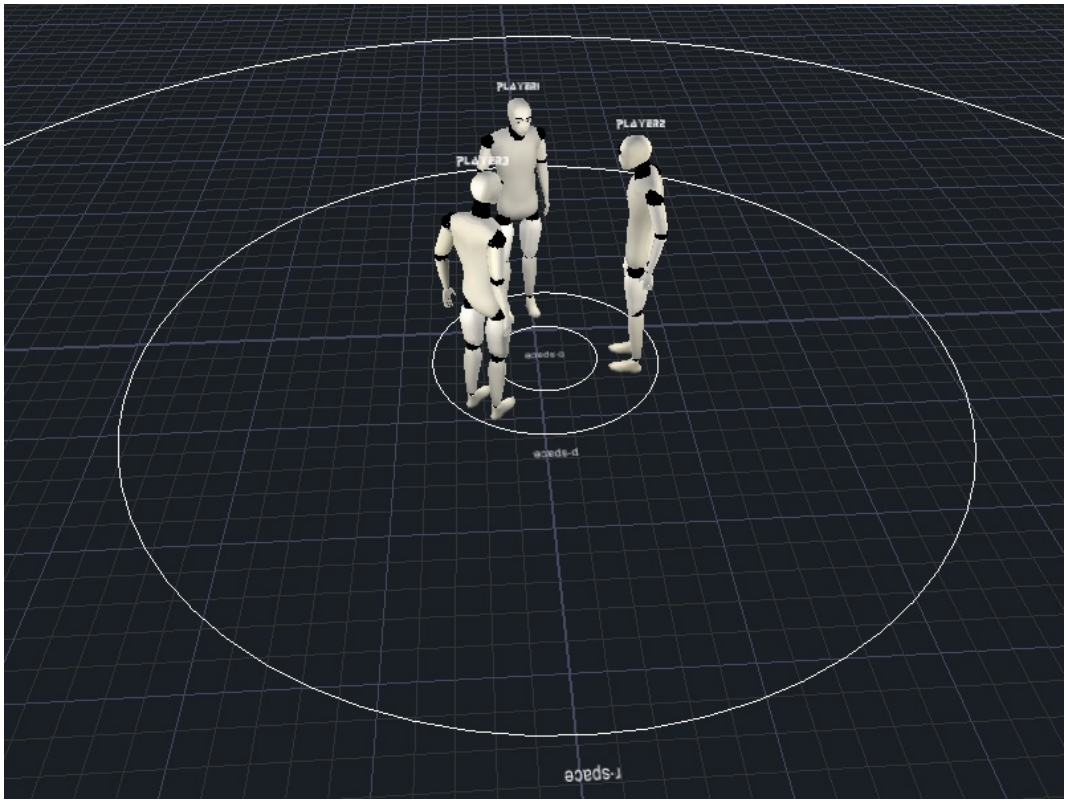

Figure 2.1: Agents in CADIA Populus in a conversation.

the agent according to the rules of the conversation model. The conversation modelling is influenced by Kendon's research on face-to-face interaction (Kendon, 1982) in which he came up with the F-formation system that explains the positional and orientational arrangements of conversation participants as well as the behavioural relationships amongst them.

Simulating the social norms of human territories in CADIA Populus has been shown to increases the believability of a scene significantly, by reducing artificiality and increasing appeal, social richness and apparent agent awareness in simulated conversations (Pedica, Vilhjálmsson, & Lárusdóttir, 2010). The agents awareness of other agents was also shown to be improved through the use of gaze behaviour and body orientation.

A feature of CADIA Populus that is particularly useful for this research is the ease of plugging in new behavioural controls. New behaviours, such as those resulting from planning can be integrated with existing reactive behaviours using several available mechanisms.

2.2 Emotional Models

Emotions play a large part in how we humans act and what we do. Our emotional state is on display every waking hour, we frown when we are sad and smile when we are happy but the face is not the only indication of emotions, for example we startle when we are surprised or take defensive stance when frightened. Just by looking at those few examples shows that creating AI agents without some kind of emotions would detract from their believability. The goal of this work is not to include emotions only for their aesthetic or dramatic value but also for how they influence the actions of the agents and the conditions that generate them. A human in a good mood is more likely to be optimistic about achieving his goals than one in a bad mood. An angry or offended person is likely to try to seek retribution or justice, while a person burdened with guilt might try to make up for what he did wrong.

In recent years researchers in the field of AI have started developing computational models for agents using various existing emotion theories. One of them is the Pleasure-Arousal-Dominance (PAD) emotional state model by Mehrabian (Mehrabian, 1996) in which he suggests that pleasure, arousal and dominance are the three faces of emotion. The PAD model was designed to describe and measure individual differences in temperament. Instead of using names or concepts for emotions, PAD gives each of the three faces a value between -1.0 to 1.0, as it is impossible to measure and average emotional traits. A subset of the PAD emotional model was used in AlphaWolf (Tomlinson & Blumberg, 2003) to model how automated pups of wolves use dominance, one variable of PAD, to find their place in the packs social order. Even though every emotion possible can be modelled in PAD there is no appraisal of those emotions, that is, there is nothing that links emotions to the events that caused them.

The OCC theory (Ortony, Clore, & Collins, 1988) models an appraisal process that assess the link between emotions and events by defining emotions as valanced (good/bad) reactions to events. The OCC model defines 22 different emotion types and 3 types of goals (active pursuit, interest and replenishment goals). The importance of goals in OCC is split into two, the importance of success and the importance of failure, as goals can have different emotional intensities whether they succeed or not. For example a person might have the goal of eating and by achieving that goal a person would probably be moderately pleased, however it will not be nearly as intense an emotion as when he can not achieve a goal which might cause him intense fear for his life. FearNot! is an Interactive Virtual Environment (IVE) that addresses the issue of bullying in schools where the player assumes the role of an advisor (friend) to the bully victim, who then tries to follow the

advice in short interactive scenes (Aylett, Dias, & Paiva, 2006). They use emotions as a key factor in achieving believability and use the OCC theory to model it. FearNot! also addresses one of the shortcomings of OCC by adding a coping mechanism.

One of the key aspects of Lazarus' appraisal theory is a coping mechanism (Lazarus, 1991). Coping, as defined by Lazarus, "consists of cognitive and behavioral efforts to manage specific external or internal demands (and conflicts between them) that are appraised as taxing or exceeding the resources of the person" (Lazarus, 1991). In other words coping determines an agent's response to the appraised significance of events. Coping is an important aspect of creating believable agents as they will have to handle situations in various manners depending on their appraisal of the situation. One of the better known computational models of Lazarus' appraisal theory is Emotion and Adaptation (EMA) (Marsella & Gratch, 2009), a system we will discuss in more detail below. EMA provides a general framework of appraisal and coping as reasoning components for believable autonomous agents. EMA introduces a coping process that is split into five stages, it starts by identifying a coping opportunity, then elaborates the coping situations using the relations between the individuals concerned as well as their responsibilities in the event. After that, coping strategies are proposed for a coping opportunity, the coping potential is then assessed and finally one or more strategies are selected and applied (Gratch & Marsella, 2004).

2.2.1 EMA

EMA is an emotional framework that models human emotions based on Lazarus' appraisal theory. Appraisal and coping are key features of EMA. The system continuously appraises how the dynamic environment affects the agent resulting in emotional and coping responses. When an event is appraised it provides the agent with information on how an event impacts others, how relevant it is to the agent, is the event desirable, likelihood of the event, does the source of the event deserve blame or credit, how much control the agent has over the event and whether it is likely that the event might change without any intervention by an agent. The agent then might need to cope with the situation he is in and there are few different ways of coping and each coping strategy affects the agent in a different manner. EMA organizes these coping strategies into four categories as opposed to the two strategies proposed by Lazarus (Lazarus, 1991), problem focused and emotional focused. Those strategies are: Attention, belief, desire and intention related coping. *Attention* related coping can either be to seek more information if something changed unexpectedly or to suppress information if the agent has low control of the situation. Shifting

responsibility towards or from someone from or towards another and wishful thinking are types of *belief* related coping. Lowering or increasing the utility are ways to increase or decrease the *desirability* of an action. *Intention* related coping can refer to form an intention to perform an action to achieve an un-achieved goal, get help from others, perform an action that makes amends if he has harmed another, delay an intention if the goal is currently unattainable, drop the goal or perform an action that removes the agent from an undesirable situation.

EMA is one of the most advanced emotional frameworks that combines planning and affect. EMA's description of appraisal makes it a fast, uniform process with no need to appeal to alternatives between fast and slow appraisal processes. The EMA model is unique among computational models of emotion in its ability to both influence and be influenced by other cognitive processes. As it is built on standard AI processes and representations it can be incorporated into autonomous agent systems. EMA has been applied to such systems as Mission Rehearsal Exercise (MRE) (Hill et al., 2003) and Stabilization and Support Operations (SASO) (Traum, Swartout, Marsella, & Gratch, 2005). MRE is a virtual military simulation in which the user takes part in a scenario. The user needs to converse with virtual characters in an extremely emotional and stressful situations and resolve the situation in an acceptable way. SASO is a negotiation training application with autonomous agents, in which negotiation strategies are seen as coping strategies and the appraisal process is used to select which strategies to use.

2.3 Planning

Automated agents must exhibit a reasonable behaviour in the environment. For the agent to act in a believable manner all his decisions and actions must be oriented to achieve his goals, this holds for both high level and low level goals such as ordering a drink at a bar and greeting a person respectively. Making these decisions and executing them in a dynamic environment requires the agent to plan his actions in a fast and dynamic way, as he has to be able to adapt his plan to the ever-changing environment.

The planning community has for a long time been interested in the design of artificial agents. Of these early planning systems, STRIPS (Fikes & Nilsson, 1971), is probably the best known. They created a system "that attempts to find a sequence of operators in a space of world models to transform a given initial world model into a model in which a given goal formula can be proven to be true" (Fikes & Nilsson, 1971). The STRIPS planner has its limitations in regards to planning with uncertainty in virtual reality simu-

lations, since a STRIPS plan is a sequence of actions that are executed blindly. To solve the uncertainty problem Sensory Graphplan (SGP) added sensing actions and conditional effects (Weld, Anderson, & Smith, 1998). SGP builds contingency plans that use sensing actions to gather information that are then used to select between different plans. In LORAX (Jónsson, McGann, Pedersen, Iatauro, & Rajagopalan, 2005) and more specifically in their planning system, EUROPA (Frank & Jónsson, 2003), they use continuous re-planning to allow its system to seamlessly modify their plans when unexpected events occur due to the dynamic environment.

The main focus of the so-called Belief-Desire-Intention (BDI) systems is reactive planning and goal deliberation. Because of this they are considered to be well suited for modelling rational behaviour in agents (Thangarajah, Padgham, & Harland, 2002). Even though they are not fully fledged deliberative planning systems there is a similarity to them (Silva & Padgham, 2004). One of the problems with pure BDI systems is that they do not provide an architectural framework for handling goal relationships, and therefore it is difficult to pursue multiple goals in parallel (Pokahr, Braubach, & Lamersdorf, 2005a). The Jadex BDI system, which we will go into more detail below, solves this problem of the BDI model by implementing a goal deliberation strategy that allows agents to pursue multiple goals in parallel (Pokahr et al., 2005a) instead of just one. As stated above BDI systems are not deliberative planners and the handicap of their reactive planning is that the plan schemata is pre-compiled and therefore not flexible. But it has been shown that it is possible to integrate a deliberate planner with a BDI system to get the flexibility of the deliberative planners and reactivity and goal deliberation of BDI (Walczak, Braubach, Pokahr, & Lamersdorf, 2006).

2.3.1 Jadex

Jadex is a BDI reasoning engine that supports building agents with *beliefs*, *desires* (goals) and *intentions* (plans) (Pokahr et al., 2005b). They automatically deliberate about their goals and pursue them applying appropriate plans. Practical reasoning is handled by two components responsible for goal deliberation and means-end reasoning respectively. Goal deliberation selects current non-conflicting goals based on agents beliefs. Means-end reasoning takes events and new goals as input to select a plan from the plan library. While plans are running they can update the belief base, trigger events or create new goals. The system uses a belief base to represent the agent's knowledge of the world. The belief base plays a crucial role in the reasoning process as the engine monitors the belief base for changes that may fulfill conditions that can trigger a new goal or drop an existing

one. Jadex provides four different types of goals: perform, achieve, query and maintain goals. Perform goals are an activity centric goals, where performing an action is most important, in contrast to achieve goals that seek to achieve a certain world state. Query goals are used for information retrieval so if the knowledge required is already available no additional work is required. Maintain goals try to maintain a certain world state and if that state gets violated the agent will by any means possible try to re-establish the desired world state. A goal deliberation strategy makes sure that conflicting goals can not be run at the same time by defining inhibition links between them so that less important goals are delayed until the completion of more important ones.

The Jadex BDI reasoning engine is designed to be extended, it is independent of the underlying platform and that makes it an easy system to use as a library for other systems. The addition of a planner makes it different from most other BDI systems as it is not only a reactive planning system but can also deliberate over more complex situations. The engine's most important feature is the full support of the two phases of the practical reasoning process (goal deliberation and means end reasoning). Jadex has not been integrated with a multi-agent social game environment before but its setup of actors, goals, beliefs and plans makes it relatively easy and implementing an external knowledge base is also straight-forward.

2.4 Summary

As can be seen here, a lot of work has been done in the field of planning and modelling social norms and emotions. But combining all three aspects in a multiplayer game environment has not been done before. By combining emotions and social norms we cover two of the most prominent aspects of what makes us unique human beings and combined with the planner will give the agents a process of thought or cognition. Adding a planner to the reactive behaviours of CADIA Populus, supports a more complex performance by our agents, as they can deliberate what actions to take to achieve their goals. In the following chapters we will go into more detail about our approach for combining these techniques.

Chapter 3

Approach

"You won't get anything done by planning." -Karl Pilkington

In this chapter our approach will be described in more detail. The goal is to improve the believability of autonomous agents using social norms and emotions. Our work is an extension on the work already done in CADIA Populus, in which human social behaviour is simulated using human territoriality. However it does not deal with interpersonal relationships, emotions, affiliation and other deeper social aspects of social interaction. Simulating those aspects has been researched and worked on by various people as seen in the previous chapter. What we want to achieve is to bring those aspects together in one place.

Our approach proposes the architecture shown in figure 3.1. We split the system into three independent modules. The first one is the virtual environment which handles the animation of the agent as well as its reactive behaviours. The cognitive aspect is represented by an appraisal module and a planning module. We decided on this three module approach as a believable agent needs be able to perceive and interact with his environment (Virtual Environment), understand what is happening around him (Appraisal) and combine various actions to achieve some goal (Planning). As stated in the previous chapter on related works we will be using the pre-existing system of CADIA Populus for the virtual environment part and the Jadex BDI system integrated with the planner for the planning module, both systems have been modified to suit our needs. The appraisal module is based on the work by Gratch and Marsella on the emotional appraisal theory of Lazarus, which we also use to model social norms.

For this work we do not provide a system with a complete range of emotions, wide range of social norms or all coping strategies defined by Gratch and Marsella. We were primarily focused on achieving a functional integration. However one of the design goals

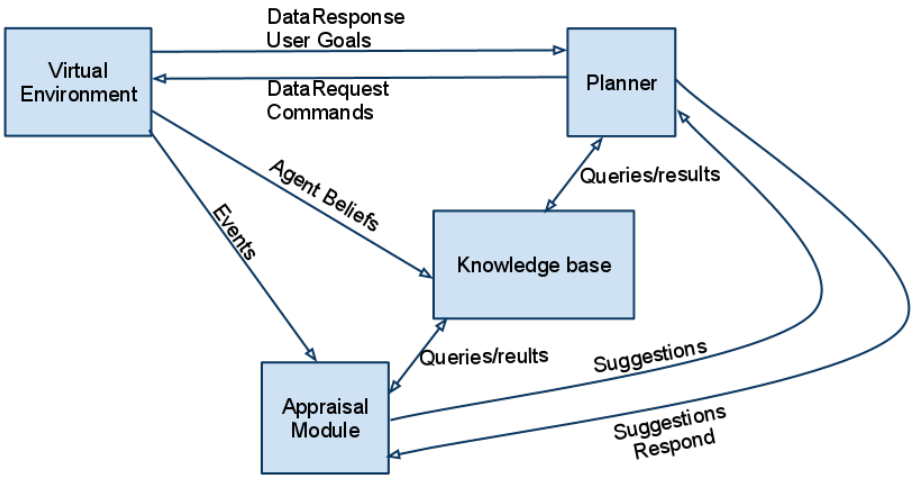

Figure 3.1: Overview of the system architecture

of this system was to make it easily extendable and therefore it is easy to integrate new emotions, social norms and coping strategies to it. It is also possible to switch out or turn off each and every one of the three main modules as they were designed be able to act independently. The appraisal module and planner module could have been more tightly coupled to make it easier to appraise plans as they are created and appraise how events affect active goals. However keeping them loosely coupled gives us more flexibility in extending each of the modules, as well as connecting them to other modules in the future. Also the limitations of not having the tightly coupled can be overcome using well defined interfaces.

In the next sections we will cover in more detail the overall architecture of the approach for modelling social norms and emotions. Each part or module will be described in detail and how they are all connected together. Detailed implementation will be covered in chapter 4.

3.1 Virtual Environment

The virtual environment, CADIA Populus, handles the visual aspect of the project, where agents are displayed in a game environment. It handles animating the agents. As mentioned in previous sections it also handles reactive behaviours modelled on Kendon's theory on face-to-face interaction formations as well as human territories focusing on conversational social situations. Those behaviours coupled with a gaze controller that controls

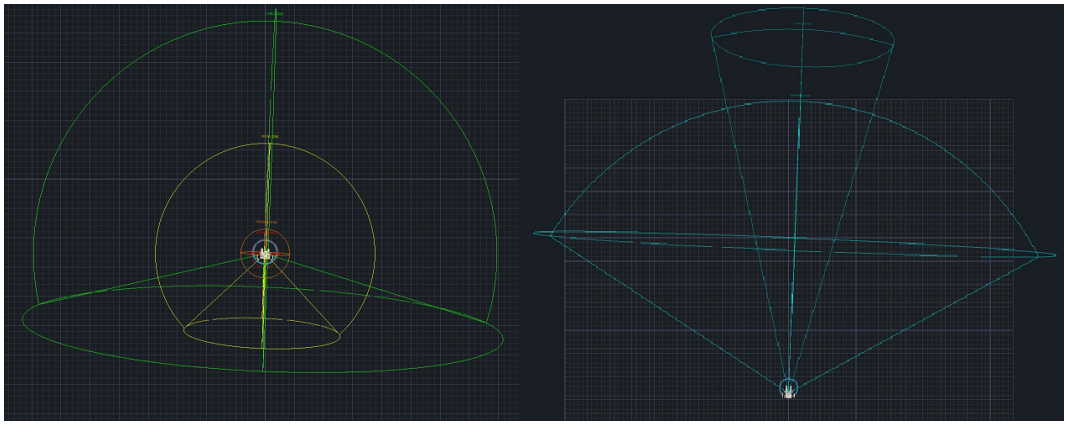

Figure 3.2: The two types of sensors in the CADIA Populus perception system. Proximity sensors on the left and visual perception on the right.

the agents idle gaze make the agents seem like they are aware of the agents they are interacting with as well as their surroundings. To generate an agent's reactive response to the environment and the social situation they are in, CADIA Populus provides a reactive behaviour framework. The framework supports the creation of a library of reactive behaviours which can be executed by other modules or by other reactive behaviours. The framework keeps track of running behaviours and will order and blend newly started behaviours with those that are already running according to priorities and weights. The sensory system of CADIA Populus, seen in figure 3.2, consists of two types of sensors. A low level visual perception (on the right) that is split into peripheral and central visual area represented by two cones in front of the agent. The second sense is a proximity sensor (on the left) that simulates a persons awareness over four distances or social zones, intimate, personal, social and public, represented by the circles around the agent. The two larger social zones have blind spots represented by two cones at the back of the agent. These two senses continuously gather information about the agent's surroundings, producing perceptual data.

In our approach CADIA Populus is responsible for updating the central knowledge base with the agent's beliefs such as the the perceptual data gathered by the perception system so other external modules can have access to the data. The perceptual data provides the knowledge base with information like position of the agents and props the agent sees using the visual perception as well as what he perceives through the proximity sensors. The knowledge base not only receives perceptual data, the knowledge base also contains information on the external status of the agent. The agent's external status information can for example be whether it is sitting or not, is in a conversation and if so what conversation is he in and with whom. The agent's profile is also stored and that has more to do with its internal status. The internal status of an agent consists of its social relationships with

other agents, who its friends and enemies are. The current emotional status of the agent is also stored in its profile along with the agents personality. Currently it is only possible to configure how emotional or social the agent is, where a highly emotional agent might take different actions in a situations then a more socially adept agent might. The agent profile information plays a huge part in the inner functionality of the cognitive aspects of this work and will be explained in more detail in the relevant sections.

Figure 3.3: CADIA Populus command menu interface

As the virtual environment provides the interface to the user of an avatar it has the goal of displaying to the user what is going on behind the scenes and to take commands from the user. The cognitive modules feed the environment with updated emotional data and social states. CADIA Populus provides the visual hints of these internal states to the user using various changes in the users avatars as well as the computer controlled agents. For example if the agent is happy he might be smiling and if it is angry the agent could furrow his brows and scowl. The agent even might clap hands and smile if an event or another agents action made the agent extremely happy or shake his fist in anger if something angered him. Some effects of changes in internal state of the agent might be more subtle and might not be directly linked to an user controlled event, for example a slight head nod and a smile when passing a friend or carefully following an agent with an evil look if it is an enemy. The virtual environment does not only receive internal status changes from the

cognitive modules but also receives commands to perform certain tasks or actions. These commands range from triggering simple animations to starting one or more behaviours such as greeting or wandering. Normally the commands received are a part of a larger plan generated in the planner module. CADIA Populus also allows the user to send commands to his avatar. The user can either just click on a certain object or place and a simple default action will be carried out or he can send a command through a command menu (see figure 3.3). The latter results in a command being sent to the cognitive modules to handle and create a plan for it. The plan is then carried out by sending back to CADIA Populus small action commands to be performed. A command can be as simple as moving to a certain object, where the plan only has one step, or a more complicated one like ordering a drink in which many small actions are needed to achieve that goal, such as getting the attention of the bartender by greeting him for example, asking for a drink, waiting for the drink to be prepared and receiving the drink and thanking the bartender.

3.2 Appraisal

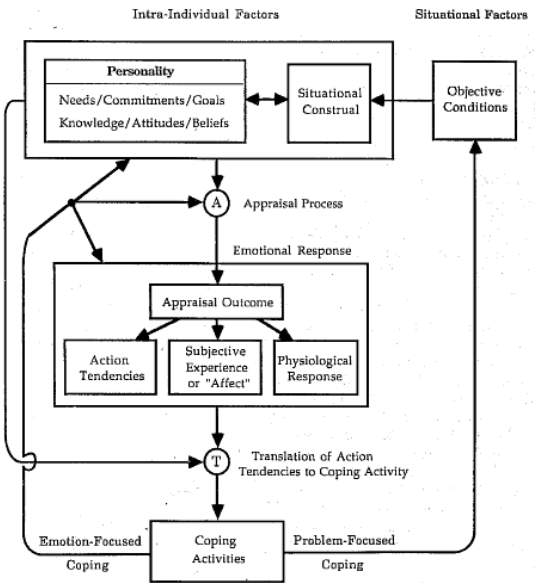

Figure 3.4: The cognitive-motivational-emotive system of Smith and Lazarus (Smith & Lazarus, 1990)

In order to create believable and life-like agents we decided that they would need to follow some kind of social norms and also show emotional responses to events happening around them. The emotions generated would influence his decision making so that the same or a similar situation would not always result in the same actions so scenarios would not

become to repetitive. To achieve this we based our modelling of emotions on the Appraisal Theory of Lazarus. "How a given individual reacts emotionally to an encounter depends on an evaluation of what the encounter implies for personal well-being" (Smith & Lazarus, 1990) is *appraisal*. Figure 3.4 depicts the model as a two step process of appraisal and coping. Our approach is to appraise events as they occur in the environment and how they affect the agent at that time, but future actions and goals are not considered during appraisal. This differs from the EMA computational model mentioned in the previous chapter, in which future actions and goals influence the agent's response to an event.

Even though the Lazarus's Appraisal Theory is a model for emotional response to events we also base our modelling of social norms on the same principles as our emotional model. In similar way as appraisal causes an emotional response to certain events, it will also result in a social response. For example If the event in question is a social action, like greeting, the appraisal process will deem greeting back as an expected response to that event as well as generating an emotional reaction.

In order to model emotions and social norms we created a dedicated appraisal module. The module is triggered by events happening in the virtual environment. An event can be anything happening in the system, it can be a action of an agent or a sound coming from an object. That is, an event needs to be triggered by someone or something and affecting or noticed by someone. As an event gets triggered, the module will notify the concerning agents, both those that are affected by the event and the one causing it. Each agent, using his event appraisal process, will then analyze the event by checking who or what triggered the event. After the cause of the event has been established the agent checks if the event fulfilled an *expectation* he had towards the source of the event.

An *expectations* is when someone expects something to happen in the future. We use expectations, for example, in such a manner that if an agent starts a two way social situation, like a greeting, he will expect that his action will be reciprocated. If an expectation is fulfilled the internal status of the agent will be updated in a positive manner but if an expectation is left unfulfilled the agent will be affected in an negative manner. For example your avatar greets an agent and if the agent greets back the avatar will both be affected emotionally (joy) and socially towards that agent, while if the agent ignores the greet the avatar might feel disappointment and dislike towards the agent. If the event in question is something that the agent had not expected, then the agent will evaluate how it affects him both socially and emotionally.

The next step of the process is the creation of the aforementioned expectations. An expectation is created only for the agent that is the cause of the event and if the event in question

triggers a fulfillment of a condition required so that the expectation can be created. One event can cause more then one expectation to be created. For example an agent might notice a friend and call him over and that would create two expectations, some kind of acknowledgment and coming over to join him. After the expectations have been created the agent evaluates all of them to see how they will affect him both emotionally and socially if they are fulfilled. Then the expectations are added to a list of active expectations which the agent checks each time he receives a new event as mentioned above. The expectations also do not live forever and each one is given a fulfilment deadline that depends on the type of expectation. For example a greet expectation has a relatively short deadline as one would expect the response to be almost immediate, while on the other hand expectations of being bought a drink in return might take longer to timeout. When an expectation expires the agent will evaluate how it will affect him and respond accordingly. Same type of expectations are also grouped together and only one of them needs to be fulfilled as they all share a similar purpose. For example in response to a greet action, one might expect to be greeted back with any of many different types of greetings and each of them would affect him differently but not receiving the greet the agent wanted the most would not affect him in a negative manner in a similar way as if he would not have been greeted at all.

The final step of the process when an event triggers the appraisal module is to create *suggestions*. *Suggestions* are the appraisal modules way of coming up with possible actions as a response to the event that just happened. Suggestions are created in the same way as expectations, that they have conditions that need to be fulfilled by the event and as with expectations many suggestions can be created from one event. They are also grouped together depending on type with one slight difference as one of the suggestions per group is always to take no action at all. This option is needed so that the agent has the possibility to select to do nothing when he wants to or can not perform the other actions. Suggestions also define a certain state that needs to be reached, called a goal state. For example a bartender agent receives an event that an agent has ordered a drink, one of his suggestions is to serve the agent and that suggestion has a goal state that the agent in question has been served or has a drink. Suggestions can have a goal state containing many such variables. As with the expectations once the suggestions have been created they need to be evaluated to set the importance of each suggestion both for social reasons and emotional. Now that the appraisal module has completed the processing of the event the suggestions are sent to the planner for deliberation. The planner chooses which suggestions should be performed and which should not and how that is done will be described in the next section. After the planner has decided it sends the results back to the appraisal module for evaluation.

The module then reviews what suggestions were accepted and updates his internal values according to their importance to him.

As has been described above the appraisal module takes care of updating the internal status of the agent, that is his emotional state, his social status towards other agents, who his friends are and who is enemies are. The internal status is updated every time an event occurs, what expectations have been fulfilled or not and what actions the agent has chosen to take. The value of change depends on the importance to the agent. How important certain things are to the agent depends on his mood, his personality and his relationship with other agents. The internal status is important both for visual aid and for the selection of actions to be taken, so the appraisal module also takes care of informing other modules regularly of the agents state.

3.3 Planning

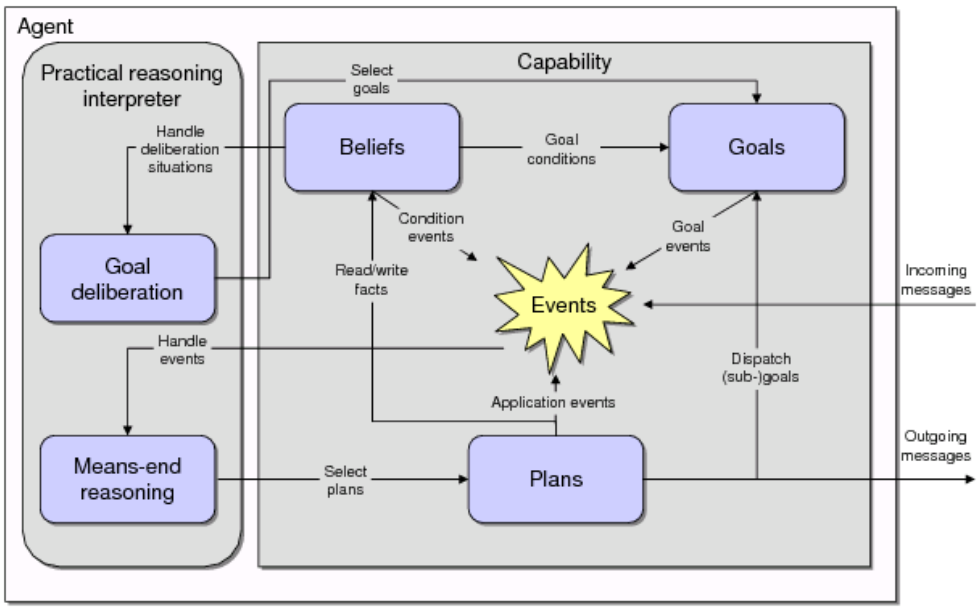

Figure 3.5: The Jadex architecture (Pokahr, Braubach, Walczak, & Lamersdorf, 2007)

Lots of what we do or want to do as humans are complex behaviours that require many different actions to be coordinated together. To create believable automated agents they also need to be able to perform more complex behaviours that require a set of actions to be performed in contrast to the simple reactive behaviours provided by CADIA Populus. Behaviours such as starting a conversation with someone are more complex than to only go up to him and start talking, at least if you want to keep some sort of a social protocol. To start a conversation with someone one needs to get his attention, wait for acknowledge-

ment, approach and greet the person and then you can start the conversation. To achieve creating complex behaviours you can either script scenarios, which might look good but will become repetitive or even in certain situations look out of place as something scripted does not take into account changes in the environment, or let the agent plan what actions to take, in what order and when. For this you need a planner and planning in a continuously changing multiplayer environment is a big challenge as the agents need to be able to respond to those changes quickly as well as intelligently.

In our approach we take a pre-existing BDI system with a planner, called Jadex (See figure 3.5). The BDI system provides us with an infrastructure to create planning agents, goals, plans and beliefs in an easy manner. Each agent in the virtual environment has a planning agent in Jadex that is completely independent of the other agents and knows basically nothing about anything except what is added to its *beliefs*. The belief base is fed information both from the virtual environment, which provides perception data and personality, and the appraisal module, which updates the internal status of the agent. The appraisal module also provides the BDI system with proposal for the *desires* or *goals* of the agent in from of the suggestions discussed in previous section. Jadex then sends its *intentions* back to CADIA Populus in form of commands that make up the *plan* for that goal. The planner works as most planners in a way that when it receives a goal it searches through all actions at its disposal to find out what actions are needed to achieve the goal. Each actions has a precondition that has to be met before it can be used and an effect which describes what will happen to the state or environment after it has been executed. The preconditions and effect are then used to determine the order of the actions when building the plan. The Jadex BDI system also has a goal deliberation strategy that allows the planner to run many non-conflicting goals at the same time as well as suspending and activating goals depending on their priority, so that when goals defined as goals of high importance become active the conflicting goal of lower importance become suspended while the other one runs its course.

The planning system receives goals from two external sources, the user issued commands in the virtual environment and the suggestions from the appraisal module. Jadex also provides a structure to create reactive goals within the system that are triggered due to changes in the belief base that fulfill create conditions of said goals. As was mentioned in previous section in this chapter the user sends his goals through a command interface in the virtual environment. The user goals are of the highest importance and override all running goals. This is done so that even though the computer controls many of the avatars action the user will always be in control. The user might therefore violate certain etiquette which might trigger unfavourable events for his avatar. The system however provides support for the computer to take some measures to mitigate such effect by either

changing the belief base so that a certain goal would be activated or to trigger an event and the appraisal module would assess the event and generate suggestions that would conclude a social situation gracefully. For example the avatar is in a conversation with an agent and the user selects to go somewhere else, that would change the belief base in such a way that the avatar was exiting a conversation, a reactive goal within the planner would then be triggered to wave or say goodbye as he leaves.

The suggestions the planner receives from the appraisal module require more work than the user issued commands as the suggestions represent many ways of responding to some situation. First of all when the planning agent receives the suggestions he will have to go through them and calculate which suggestions he wants to take, based on for example his mood or personality. The suggestions are already grouped together so the agent only has to select one suggestion from each group he receives. After the agent has chosen which suggestions to take he has to create a goal state from the combined goal states defined in each suggestion. That new goal state has to be evaluated to see if the planner can find a plan to fulfill it. If every thing is fine a new plan has been generated but if the planner finds no solution the planning agent needs to identify what part of the goal state can not be achieved and go through the suggestions again with that in mind and choose again. For example there could be two suggestions to greet another agent and one would be to wave and the other to shake his hand. The latter would be a more favourable action and would be selected in the first round but the planner finds out that the agent has to be able to touch the other agent to shake his hand and he is on the other side of a glass wall. The planner would then reject all suggestion that required him to be near the other agent and would choose the wave greet suggestion instead. As was mentioned in the section above each group of suggestions always has a no action suggestion that has no influence on the goal state so there is always at least one combination of suggestions possible. After the planner has created a plan from the suggestions the planner sends to the appraisal module what suggestions he chose for evaluation and sends the actions one by one to the virtual environment.

As the planner is independent from the virtual environment and has no idea how long actions take to perform it sends only one action from the plan at a time to the virtual environment. This means that the planner has to monitor the belief base of changes to see if the preconditions of the action have been fulfilled before sending the next action. The planner uses the same preconditions as when selecting the order of the actions when creating the plan. By making the planner send the actions one by one the flexibility of changing or interrupting a running plan becomes more easy as the virtual environment is not running a whole plan at once but just one small part at a time.

3.4 Summary

Combining these three modules discussed in this chapter helps us achieve our goal of creating agents that are able to react to changes in their environment both in an reactive manner and a manner that needs a more complex set of actions. We also have avatars that do not need micro managing as they can be given high level goals which they will then discover themselves how to achieve. Their actions are not performed in any manner the agents see fit but are guided by social rules of their game world. However, as with humans, the social awareness of the agents can be clouded by emotions and their feelings towards other agents.

Chapter 4

Implementation

In the following sections we will go into more detail regarding our implementation of the approach described in the previous chapter and what technology is used. We will not go into detail of the third-party software used, such as Jadex BDI System and PowerLoom. Our implementation goals were to create a flexible and easily extendable system with modules that are independent of each other.

4.1 Shared Modules

In this section we will discuss smaller modules, utilities and data objects that are shared amongst the three major modules, in particular the planner and the appraisal module. Modules are written in Java unless stated otherwise.

4.1.1 Launcher

Everything except CADIA Populus is started by running the *Launcher*. The *Launcher* starts by reading the setup configurations (see figure 4.1) to determine what modules to initialize and which parameters to use. If the knowledge base module is configured to start (*knowledgeBase*) then a XML-RPC web server for PowerLoom is started running on the port configured (*knowledgeBasePort*), with initial data loaded from a file (*knowledgeBaseFile*) and setting which knowledge base module to use (*knowledgeBaseModule*). The *Launcher* will not continue until CADIA Populus has registered itself with the knowledge base as there is no point of starting other modules until the virtual environment is up and running. Finally the appraisal module (*appraisalModule*) and the planner (*planner*) are

started. The planner can also be started in a graphical user interface mode (*plannerGUI*) in which goals, plans and beliefs can be monitored. As each module is started they register with the knowledge base so the other modules can know if they are running or not.

```
knowledgeBase = true
knowledgeBasePort = 8081
knowledgeBaseFile = cadia.plm
knowledgeBaseModule = CADIA
socketHost = 127.0.0.1
socketPort = 4545
planner = true
plannerGUI = false
appraisalModule = true
appraisalImports = is.ru.cadia.knowledgebase.PowerLoom
socialModule = true
socialRuleFile = social-rule.xml
emotionModule = true
emotionRuleFile = emotion-rule.xml
```

Figure 4.1: Example of a setup properties file for the system

4.1.2 PowerLoom

```
;;; Concepts
(defconcept entity)

(defconcept agent (?e entity))
(defconcept prop (?e entity))
(defconcept bartender (?e agent))
(defconcept customer (?e agent))

(defconcept conversation)

;;; Relations
(defrelation in-conversation ((?e entity) (?conv conversation)))
(defrelation greeted ((?e entity) (?e2 entity)  (?x STRING)))

;;; Functions
(deffunction pos-x ((?el entity) (?e2 entity)) :-> (?x FLOAT))
(deffunction pos-y ((?el entity) (?e2 entity)) :-> (?y FLOAT))

(deffunction sees ((?e entity)) :-> (?s STRING))
(deffunction detects ((?e entity)) :-> (?s STRING))

(deffunction sitting ((?el entity) ) :-> (?x STRING))
```

Figure 4.2: A few PowerLoom concepts, relations and functions used

For a knowledgebase we use PowerLoom (Chalupsky, MacGregor, & Russ, n.d.). PowerLoom is a knowledge representation system that uses a fully expressive, logic-based

representation language. PowerLoom uses a natural deduction inference engine that combines forward and backward chaining to derive what logically follows from the facts and rules asserted in the knowledge base. As was mentioned in previous chapters the knowledge base stores information regarding various things like the type of agent, conversations, actions performed, locations, status and what the agents perceive. Some of the *concepts*, *relations* and *functions* defined in PowerLoom can be seen in figure 4.2. PowerLoom is programmed in a programming language called STELLA that can be translated into Lisp, C++ and Java. We use the Java version of PowerLoom as the planner and appraisal module are both programmed in Java and connect therefore with the knowledge base using the Java API provided. Figure 4.3 shows the class diagram of the PowerLoom connection to the three major modules. We created a *PowerLoomConnection*, that provides an easy to use interface against PowerLoom. The class encapsulates all major functionality of PowerLoom, such as asserting and retracting propositions, retrieving knowledge, ask whether something is true or not, runtime creation of concepts, evaluation of commands and loading a file of pre-defined structures. The planner and appraisal module share a helper class, *PowerLoom*, that encapsulates the PowerLoom syntax and uses *PowerLoomConnection* to connect to PowerLoom. The connection on the CADIA Populus side is done in a similar fashion with a *PowerLoom* helper class that connects to the *PowerLoomConnection*. The only difference is that it sends the PowerLoom commands through an XML-RPC gateway (*PowerLoomXMLRPC*). However this causes a performance issue because of the overhead of sending messages over XML-RPC, to solve this we group the commands together in batches before sending them over instead of sending them one by one.

4.1.3 Communication

All communication between CADIA Populus and the external modules goes through sockets. The CADIA Populus side is the socket server and all external modules connect to it using socket clients. The class structure of the communication layer can be seen in figure 4.5. Within CADIA Populus it is the *PlannerSubsystem* that manages the data transfer to and from the socket server. Google's Protocol Buffers[1](Protobuf) are used to encode the data that is transferred through the sockets. Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data. The data structure created is then generated into source code for the programming language being used. Protobuf currently supports three programming languages, C++, Java and Python. The fact that our system uses both Python and Java modules made using Protobuf an easy choice. We categorized our data messages into three groups: request, response and commands. For this

[1] http://code.google.com/p/protobuf/

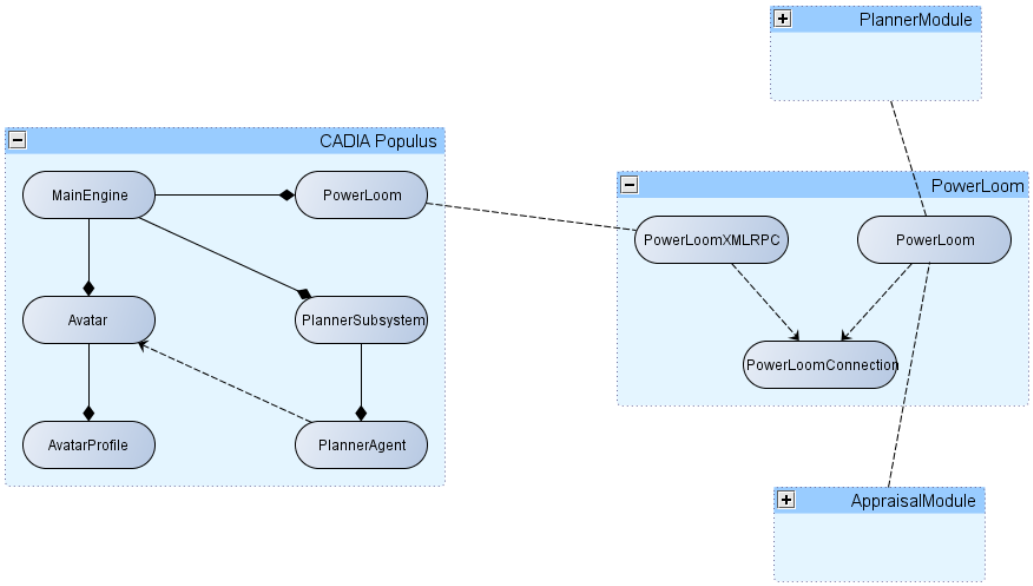

Figure 4.3: A class diagram showing the PowerLoom classes and connection to the three major modules. CADIA Populus connects to the PowerLoom interface using an XML-RPC interface. The Appraisal and Planner module use a Java API to communicate with PowerLoom.

we created a data structure consisting of a wrapper object, *DataTransfer*, that can contain an instance of the three message types, *DataRequest*, *DataResponse*, *DataCommand*, as seen in figure 4.4.

DataRequest and *DataResponse* work in pairs. The external modules request data from CADIA Populus by sending *DataRequest* and get the data back as *DataResponse*. Both messages are very similar and can be of four different types, registered agents, goals, remove agent and events, but the *DataResponse* also sends additional data depending on the type. A registered agents message contains information on agents in the virtual environment, goals have information on goals created by the player himself, remove agent informs when agents have been deleted and events have data regarding events triggered in the system.

Handling of communication for CADIA Populus is written in Python and is done through the *PlannerSubsystem* class. *PlannerSubsystem* works as a subscription service, whenever it receives a *DataRequest* the sender is put into a list of subscribers and when CADIA Populus sends out information the data is sent to all those that are registered to that type of a message. The *DataCommand* is used by the external modules to send action commands to an agent. The CADIA Populus *MainEngine* takes care of informing the *PlannerSubsystem* of registered agents and when agents are removed from the environment as well as providing an access point for other classes, such as *Avatar*, to the *PlannerSubsystem*.

```
message DataTransfer
{
    enum DataType { REQUEST = 1; RESPONSE = 2; COMMAND = 3; }
    required DataType dataType = 1;

    optional DataRequest request = 2;
    optional DataResponse response = 3;
    optional DataCommand command = 4;
}

message DataRequest
{
    enum Type { REGISTERED_AGENTS = 1; GOALS = 2; REMOVE_AGENT = 3; EVENTS = 4;}
    required Type type = 1;
}
```

Figure 4.4: The Protocol Buffers messages DataTransfer and DataRequest

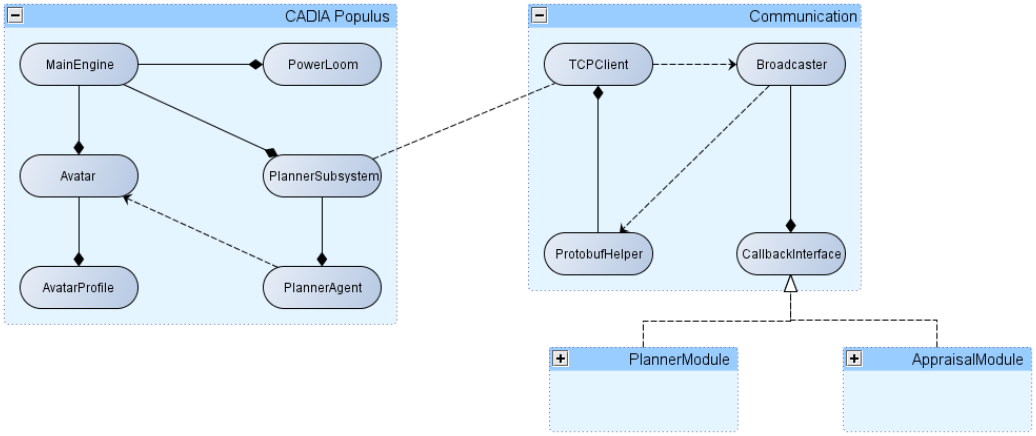

Figure 4.5: A class diagram showing the communication layer. CADIA Populus communicates with the communication layer using Protobuf messages. The other two modules use the ProtobufHelper to send messages and subscribe to messages with the Broadcaster.

In the *MainEngine* game-loop a call is made to the *PlannerSubsystem* so that data that is not urgent can be gathered and sent at a certain time interval.

To handle communication for both the planner and the appraisal module we created a Java *TCPClient* that handles all socket communication with the socket server in CADIA Populus, using Google Protocol Buffers as mentioned above. The socket client is asynchronous so that when it sends a message it will not wait until it receives a response. What socket server to connect to (*socketHost*) and on what port (*socketPort*) is configured in the setup properties file (see figure 4.1). As soon as a message arrives it is broadcast through the *Broadcaster*. The *Broadcaster* handles all messages going to the planner and appraisal module, both messages coming from CADIA Populus as well as messages between the modules. For the modules to get a message from the *Broadcaster* they have to subscribe

to the message types they are interested in and implement the *CallbackInterface* used to broadcast messages to the modules. When a message is received through the socket connection the *Broadcaster* uses *ProtobufHelper* to convert Google Protocol Buffer messages to data classes (see section 4.1.5 Data Classes). *ProtobufHelper* is a helper class that can be used for both converting data classes to messages and vice versa and therefore hides Protocol Buffers code from other modules. *ProtobufHelper* also provides functions for the modules to send Google Protocol Buffer messages to CADIA Populus. This means that all communication between modules goes through a single point.

4.1.4 Utilities

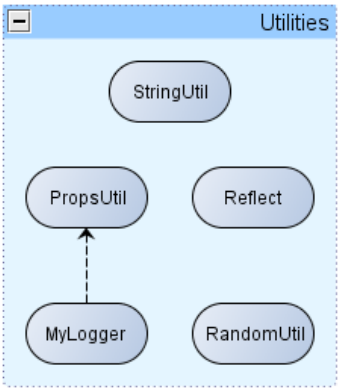

Figure 4.6: A class diagram showing the Java utility classes

Few utility classes have been created to keep commonly used functionality in one place (see figure 4.6). *StringUtil* provides functions when working with strings such as to parse strings to various arrays and lists. *Reflect* has various static reflection methods to find classes, construct classes or invoke class methods when code is stored in non compiled files like XML or properties files. This utility is mainly used when converting condition clauses in the *rule* xml files when evaluating them. *PropsUtil* provides a singleton class for loading and storing a property file as well as access methods that return default values instead of null when property is not found in file. *RandomUtil* is a static class with access to random generator so that only one instance of provides the system with random numbers. This is mainly done so that objects populated with random numbers will not have the same data if created at the same moment. The final utility is *MyLogger* which, as the name might indicate, handles logging in the system except for CADIA Populus. *MyLogger* uses the log4j library from Apache[2]. The log utility is a singleton so that all modules

[2] http://logging.apache.org/log4j/

log into the same file, an example can be seen in figure 4.7. Log pattern (*logPattern*), level (*logLevel*) and file (*logFile*) for the logger are configured in the setup properties file (see figure 4.1). *MyLogger* also provides static logging functions for all log levels, debug, info, error and fatal.

```
INFO 2010-12-28 18:47:07,531 Starting PowerLoom/XML-RPC web server at port 8081
INFO 2010-12-28 18:47:07,578 PowerLoom web server running.
INFO 2010-12-28 18:47:07,578 Waiting for CADIAPopulus
INFO 2010-12-28 18:47:20,187 AppraisalController - AppraisalController created
INFO 2010-12-28 18:47:21,718 SceneController - SceneController created
INFO 2010-12-28 18:47:30,937 AppraisalController.addAgent - Adding agent Barkeep
INFO 2010-12-28 18:47:30,937 SocialNormModule.loadRules - Loading social rules for Barkeep
INFO 2010-12-28 18:47:31,000 EmotionModule.loadRules - Loading emotion rules for Barkeep
INFO 2010-12-28 18:47:31,000 AppraisalController.addAgent - Adding agent Player1
INFO 2010-12-28 18:47:31,000 SocialNormModule.loadRules - Loading social rules for Player1
INFO 2010-12-28 18:47:31,015 EmotionModule.loadRules - Loading emotion rules for Player1
INFO 2010-12-28 18:47:31,015 SceneController.registerAgent - starting agent Barkeep
INFO 2010-12-28 18:47:31,015 SceneController.startAgent - Starting agent Barkeep
INFO 2010-12-28 18:47:31,171 AgentInitPlan.body - Initializing agent Barkeep
INFO 2010-12-28 18:47:31,203 SceneController.register - Registering agent - Barkeep
INFO 2010-12-28 18:47:31,250 SceneController.registerAgent - starting agent Player1
INFO 2010-12-28 18:47:31,250 SceneController.startAgent - Starting agent Player1
INFO 2010-12-28 18:47:31,250 AgentInitPlan.body - Initializing agent Player1
INFO 2010-12-28 18:47:31,250 SceneController.register - Registering agent - Player1
```

Figure 4.7: Examples from the systems log file

4.1.5 Data Classes

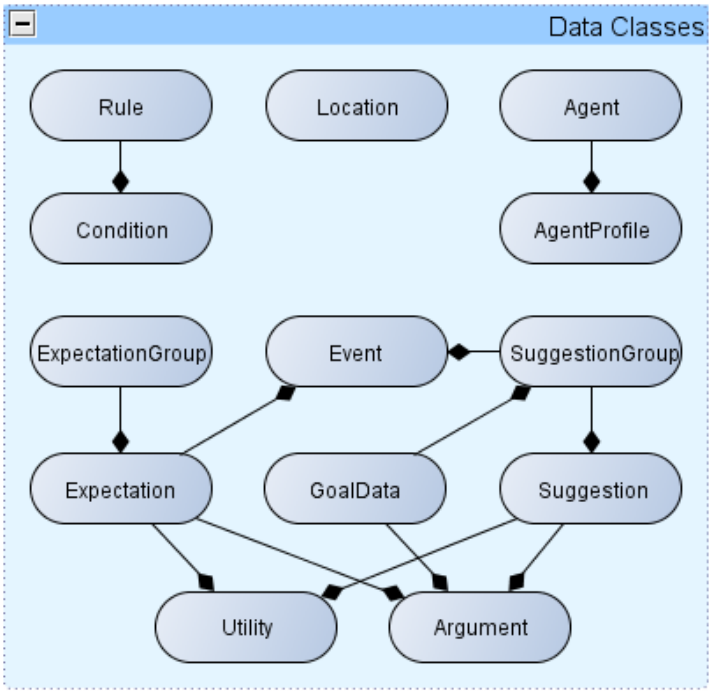

Figure 4.8: A class diagram showing the Java data classes

The appraisal module and the planner share some data classes which can be seen in figure 4.8. *Agent* stores information for agents such as his unique id, name, type and the agents

profile (*AgentProfile*). The *AgentProfile*, as mentioned in previous sections, holds information regarding the agents' emotional and social state, list of friends and enemies, his attitude towards other agents as well as configurations of the agents personality. *Location* stores coordinates of entities in CADIA Populus and is used when fetching their position from the knowledge base. The expectation and suggestion concepts mentioned in previous sections both have class representations, *Expectation* and *Suggestion* respectively.

Expectation has information on the type of expectation, what sub-type it is, who is the target, what event caused the expectation, which extra arguments there are if any, how much utility it will provide to the agent if fulfilled or not and how long it will be valid once it is activated. *ExpectationGroup* is used to group together expectations of the same type and target, so that only one such expectation needs to be fulfilled. *ExpectationGroup* also keeps track of which *Expectation* in the group will expire last, as the expectation group is valid until all expectations have expired. Similar data is stored for suggestions except *Suggestion* does not have a deadline and additionally stores the status of the suggestion (Accepted, rejected, unprocessed) and what changes to the world state are required to achieve the suggestion. *SuggestionGroup* groups together suggestions of the same type that are relevant to a certain event.

Event contains information on events that occur in the environment. An event consists of a type, sub-type, trigger and list of agents that are affected by the event. *Argument* is used when storing extra arguments for expectations and suggestions as well as goal state values of suggestions. *Argument* consists of a name, expression and value. The expression can be (almost) any value or Java statement. The expression is parsed to set the value of the argument. The expression is only parsed once for each argument. A *Utility* class is used to store type and value of utilities for suggestions and expectations. Information regarding goals are stored in *GoalData*. Information on goals includes the id of the agent who's goal it is, what type and sub type of goal it is, additional arguments, state required to achieve the goal and if applicable the group of suggestions used to create the goal.

Condition and *Rule* classes will be described in the next section.

4.1.6 Parsers

We created two parsers for XML files that store social and emotional rules for the appraisal module. First we created *XMLParser*, a generic xml parser using the SAX XML parsing library, to parse the rule XML file (see figure 4.9). The parser is then called to parse the rule XML file using *RuleHandle*, a XML parsing helper class. The *RuleHandler* contains the logic to parse the specific structure of the rule XML file. The helper

is invoked when a start or an end element is located in the XML file. The name of the elements is then used to construct instances of *Rule*, the rules' *Condition*, its suggestions, expectations and what effect its fulfillment will have. When the end element of the rule is encountered all objects are gathered together and added to the *Rule* instance and added to a list of rules.

```
<rule>
    <condition>
        <!-- self not trigger of event AND event is of type greet AND self not greeted
             trigger with same type of greet-->
        !$self.getId().equals($event.getTrigger()) &amp;&amp;
        "greet".equals($event.getType()) &amp;&amp;
        !PowerLoom.hasGreeted($self.getId(), $event.getTrigger(), $event.getSubType()) &amp;&amp;
        "distant".equals($event.getSubType())
    </condition>
    <suggestions>
        <suggestion type="greet" subType="distant">
            <utilities>
                <utility type="social" utility="5" />
                <utility type="emotion" utility="1" />
            </utilities>
            <arguments>
                <!-- value can be expression-->
                <argument name="isResponse" value="true" />
                <argument name="target" value="$event.getTrigger()" />
            </arguments>
            <goalState>
                <state name="hasDistantSaluted" value="$event.getTrigger()" />
            </goalState>
        </suggestion>
        <suggestion type="greet" subType="noAction">
            <utilities>
                <utility type="social" utility="-7" />
                <utility type="emotion" utility="-1" />
            </utilities>
            <arguments>

            </arguments>
        </suggestion>
    </suggestions>
    <!-- i was greeted this makes me happy -->
    <event-effect>
        <effect name="angerJoy" value="5"/>
    </event-effect>
</rule>
```

Figure 4.9: Example of a rule from XML file. At the top there is the condition that needs to be fulfilled so that the rule is used. Rules can have a list of suggestions with utility, arguments and goal state. Not shown in this example but some rules also have expectations. Last part of the rule is the event effect on the internal state of the agent.

As seen in figure 4.9 the condition of the rule is very similar to Java code. To be able to evaluate these conditions another parser was made called *ExpressionParser*. The parser is generated using JavaCC[3], a parser generator, and its add-on JJTree, which allows the generated parsers to produce syntax trees. The expressions to be parsed, such as the condition, are then passed to the parser. The parser then breaks the expression down to smaller units, which are then recursively parsed until the whole expression has been broken down to single statements. Each statement is then evaluated to find out of what type it is using

[3] http://java.net/projects/javacc/

reflection. The expression in whole is not evaluated until the value of the expression is requested. The parser offers the possibility of including import statements when evaluating an expression that contain classes that are not part of the standard *java.lang* package. These import statements can be added in the system setup properties file mentioned above (see figure 4.1) using the *appraisalImports* property. Example of this can be seen in the example provided where a static call to the *PowerLoom* class is made. A major part of the expression is the possibility to include parameters, marked with a '$' symbol in front of it. When an expression has such parameters they need to be included when the expression's value is requested. In the example the *$self* parameter is replaced with the *Agent* object of the agent evaluating the condition. Not only conditions are evaluated in this manner, both values of arguments and goal states can be expressions as well.

4.2 CADIA Populus

CADIA Populus is a virtual social game environment written in python that supports fast development of various different social situations to try out new agent behaviours. The graphical aspect of the environment in created using Panda3D[4]. Panda3D is a game engine, a framework for 3D rendering and game development from Disney and Carnegie Mellon University. The NVIDIA PhysX engine is used to create a reliable physical simulation of scenarios in CADIA Populus. PhysX allows rigid body dynamics, fast spatial queries and high performance physical simulation.

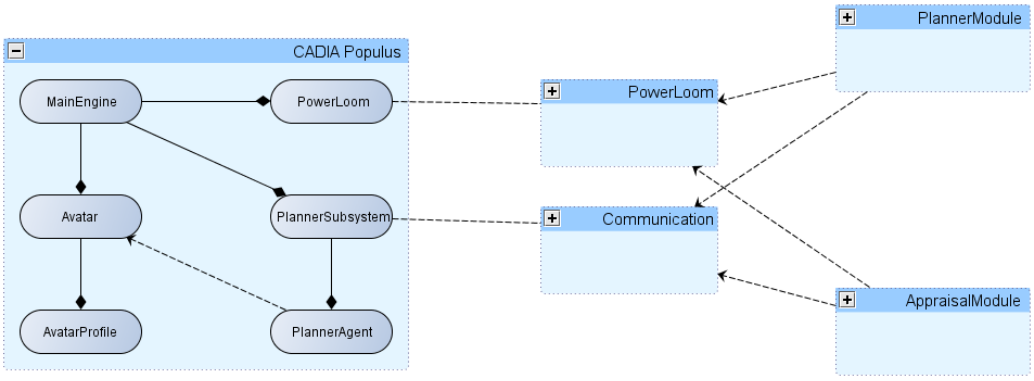

Figure 4.10: A class diagram of new and altered classes in CADIA Populus

To be able to integrate the planning and appraisal module with CADIA Populus some additions were needed, see class diagram in figure 4.10 for new or altered classes. First of all a layer was added between the *Avatar* class in CADIA Populus, which controls the

[4] http://www.panda3d.org

reactive behaviours and animation of the virtual agents and avatars, and the new external modules (appraisal module and planner). We created a *PlannerAgent* that has the purpose of receiving commands from the external modules and play them out through the *Avatar* class. This is done by mapping each command received to starting and/or stopping reactive behaviours, play or stop animations or gestures and issuing commands to speak. Doing it this way the external modules issues a single command that can start one or more behaviour depending on the complexity of the action, for example a greet command is issued the *PlannerAgent* glances at the other agent, greets him both verbally and with a wave gesture. Each agent of the virtual environment is registered with the external modules and will have a *PlannerAgent* instance that it belongs to. The class also takes care of updating the knowledge base when a performance of action by the agent needs to be stored in the knowledge base so that the agent can know that he has performed certain actions, like when an agent greets another it will be stored so that he will not greet him again within a certain time period. We do this so that the agents do not look silly by constantly greeting each other. As well as updating the knowledge base the *PlannerAgent* also receives new internal status information from the external modules to update the *AvatarProfile* of the *Avatar*. The *AvatarProfile* contains information on the agents friends and enemies which is a list of the agents names in question, his personality, his emotional status and status towards other agents. Currently the personality consists of what type the agent is, for example a bartender, how emotional and social the agent is. The emotional and social variables are represented as float numbers that can range from 0.0(lowest) to 1.0(highest). Meaning an agent with emotional value of 1.0 and social value of 0.3 would be controlled by his emotions and take emotional decisions instead of maybe more socially acceptable ones. For example if he bumps into another agent, he would shout obscenities at him for being in his way instead of apologizing. For now these values are constant as the persons personality does not change much over short time which are current focus is on. However the emotional status and how an agent feels about other agents are ever changing variables. Those variables are floating numbers that range from -10.0 to 10.0 if the variable has two sides and 0.0 to 10.0 if it only has one side. Currently the variable we use are all two sided like *angerJoy* represents the agent emotional spectrum from furious(-10.0) to ecstatic(10.0) with 0.0 being a neutral and the same for agents opinion on other agents, where -10.0 represents utter dislike and 10.0 means that the agent is in awe of the other agent.

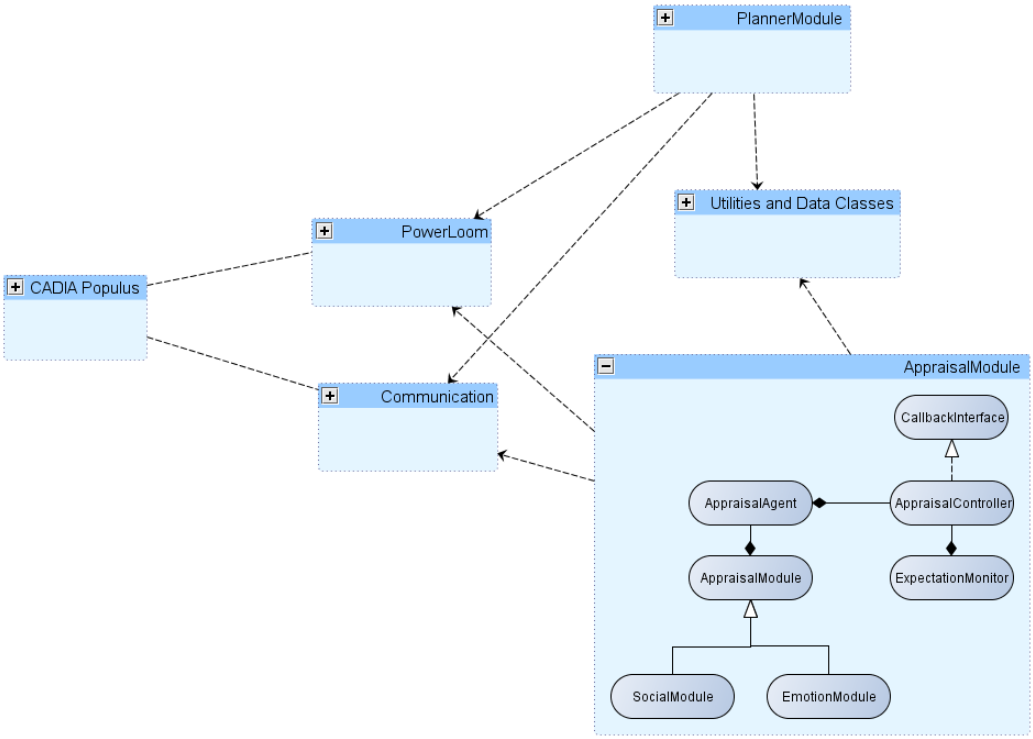

Figure 4.11: A class diagram of the new Appraisal Module

4.3 Appraisal Module

The class diagram of the new Appraisal Module is shown in figure 4.11. The *Appraisal-Controller* is a singleton class and is the center of the appraisal module. The controller keeps track of all agents, maintains the *ExpectationMonitor* and is able to receive broadcasts from the *Broadcaster* by implementing the *CallbackInterface* as well as registering that the appraisal module is active with the knowledge base. The controller subscribes to registered agents, removed agents, events and suggestion responds messages. When a new agent is registered with controller an *AppraisalAgent* is created. The remove message deletes the appraisal agent from the list of agents. When an event message is received the controller checks which agents triggered and are influenced by it and sends it to them for processing. The same is done for suggestion respond messages from the planner.

ExpectationMonitor is a thread that runs every second and checks whether expectations have expired. The monitor fetches all *AppraisalAgent* instances from the *AppraisalController*. Each appraisal agent is then queried for his *ExpectationGroups*, which the monitor will run through and checks if the last *Expectation* in the group has expired. The monitor gathers together all expired expectation groups and informs the appraisal agent if there are any.

Each *AppraisalAgent* consists of the agents information (*Agent*) and the expectations of the agent. Each appraisal agent can also have *AppraisalModules*. The modules are used when the agent is processing an *Event* and creating and evaluating expectations and suggestions in response to that event. Currently we there are two appraisal modules and they can be configured in our out using the system setup properties file. The agent can have an *EmotionModule* and a *SocialModule*, if the *emotionModule* and *socialModule* properties are set respectively. Both modules extend the *AppraisalModule* and share most of its functionality. Each module has its own set of rules, which will be explained below. When the *AppraisalAgent* processes an event he gets expectations and suggestions from the appraisal modules which he then groups together as mentioned above. After the appraisal agent has processed the event, the suggestion groups are sent to the *Broadcaster* which ultimately provides the suggestions to the planner. When the *ExpectationMonitor* informs the *AppraisalAgent* of expired expectations the agent starts by removing them from his list of active expectations. After that the expectation, from the group of expectations, that would have given the agent the lowest utility is taken, negated and his internal status updated accordingly.

After the planner has chosen which suggestions to use, a response with the results is sent back to the agent. The agent goes through the suggestions and updates his internal status depending on the utility of the chosen suggestions. As there is always one suggestion in each group which is to take no action with appropriate utility, there is no need to think about negative utility for actions not taken. When calculating the changes in the agent status the utility is altered depending on the agent's personality. A highly social agent will be more affected by social utilities. After the agent has updated his internal values an update is sent to the other modules.

4.3.1 Social and Emotional rules

The *AppraisalModule* and its subclasses *EmotionModule* and *SocialModule* load the XML rules files mentioned before. These rules are used when appraising events. When an unexpected event occurs the appraisal modules evaluate each rule to see if the conditions are met. If a condition is met the utility of the configured effect of the rule is used to alter the agents status. Same process is used when creating expectations and suggestions, if the rule's conditions are fulfilled then the expectation or suggestion of that rule are added to their respective lists.

As can be seen in figure 4.9 rules consists of conditions that need to be true so that rule can become active, suggestions that have a type so they can be grouped together, sub-

type to distinguish between them and list of utilities. The utilities tell how important the suggestion is, both positively and negatively, to the agent. Each suggestion can also have additional arguments, that are not part of the standard suggestion, that might be needed when working with them. For example in the case of a greet suggestion knowing whether it is a response or who the agent is greeting could be important. The suggestion might also define a goal state that needs to be achieved so that the suggestion can be considered achieved. If the goal state has more than one variable all of them need to be fulfilled. Like suggestions, expectations of a rule (see figure 4.12) have a type and subtype but they also have a deadline. A deadline is defined as the number of milliseconds that can pass from activation until the expectations has expires. Expectations also have utility and arguments but no goal state. The final part of the rule is that it can have an event effect, which defines in what way fulfilling the condition of the rule will effect the agent.

```
<rule>
    <condition>
        <!-- self trigger of event AND event is of type greet AND
             target has not greeted trigger before-->
        $self.getId().equals($event.getTrigger()) &amp;&amp;
        "greet".equals($event.getType()) &amp;&amp;
        !PowerLoom.hasGreeted($target, $event.getTrigger())
    </condition>
    <expectations>
        <!-- A greet of no specific type, default used if he gets an
            unexpected greet but a greet of some sort-->
        <expectation type="greet" subType="default" deadline="10000">
            <utilities>
                <utility type="social" utility="1" />
                <utility type="emotion" utility="1" />
            </utilities>
            <arguments/>
        </expectation>
        <!-- A greet of type distant -->
        <expectation type="greet" subType="distant" deadline="5000">
            <utilities>
                <utility type="social" utility="3" />
                <utility type="emotion" utility="5" />
            </utilities>
            <arguments/>
        </expectation>
        <!-- A greet of type close -->
        <expectation type="greet" subType="close" deadline="10000">
            <utilities>
                <utility type="social" utility="4" />
                <utility type="emotion" utility="6" />
            </utilities>
            <arguments/>
        </expectation>
    </expectations>
</rule>
```

Figure 4.12: Another example of a rule from XML file. Unlike in figure 4.9, this rule has a list of expectations that are activated if the condition is met.

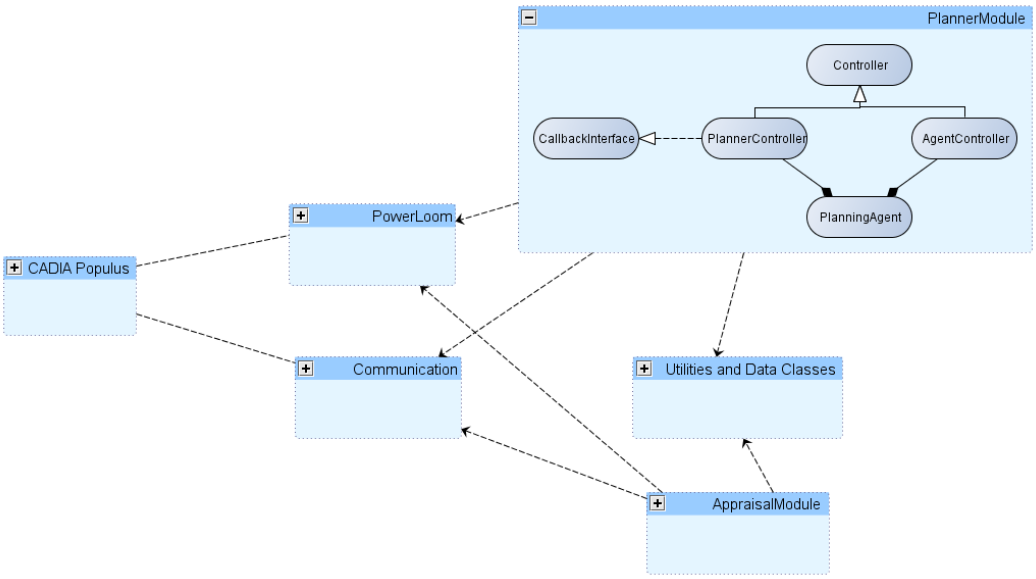

Figure 4.13: A class diagram of the new Planner Module

4.4 Planner Module

For the planning part of the system we use the Jadex BDI system with a planner integrated, which is written in Java. The class structure of the Planner Module can be seen in figure 4.13 (the diagram only shows our classes and not the classes of the Jadex system). In a similar setup as the appraisal module the central piece is a *PlannerController* which handles broadcasts from the *Broadcaster* by implementing the *CallbackInterface* and manages the agents. The controller is registered with Jadex as a Jadex agent and as such has a connection to the Jadex system, has a belief base and can create goals and plans. When the controller has been created it registers the planner module in the knowledge base and subscribes to registered agents, removed agents, goals and suggestion messages from the *Broadcaster*. When an agent is registered with the controller a *PlanningAgent* is created. Also the controller creates a goal in Jadex to create a new Jadex agent. A Jadex agent is created using an XML file which defines what beliefs, goals and plans the agent can have. Each part of the Jadex agent XML file will be described in the following sections. When a new Jadex agent is initialized an *AgentController* is created to manage the connection to the Jadex agent, so that it is possible to create goals, receive plans and access the belief base of that agent. The *PlanningAgent* is then connected with the *AgentController* so that the relevant Jadex agent is accessible to the agent. A remove agent messages both deletes the agent from the list of agents stored in the *PlannerController* and un-registers the agent in the Jadex system. The goal messages are, as mentioned in previous sections, a user triggered goal. The controller finds which agent this goal belongs to and dispatches it to the

PlanningAgent in question. Similarly, the suggestions are sent to the agent for processing. The *AgentController* also takes care of updating the Jadex belief base of the agents with some information stored in the knowledge base such as the location of the agent, what entities he sees and what he detects. Both the *PlannerController* and the *AgentController* extend the *Controller* class which encapsulates the access to the Jadex agents.

The *PlanningAgent* consists of a connection to its Jadex agent and information about the agent (*Agent*). The *PlanningAgent* has the ability to drop running goals and create new goals. Active goals can be dropped if a new conflicting goal of higher importance is generated. The agent has two ways of creating new goals, either a user triggered goal which is of the highest importance or a goal created from suggestions. The agent receives *SuggestionGroups* from the appraisal module. The suggestions are then evaluated to see which combinations of suggestions will give the most desired result. The agent also checks if it is possible to achieve the goal state defined in each suggestion. All suggestions that lead to an unreachable goal state are rejected. When the agent has selected which suggestions he wants to use they are marked as accepted and a new goal is created using the goal state of each suggestion. As soon as a valid plan has been generated from the suggestion goal, the planning agent sends a suggestion response back to the appraisal module, informing it of which suggestions were accepted and rejected.

4.4.1 Beliefs

```
<beliefs>
    <belief name="ctrl" class="AgentController"/>
    <belief name="id" class="BasicAgentIdentifier"/>

    <belief name="sees" class="String"  updaterate="500">
        <fact>$beliefbase.ctrl==null?"":$beliefbase.ctrl.getSeenEntities()</fact>
    </belief>
    <belief name="detects" class="String" updaterate="500">
        <fact>$beliefbase.ctrl==null?"":$beliefbase.ctrl.getDetectedEntities()</fact>
    </belief>

    <belief name="playerGoalData" class="GoalData"/>
    <belief name="playerGoal" class="String"/>
</beliefs>
```

Figure 4.14: Beliefs from a Jadex agent XML. Each belief needs ot define a name and the class type of its value. Beliefs can be defined to update itself automatically by setting an update rate.

The Jadex system provides each agent with a belief base which is used to maintain the agents world state. The belief base is defined in the Jadex agent XML file (see figure 4.14).

The belief base can both contain a single value beliefs or a belief set with multiple values for a single belief. Beliefs can also be updated both manually and automatically. Even though we mainly use an external knowledge base to store the agents world state, we also use the belief base. The belief base stores the connections between the Jadex agent and our agent classes through the *id* and *ctrl* belief variables respectively. The belief base is also useful as a trigger for goals in Jadex. We use the beliefs both when we want to trigger dynamic goals and reactive goals. The *playerGoal* belief is used when the *PlanningAgent* creates a new goal, which can be either a goal triggered by the user or a goal the agent creates from suggestions received from the appraisal module. The *playerGoalData* belief is an instance of *GoalData* and stores all information regarding the goal defined in the *playerGoal* belief. The reactive goals on the other hand are goals that are started by the Jadex agent when a certain condition is fulfilled. We can use the *sees* and *detects* beliefs to trigger such goals. For example an agent detects another agent but does not see it could mean that the other agent is behind him, a reactive goal to turn around could be created. The two perception beliefs are updated automatically every half a second by fetching data from the knowledge base using the expression defined as the *fact* of the belief. These facts can also be used to give a belief an initial value at startup.

4.4.2 Goals

```
<goals>
    <achievegoal name="Suggestion" exclude="never">
        <creationcondition>$beliefbase.playerGoal.equals("suggestion") == true</creationcondition>
    </achievegoal>

    <achievegoal name="GoToTarget" exclude="never">
        <parameter name="target" class="String" optional="true"/>
        <unique/>
        <creationcondition>$beliefbase.playerGoal.equals("goTo") == true</creationcondition>
    </achievegoal>

    <performgoal name="TurnAround" exclude="never">
        <deliberation cardinality="1">
            <inhibits ref="GoToTarget"/>
        </deliberation>
        <creationcondition>$beliefbase.sees.equals($beliefbase.detects) == false</creationcondition>
    </performgoal>
</goals>
```

Figure 4.15: Goals from a Jadex agent XML. This example has two types of goals achieve goal and perform goal. Goals define create condition which activates the goal when fulfilled. The deliberation configuration helps with the goal selection process.

As was mentioned in a previous chapter the Jadex system has four different types of goals. An example of the agent's goals can be seen in figure 4.15. The perform goal is an activity centric goal, where performing an action is most important, in contrast to the achieve goal

that seeks to achieve a certain world state. Query goal is used for information retrieval so if the knowledge required is already available no additional work is required. Maintain goal tries to maintain a certain world state and if that state gets violated the agent will by any means possible try to re-establish the desired world state. Currently we mainly use the achieve goal as we are usually looking to get to a certain world state. Perform goals are more suited for reactive goals for example to turn around when an agent senses something behind him. Maintain goals could be used if we wanted our agents for example to get a drink whenever they would become thirsty, then the state that the maintain goal would be trying maintain would be that the agent is quenched. As was mentioned in the beliefs section above the goals can be created using the belief base. As can be seen if figure 4.15 we use the *playerGoal* belief to create either a user triggered goal (*goTo*) or a suggestion generated goal (*suggestion*) depending on its value. The reactive goal to turn around uses the the changes in the *sees* and *detects* beliefs as a create condition. The goals can be configured to block each other based on how important they are. This is done by defining a *deliberation* with a *cardinality* value for how important it is and then listing up what goals the goal *inhibits*. The goals can also block creating other instances of themselves by defining them as *unique*.

Most of the goals we use are dynamic goals and for each one them we need to create a class. The class needs to extend the *DynamicGoal* abstract class which implements the Jadex planner *PlanningGoal* interface. Each goal class needs to define a function that checks for each step of the plan how close it is to fulfill the goal state. If the goal is unable to find a plan it marks all unattainable goal states so that if the goal is a suggestion goal a new set of suggestions that do not have the bad goal state variables can be tried to achieve the goal.

4.4.3 Plans and Operators

The Jadex system defines plans available to the agent in the agent XML file. Figure 4.16 shows how plans are defined in the Jadex agent XML file. In Jadex there are two kinds of plans, the predefined plans and the dynamic plans. Both types of plans need to create a class to represent them. The predefined plan classes need to extend the Jadex *Plan* class while the dynamic plan classes extend the Jadex *DynamicPlan*. Predefined plans, such as the *AgentInitPlan*, do not use the planner to find out what actions to perform. All actions of such plans have been set and are always performed in the same way and order each time. Such plans are normally only useful as simple reactive plans, such as to turn around, which only requires one action. The dynamic plans on the other hand

```
<plans>
    <plan name="SuggestionsDynamicPlan">
        <parameter name="time" class="int" direction="fixed">
            <value>50</value>
        </parameter>
        <parameter name="operator1" class="String">
            <value>"new DistanceSalutationOperator()"</value>
        </parameter>
        <parameter name="operator2" class="String">
            <value>"new FindTargetOperator()"</value>
        </parameter>
        <parameter name="operator3" class="String">
            <value>"new ApproachTargetOperator()"</value>
        </parameter>
        <parameter name="operator4" class="String">
            <value>"new CloseSalutationOperator()"</value>
        </parameter>
        <parameter name="operator5" class="String">
            <value>"new StartConversationOperator()"</value>
        </parameter>
        <parameter name="operator6" class="String">
            <value>"new StandUpOperator()"</value>
        </parameter>
        <parameter name="operator7" class="String">
            <value>"new OrderDrinkOperator()"</value>
        </parameter>
        <parameter name="operator8" class="String">
            <value>"new ServeOrderOperator()"</value>
        </parameter>
        <parameter name="operator9" class="String">
            <value>"new FetchOrderOperator()"</value>
        </parameter>
        <body>new SuggestionsDynamicPlan()</body>
        <trigger>
            <goal ref="Suggestion"/>
        </trigger>
    </plan>

    <plan name="init">
        <body>new AgentInitPlan()</body>
    </plan>
</plans>
```

Figure 4.16: Plans from a Jadex agent XML. A dynamic plan needs to define which operators can be used to generate a plan. All plans need to define which class is used when constructing a plan. A plan can also define which goal triggers the plan and how much time it can take to find a plan before aborting the.

use the Jadex planner to find out which actions to take. In Jadex the actions of a plan are called operators. Each operator can in fact issue more than one action to the virtual environment. The plans are triggered when a goal becomes active by linking the plans with the goal using the *trigger* tag.

Each dynamic plan needs to define what operators can be used in the plan. This is done to make it easier for the planner to find plans. If the planner only needs to check the operators relevant to the active goal instead of checking all possible operators a lot of time is saved and the plan returned more quickly. Saving time is extremely important when planning in a real time environment as it would reduce the believability of the scene if an agent would stop every now and then while a new plan is being generated. The *Suggestion* plan on the other hand needs to have access to all available operators as it is unknown which operators are relevant. It can also be configured how long the dynamic plans have to come up with a plan. The plan classes need to handle what to do when a plan has been found, when a plan failed or when a plan was aborted. The plan classes also construct a planning domain so that a plan can be tested without affecting the world state. An *AgentDynamicPlan* super class, which extends the *DynamicPlan* was created to encapsulate that functionality so that each planning class only needs to implement logic specifically for their needs. The *SuggestionDynamicPlan* calls the *PlanningAgent* to reevaluate the suggestions when no plan has been found.

Each operator is a class that implements the Jadex *Operator* interface. The operator needs to implement three functions. First it needs a function to get the description of the domain it is working in. Secondly a function which updates the planning domain with information regarding what happens when the operator is executed. This function also defines the preconditions of the operator so that the operator will not be tried until all of its conditions have been fulfilled. Finally the operator defines a function to execute its actions in the virtual environment when a plan has been found. An operator is selected as a part of a plan if it effects the planning domain in such a way that in gets the plan a step closer to achieving the goal state.

Chapter 5

Results

There are various ways to evaluate our resulting software. We can evaluate the system as a whole or we can evaluate each part of the system separately. Even though a lot has been accomplished, the system is by no means complete as will be discussed in the next chapter. The ultimate contribution of the system is the support for increased believability of agents without the user needing to micro-manage their behaviours. So believability is what we would want to evaluate in the end. Just as important though is evaluating the architecture of the system. The architectural qualities of the technology delivered should be evaluated with the following questions in mind: is it easy to use, can it be extended, are the modules independent of each other and how does the system perform.

5.1 Believability

Believability is something that people sense when they see characters interacting within the context of a particular environment. The observed behaviour needs to fulfill certain expectations based on the norms established by the game or story. Evaluating believability is not something that can be easily measured with standard tools. It can only be evaluated by capturing the sensations and experiences of users of a finished interactive environment. Do our agents look more believable than before we applied our approach? This is the question we would like to answer when evaluating the increase in believability.

One of the major dangers when creating believable characters is that they will fall into the Uncanny Valley (Mori, 1970). The Uncanny Valley is a hypothesis that when a robot, or in our case an agent, looks almost human but not quite, then the observer will feel slight discomfort towards it. This feeling is magnified when the human-like entity starts

moving. As our system has not yet been connected to a high quality graphics and animation engine with life-like human agents, being concerned about hitting the the Uncanny Valley would be premature as well as comparing it to state of the art game systems. However that is something that should be done when those standards of graphics have been achieved.

For now we will compare the current system with the system prior to the addition of our work. Believability is hard to evaluate and strongly depends on the quality of the generated agents and their animation. At the current state a full formal evaluation has not been performed but should be done in the future in a controlled environment using independent human subjects to rate the system's believability. We will instead start by taking a close look at the behaviours generated by the system and discuss how well they meet our expectations. We will do this through informal tests on few specific scenarios. The scenarios are used to evaluate how well the system handles the original requirements of having agents that personally appraise and react to situations, dynamically plan their actions, follow social rules of the society they are in and show emotions as well as use them when making decisions.

5.1.1 Test scenarios

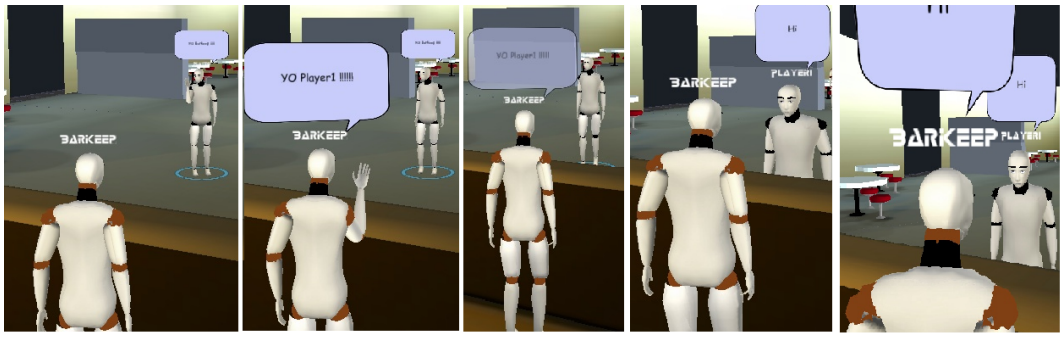

Figure 5.1: A scenario where an avatar starts a conversation with an NPC. From left to right: (1)An approaching avatar turns towards the NPC and greets by waving to him. (2) The NPC greets back. (3)The avatar continues to approach the NPC. (4)Avatar stops close to the NPC and issues a close salutation. (5) The NPC greets back and the conversation can start.

In our first scenario the avatar is supposed to start a conversation with an NPC. Previously the avatar's user would click on the NPC he wants to talk to, the avatar would go straight up to the NPC and start a conversation. With the addition of the planner and appraisal module a more detailed approach is automated. As can been seen in figure 5.1 the avatar starts by turning towards the NPC, in this case the bartender, and waves his hand (a distant

salutation), he then waits for a response and then starts approaching the NPC. When the avatar has moved to within a certain distance he stops and greets him again by bowing his head slightly and saying 'hi' (a close salutation). The NPC reciprocates the gesture and the conversation can start. The emotional part of the system can influence the situation in few ways. It can show visual cues of the emotional state or how an event affects the agents. For example the avatar might become happy and smile when his salutation is reciprocated by the NPC. Also if the NPC would be angry or not like the avatar he might decide to ignore him instead of greeting him back.

Player	Bartender	Bob
GOAL: Order drink (User) → *Approach Bartender*		
Suggestions Created		**GOAL: Greet Agent Goal (User)** → *Greet Player(Event triggered)* Expectation created
GOAL: Greet (Suggestion) → *Greet Bob(Event triggered)*		Expectation fulfilled
→ *Approach Bartender (cont.)*		
→ *Greet Bartender(Event triggered)* Expectation created Expectation fulfilled	Suggestions created **GOAL: Greet (Suggestion)** → *Greet Player (Event triggered)*	
→ *Order drink(Event triggered)* Expectation created	Suggestions created	
	GOAL: Serve (Suggestion) → *Make drink*	
Expectation fulfilled Suggestion created	→ *Serve drink(Event triggered)*	
GOAL: Say Thanks (Suggestion) → *Thank Bartender*		

Figure 5.2: A step-by-step of what goes on in the system for a scenario where an avatar orders a drink.

In the second scenario, which we will explain in more detail to show the inner workings of the system (see figure 5.2), we have three agents, the player controlled avatar, a bartender NPC and another avatar called Bob. In this scenario the player would like to order a drink from the bartender. The user starts by giving his avatar an order drink command, which creates a new *user goal*, the avatar creates a plan how to achieve that goal and his first action is to approach the bartender. As the player's avatar is approaching the bartender, Bob's user issues his avatar a command to greet the player's avatar and a *user goal* is created for Bob. Bob creates a plan and executes the only action of the plan, greet player. This action triggers an *event* which creates a *expectation* for Bob to be greeted

back and *suggestions* for the player to respond to that event. The suggestions creates a new *suggestion goal* as the player's avatar decided to follow the social rule and greet Bob back. Now the player has two goals but as they do not conflict, his plans are dynamically altered to fulfill both goals. The agent has a new action planned to greet Bob which is executed. This also triggers an *event* and that event fulfills Bob's *expectation* of being greeted back. This interaction between the player's avatar and Bob affects the two avatars socially and emotionally both internally and towards each other. That Bob greeted the player's avatar made the player's avatar like Bob a little more as well as improve his mood a little and the same for Bob when the player's avatar fulfilled his expectation by greeting him back. The player's avatar has now fulfilled his *suggestion goal* and can continue on with the previous *user goal* of ordering a drink. The player's avatar continues his approach towards the bartender. When the player's avatar is close enough to the bartender the next planned action is taken which is to greet the bartender, this triggers a very similar situation as happened between Bob and the player's avatar as the bartender greets back. The third and final action, order drink, of the original goal is then executed and that goal is now completed. This action triggers an *event* which creates an *expectation* for the player's avatar to get a drink and *suggestions* on how to respond for the bartender. The bartender decides to do the acceptable thing and serve the order so a new *suggestion goal* is created. This goal requires a plan of two actions to make the drink and to serve it to the player's avatar. The latter actions triggers an event. This *event* fulfilled the player's avatar *expectation* of getting a drink and this makes him very happy. The *event* also creates *suggestions* for the player's avatar and as he is quite happy about getting his drink and he starts to smile, he also decides it is the socially acceptable thing to thank the bartender. The player's avatars creates a new *suggestion goal* and thanks the bartender.

The third scenario is a duplicate of the second scenario except now the player's avatar does not like Bob. Here we will only describe in detail what differs from the scenario above. The player gives his avatar a command to order a drink. The avatar starts approaching the bartender. Bob's user gives a greet player command, Bob greets the player's avatar and as before this triggers an *event*. Bob creates an *expectation* to be greeted back and *suggestions* are created for the player's avatar on how to respond to the event. The player's avatar generated the same set of suggestions however as he now dislikes Bob, he decides based on his emotions to not select the socially acceptable way to greet him but instead selects to ignore Bob, so no new goal is created. This means that Bob does not receive any event to fulfill his expectation of being greeted back. When the deadline for the expectation to be fulfilled passes Bob's attitude towards the player's avatar worsens and he shakes his fist at him in fury. The player's avatar however just continues on with his goal to order a drink.

The final scenario is again a duplicate of the second scenario except now the social rule of how to respond to a greet is to stop and have a chat with the other agent. The player's avatar still has the goal to order a drink, so he starts by approaching the bartender. Bob then greets the player which triggers an *event* which creates an *expectation* for Bob as in previous scenarios and *suggestions* for the player's avatar, however now as is defined in the social rules for this scenario the player's avatar must also stop and have a chat with Bob. Here we encounter a problem as the new goal and the order drink goal can not be achieved simultaneously as was the case in the second scenario but the avatar still tries and gets stuck in a limbo between both goals.

These scenarios show that the agents can personally appraise and react to situations as could be seen when Bob greeted the player's avatar while he was approaching the bartender. The player's avatar appraised that event, it affected him emotionally and socially and he responded by greeting Bob back in one scenario and in another he decided to ignore him. The agents can create complex plans and are able to alter them if the situation they are in demands it. The final scenario however shows one of the limitations that the system currently has, which is how to dynamically handle conflicting goals. The scenarios also show on several occasions that the agents can follow the social rules of the society they are in as long as they are encoded as social rules. The agents are also able to show emotional responses as when the player's avatar smiled when he received his drink and also Bob shaking his fist in fury when the player'a avatar ignored him. The emotions also play a part in deciding what actions to take as happened when the player's avatar decided not to greet Bob back but to ignore him as described in the third scenario.

5.1.2 Informal User Evaluation

The system was given to a few naive users for an informal evaluation, where they were free to walk around the environment populated with NPCs and strike up conversations with them, and they were asked to comment on their experience. These informal tests indicate that the addition of the planning and appraisal module gives an impression that the agents are following some sort of social protocol. In addition to the agent's awareness of being in a conversation, as in the previous system, the agent now seems to plan ahead and interact with other agents in a more complex and what we consider a more believable manner.

Compared with the social norm improvements the emotional aspect was not immediately recognised by the users. The visual aid for the display of emotion is that the agent will smile or frown depending on the situation, at first glance this can be quite hard to see if

the user is not looking directly at the face. The facial expression also stays until the agent is no longer happy or angry so the expression is not highlighted by appearing and then going away in quick succession. The emotional state is also more internal which is hard to evaluate. Moreover when users perform an action that has not yet been configured for the planner or the appraisal module in forms of rules, actions, goals and plans the believability of the agents takes a hit. Also when users figured out what affects the emotional state of the agent they started to abuse that knowledge which should be taken into consideration when configuring and improving the system, as players in a massive multiplayer game would most likely also abuse such limitations. Implementing some sort of short term memory helps the agent to prevent repetitive actions, but more work needs to be done in the future in that area.

5.2 System architecture and performance

5.2.1 Architecture

One of the goals of this project was to create a basic system for dynamic planning that could be easily extended. The system can be extended in various ways, such as by adding new rules to the appraisal modules, adding operators, plans and goals to the planner. It is also quite easy to plug an altogether new module into the system, it only needs to implement the communication interfaces and from there it can receive and send information to the other modules.

Adding new rules to the appraisal module can range from being quite easy to being required to alter the other modules as well. A new rule that will only need to use data and actions that are already in the system, will only have to be added to the rule configuration file. A more complex rule which adds completely new behaviour in the agents and uses data not being generated and stored already, might have to add new goals, plans and operators to the planner, add new animations and steering behaviours to the virtual environment and add data to the knowledgebase or a new way to query existing data. Even though it sounds difficult to add a new complex rule to the system, it is in fact not so. Adding and retrieving data from the knowledge base is done through well defined interfaces so adding new methods to retrieve or store the new data is quite easy. Adding new concepts, relations and functions to the knowledge base is done by adding them to the system's initialization file of the knowledge base. Same goes for the planner, adding new operators, plans and goals requires an update to a configuration file for the planner agents. A small class for each new operator, plan or goal is also needed. These classes need to

implement an interface that the planner uses when constructing and executing a plan. If a new operator is created a new command might be needed to be sent and received by the virtual environment and acted out by the agent. This requires adding a new command message to the communication layer, a method that handles that message on the virtual environment side and then adding animations and possible even new steering behaviours to the agent.

The modules of the system were also designed to be independent of each other. This is achieved by having all communication between the modules go through a communication layer so that if one module is not running the other modules will not stop. This also provides the possibility to easily replace a module with a new one. For example a new planner could be added in place of the current one. All that would be needed is for the new planner to implement the communication interfaces and no alteration would be needed to the other modules. As was mentioned above this also makes it easy to integrate a completely new type of module to the system.

5.2.2 Performance

For a system to be able to function in a massive multiplayer game, it needs to be able to handle lots of data and perform its tasks in a fast and reliable manner. Currently the virtual environment in use does not perform too well with high a number of agents so that overall system performance test has not been performed, however some performance measurements have been performed on various sub-parts of the system. The tests were all performed on a Dell Latitude D820 laptop with a 2.0 Ghz Intel dual core processor and 2 GB of RAM.

We performed time measurements on the parsing of the expressions in the rules configuration file in the appraisal module. We also measured the time it takes to evaluate them as it needs to be evaluated each time it is invoked as data might have changed while an expression only needs to be parsed once. As can be seen in table 5.1 parsing the expressions takes the longest time but as the expressions are only parsed when the system starts it has low impact on the performance of the system. The evaluation of the expression is more important and being able to evaluate 100.000 expression in less then a second is very acceptable. In the current system there are currently 6 complex expressions (similar to those measured) and around 30 simple expressions. The complex expressions are evaluated each time an event occurs for each agent involved with that event and the simple expression are evaluated more often.

Table 5.1: Rule parsing and evaluation time measurements

Case	X 1.000	X 10.000	X 100.000
Parsing	203 ms	1.406 ms	13.234 ms
Evaluating	15 ms	109 ms	825 ms
Evaluating & Parsing	234 ms	1.531 ms	14.594 ms

Inserting and retrieving data from the PowerLoom knowledge base was tested for both the virtual environment and the planner and appraisal module. The virtual environment connects to the knowledge base from a python client to a java XML-RPC interface and the other two modules connect using Java API calls as both the modules and the knowledge base run as Java programs. The results shown in table 5.2 demonstrate some concerns regarding retrieval of data from the knowledge base. Also there was a big concern regarding inserting data from the virtual environment but that was tackled by sending the data to the knowledge base interface in chunks, that is all data for every agent was gathered together and sent as one big chunk over the XML-RPC connection and then on the Java side each insert command is then sent to PowerLoom through the Java API. As can be seen in table 5.3 this method shows considerable improvement. The virtual environment mainly adds data to the knowledge base so that the bad performance in getting data is irrelevant at this time, but might be a problem at later stages if the environment will depend on data created by other modules. The planner and the appraisal module only fetch data when they need it except for the planner which gets information on what each agent detects every 200 milliseconds and that is too infrequent to be of any concern.

Table 5.2: Inserting and retrieving agent position from knowledge base measurements

Case	X 1.000	X 10.000	X 100.000
Java retrieve	250 ms	1.187 ms	9.875 ms
Java insert	63 ms	328 ms	2.672 ms
Python retrieve	2.503 ms	24.365 ms	239.907 ms
Python insert	2.485 ms	23.061 ms	228.150 ms

Table 5.3: Inserting chunks of data to the knowledge base measurements

Case	X 1.000	X 10.000	X 100.000
Python chunk insert	80 ms	818 ms	7.902 ms

An important feature of the system is the sending and receiving data through a socket communication between the virtual environment and the appraisal and planner modules. As was mentioned in chapter 4 we use a Google Protocol Buffer to encapsulate the data

before sending it over the socket connection. Table 5.4 shows that in a system with massive numbers of agents some alterations might be needed to increase the performance of the socket communication, for example by putting the data into chunks before sending. In the current implementation data is mainly sent from the planner to the virtual environment in the form of commands and sending events from the environment to the appraisal module. Both of these messages are sent only when an agent needs to perform an action or has caused something to happen by performing an action, so the average agent is rarely sending more then a few messages each second.

Table 5.4: Sending and receiving data through a socket connection

Case	X 1.000	X 10.000	X 100.000
Sending & receiving	492 ms	3.657 ms	37.360 ms

The time it takes from creation of a goal to finding an executable plan takes around 50 milliseconds for the most complex plans currently available in the system, such as the one described in the scenario in previous chapter, to around 10-20 milliseconds for easier plans. A complex plan can consists of needing to select 5 or more different operators from 9 available operators to achieve the goal. The current performance of the system gives no cause for concern, however it has not been tested using more than 15 agents due to the limitations of the virtual environment.

Chapter 6

Conclusion and Future work

6.1 Contributions

This project contribution is primarily in the field of interactive social game environments. The designers of these game environments have been over the last few years focusing on creating picture perfect environments with realistic looking characters and agents with impressive game play AI. Even though agents provide high quality game play, there is something missing as they move around and interact with the environment and other agents. The agent does not react and behave in a manner that is appropriate according to the agents game world and this affects the users' ability to immerse themselves in the game.

The work done as part of this thesis is an addition to previous and ongoing work done by Claudio Pedica and others on CADIA Populus, which is a tool to design social situations in an interactive virtual environment, where the agents contribute and react to social stimuli around them, making them look more aware and life like. Our addition extends the reactive behaviours already in the system, with more complex social scenarios that require planning. These social scenarios are achieved by introducing social norms into the agents as goals that need to be achieved. The agent appraises the environment and the situations he is in and uses the rules of his social norms to interact with other agents in an appropriate way by executing plans created by the planner. Emotions have also been added to the system. These emotions affect the agents in the way they behave and which actions they take. Having the addition of social norms and emotions the agents will now be able to alter their actions due to events occurring in the virtual environment. The user's experience will improve as the agent will perform actions in accordance with social rules

familiar to the user, which will help inform the user of what kind of social situations his avatar is in.

6.2 Limitations and Future work

As this work should be considered a contribution to a larger project, still in progress there are several limitations to what can currently be delivered. The idea behind the appraisal module comes from the appraisal theory of Lazarus and to achieve the full power of that theory more effort must be put into adding more coping strategies. Currently only planning based coping has been implemented. In addition some work needs to be done on the appraisal of an event in a similar way as is done in EMA and was described in a previous chapter. The calculations done in the appraisal module on how an event affects an agent, as well as the calculations of the desirability of the suggestions generated from an event are quite simple and could be improved through further research.

The current emotional module can easily be manipulated to invoke certain emotional state as was mentioned in the evaluation chapter. This could be fixed by improving the calculations as mentioned above as well as by introducing discounting of emotions. That is the value of emotions will, over time, approach neutral state again.

The expansion of the emotional spectrum is needed. The current implementation only has two emotions, anger and joy. The addition of some basic emotions such as fear, surprise, sadness and disgust would increase the emotional believability of the agents.

The system's current limitations of handling the creation of a new goal when another goal is currently active needs to be addressed in such a way that the more important goal will be put into focus and the second goal will be dropped or suspended if it is possible to go back to that when the primary goal has been achieved. Also it should be checked if both goals can be run at the same time, by creating a plan that fulfills the goal state of both goals.

Architecturally the system's use of several different communication protocols is something that should be addressed to make the connection between the modules simpler. This could be achieved by using black- or whiteboards for messages between modules. The boards could also be used as a knowledge base and therefore eliminating the need for another 3rd party system. This could also help spread the modules and even parts of a module to more machines for better scalability.

6.3 Conclusion

Even though the system is not perfect it gives a really good starting point for implementing emotions and social norms into massive multiplayer game environments. It is possible to extend the current modules for years to come. The architecture of the system also provides a good support for adding completely new modules to it. The three modules we have currently included provide us with a system where agents are able to show emotional responses to dynamic events occurring in the environment. The agents can perform complex social behaviours and react to the social behaviours of other agents. The user can give his avatar high level commands, who will then try to achieve that goal in a socially believable manner. This system also provides us with an excellent platform for further research into integrating human behaviours into autonomous agents.

Bibliography

Aylett, R., Dias, J., & Paiva, A. (2006). An affectively driven planner for synthetic characters. In D. Long, S. F. Smith, D. Borrajo, & L. McCluskey (Eds.), *Icaps* (p. 2-10). AAAI. (conf/aips/2006; 2007-11-23)

Cassell, J., Bickmore, T., Campbell, L., Vilhjalmsson, H., & Yan, H. (2001). More than just a pretty face: conversational protocols and the affordances of embodiment. *Knowledge-Based Systems*, *14*, 55-64. (ID: 257; written)

Chalupsky, H., MacGregor, R., & Russ, T. (n.d.). Powerloom manual [Computer software manual]. (Available at http://www.isi.edu/isd/LOOM/PowerLoom)

Fikes, R. E., & Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189-208.

Frank, J., & Jónsson, A. K. (2003). Constraint-based attribute and interval planning. *Constraints*, *8*(4), 339-364. (2004-02-12)

Gillies, M., & Ballin, D. (2004, July 19-23). Integrating autonomous behavior and user control for believable agents. In *Autonomous agents and multi-agent systems* (p. 336-343). ACM Press. (ID: 481)

Gratch, J., & Marsella, S. (2004, 12). A domain-independent framework for modeling emotion. *Cognitive Systems Research*, *5*(4), 269-306.

Hall, E. T. (1966). *The hidden dimension*. New York, NY: Doubleday.

Hill, R. W., Gratch, J., Marsella, S., Rickel, J., Swartout, W., & Traum, D. (2003). Virtual humans in the mission rehearsal exercise system. *KI Special Issue on Embodied Conversational Agents*, *03*(4), 5-10. (ID: 466)

Hofstede, G. H. (2001). *CultureŠs consequences : comparing values, behaviors, institutions, and organizations across nations*. California: Thousand Oaks, Calif : Sage Publications,.

Jónsson, A. K., McGann, C., Pedersen, L., Iatauro, M., & Rajagopalan, S. (2005). Autonomy software architecture for lorax (life on ice robotic antarctic explorer). In B. Battrick (Ed.), *Proceedings of the i-sairas 2005*. AG Noordwijk, The Netherlands: ESA Publications Division.

Kendon, A. (1982). *Conducting interaction: Patterns of behavior in focused encounters*. Cambridge University Press.

Lazarus, R. S. (1991). *Emotion and adaptation*. New York: Oxford.

Marsella, S. C., & Gratch, J. (2009). Ema: A process model of appraisal dynamics. *Cognitive Systems Research*, *10*(1), 70-90. (Modeling the Cognitive Antecedents and Consequences of Emotion)

Mascarenhas, S., Dias, J., Enz, S., & Paiva, A. (2009). Using rituals to express cultural differences in synthetic characters. In S. Decker Sichman & Castelfranchi (Eds.), (p. 305-312). IFAAMAS.

Mehrabian, A. (1996). Pleasure-arousal-dominance: A general framework for describing and measuring individual differences in temperament. *Current Psychology*, *14*(4), 261-292.

Mori, M. (1970). The uncanny valley. *Energy*, *7*(4).

Ortony, A., Clore, G. L., & Collins, A. (1988). *The cognitive structure of emotions*. Cambridge University Press.

Pedica, C. (2009). *Spontaneous avatar behaviour for social territoriality.*

Pedica, C., & Vilhjálmsson, H. H. (2009). Spontaneous avatar behavior for human territoriality. In Z. Ruttkay, M. Kipp, A. Nijholt, & H. H. Vilhjálmsson (Eds.), *Iva; lecture notes in computer science* (Vol. 5773, p. 344-357). Springer. (conf/iva/2009; 2009-09-21)

Pedica, C., Vilhjálmsson, H. H., & Lárusdóttir, M. (2010). Avatars in conversation: The importance of simulating territorial behavior. In J. M. Allbeck, N. I. Badler, T. W. Bickmore, C. Pelachaud, & A. Safonova (Eds.), *Iva* (Vol. 6356, p. 336-342). Springer.

Pokahr, A., Braubach, L., & Lamersdorf, W. (2005a). A goal deliberation strategy for bdi agent systems. In T. Eymann, F. Klügl, W. Lamersdorf, M. Klusch, & M. N. Huhns (Eds.), *Mates; lecture notes in computer science* (Vol. 3550, p. 82-93). Springer. (conf/mates/2005; 2006-01-26)

Pokahr, A., Braubach, L., & Lamersdorf, W. (2005b). Jadex: A bdi reasoning engine. In G. Weiss, R. Bordini, M. Dastani, J. Dix, & A. Fallah Seghrouchni (Eds.), (Vol. 15, p. 149-174). Springer US.

Pokahr, A., Braubach, L., Walczak, A., & Lamersdorf, W. (2007). Jadex: Engineering goal-oriented agents. In D. G. Fabio Luigi Bellifemine Giovanni Caire (Ed.), *Developing multi-agent systems with jade.* Wiley.

Si, M., Marsella, S., & Pynadath, D. V. (2006). Social norm models in thespian: Using decision theoretical framework for interactive dramas. In *Proceedings aisbŠ06* (p. 70-77). AISB.

Silva, L. de, & Padgham, L. (2004). A comparison of bdi based real-time reasoning and htn based planning. In G. I. Webb & X. Yu (Eds.), *Australian conference on artificial intelligence; lecture notes in computer science* (Vol. 3339, p. 1167-1173). Springer. (conf/ausai/2004; 2004-12-10)

Smith, C. A., & Lazarus, R. S. (1990). Emotion and adaptation. In L. A. Pervim (Ed.), (p. 609-637). NY: Guilford Press.

Thangarajah, J., Padgham, L., & Harland, J. (2002). Representation and reasoning for goals in bdi agents. In M. J. Oudshoorn (Ed.), *Acsc; crpit* (Vol. 4, p. 259-265). Australian Computer Society. (conf/acsc/2002; 2004-08-24)

Thorisson, K. R. (1999). A mind model for multimodal communicative creatures & humanoids. *International Journal of Applied Artificial Intelligence*, *13*, 449-486.

Tomlinson, B., & Blumberg, B. M. (2003). Alphawolf: Social learning, emotion and development in autonomous virtual agents. In (Vol. Vol. 2564, p. 35-45). Springer Berlin / Heidelberg.

Traum, D. R., Swartout, W. R., Marsella, S., & Gratch, J. (2005). Fight, flight, or negotiate: Believable strategies for conversing under crisis. In T. Panayiotopoulos, J. Gratch, R. Aylett, D. Ballin, P. Olivier, & T. Rist (Eds.), *Iva* (Vol. 3661, p. 52-64). Springer.

Vilhjalmsson, H. (1997). *Autonomous communicative behaviors in avatars.* (ID: 414; Media Arts and Sciences)

Vilhjalmsson, H. (2004). Animating conversation in online games. In M. Rauterberg (Ed.), *Entertainment computing icec 2004.* Berlin: Springer.

Walczak, A., Braubach, L., Pokahr, A., & Lamersdorf, W. (2006). Augmenting bdi agents with deliberative planning techniques. In R. H. Bordini, M. Dastani, J. Dix, & A. E. Fallah-Seghrouchni (Eds.), *Promas; lecture notes in computer science* (Vol. 4411, p. 113-127). Springer. (conf/promas/2006; 2007-09-10)

Weld, D. S., Anderson, C. R., & Smith, D. E. (1998). Extending graphplan to handle uncertainty & sensing actions. In *Aaai/iaai* (p. 897-904). (2002-01-03)

School of Computer Science
Reykjavík University
Menntavegi 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.reykjavikuniversity.is
ISSN 1670-8539