Interaction in Multi-Agent Systems

Thomas Kjærgaard Malowanczyk

Kongens Lyngby 2014
Compute-BSc-2014

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet, building 303B,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk Compute-BSc-2014

Summary (English)

The goal of the thesis is to implement emotion in a multi-agent system to improve the interaction between human and computer. This is done through examining existing theories and models of emotions used to produce a logical formalisation of emotion for the basis in creating a framework used in the agent-programming language in GOAL. This framework gives an agent the capabilities to experience 22 emotions and express them both through visual representation and natural language. An agent is produced with the framework implemented along with an environment with the focus on interaction with a user in order to test the framework. The new agent is able to experience 18 of 22 emotions possible in the framework and can express these emotions both through language and visual expression.

Summary (Danish)

Målet for denne afhandling er at implementere følelser i et multi-agent system for at forbedre interaktion mellem menneske og maskine. Dette er gjort ved at eksaminere eksisterende teorier og modeller for følelser. Disse er brugt til at lave en logisk formalisering af følelser og til at udvikle et framework som er brugt i agent-programmeringssproget GOAL. Dette framework giver en agent mulighed for at opleve 22 følelser og er i stand til at udtrykke disse følelser både igennem visuelle repræsentationer og naturligt sprog. En agent er produceret med dette framework samt med et environment med fokus på interaktion mellem agent og bruger for at teste frameworket og er i stand til at oplever 18 af de 22 mulige følelser samt udtrykke dem.

Preface

This thesis was prepared at the Department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfilment of the requirements for acquiring an BSc in Software Technology.

The main subject for this thesis is artificial intelligence but it is however not my first time working with AI. At my third semester I had the course 'Introduction to Software Technology' where we had 3 projects during the course and one of the projects was to make a simple tic-tac-toe game, and after it was made I spend large amount of hours programming an AI for the game. The AI was very simple but it did provide some challenge to the player, however if the player was a bit smart the AI could be tricked into a situation where the player would always win. The next AI I made was last year in a project where a larger software program was to be produced. My group was given the task to produce an AI for a strategy game which was intended to compete against 2 other groups. Sadly the competition never took place but we did manage to develop an AI capable of dealing with multiple goals and analysing its surroundings. However these AIs was not that advance or intelligent and simply consisted a series of algorithms and conditions.

I have since taken the courses 'Introduction to Artificial Intelligence' and 'Logical Systems and Logical Programming', and in parallel to this thesis I also had a course for learning to program in GOAL, an agent-programming language used to make multi-agent systems with intelligent agents. With these three courses I had the basis for developing more intelligent and rational AI's.

Emotions, however, was not determined to be the focus at the beginning of the thesis as all I received at the start was the title; "Interaction in Multi-

Agent System" and the master thesis [Spu13] as inspiration. The master thesis examined interaction in organized-oriented multi-agent system and focused on modelling a theatrical performance. In this thesis the agents was implemented to show simple emotions but it was a very small part of the thesis and not explored fully. It was here that I got the inspiration to implement emotions in agents used for interaction in multi-agent systems.

Lyngby, 01-July-2014

[illegible]

Thomas Kjærgaard Malowanczyk

Acknowledgements

I would like to thank my supervisor Jørgen Villadsen for accepting me as one of his many bachelor student projects on such a short notice and his support through the project. He has provided me with material in order to establish what this thesis should be about and assisted in finding relevant material for the subject.

I would like to thank Salvador Jacobi in the time we spent understanding and learning to program in GOAL and his input and contribution at the initial part of the project.

I send my thanks to Drude Hargbøl Hundevadt for proofreading the thesis and contribute with inputs for improvements.

Finally I want to thank all the people who have listened and shown interest in the project which has maintained my motivation and interest in the subject.

Contents

Chapter 1

Introduction

In today's world we see that an ever increasing focus in the development of software has been in the area of human-computer interaction (HCI). Especially since smartphones have been developed, where the interaction between the user and the software has grown. Especially since the user interaction more directly with the software instead of through real world interfaces such as buttons. The next step in HCI is already starting to emerge as it starts to merge with the field of artificial intelligence (AI) or what appears to be AI.

Apple launched Siri in 2011, which is a personal assistance for the Iphone that is operated through conversational interface as Siri is able to understand natural language. This means that request or commands can be given to Siri to operate the device such as make phone calls, set reminders and navigations to a desired destination. This form of personal assistance is not exclusively found in Apple products as android devices now has the same functionalities as Siri. Microsoft has recently come out with their version of Siri called Cortana and their personal assistance avatar is taken from the their game ”Halo” where in Cortana is a portrayed as a human like AI, assisting the protagonist/player on his quest. Unlike Siri, Google's and Microsoft's versions has improved the aspect of ”personal assistance” as they are able to learn the users habits and interests in order to provide relevant information to the user. The idea of having an AI personal assistance is starting to be an reality and the movie ”Her” released in 2013 gives an image of where this is taking us. In this movie the protagonist, Theodore, acquires a new talking operation system equipped with a human-like AI with feelings designed to adapt and evolve to its user. Theodore and his new OS develops a romantic relationships and the movie centralize about the possibility of human-AI relationships.

In the gaming industry the idea of giving AI emotions is also part of the future. The gaming industry has grown tremendously the last decade and with it the demand for better graphics and more immersive gameplay[Cho]. One of the aspect to make more immersive gameplay is in the AI and as the computation power on the new generation of console has increased, the option to make more complex AI has also increased. The game ”The Last of Us” released in 2013 relies heavily on complex AI as the player is followed by an AI named Ellie throughout the game and one of the key things for this AI was to make it believable in the way it acted in the environment in order to not break immersion[Dyc]. However the direct interaction with the player is not in focus as most of the interaction is scripted, so when a condition is reached an interaction is played out. Besides interaction, emotion can also be used in the AI's decision making which will remove it from the cold rational thinking and give it a more human touch. It is believed that to improve the immersion in games the next step is to improve the AI's behaviour and one way is to simulate emotions[Lyn].

Complex AI able to show emotions is not just useful to improve immersion in game but also in e-learning. With the overflow of devices such as tables we see that the these devices are becoming entertainment for kids but they are also used as a learning tool. Most e-learning tools consist of a tutor that interacts with the user and making a believable AI that is able to recognize and respond with emotions desired as it can improve the users emotions and in turn increase the users learning capabilities. This is evident as emotions plays a large role in human learning and decision and are closely tied to a person decision making [ME12].

In order to implement emotions in AI it is important that we understand how emotion in human works and this has been a subject for many year. Already back in the 1950 have there been effort to understand emotion and through the year multiple theories have been proposed, some of them with wild differences of what emotions are. However these theory are made from a psychological approach and does not align with any form of computational approach. But the last few decades models have been proposed using cognitive psychology of appraisal wherein emotions are extracted from the evaluations of events, actions and objects but still far from the conventional AI consisting of complex algorithms but demands more human like structured intelligence.

An agent is considered intelligent if it is able to sense its environment and act upon it, yet it does not define human-like intelligence. In order to make more sophisticated intelligence, formal logic has been proposed that provide ways to handle data through knowledge representation and reasoning which uses natural language. Formal logic uses inference to derive a logical conclusion and provides ways for computers to reason about existing information. Besides logical formalization an agent can be build with the BDI-model which is a model of human practical reasoning. This is designed for programming intelligent agents wherein the agent is equipped with beliefs, desires and intention. The agent then has beliefs about it's environment, desires of what it wants to obtain and from these two it can derive intention or action to act upon the environment. This type of agents provides a good foundation of implementing emotions as it give the agent capabilities to evaluate events, actions and objects.

The BDI-model has been the basis for new areas of artificial programming and especially in the field of agents and multi-agent systems where new agent-programming language has surfaced such as Jason and GOAL. These language makes a perfect candidate for emotions but the next is to translate emotions into logical formalization that agents can understand and use. But how close will the agents emotion be to a humans emotions and is it able to express the same variety as a human?

The thesis deals with;
understanding theories and models of emotion and create a logical formalization that can be applied to agent-programming language.
Develop and evaluate an agent with the capabilities of showing emotions, empathising on the interaction between human and computer.

The thesis begin with two sections on how to program in multi-agent system in the agent-programming language GOAL and establishing the theories and models of emotions. These are followed by two sections where a logical formalization of emotion is produced and used to program a framework made in the agent-programming language GOAL. Lastly, an agent along with an environment is made from the framework in order to test the framework. The thesis ends with a discussion of the implementation of emotion along with the agent and environment and a conclusion of the thesis.

Chapter 2

Multi-Agent System

Multi-agent system also known as MAS is a computational system wherein multiple agents are connected to an environment. These agents work together in order to solve problems faster than a single agent can or even solve problems that requires more that one agent.

But what is an agent?
In [RNC$^+$10] an agent is defined as anything that can perceive in its environment through sensors and act in this environment through actuators. However it is a very loose definition that doesn't tell much as anything that can sense and act can be seen as an agent. In the article [WJ95] the term *agent* is distinguished by two notions, a weak notion and a stronger notion.

The weak notion of an agent is defined by the following four properties.

- *autonomy*; meaning that the agent acts without any intervention from outside
- *reactivity*; the agent perceive the environment and responds to the changes that may occur in this environment
- *pro-reactivity*; which means that the agent take the initiative to perform action instead of just responding to the environment

- *social ability*; simply meaning that the agent interacts with other agents in the environment

With this, the previous definition is expanded as it simply consisted of the term *reactivity*. The new definition fits much better within MAS as the property of *social ability* is a key element for multiple agents.

The stronger notion of an agent is used mainly in the field of AI where an *agent* is defined as a computer system containing the same four properties as the weaker notion but is also implemented using human-like concepts. These concepts can for example be emotional agents using affective computing or an agent programmed with the the BDI-model.

It should be noted that neither the weak or strong notion of an agent exclude humans for being defined as an agent since they follow the four properties. This means that when developing an MAS it could just as easily be agent to agent interaction as agent to human interaction.

Another well used concept is the BDI-model as mentioned before, which will be the basis of the agents used in this paper. In this model the agent has three mental attitudes; beliefs, desire, and intention, hence the name BDI.

- *Beliefs* represents the agents beliefs about how the world is.
- *Desires* represents the agent desires describing states that the agent would like to reach.
- *Intention* is the agents commitment to follow a plan to obtain its desires. These plans are derived from the agents beliefs and desires

The big advantage of the BDI-model is that the beliefs and desires of the agent can easily be written in first-order logic suited for declarative programming language such as GOAL.

2.1 GOAL

Currently there exists a large amount of agent based modelling softwares. Some of these are focused on multi-agent systems. Some are mentioned in [SD02], but only a few have agents implemented after the BDI-model, e.g. Jason, Jadex

```
environment {
  env = "environment.jar".
  init = [variable1 = 5, variable2 = true].
}

agentfiles{
  "agent1.goal".
  "agent2.goal".
}

launchpolicy{
  when [type = agentType, max = 1]@env do launch agent1:Agent1.
  launch agent2:Agent2.
}
```

Figure 2.1: Example of a `mas2g` file

and GOAL. In this paper the agent programming language that will be used is GOAL.

GOALs main feature is that it uses the logical language Prolog so the agent has both declarative beliefs and goals which makes it an intuitive language to write in and easy to develop in. This paper will not go in depth with how to program in GOAL but it will go through the very basic of GOAL and for a more detailed guide the reader is referred to [Hin14].

A GOAL Program is defined in a MAS module file with the extension `.mas2g`. This file is very basic and defines the environment that the system uses and what agents that are connected to it. An example of a MAS module can be seen in 2.1.

In the `.mas2g` example we can see that it consist of three blocks.
The first block *environment* defines what environment that is loaded from a `.jar` file the system are to use and also defines some initial variable that the environment will start up with. It is however not necessary to have an environment in order to develop a MAS in GOAL.

The second block *agentfiles* defines what agent files with the extension `.goal`, are to be used in the system and in the example we see that there are two agent files that are going to be used in the system.

The third block is *launcpolicy* that defines a policy between the the agent and the environment in the system. These policies consists of conditions for how the agent should be connected to the environment and its entities, however an agent can be launched without being connected to an environment. In the example we see that the MAS file launches the two agents, the first is to be connected to the environment with the condition that if the environment has an entity with the type `agentType`, then there can be a maximum of 1 of that agent type. The second are launched in the system without being connected to the system. It is also possible to connect multiple agents to a single entity in the environment, this is well suited for an entity composing of multiple independent systems that can communicate with each other such as a robot.

2.1.1 Agent

Agents in GOAL are implemented after the BDI-model as mentioned previously, this means that the agent has a set of beliefs and desires it uses to define intentions to obtain these desires. In GOAL an agent's mental state is composed of a knowledge base, belief base, and a goal base which will be discussed further on. With the previous definition of an agent in chapter 2 it should be able to react to the environment it is situated in and in so should be able to percepts it's surrounding. For that the agent is equipped with a percept base that contains informations received from the environment the agent is connected to. The agent is also required to have social abilities and in GOAL agents can send messages to each other and to receive or send these messages the agents is equipped with a mail-box.

An agent should be able to react to the environment by executing actions and to derive what action to perform which requires the ability to check the agent's mental state. GOAL gives the mean in the form of *action rules* that uses a *mental state condition* in order to decide what action to perform. To be able to inspect the agent's mental state GOAL has some inbuilt predicate that queries the agent mental state using mental atoms. There are two type of mental atoms and one is **`bel`**(φ) that queries the agents belief base and knowledge base and checks if the condition φ is in either of these bases. The second mental atom is **`goal`**(φ) that queries the agents goal base. It should be noted as GOAL uses Prolog the contents of the mental atoms must be a valid Prolog query. To be able to inspect the percept base the predicate **`bel`** is also used, the only change is that a the query is written as `percept`(φ) (e.i **`bel`**(`percept`(φ))) specifying to GOAL that the percept base it be to inspected. It is also possible to check if a condition φ is not in the base by negating the query using the predicate **`not`** like **`not`**(**`bel`**(φ)).
These mental atoms can be used to build a mental state condition that are a conjunction of mental literals.

In GOAL there are three forms of action rules.

if <mental_ state_ condition> **then** <action>.

This action rule is the most basic and and simply says that if the mental state condition is true then perform the action.

forall <mental_ state_ condition> **do** <action>.

This action rule is similar to the first but instead of executing the action rule once, the action is executed for every instances where the mental condition succeeds. This rule is mainly used when updating the belief base with received percepts from the environment as the agent can receive multiple percepts with the same predicate.

listall <Listvar> ← <mental_ state_ condition> **then** <action>.

This action rule is somewhat similar with **forall** but instead of executing the action rule for every time the mental state condition succeeds, it instead stores all variables in the mental state condition for which it succeeded in a list that can be used in `<action>`

Nesting these action rules is possible as the `<action>` can just be a block containing new actions rules.

```
forall <mental_state_condition> do {
  if <mental_state_condition> then <action>.
}
```

To update the agents belief base, GOAL as two built-in actions; **insert** for inserting new information into the belief base and **delete** for removing information from the belief base. The same can be done with the goal base where the action **adopt** insert a goal to the goal base and **drop** to remove a goal from the goal base. Yet the **drop** action is not used that often as goals are automatically removed from the goal base when they are achieved which happens only when both the belief and goal base has the same literal. This is called *blind commitment strategy* as the agent is committed to achieving their goals and should not drop goals if there aren't a valid reason.

An agent file `.goal` can consist of modules, which are used for sectioning the agent file. An agent is composed of at least three modules; *init*, *main*, and *event* module. The *init* module is the first module that is executed when the environment starts and it is here that initial beliefs and goals are set. After that, a cycle starts where each cycle where the event section is executed first followed by the program section. The main module is where the agent decides what action to perform from it's knowledge, beliefs, and goals and the event module is where mostly beliefs are updated.

Each module may contain the following five sections;

knowledge
Here the agent is given the domain logic, e.i. facts, it has of the environment it is situated in. This section is written in pure Prolog syntax and the rules are static and cannot be changed at run time.

beliefs
This section gives the agents its initial facts of how the world is and unlike the knowledge sections these facts can be changed at run time.

goals
This section is designed to give the agent initial goals but is mainly used in the init module as it is only executed once.

actionspec
This section specifies what action the agent can perform, each action has a pre- and post-condition where the pre-condition are a mental state condition of the belief base that has to succeed before the agent can perform that action. The post-condition contains a conjunction of literals and are facts that are inserted into the belief base.

program
Here a set of action rules is defined, governing the strategy of the agent and executing action from the *actionspec* section based on this strategy.

In 2.2 an example of how an agent file looks is given and this agent has the simple task of going to work. From the example it can be seen that the agent has the modules init, main, and event module and in the init module, all of the 5 modules are present.

A program section can follow a specific rule evaluation order deciding how the rules are to be evaluated and executed as more than one action rule may be applicable to be performed. In the main module where the agent decides what action to perform only one action should be executed, however in the event module, multiple action rules should be performed as it is desired to update the belief base with all the received percepts and messages in the same cycle.

This paper will only be using the following two of the rule evaluations that is possible in GOAL:

Linear; With this order the action rules are evaluated in the order they are

```
init module{

  knowledge {
    weekend(saturday).
    weekend(sunday).
  }

  beliefs{
    home.
    day(tuesday).
  }

  goals{
    work.
  }

  program{
    if bel(percept(day)) then insert(day).
  }

  actionspec{
    gotowork{
      pre{home}
      post{not(home), work}
    }
  }
}

main module{
  program{
    if bel( day(D), \+weekend(D) ) then gotowork.
  }
}

event moduel{
  if bel(percept(day), day) then delete(day) + insert(day).
}
```

Figure 2.2: Example of module

written and the first rules in which the mental state condition is true the action is performed and no further evaluation is performed and exits the program section. This order is default for the main module.

Linearall; Much like the linear order this order goes through each action rules in the order they are written but instead of only executing one action, all action rules where the mental state condition succeeds is executed. This order is default for the init and event module.

Beside the three mentioned modules that are default additional modules can be made, either in the agent file or in a new file called module file with the extension `.mod2g`. Unlike the default modules these new modules needs to be called from the agent file in one of the default modules and are simply called as an action in an action rule using the modules name.

```
if true then NewModule
```

To use a module file in the agent file the module first needs to be import which is done with the command `#import NewModule` at the top of the agent file.

GOAL also provides a way to easier define mental state conditions with the use of macros. An macro is a function defined at the top of the program section with the command `#macro` and is used to define conjunction of mental literals into easier understandable commands. Using the example in 2.2 the mental state condition in the event module can instead be made with a macro.

```
program{
  #macro workday(D) bel(day(D), \+weekend(D))

  if workday(D) then NewModule
}
```

A macro should not be confused with a knowledge function as a macro can only contain queries to the agents belief base and can't contain prolog functions and a macro function is used outside a mental atoms, e.i **`bel`** and **`goal`**.

2.1.2 Environment

As mentioned before the system the agent is situated in can be an environment which is defined in the `.mas2g` file and is provided by a `.jar` file. GOAL uses a environment interface standard (EIS) to connect between GOAL and the environment and has a set of requirements in order for the environment to work with the interface. The proposed EIS and it's requirements to develop an environment using EIS can be read in [BHD11] and [BHD].

The EIS has the task of linking the agent in the agent-programming language (APL) with free controllable entities in the environment, but it's main task is to act as a medium in which the APL and the environment communicates. The EIS also has the job of connecting the Environment Management System (EMS) which provide actions to control the environment between the APL and environment. This controller can either be in the environment or the APL but in this case GOAL controls the EMS. These controls could be initializing the environment using initial configurations as mentioned before or be able to start, pause, or end the environment along with pausing individual entities and killing the connection between agent and entity. As mentioned the environment contains the representation of the state of the world and provides the APL with ways to interact with the world and sense it. It is the environment task of regulating what each agent can perceive in the environment and providing the correct information when responding to a percept request.

The advantage with the EIS is that there is no general rule for how the environment is developed, the only thing that is required is implementing methods defined by the EIS interface. This means that how the environment looks and works is up to the developer. The methods that is required for EIS to be connected are the following;

```
public void init(Map<String, Parameter> parameters)
```

This method is used by the EMS to initialize the environment with the parameters given in `.mas2g` as mentioned previously.

```
Percept performEntityAction(String entity, Action action)
```

This method is when the agent has performed an action. The environment should then alter the current state by the received action. The environment may also provide a respond to the action in the form of a percept. The received action can contain `Parameters` that can be of four types defined by EIS, `Identifier` which is the same as strings, `Numeral` says it self, `Function` that can have the form "name(arg1,arg2)" and last `ParameterList` that is simply a list of parameters.

```
LinkedList<Percept> getAllPerceptsFromEntity(String entity)
```

Here an entity has requested a percept and the environment responds with a list of percepts that the agent can perceive in the current state. The percepts, defined in a class called `Percept` provided by EIS, holds the name of the percepts in the form of a string and may contains any number of `Parameters`.

```
boolean isSupportedByEnvironment(Action action)

boolean isSupportedByType(Action action, String type)

boolean isSupportedByEntity(Action action, String entity)
```

As action are defined in the environment the EIS should be able to tell if the received action from an agent is supported by the environment. This is where these functions comes in, the first is all the actions that is in the environment, the second is the actions that entities of certain type can perform and the last is what a specific entity can perform. This gives the ability to provide different entity types and individual entities with their own set of actions.

Chapter 3

Emotions

This chapter will first review existing work and theories of emotions along with proposed models for emotions in order to find the most suited for modelling emotions in GOAL.

3.1 Existing work

In the field of computational modelling of emotion there have been multiple and widely different approaches as there are currently no generally accepted theory of emotions, however, there is an acceptance that such emotional states as joy and fear are normal reaction to perceived events and prospect of events [RHD$^+$]. This means that even though the approaches are different the final product is somewhat similar in the emotional states that are expressed. Another reason for the different approaches is the multitude of intended function in the emotion in an agent, these functions can generally be classified into three categories [RHD$^+$].

Informational or epistemic function of emotions
Here the emotions are seen as informational such as changes in the agent or information related to an object or an event that is shared between agents.

Attention or resource-allocation function of emotions
Here the agent uses the emotion to focusing on relevant events currently affecting the agent in order to use more resources to process and deal with these events.

Motivational function of emotions
Here the agent uses the emotions in its decision making by having maybe a hedonistic desire to avoid negative and only produce positive feelings.

As this thesis focus on the human-computer interaction, then it will only deal with informational function of emotions but it should be possible to extend it further to one of the other categories.

The different theories about how emotions functions can also be divided and has the following three disciplinary groups;

Physiological theory of emotions which states that activities within the body that responds to events will elicit emotions e.g. the event of immediate danger result in the physiological responds of elevated heart rate which in turn elicit the emotion fear.

Neurological theory of emotions defines emotions as hard-wired in the brain and that it is the activity within the brain that elicit emotions.

And lastly *cognitive theory of emotions* states that it is the thoughts, memory and mental activity that elicit emotions.

Since the thesis works with artificial intelligence in MAS and the agents are modelled after the BDI-model that follows the cognitive theory of emotions.

3.1.1 Appraisal theory

An aspect of cognitive theory is the appraisal theory that has the theory in which it is the evaluations of events that causes specific reactions in people. An example of this is a student who is taking an exam and if the event is perceived as positive then the student may feel joy, happiness, or even anticipation as the event may be long term such as the student receiving top grade, finishing his education with high remarks and potentially landing a great job. Appraisal theory has two basic approaches which gives an explanation for the appraisal of emotions and how these emotions can develop.

One of these basic approaches is structural model of appraisal which splits the appraisal process up in to two categories, primary and secondary appraisal. In the primary appraisal the person is to evaluate an event by the motivational relevance and its congruence to ones goals. The former is how relevance the event is to the persons needs and is shown to influence the intensity of emotions and the latter being the how these events align with the persons goals [SK09]. The secondary appraisal focus on the person evaluations of their resources such as who is to blame, as a person may blame himself or another or a group of people. Another focus is the person's ability to cope with emotions or problem-focused coping which refers to that person ability to take action and change the situations to align with ones goals [SK09].

Besides using appraisal as an approach other uses arousal as an approach to emotion as stated before with the physiological theory of emotion where the bodily function are a key element. Other theory states that arousal and emotions are interchanged and are both equally part of emotions [Onl].

3.2 The OCC model

In 1988 a book titled The Cognitive Structure of Emotions written by Ortony, Clore and Collins [OCC88], explored if the cognitive psychology could provide a foundation for the analysis of emotions. A model of emotions was proposed defining three aspect to which humans react emotionally; consequence of event, action of agents and aspects of object. This model was named OCC model taken from the first letters of each author and is a widely accepted cognitive appraisal model for emotions. In this model 22 emotions was defined into 6 groups, that are systematically structured as seen in figure 3.2.

Beside structuring the emotions the OCC model also defines intensity of the emotions in order to make a computational tractable motions. Each emotions has a set of variables that defines the intensity of emotions such as the desirability or likelihood of an event, the effort in attaining an event or the praiseworthiness of an action.

Emotions in the OCC model are structured after an affective reaction being either positive or negative so each emotion has a related opposite reaction. This can be seen in figure 3.1 where in the aspect consequence of event, the agent can either have a pleasing or displeasing reaction to an event, in actions of agents the agent can have an approving or a disapproving reaction to an action and in aspects of objects the agent either likes or dislike an object.

Consequence of events can then be split up in to two branches where the event is either focused on the agent itself, consequence of self, or another agent and

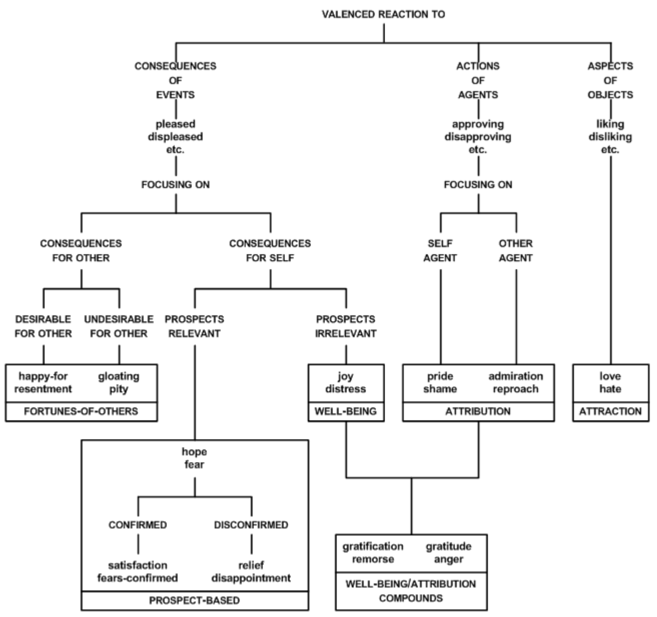


Figure 3.1: Structure of emotions in the OCC model. [OCC88, p. 19]

consequence of other. Consequence of others give rise to the first group of emotions called fortunes of others where an event can either be seen as desirable or undesirable and this group contains four emotions.

The aspect, consequence of self, is a bit different as it also deals with the prospects of an event and this give rise to two branches as the prospect of an event is either seen as relevant or irrelevant for the agent. The branch for a irrelevant prospect contains the group called *well being* which only contains two emotions and these emotion are the default case of being pleased or displeased about an event and does not consider prospect as part of the emotion. The other branch regarding relevant prospects leads to the group *prospect based* and deals with events that the agents looks forward to and contains six emotions. The first two are *hope* and *fear*, these can then be confirmed or dis-confirmed leading to four new emotions.

Going back to the branch, action of agents, the agent can either approve or disapprove an action. This aspect leads to two new branches where the agent in focus is either the agent it self or another agent but both of these branches leads to the same group called *attribution* which contains four emotions. However two of the emotions are relating to the the agents action and the other two relates to the action of another agent.

These emotion can be combined with the emotions in the well-being group to produce a new group *well-being/attribution compounds* where an example would be that if the agents performs an approving action leading to the the agent having the emotion pride and that action produces a pleasing event resulting in the agent feeling joy, these emotion will then elicit the agent feeling gratification.

The last aspects, aspects of objects, is very simple, the agent can either like or dislike an object and that leads to the last group *attraction* that contains two emotions, *love* and *hate*.

Implementing these emotions also requires an intensity as as themselves they will only produce an agent where each emotions has an equal value which is not suitable as agent should feel a more prominent feeling of joy by winning 1 million dollars than finding 10 dollars on the street. To implement intensity each emotion has a set of variables affecting the intensity of emotions, some of them are global and affects all emotions and others a local.

Global variables
The OCC model introduces four global variables, *sense of reality*, *proximity*, *unexpectedness* and *arousal* , where the first defines how the agents feels that the event, action or objects seems real. Example of this is that an event can seem unreal to an agent at first which will not produce an action, but it is first when the agent has come to terms with reality that an emotion is produced.
The second variable, *proximity*, is the proximity of time which says how close to the present an event, action or object are. If an agent is first told of an event years latter then it should not give rise to a large intensity of emotions compared than an event that is happing in the present.
Unexpectedness defines if an event, action or object was unexpected meaning that the agent has never considered and event or action to happen at all.
The last variable is *arousal* and unlike the others that are mainly cognitive, this variable deals with the agents physiology. If an agent experience negative events such as burning the toast for it's morning breakfast and forgetting to make coffee, these events will most likely increase the agents arousal and give rise to a feeling of frustration. This frustration and increased arousal may produce more intense action and emotions where the agent may pour it's frustration on objects or other agents.

Local variables
Turning to the local variables each of the three aspects discussed before (e.i. event, action, object) has a central variable used in all the relating groups. For event based emotions the central variable is *desirability* which defines how much the event is considered desired or undesired for the agent. For the aspects of actions the central variable is *praiseworthiness* that tells if the action is to be praised or not. The last aspect regarding objects has the central variable is

appealingness which is straightforward and defines if a object is appealing or not.

Continuing in the event based emotion the intensity of the emotions in the group fortunes of others are affected by the variables *desirability for other*, *deservingness*, and *liking.* For example if an agent feels sorry for another agent failing an exam, the intensity is determined by the desirability for the other agent to succeed, if the agent deserves to succeed depending on how much effort he has put in to it or if the agent likes the other agent. Or it may simply be if the agent is concerned with the other agents well being the agent has a *desirability* for the agent to succeed.

Looking at the prospect based group the two first emotions, hope and fear, the *likelihood* of an event to happen is the only variable besides the *desirability* of an event to affect these emotions. Looking at the more specialised emotion resulting from hope and fear, such as satisfaction and relief, the two variables, *effort* and *realization* along with the intensity of the relating original emotion will affect the new emotion. *Effort* is simply the effort the agent has spend on realizing the event, so an agent spending a large amount of effort in making a event happen should feel a an intense emotion than if no effort is spend. The other is *realization* which defines the degree of which the event is realized meaning that if an event is only partially realized a lesser intensity should arise than if the event is fully realized. An example could be for an agent to clean the house before his quest arrives but does not manage do complete the task fully. Then the he should not have the same intense feeling of satisfaction than if he completed the task fully. This variable, however demands that the event can be partially realized.

The well-being group has only one variable that affect the related emotions which is the central variable *desirability.*

The emotions in the attribution group has the central variable *praiseworthiness* and the two variables, *expectation deviation* and *strength of unit.* When dealing with such emotions it is not necessarily emotion that arise from the agents own action. it could very well be from the action from an agent or organization that the agent has a strong association with. A mother can be proud of it's child for accomplishing a great goal or a worker in a company may feel shame if the division he works makes a mistake but he is not directly to blame, however, he has a strong affiliation with his division. It is the *strength of unit* that determines if the the action of an agent should be seen as part of on self and in so feel pride/shame. The *expectation deviation* variable is when an agents action deviate from what was expected of an agent, or an agent in a specific role. For example a person would most likely admire another stranger for performing CPR on a person experiencing cardiac arrest more than if it was a paramedic.

For the well-being and attribution compound group no new variable is introduced as the intensity is simply derived by the attending emotions from the well-being and attribution group.

The last group attraction has a single local variable that defines the intensity of its emotion besides the central variable *appealingness* called *familiarity*. The idea is that if an agent has an increased exposure to an objects the more familiar it would become and in so the intensity of the liking or disliking are increased.

Each variable also have assigned a *weight* to them as an emotion with multiple variable then each variable is not sure to have the same precedence yet the model does not go into detail how these weights are determined. These variables and their *weight* is not the only deciding factor for emotions as the model also introduces a threshold. Each emotion has a context sensitive thresholds to ensure that emotions are only experienced if they exceeds this threshold and can be used to introduce the concept of moods in the agent. An example would be when a person is in good mood due to previous positive emotions, it's most likely that lesser negative events or action would not give rise to a negative respond in that person. Using threshold we can lower or increase the threshold value to visualize emotions depending on the emotion the agent is feeling, so a series of positive emotion can increase the value of some negative thresholds and reduce the value of some positive thresholds and vice versa. Besides filtering out low intensity emotions, the thresholds also impact the intensity so the actual intensity value is the amount that an emotions intensity exceeds the threshold.

Example from the OCC model of calculating the intensity value of the emotion joy would be the following pseudo-code where the potential intensity of the emotion is computed first.

```
if DESIRE$(p, t, e) > 0$ then
    set JOY-POTENTIAL$(p, e, t)$ = $f_j$[|DESIRE$(p, e, t)$|,$I_g(p, e, )$]
end if
```

Where DESIRE(p, t, e) returns the value of desire a person p, assigns to a perceived event e, at the time t. I_g is the function that returns the value of the combined global variables. f_j is the function that returns the intensity value for the emotion joy.

Next the potential intensity is checked against the related threshold.

```
if JOY-POTENTIAL$(p, e, t)$ > JOY-THRESHOLD$(p, t)$ then
    set JOY-INTENSITY$(p, e, t)$ = JOY-POTENTIAL$(p, e, t)$− JOY-THRESHOLD$(p, t)$
else
    set JOY-INTENSITY$(p, e, t) = 0$
end if
```

Here we see that if the potential intensity does not exceed the thresholds then the intensity it set to zero in order to indicate for the system that an emotion of joy is existing but the agent didn't experience it.

Another usefulness for the intensity is in the case of the agent expressing its emotion in a language as each emotion can have a token that reflects the current intensity of an emotion. So if the agent is feeling low intensity of joy it may use the token "pleased" or "glad" but if it was a high intensity it could use "ecstatic".

3.2.1 Revisited model

However hard they tried to make the OCC model computational it did contain ambiguities and the lack of logical approach has made it harder to implement. Looking at 3.1 we see that the model is a flow diagram of how the emotions are constructed but from a computer science perspective the figure could be formed as a inheritance diagram, especially since that joy and distress are default emotions for the branch "consequence of self".

Through out the book they use "desirable event" but looking at it we are not interested in the event but rather the consequence or outcome of an event. An earthquake in it self is not of relevance but rather the consequence it produce has such as the damage it carries or lost of life.

Another problem is in the Prospect-based emotions where fear can lead to disappointment and satisfaction but neither of these two emotions spawns from fear as both arise from a desirable event. The same can be said about hope as neither relief or fear-confirmed has anything to do with hope.

The fortunes of other emotions can be related to the well-being emotions since "happy-for" is when an outcome is considered desirable for others and in so must also to some degree be desirable for on self and the thus the feeling of "joy" is experienced. For example, being happy for another for completing a difficult task since they desired it also means that it was desired that they succeeded an happy that they did. This means that the term focusing on and prospect irrelevant is not suitable any more and logical incorrect, since "happy-for" implies "joy" then the focus is also on oneself.

The aspect of objects is also considered to be in need of improvements as there are no conditions to distinguish love/hate from the generalized liking/disliking. For approving/disapproving, pride/shame and admiration/approach is differentiated by who performed the action and for pleased/displeased the emotions hope/fear and joy/distress is differentiated by the one being a prospect and the other an actual consequence. love/hate does use familiarity but is only used to further the intensity where the more familiar an agent is with an appealing object the more it is loved and the more familiar a unappealing object the more it is hated.

In 2009 a revisited model of the OCC model was proposed by three computer scientist in order to create a standardized interpretation of the psychological OCC model for other computer scientist wishing to formalize or implement emotions [SDM09]. This revisited model removed and clarified the ambiguities mentioned above along with other ambiguities and restructured the model to a more logical structure in order to make the model easier to implement and work with.

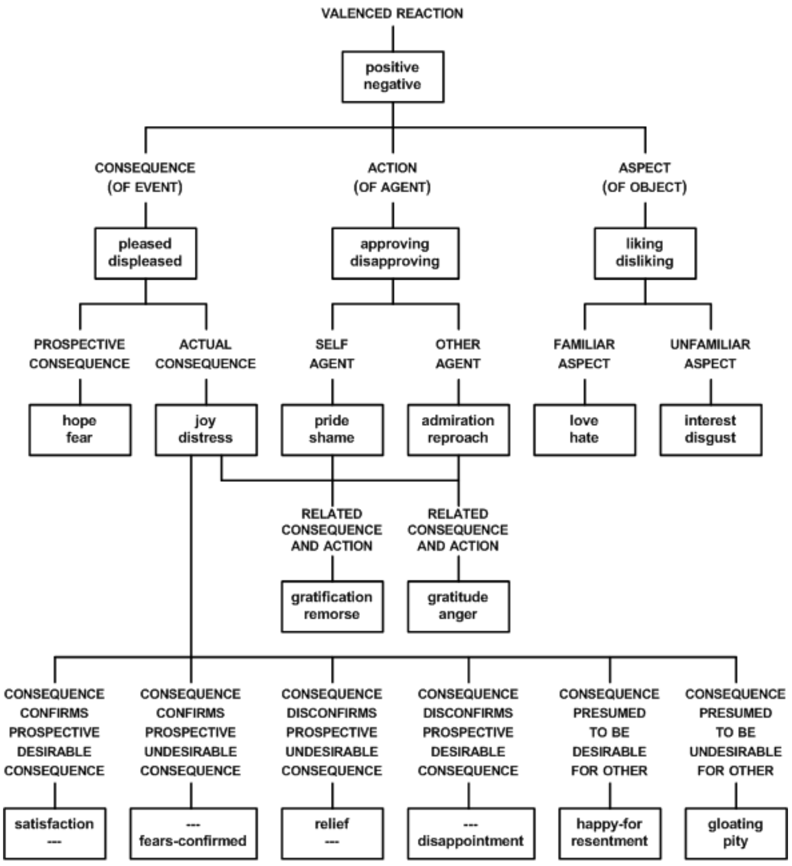


Figure 3.2: Revisited structure of emotion in the OCC model. [?]

The new inheritance model of the OCC model seen in figure 3.2 have removed the idea of groups and many of the emotions are now split up. All of the mentioned ambiguities are solved so now the emotions such as satisfaction and relief along with happy-for and resentment or now specializations of the feelings joy and distress. With that, hope and fear are now for themselves but as it is an inheritance diagram then hope is still not excluded as logical part of satisfaction. The aspect of objects are also more defined with the aspect of familiarity introduces which give rise to two new emotions interest/disgust where familiarity is used to distinguish between love/hate.

Chapter 4

Modelling Emotions

Multiple articles have constructed a logical formalized emotions for BDI-agents using the OCC model as a basis. All of these framework establish a syntax with a language containing predicates similar to the predicates in a BDI-agent and GOAL-agent such as **Bel** for beliefs and **Goal** for goals of an agent.

Meyer who also proposed the revisited OCC model had formalized emotions before using the original OCC model along with an existing an framework called KARO in 2 different papers [Mey06, SDM07]. In both papers the formalization was focused heavily on the agents planning and action as the formalization has definition for the agents commitment to a plan along intention of accomplishing a goal through execution of a plan. However the formalizations was made with the OCC model in mind yet the only defined emotions in the article [Mey06] were **Happines**, **Sadness**, **Anger** and **Fear**. The [SDM07] formalization had even less as it only dealt with 2 emotions, **Hope** and **Fear**.

Another formalization was made by Herzig and Longin based on the original OCC model in 2006 ([AGHL06]) and refined in 2009 ([AHL09]). Both paper defines operators with temporal logic such as $(H\varphi)$ that reads "φ has always been true" and $(G\varphi)$ that read "Henceforth φ is going to be true" in order to define all the 22 emotions in the OCC model. A unique operator is also defined called **Expect** which simply defines if an agents expects an event and is used to in the logic for prospect based emotions e.i. hope and fear.

Another paper, [GLL$^+$11], formalized emotions with very few operators unlike the other two formulations. It is also based on the OCC model but only implements 12 emotions with only a few of them the same as the OCC model, besides formalizing emotions a wide variety of expressions between agents are made. In this paper three operators deviate from the other formalizations: **Ideal** defining the moral state of the agent, **Cd** defining the the agents action and choices and **Exp** defining that an agent express a formula to another agent. The operator **Cd** is used to define a new operator called **Resp** which states that an agent is responsible for an outcome.

4.1 Formalizing the OCC model

Based on the OCC model and the different proposed formalization of the OCC model, a new logical framework will be proposed in trying to formulate emotions in the revisited OCC model. However, as the formalization will be used to implement emotions in agents in GOAL it has to be compatible with GOAL. But some of the proposed intensity variables in the OCC model can't be implemented in GOAL in it's current form as GOAL lack the function needed to define the some of the variables.

There are generally two approaches the OCC model can be implemented in a BDI-agent, one is a form of interpretor for the agent state of mind in to emotions. Implemented in GOAL means that the agents state of mind is interpreted each cycle and the emotions intensity are calculated each time and checked with the threshold in order to determine if they are still relevant. However has GOAL removes goals when they are achieved then the agent will only be able to feel emotions such as satisfaction and relief as it is possible to see an prospected event has been realized but any recurring cycle will not be able to see this making it an unsuitable approach for GOAL. The other way is to use the state of mind to create persistent emotions in the agents state of mind where the emotions are given a intensity when they are realized and will slowly decay until they are not relevant any more.

Emotions regarding appraisal of agents action make use of the variable "Strength of unit" and this variable requires that the agents has some sort of model between the agent it self and others defining social groups or organizations such as family, workplace, friendships. This model does not exist in GOAL and implementing it in GOAL is out of this scope, however previous work has been done in this area such as [Spu13] which could be used in order to implement this variable.

Another variable that cannot be realized in GOAL is the variable "Deviations of the agent's action from person/role-based expectation", i.e the unexpectedness of an agent. In order to have this variable the agent is required to have a model of other agents or place it self in another agents position and attempt to predict what that agent would do in its given position and be able to compare it with what the agent actually did and be able to quantify it

The same is the problem with the global variable "Unexpectedness". It is possible if it was just used to decide if something was expected or not but as it used as a variable the agent should be able to quantify the unexpectedness.

The "proximity" variable is somewhat possible to implement with linear temporal logic but it requires that the outcome of events has a timestamp so it is possible to determine the proximity to the agent but the added information will make it difficult to store in GOAL as it needs to store the timestamp as well as info of the event.

Additionally it was proposed in the OCC model that the variables for emotions should be weighted as they don't have the same impact on an emotion but never stated what this distribution should be, so in an attempt to limit the amount of work in this thesis all variables are weighted equally.

Love and hate along with interest and disgust is not going to be part of the logical formalization as these emotions requires first order modal logic to be implemented as stated in [AHL09], however an alternative implementation for love and hate will be proposed later.

The last variable that will not be present in the framework is the global variable "arousal" as this is a physiological state of a person and as the agent is purely cognitive. Such a physiological state can be simulated in the agent but is out of the scope of the thesis.

Removing these variables from the framework would reduce the correctness of the intensity but will not render the intensity for the affected emotions obsolete as these emotions has other variables that factor in to their respectively intensity.

Finally, as neither GOAL or Prolog support temporal logic then the logical formalization will refrain from using these.

Based on these limitation a new inheritance model for the emotions for this thesis can be seen in figure 4.1.

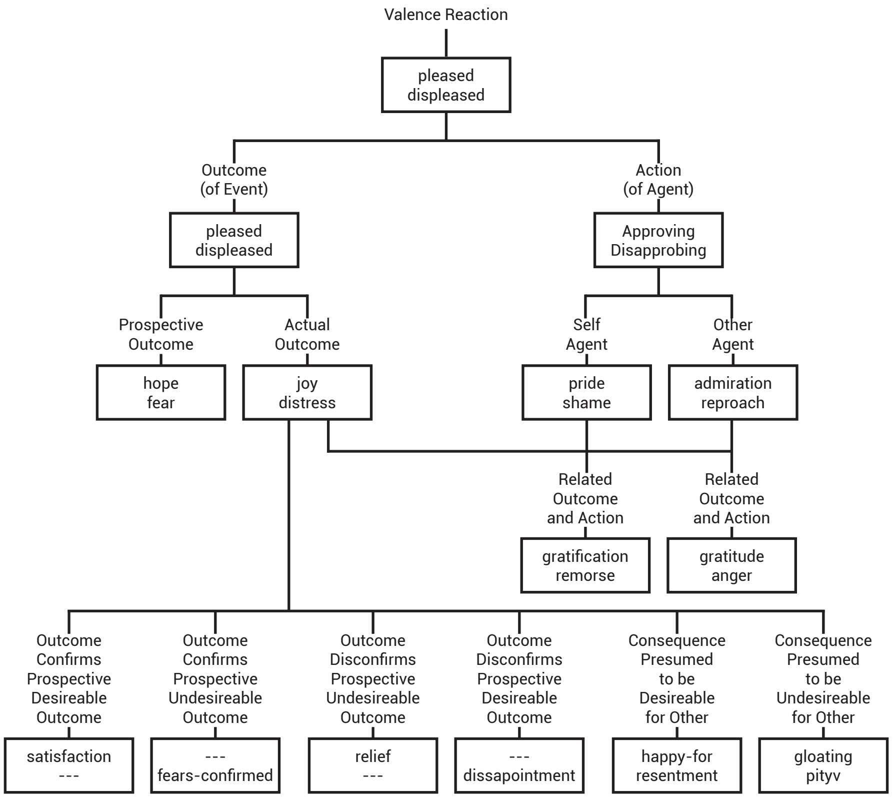


Figure 4.1: The structure of emotion for the framework

4.2 Logical Formalization

The logical formalization will be based mostly on the formalization proposed in [GLL$^+$11] as it avoids most of the limitation in GOAL as it doesn't uses temporal logic. So the formalization will be using the operators **Ideal** and **Resp** besides the operators **Bel** and **Des** which are standard for all formalizations. Another predicate will be used in order to implement prospect based emotions which is the operator **Expect** defined in the papers [AGHL06, AHL09].

It should be noted that the formalization uses the notion consequence or outcome instead of event as mentioned in section 3.2.1 as it is not the event that is interesting for the agent but the consequence/outcome.

The framework will then consist of the following language $\mathcal{L}$ defined in by the

BNF;

$$\varphi ::= p|\neg\varphi|\varphi \wedge \varphi|\mathbf{Bel}_i\varphi|\mathbf{Des}_i\varphi|\mathbf{Undes}_i\varphi|\mathbf{Ideal}_i\varphi|\mathbf{Resp}_i\varphi|\mathbf{Expect}_i\varphi$$

where p ranges a set of propositions, i,j ranges over a finite non-empty set of agents.

The operators $\mathbf{Bel}_i$ represent agent i's beliefs

$\mathbf{Des}_i$, $\mathbf{Undes}_i$ represent agent i's desires. Others such as [GLL+11, AHL09] uses the operator **Goal** or $\mathbf{Des}_i$ and simply write $\mathbf{Goal}_i\varphi$ when something is desired by agent i and $\mathbf{Goal}_i\neg\varphi$ when something is undesired but the way GOAL works this will lead to problems.

Say that an agent in GOAL has the goal $\mathbf{Goal}\neg \mathit{stealing}$ then when queried with the goal base with

```
Goal(goal(neg(X)))
```

which would return the value X = stealing, however making the query where without negation

```
Goal(goal(X))
```

would return X = neg(stealing) which was not the intention. In order to avoid this the formalization will simply refrain from using negation when queering the goal base and instead use the two predicate **Undes** and **Des**.

$\mathbf{Ideal}_i$ represent the moral state of agent i's, so for the formula $\mathbf{Ideal}_i\varphi$ expresses that agent i wants to achieve or maintain φ. This moral is different from agent to agent as their moral attitude may differ so an agent may see it as morally wrong to steal and prefer to maintain that neither him nor other steals. But another agent has no moral about stealing so if that agent performs or sees another agent stealing it will not be affected by it.

The operator $\mathbf{Resp}_i$ represent the responsibility of an agent has ie. the formula $\mathbf{Resp}_i\varphi$ states that agent i is responsible for φ. Responsibility can be have two notions, a weak and a strong notion. The weak notion is that the agent responsible for an outcome of an event is the one whose action lead to it as seen in [AHL09]. In short if agent i views another agent j performing an action that leads to an event which consequence inflict i then i believes that j is responsible for the consequence. The problem with this view is that an agent is not necessarily responsible for his action as his hands could have been forced as he had no other choice. This is where the strong view comes in as an agent is only responsible for the consequence of an outcome if and only if he could have prevented it as seen in [GLL+11]. The problem with the strong view is that it requires the agent to know that no matter what it does the consequence will always happen which can only be expressed in first-order modal logic which is not part of GOAL or Prolog. This means that only the weak view of responsibility can be implemented.

Expect simply state that an i expect an outcome from an event so the formula $\mathbf{Expect}_i\varphi$ says that agent i expects φ to happen. A simple example would be an agent that plans to go shopping for food then that agent expects that he would obtain food. This operator is a key element to define prospect based emotion and to be able to distinguish between prospect based and well-being emotion emotion as you will see in the next section.

Next is the logic of the formalization and will follow the structured model defined in the revisited model.

4.2.1 Basic Emotions

The emotion are structured after a inheritance model so the emotions can be split into two, basic emotions and complex emotions. Basic emotions are defined mainly from the language $\mathcal{L}$ while complex emotion are defined as compounds of the basic emotions and from $\mathcal{L}$. The only deviation is the fortunes of others as these are not compounds of other emotions but still demands complex logic and as specified by the revisited OCC model is specialisations of the emotions joy and distress.

4.2.1.1 Outcome of event

The first basic emotions that will be defined are the outconme of event.

First a outconme of event can be defined with the following relevance to the agent.

$$\mathbf{Pleased}_i\varphi = \mathbf{Des}_i\varphi \tag{4.1}$$
$$\mathbf{Displeased}_i\varphi = \mathbf{Undes}_i\neg\varphi \tag{4.2}$$

Here the agent i is either pleased or displeased about a desired outcome of an event.

Prospective Consequence

If an event has not yet happened then agent i hope for the event to happen only if the outcome is pleasing for i else i fear the outcome of an event only if that outcome is displeasing.

$$\mathbf{Hope}_i\varphi = \mathbf{Des}_i\varphi \wedge not(\mathbf{Bel}_i\varphi) \wedge \mathbf{Expect}_i\varphi \tag{4.3}$$
$$\mathbf{Fear}_i\varphi = \mathbf{Undes}_i\varphi \wedge not(\mathbf{Bel}_i\varphi) \wedge \mathbf{Expect}_i\varphi \tag{4.4}$$

Actual Consequence

If an event does happen and it is pleasing to agent i then i feels joy, however if the outcome is displeasing then i feels distress.

$$\mathbf{Joy}_i\varphi = \mathbf{Des}_i\varphi \wedge \mathbf{Bel}_i\varphi \tag{4.5}$$
$$\mathbf{Distress}_i\varphi = \mathbf{Undes}_i\varphi \wedge \mathbf{Bel}_i\varphi \tag{4.6}$$

4.2.1.2 Action of agent

Next is the action of the agent and here the outcome of an event is defined relevant by;

$$\mathbf{Approving}_i\varphi = \mathbf{Ideal}_i\varphi \wedge \mathbf{Bel}_i\varphi \tag{4.7}$$
$$\mathbf{Disapproving}_i\varphi = \mathbf{Ideal}_i\neg\varphi \wedge \mathbf{Bel}_i\varphi \tag{4.8}$$

Agent i either approves or disapproves of an action based on i's moral norms towards φ defined by the predicate **Ideal**.

Action of Self Agent

If agent i belief it is responsible for that action then i either feels pride but if it's an approving action or shame if it's a disapproving action.

$$\mathbf{Pride}_i\varphi = \mathbf{Ideal}_i\varphi \wedge \mathbf{Bel}_i\mathbf{Resp}_i\varphi \tag{4.9}$$
$$\mathbf{Shame}_i\varphi = \mathbf{Ideal}_i\neg\varphi \wedge \mathbf{Bel}_i\mathbf{Resp}_i\varphi \tag{4.10}$$

Action of Other Agent

However if agent i belief an other agent j is responsible for an approving action then i admire j, if it is a disapproving action then i reproach j.

$$\mathbf{Admiration}_{i,j}\varphi = \mathbf{Ideal}_i\varphi \wedge \mathbf{Bel}_i\mathbf{Resp}_j\varphi \tag{4.11}$$
$$\mathbf{Reproach}_{i,j}\varphi = \mathbf{Ideal}_i\neg\varphi \wedge \mathbf{Bel}_i\mathbf{Resp}_j\varphi \tag{4.12}$$

As mentioned before the formalization will not contain the emotions love/hate/interest/disgust and so all necessary basic emotions have been defined.

4.2.2 Complex Emotions

Complex emotions are defined using both the language $\mathcal{L}$ and basic emotions. The first complex emotions that will be defined are emotions where an action leads to a consequence. Do note that the eliciting emotions are replaced with the new realized emotion.

Related Consequence and Action

Self Agent

If an agent i performs an action that it approves (Pride) and leads to a consequence that is pleasing (Joy) then i feels gratification. However if i disapproves of it's own action (Shame) and the consequence is displeasing (Distress) then i feels remorse.

$$\mathbf{Joy}_i\varphi \wedge \mathbf{Pride}_i\varphi \rightarrow \mathbf{Gratification}_i\varphi \tag{4.13}$$
$$\mathbf{Distress}_i\neg\varphi \wedge \mathbf{Shame}_i\varphi \rightarrow \mathbf{Remorse}_i\varphi \tag{4.14}$$

Other Agent

If an agent j perform an approving action (Admiration) that leads to a consequence that is pleasing to agent i (Joy) then i feels gratitude towards j. If i disapproves of agent js action (Reproach) and that action leads to a displeasing consequence for i (Distress), then i is angry with agent j.

$$\mathbf{Joy}_i\varphi \wedge \mathbf{Admiration}_{i,j}\varphi \rightarrow \mathbf{Gratitude}_{i,j}\varphi \tag{4.15}$$

$$\mathbf{Distress}_i\neg\varphi \wedge \mathbf{Reproach}_{i,j}\varphi \rightarrow \mathbf{Anger}_{i,j}\varphi \tag{4.16}$$

Consequence Confirms Prospect Desirable Consequence

If agent i has the prospect of a desirable outcome φ and it's confirmed then the feeling of hope is replaced with the feeling of satisfaction that the desired consequence was achieved.

$$\mathbf{Hope}_i\varphi \wedge \mathbf{Bel}_i\varphi \wedge \mathbf{des}_i\varphi \rightarrow \mathbf{Satisfaction}_i\varphi \wedge \mathbf{Bel}_i \tag{4.17}$$

Do note that $\mathbf{Bel}_i\varphi\wedge\mathbf{des}_i\varphi$ is the same as the emotion joy and could be replaced with $\mathbf{Joy}_i\varphi$, given the new formula;

$$\mathbf{Hope}_i\varphi \wedge \mathbf{Joy}_i \rightarrow \mathbf{Satisfaction}_i\varphi \wedge \mathbf{Bel}_i \tag{4.18}$$

However this does give rise to problems as the emotion joy could have been realized before hope which is not possible so in order to solve this problem temporal logic is needed. As the formalization refrains from using temporal logic the first formula is the one used later on.

Consequence Disconfirms Prospect Desirable Consequence

However if a prospect of a desirable outcome is disconfirmed then the feeling of hope is replaced with the feeling of disappointment as the desired consequence was not achieved.

$$\mathbf{Hope}_i\varphi \wedge \mathbf{Bel}_i\neg\varphi \wedge \mathbf{des}_i\varphi \rightarrow \mathbf{Dissappointment}_i\varphi \wedge \mathbf{Bel}_i \tag{4.19}$$

Consequence Disconfirms Prospect Undesirable Consequence

If agent i fears φ will happen and it is disconfirmed then the feeling of fear is replaced with the feeling of relief.

$$\mathbf{Fear}_i\varphi \wedge \mathbf{Bel}_i\neg\varphi \wedge \mathbf{undes}_i\varphi \rightarrow \mathbf{Relief}_i i\varphi \wedge \mathbf{Bel}_i \tag{4.20}$$
$$\tag{4.21}$$

Consequence Confirms Prospect Undesirable Consequence

However if it is confirmed that the prospect of a undesired consequence has happened then feeling of fear is confirmed.

$$\mathbf{Fear}_i\varphi \wedge \mathbf{Bel}_i\varphi \wedge \mathbf{Undes}_i\varphi \rightarrow \mathbf{FearsConfirmed}_i\varphi \wedge \mathbf{Bel}_i \tag{4.22}$$
$$\tag{4.23}$$

Do note that $\mathbf{Bel}_i\varphi \wedge \mathbf{Undes}_i\varphi$ is the same as the emotion distress and could be replaced with $\mathbf{distress}_i\varphi$. But the same problem as before arises where the distress may have been realized before the emotion fear.

As mentioned before the **Expect** operator is important for these emotions as without this the agent would hope/fear every thing that it desire/undesire. This will lead to the agent never feeling joy as each time it felt joy there would always exist a related hope which would lead to satisfaction. The same would happen for the emotion fear confirmed.

The next four emotions are a bit different as they are about the fortune of other agents, so here the agents beliefs about another agents beliefs and goals are relevant. These emotions focus on the concept of an agent believing that another agent desire the outcome of an event or that another agent deserves the outcome of an event.

Deservingness is defined by the following;

$$\mathbf{Deserving}_{i,j}\varphi = \mathbf{Des}_i\mathbf{Bel}_j\varphi \tag{4.24}$$

where agent i desire that agent j beliefs φ.

Desirability of other is defined as such;

$$\mathbf{Desired}_{i,j}\varphi = \mathbf{Bel}_i\mathbf{Des}_j\varphi \tag{4.25}$$

here agent i beliefs agent j desires φ. This can also be defined as agent i beliefs that agent j hopes or fear φ of happening.

Consequence presumed to be desirable for other

If agent i beliefs event φ has happened and that agent j desires and deserves φ, then i is happy for j, however if i does not belief j deserves φ then i resents j.

$$\mathbf{HappyFor}_{i,j}\varphi = \mathbf{Bel}_i\varphi \wedge \mathbf{Bel}_i\mathbf{Goal}_j\varphi \wedge \mathbf{Goal}_i\mathbf{Bel}_j\varphi \tag{4.26}$$

$$\mathbf{Resentment}_{i,j}\varphi = \mathbf{Bel}_i\varphi \wedge \mathbf{Bel}_i\mathbf{Goal}_j\varphi \wedge \mathbf{Goal}_i\mathbf{Bel}_j\neg\varphi \tag{4.27}$$

$$\tag{4.28}$$

Consequence presumed to be Undesirable of other

There is not a big difference when the agent beliefs an event is undesirable for another agent.

If agent i beliefs event φ has happened and that agent j does not desires but i beliefs j deserves φ, then i is gloating j, however if i does not belief j deserves φ then i pities j.

$$\mathbf{Gloating}_{i,j}\varphi = \mathbf{Bel}_i\varphi \wedge \mathbf{Bel}_i\mathbf{Goal}_j\neg\varphi \wedge \mathbf{Goal}_i\mathbf{Bel}_j\varphi \tag{4.29}$$

$$\mathbf{Pity}_{i,j}\varphi = \mathbf{Bel}_i\varphi \wedge \mathbf{Bel}_i\mathbf{Goal}_j\neg\varphi \wedge \mathbf{Goal}_i\mathbf{Bel}_j\neg\varphi \tag{4.30}$$

4.2.3 Relation

As stated before this framework will not deal with love and hate the way it is done in the OCC model due to to GOAL being unfit to implement them the way they are defined through appealingness and familiarity.

Some say that action defines who we are which provides a possibility to introduce something to define relations between agents as they can perform action that can influence others. This is a slight combination of familiarity and appealinngness as the action of an agent tells a bit about it's attitude and repeatedly exposure an agent has to another agents action the more familiar that agent becomes. The emotions dealing with action of others agents can be used for this purpose as when an agent realize emotions such as admiration or gratitude the agent must begin to like the other agent or if it is reproach or anger the agent may begin to dislike that agent. This can be done by keeping a relation value of each agent updated the value of an agent when an emotion related to that agent is realized. This value can then determine how much an agent likes or dislike a person depending on the value.

4.2.4 Intensity

Now that the logical formalization of emotions is defined the next is to go through how the intensity of the emotions is computed. These variable will consist of the variables defined in the OCC model and used the same way as only few other formalization has dealt with intensity and does who have only uses the variable likelihood. So the variables used for intensity are the following from the OCC model excluding the ones in section 4.1 that was not possible to implement in GOAL;

Desirability: The desirability will be much the same as the OCC model, an agent will have a set of desires of a given event and a related value of how much the event is either desired or undesired.

Ideal(Praiseworthiness): Praiseworthiness is subjective to each agent and is defined by what the agents beliefs is normal so an agent performing an ideal action would be praised while an agent performing unideal action would not be praised. The value of the variable will be part of the predicate **Ideal** so an agent will have a set of ideals for action and a value associated with this ideal relating to how big a moral impact it has on the agent.

Probability(Likelihood): Much the same as the OCC model or any other implementation of the OCC model this variable is simply the likelihood or probability of an event happening.

Desirability of other Much like the desirability variable but in this case it is the presumed desirability of another agent.

Derservingness This simply state how much the agent thinks that another agent deserves an outcome of an event.

Next is how the intensity of emotion are computed and looking at the basic emotions the first emotion are hope and fear that are influenced by the desirability and probability of an event. In [GM04] the intensity for these are defined as such:

$$Intenisty(p)_{hope,fear} = Desirability(p) \times Probability(p)$$

where $Probability(p)$ ranges between [0,1] and $Desirability(d) \in \mathbb{R}$ and .

When dealing with emotions in actual consequence such as joy and distress the same formula can be used for intensity, however here $Probability(p)$ is 1 since the event has happened so the intensity is only affected by the desirability.

$$Intenisty(p)_{joy,distress} = Desirability(p) \times 1$$

For action it is only influenced by the agent ideals just as desirability.

$$Intenisty(p)_{shame,pride,admiration,reproach} = Ideal(p)$$

For complex emotions based on existing basic emotions in the agent the intensity is calculated from the average intensity of the eliciting emotions. Using gratification as an example the intensity is calculated by

$$Intenisty(p)_{gratification} = \frac{Intenisty(p)_{pride} + Intenisty(p)_{joy}}{2} \tag{4.31}$$

$$\tag{4.32}$$

The intensity of other compound emotion are computed the same way.

Next is the emotions for expected outcome and unlike the compound emotions these only have a single eliciting emotion but as these are specializations of joy/distress then the desirability should also affect these emotions. The intensity can then be computed using the average value between the intensity of the eliciting emotion and the desirability of the outcome. Using satisfaction as an example the intensity is computed as so;

$$Intenisty(p)_{satisfaction} = \frac{Intenisty(p)_{pride} + Desirability(p)}{2} \tag{4.33}$$

$$\tag{4.34}$$

The intensity of the other fortunes of other emotions are computed the same way.

4.2.4.1 Emotion Decay

As it was decided that emotion are implemented as persistent then a decay function is needed in order to reduce the emotions valence in the agent as a person experience an emotion it never exist forever but they do linger for a while before they disappear. The question however is how long should an emotion linger, do some emotions stay longer than other, is the decay linear, or does it follow another function? Only few that have implemented emotions have discussed decay and those who does simply talk of a function applied to decrease the intensity of each emotion. In [Vel97] a decay function is mentioned but the function it follows is not defined and in [DP05] the function mentioned is a simple linear function which is applied to all emotions.

Then there is the question, should all emotions have the same decay function and should all have one? Some emotion has more valence than other and as time goes on some still has valence while others don't, which means the decay should work on the valence of an emotion.
Looking at hope that is prospect based and has valence until the event is realized in which case it is turned into either satisfaction or disappointment, so hope should not disappear unless the agent forgets about the event or the event is not relevant any more. If the event has a deadline where it is realized, the valence of that emotion can increase the closer the event is which is defined in the OCC model as global variable proximity, however not all events has a deadline so it is hard to generalize.

This means that the best solution would be a generalized linear decay function that reduce the intensity each cycle of all emotions until they reach a lower bound in which they are removed. The only exception are the prospect based emotions which should not disappear without valid reason so a different function could be used were the lower bound is reached but the emotion is not deleted.

4.2.4.2 Threshold and Mood

Not all emotions that the agent can potential feel should be realized as some of the potential emotions could have a very low intensity so it's not worth dealing with and not that emotions are quantifiable it is possible to filter out these emotions.

This can simply be done with a threshold which was mentioned in the OCC in the previous chapter which was is to filter out these low intensity emotions but also to define moods in an agent. When the agent experience a positive emotions it should affect the thresholds of negative emotions simulating a good mood and is more prone to experience positive emotion and less likely to experience negative emotions so that only higher intensity of negative emotions should be able affect the agent and vice versa. This will result in the emotion having an effect on the agent and other emotions instead of simply be a interpretation of the agents state of mind. However a change in the mood should not be permanent so if a agent is in a good mood the mood should at some point return to neutral mood without the need for the agent to experience negative emotions. To fix this the decay of emotion can be implemented where the it slowly decays toward a default value.

4.2.5 Expression

Implementing these emotions for an interactive agent is obsolete if the agent can't express them, and is one of the most important aspects. There are generally two ways to express feelings:

- `Verbal expression` which can be through language or it may be non-language sounds such as a grunt or laughter.
- `Visual expression` is done through either facial expression or body language.

The verbal expression through language is able to express the 22 emotions in the framework, however the feeling conveyed this way will most likely have a reduced potency when implemented in a computer agent as language are not perfect at conveying emotions. Combining this with a visual expression can increase the effect of expressing emotions. Implementing visual expression can be done by imitating human facial expression but unlike verbal, facial expression cannot express the 22 emotions independently.

In [EO79] 6 facial expression are defined as distinctive and universal, *anger*, *disgust*, *happiness*, *sadness*, *fear* and *surprise*. This means that each of the 22 emotions in the framework needs to be be mapped as mentioned in the article [Bar02], but as pointed out in the same article there are only one positive expression for the 11 positive emotions in the OCC model and non of the emotions can be linked to *surprise* as surprise is not an emotions in the OCC model as it is not considered a surprise but a cognitive state. This means that the remaining 11 negative emotions needs to be mapped to 4 facial expressions. Looking at the facial expression for *fear* it is clear that there is only the emotion fear that can be mapped to this facial expression as no other emotions deals with a negative prospect. *Disgust* might be hard to link to any of the emotions but disgust can very well arise from the experience of an offensive moral sense, where the emotions reproach and shame can be placed as it is defined by the agents moral norms. The emotions anger and dislike can immediately be placed in the *anger* expression, additionally the emotion resentment can also be placed in this expression since the agent may hate another agent for achieving something that was not deserved. Since the emotion anger is mapped this way then one could think that remorse should also be linked to the same expression, but that is not the case as remorse is rather regret in ones own action than anger towards oneself. The last expression is sadness and this is where the remaining 5 emotions, distress, fearConfirmed, dissapointement, remorse and pity are mapped to.

Of course the agent can have multiple emotions and only one type of facial expression is to be shown. One way would be to show that expression where the sum of the intensity of all the mapped emotions is greatest thereby showing the agents general feeling. But since the emotions are not equally split amongst the 5 type of emotions, and happy dominating with most emotions, the agent would most likely show a happy expression. Going in the other direction the emotions with the highest intensity can be the one defining that expression the agent should show, but this means even if the agent is experience mostly positive emotions a single negative emotions can change the agents expression resulting in a bi-polar agent. A solution would then instead be to find the sum of the 2 or 3 highest emotions in each expression types and show the expression with the highest intensity.

Chapter 5

Implementing Formalization

Next is to figure out how to implement the formalization defined in the previous section into GOAL and make it into a usable framework. The first thing that will be discussed is the emotions themselves where after their logic is implemented, thereafter the decay and mood of the emotions followed by expressing emotions. This will all be implemented in the module called `emotions.mod2g` as seen in the appendix A.1.

5.1 Emotions in GOAL

When the agent experience an emotion then there are no doubt that these should be stored in the belief base of the agent and to make it easier to generalize and get emotions it is simply store with the predicate **emo**. Besides the predicate the emotions has the following information to store;

- *Name* of the the emotion
- *Agents* that are involved with the emotion
- *Subject* that gave rise to the emotion

- *Intensity* of the emotion in the range [0,100]

This give a declarative structure of the emotion and can be easily read with out further information needed. With the emotion joy as an example, it would defined in GOAL as;

```
emo(joy,A,X,I).
```

and read as "Agent A feels joy about X with an intensity of I".

Gratitude deals with multiple agents and the relation would instead be represented by a touple with the relevant agents like so;

```
emo(gratitude,(A,B),X,I).
```

read as "Agent A feels gratitude towards agent B for X with an intensity of I".

The emotions like and dislike are a bit different as the subject is another agent rather than an outcome.

```
emo(like,A,X,I).
```

read as "Agent A likes B with an intensity of I".

5.1.1 Realizations of Emotions

In order for the agent to realize the emotions it needs to be able to handle the predicate defined in the language $\mathcal{L}$ in section 4.2. The first operator **Bel** defines the agent's beliefs and should be stored in the agents belief base, however since goal is a declarative it give rise to a problem here as it is not possible to query with **`bel`**`( X )`. A workaround is to simply add the predicate **`bel`**`( bel(X) )` in the query. An example of mental condition using this predicate would then be written in GOAL as.

```
bel( bel(X) ).
```

This workaround does give some benefits as it makes it possible to distinguish between the agents belief base and beliefs used for emotions. As mentioned before the beliefs used for emotions are the outcome of events or action so the additional predicate **`bel`**`(bel(X))` can then be defined as the agents beliefs of experienced outcome and deleted when that beliefs has been handled. This solve some problems as some desires are persistent even after an emotion for the desire is realized. For example an agent can have the desire of finding money and happens to find money on the street which gives rise to the emotion joy, then the desire for money should still persist as he still wants to find money so the desire should be inserted in the goal base again. But if the belief is reinserted as he previously did found money then the feeling of joy would be realized again, so it is important that the belief of the outcome is only inserted once.

The next operator is **Des** and **Undes** but the same problem appears as with the belief operator nevertheless the same workaround can be used where the predicates `des` and `undes` is used and stored in the agents goal. This give the ability to define the desires differently from the goals as goals are states the agent strive to obtain but desires have a weaker definition and simply define what the agent would like should happen. These predicates also have the related variable *desirability* which will also be defined along with the predicate. An example of mental condition using either of these predicates would be.

```
goal( des(X,D) ).
goal( undes(X,D) ).
```

where `D` are the desirability the agent has to for the outcome `X`.

As both the belief and goals for emotions are different from GOALs then the term emotional beliefs and emotional desires can be used to distinguish between them.

As mentioned in section 2.1, GOAL removes goals when they are reached but since emotional desires and beliefs are used then they aren't removed by GOAL. As the emotions are structured as inheritance then it is important to not delete the emotional desires and beliefs when an emotion has used them in it's realization as other potential emotions may still need them, so after all the potential emotions have been computed then reached emotional goals are to be removed.

For confirmed and uncomfirmed desires the code then as followed;

```
forall goal( des(X,D) ) do {
 % confirmed
 if bel( bel(X) ) then delete( bel(X) ) + drop( des(X,D) )
 % disconfirmed
 if bel( bel(neg(X)) ) then delete( bel(neg(X)) ) + drop( des(X,D) )
}
```

and for undesires it is;

```
forall goal( undes(X,D) ) do {
 % disconfirmed
```

```
  if bel( bel(X) ) then delete( bel(X) ) + drop( des(X,D) )
  % confirmed
  if bel( bel(neg(X)) ) then delete( bel(neg(X)) ) + drop( des(X,D) )
}
```

Since emotional beliefs should only exist in the cycle they where they where inserted all emotional beliefs that was not used is then deleted with the code;

```
bel( bel(X) ) do delete( bel(X) ) .
```

The next predicate **Ideal** is a personal believed norm the agent has, so it should be expressed in the belief base. As the predicates **Des** and **Undes**, **Ideal** has a related variable, this case the variable *ideal*. An example of mental condition using this predicates would be;

```
bel( ideal(X,D) ).
```

here `D` is how ideal the agent beliefs the consequence `X` is.

The predicate **Resp** was defined with a weak view mentioned in section 4.2 and requires the agent to perceive other's actions which means it is up to the environment to tell the agent who has performed what and from there it is up to the agent to deduce if an agent is responsible for the consequence of the action he performed. Example of this on GOAL would be;

```
bel( resp(A,X) ).
```

where agent `A` is responsible for the consequence `X`.

The last percept to handle is **Expect** and is defined as the agent expect a consequence to happen. One clear way that the agent expects something is when it has a plan of a series of action in order to reach a certain state. Then it is easy to see what to expect, however it is not certain that it will happen as other agents can interrupt this plan. Expectation could also be related to probability as the agent can come to expect an event if there is a high enough probability of it happening. The environment or another agent could also inform the agent of events that will happen in the future and it is up the agent to decide what to expect. Example of expectation in GOAL would be;

```
bel( expect(X) ).
```

where the outcome `X` is expected.

Besides these predicates, in order to handle emotions the agent should also be able to handle the variables defined in the formalization. Some of these variables is already implemented along with the predicates as just seen.

The variable probability state what the likelihood of a consequence happening. In GOAL this would be implemented like so;

```
bel( prob(X,R) ).
```

where `R` is the probability of the consequence `X` to happen.

To implement these the agent has to keep track of any event that may be quantifiable. One way would be to use time to keep track of how many times a particular event has happened during the start of the environment. Another would be for the environment to give the agent information regarding the maximum numbers of time an action would lead to an event e.g. how many items that could be found when searching through a finite number of boxes.

Deservingness is the same as the implementation of desires with the only different being that instead it is what outcome another agent should experience at some point. In GOAL this would be;

```
goal( des(bel(A,X),D) ). goal( undes(bel(A,X),D) ).
```

where `D` is how much the agent think that another agent A deserves the outcome `X`.

Desirability of others is not much different from deservingness, here the agent simply has a belief of what the other agent desires. An example would be;

```
bel( bel(des(A,X,D)) ). bel( bel(undes(A,X,D)) ).
```

where the agent beliefs that another agent `A` has a desirability of `D` for the consequence `X`.

Now with the predicate defined along with the variable the agent can show emotion. Following the formalization, the realizations of emotions in can be set up in GOAL. Since multiple emotions can potential be realized in one cycle the module will use the rule evaluation order ”linearall” so all action rules will be evaluated. The module will mainly use the ”forall” action rule when evaluating if emotions should be realized as multiple realizations of the same type of emotion can happen in a single cycle.

The framework have some defined logics such as pleased and disapproving that can be defined as macros to make it easier to read the action rules for the emotions.

pleased and displeased will be defined by the macros;

```
#define pleased(X,D) goal( des(X,D) ).
#define displeased(X,D) goal( undes(X,D) ).
```

Approving and disapproving has the macros;

```
#define approving(X,Id) bel( ideal(X,Id), bel(X) ).
#define disapproving(X,Id) bel( ideal(neg(X),Id), bel(X) ).
```

Do note that the disapproving action is defined with a negated outcome of event in the ideal predicate.

Beside these the deservingness and desire of other agents can also be defined by macros, however these need to be defined with both positive and negative macros as emotional desires are defined by `des` and `undes`.

```
#define deserving(A,X,D) goal( des(bel(A,X),D) ).
#define undeserving(A,X,D) goal( undes(bel(A,X),D) ).
#define otherDesire(A,X,D) bel(bel(des(A,X,D))).
#define otherUndesire(A,X,D) bel(bel(undes(A,X,D))).
```

In section 4.2.4, functions was defined to calculate the intensity of some emotions, function for prospect based emotions and on for complex emotions finding the average of the eliciting emotions. These can be written in the knowledge base of the agent with the following Prolog code;

```
prospectInten(Des, Prob, Int) :- Int is min(abs(Des * Prob),100).
averageInten(Int1, Int2, Int3) :- Int3 is (Int1 + Int2)/2.
```

Well being

Looking at the basic emotions the realization of joy can be implemented in GOAL with the action rule

```
forall pleased(X,D), bel( bel(X), me(A) ) do insert( emo(joy,A,X,D) ) .
```

As the intensity of the emotions is only defined by how desired the outcome is then it is simply the desirability. Distress will be implemented in the same way but uses the `displeased` macro instead.

Prospect

The next emotion to look at is the prospect based emotions such as hope and fear, and the action rule would be;

```
forall pleased(X,D),not(bel( bel(X) )), bel( expect(X), prob(X,R),
    prospectInten(D,R,I), me(A) ) do insert( emo(hope,A,X,I) ).
```

For both hope and fear it is important to perform the query **not**(**bel**(bel(X))) checking that the outcome of the event has not yet arisen. Calculating the intensity requires the desirability and the probability for the event to happen and these are calculated using the function prospectInten defined previously. However probability is the only variable that can be omitted and the emotion can still be realized since it only affects the intensity of the emotions. To make it easier to program with this module the action rule can be altered so that it is not required to provide the responsibility variable. If it is not provided it can be assumed that the probability is 1, that is that it is assumed that the event will happen and resulting in the following code

```
forall pleased(X,D), not(bel( bel(X) )), bel( expect(X), me(A) ) do {
  if not(bel( emo(hope,A,X,_) )) then {
    % No probability exist
    if not(bel( prob(X,_) )), bel( prospectInten(D,1,I) ) then
      insert( emo(hope,A,X,I) ).
    % Probability exist
    if bel( prob(X,R), prospectInten(D,R,I) ) then
      insert( emo(hope,A,X,I) ).
  }
}
```

After the emotions is know to be realized the existing of a probability variable is checked, if no variable exist the intensity is computed with the probability of 1, if there is a probability variable then that is used instead.

As with the well being emotions, realizing fear is much the same but uses displeased macro instead.

Actions

The last basic sets of emotions is the actions based and as defined in the formalization the action rule would contain the predicate ideal and responsibility and for the agent self the responsibility should state that it is the agent itself.

```
forall approving(X,Id), ( resp(A,X), me(A) ) do insert( emo(pride,A,X,Id)
) .
```

If the emotions is focused on another agents action then is not that different as it is the responsibility predicate that simply states that another agent is responsible for the outcome.

```
forall approving(X,Id), ( resp(B,X), me(A) ) do insert(
    emo(admiration,(A,B),X,Id) ) .
```

For all cases it is only the ideal that is used to calculate the variable so just as for the well being emotion it is simply how ideal the action is that is used for the intensity.

Compound emotions

Now that the realization of basic emotions are defined the next step is the complex emotions and first up is the compound emotions made of an eliciting basic action and well-being emotion.

These are fairly simple as the action rules checks if the two emotions eliciting emotions required for the compound emotions exist and then calculates the intensity.

```
forall bel( emo(pride,A,X,Ip), emo(joy,A,X,Ij), me(A),
    averageInten(Ip,Ij,I) ) do insert( emo(gratification,A,X,I) ) .
```

It is much the same with the three other emotions just with different eliciting emotions and the function is also simple the function for finding the average between the eliciting emotions.

Prospect Outcome

Next is when a prospect based emotion is either confirmed or dis-confirmed, these emotions has a single eliciting emotions which is either hope or fear. Beside the eliciting prospect based emotion the query **bel**(bel(X)) or **bel**(neg(bel(X))) is also needed to check if the outcome has happened. For satisfaction the action rule is;

```
forall goal( des(X,D), bel( emo(hope,A,X,Ip), bel(X) me(A), averageInten(D,Ip,I)
) do insert( emo(satisfaction,A,X,I) ) .
```

Additionally it is required to check the goal base in order to get the desire or undesire the agent had for the consequence to be able to calculate the intensity as the intensity is the average of the eliciting emotion and the desire for the outcome.

For the other tree emotions the reader is referred to appendix A.1.

Fortunes of Others

Last is the emotions for the fortunes of others.

Here the two terms the deserving and desire of another agent are used to realize the emotion. The first emotion is when the agent beliefs that the other agent deserves the outcome of an event and beliefs the agent desires that outcome then the emotion happpy-for is elicited.

```
forall deserving(A,X,Deserv), otherDesire(A,X,Desire) ( bel( bel(X), me(Me),
function(Deserv,Desire,I) ) do insert( emo(happyFor,(Me,A),X,I) ) .
```

Here the intensity is computed from how much the agent thinks the other agent deserves the outcome and how it desired by the other agent and is the average of the two variables. Looking at the other three emotions it is the same but uses different combination of deservingness and desire of other.

5.1.1.1 Relations

As mentioned in section 4.2.3 the emotions such as admiration and anger can be used to define the relations between agents. This can be done fairly easy where a predicate is kept with the relation value for each agent in the system beside the agent it self. The relation value is in this case in the range [-100,100] where positive defines when an agent is liked and negative is when an agent is hated. This can be done in the initial module of the emotion module like so;

```
forall bel( agent(X) , me(A), X \== A ) do
 insert( relation(X,0) ).
```

Then each time one of the emotion relating to other agents the relation is updated where the intensity of the emotion is used and the average is found between emotion intensity and the relation intensity. For positive emotions the code is

```
if bel(relation(A,Rel), average(Rel,NegI,NewRel) ) then
 delete( relation(A,Rel) ) + insert( relation(A,NewRel) ).
```

and for negative emotions the update code is

```
if bel( relation(A,Rel), NegI is -1*Id, average(Rel,NegI,NewRel)) then
  delete( relation(A,Rel) ) + insert( relation(A,NewRel) ).
```

The reason there are two different update methods is that the relations value is in the range of [-100,100] so when reducing the relation the intensity needs to be negated first.

With the relation and it's value in place the next part is to turn it into emotions and as discussed these emotion can surface when when they reach a lower bound. In order to increase the intensity of where an agent is neither hated or liked a lower bound can be set for both hated and liked, in this case the value 20 will be used for the lower bound of the positive relation and -20 for the negative relation. This will result in the following code where that if there is a potential like emotion, then it checks if there is a existing like emotion and update it if the intensity is different, else if there aren't any existing like emotion then a new is inserted. The same is done with the hate emotion, but if the relation value is not above the threshold then existing like or dislike are removed.

```
forall bel( relation(A,NewI), me(Me) ) do {
  if bel( NewI > 20 ) then{
    if bel( emo(like,(Me,A),OldI), NewI \= OldI ) then
      delete( emo(like,(Me,A),OldI) ) + insert(emo(like,(Me,A),NewI) ).
    if not(bel( emo(like,(Me,A),_) )) then insert(emo(like,(Me,A),NewI)
        ).
  }
  if bel( NewI < -20, PosI is abs(NewI) ) then {
    if bel( emo(hate,(Me,A),OldI), PosI \= OldI ) then
      delete( emo(hate,(Me,A),OldI) ) + insert(emo(hate,(Me,A),PosI)).
    if not(bel( emo(hate,(Me,A),_) )) then
        insert(emo(hate,(Me,A),PosI)).
  }
  if bel(NewI < 20, NewI > -20) then{
    if bel( emo(hate,(Me,A),OldI) ) then delete(emo(hate,(Me,A),OldI)).
    if bel( emo(love,(Me,A),OldI) ) then delete(emo(hate,(Me,A),OldI)).
  }
}
```

5.1.2 Decay

Now that emotions can be realized and the intensity calculated they need to dissipate as time goes on and be removed when they don't have any valence for the agent. As they are implemented with the structure

```
emo(Name,Agents,Subject,Intensity)
```

It is easy to generalize and make a simple action rule to decay all of the emotions in the agents belief base.

```
forall bel( emo(E,A,X,I) ) do {
  if bel( I =< 1) then
    delete( emo(E,A,X,I) ).
  if bel( I > 1 , decayEmotion(I,NewI)) then
    delete( emo(E,A,X,I) ) + insert( emo(E,A,X,NewI) ).
}
```

Every emotion intensities in the agents belief base is reduced with the function **decayEmotion** that reduced the intensity by 10% and the emotions is updated with the new intensity. A lower bound is set to 1 so when the intensity is 1 or lower then it is considered of no valence to the agent and is deleted. The function can be changed to make emotions have a slower or faster decay time and the lower bound can be raised to reduce the time before a emotion is deleted.

One problem with the current implementation is how to handle recurring emotions. Currently a recurring emotion will be realized as long as the existing emotions doesn't have the same intensity. This can potentially clutter the belief base and if the agent expresses these feelings it is not ideal for it to express the same feeling for the same outcome multiple time. Instead it would be better to simply update the existing with the new intensity, but if the new emotions intensity is lower than the existing emotions then it would be weird for the agent to have a reduced feeling so in this case the new emotions should be overlooked. To do this it is necessary to check if an existing emotion exist in the agent belief base, if so then it is necessary to check which emotion has the highest intensity, so if the new instance of the emotion has a higher intensity then replace the old emotions but if not then the old emotions is kept.

For the emotion joy the code will now look like this;

```
forall pleased(X,D), bel( bel(X), me(A) ) do {
  if bel( emo(joy,A,X,OtherI), OtherI =< I ) then
     delete( emo(joy,A,X,OtherI) ) + insert( emo(joy,A,X,I) ).
  if not(bel( emo(joy,A,X,_) )) then insert( emo(joy,A,X,I) ).
```

This goes with an exceptions for the prospect based emotions, hope and fear. Unlike the other emotions that are influenced by a related emotional goal and desire that only happens in one cycle where after they are removed, hope and fear are influenced by an emotional goal and expectation which exist in each cycle until the emotional goal is confirmed or disconfirmed. Implementing the this update function will result in the emotion being realized in every cycle and have the same intensity and making the decay obsolete for these two emotions.

5.1.3 Mood

The last thing that needs to be implemented from the formalization in order for the emotions to functions as intended is threshold and mood mentioned in section 4.2.4.2.

Given that the mood is defined as either positive or negative then there are multiple ways to implement mood in GOAL. One way would be to have a variable in the belief base, ranging from .100 to 100 following the intensity range where -100 is bad mood and 100 is good mood. A positive emotion will then provide an increase in the mood and a negative emotions will decrease it. However this cannot be used as a threshold to filter out low intensity so additional variables would be needed to define what the threshold is for positive and negative emotions. Another way would be to have two variables, one for the threshold of positive emotions and one for negative emotions. These two variables would then be linked so when the agent experiences a positive emotion the positive threshold would decrease and the negative threshold would increase. Thereby also working as a mood since a low positive threshold and a high negative threshold would imply positive mood and would filter out more negative emotions and vice versa. When the agent has a neutral mood the threshold should still contain a default value where they still filter out low intensity emotions.

Implementing this require that the the positive and negative moods are inserted with the default in the agent at the init module. The default value of the threshold is set to 20 so only emotion in the range $[20, 100]$ is perceived by the agent.

```
if true then insert( mood(20,20) ).
```

Then when an emotion is realized, both the negative and the positive mood is update. The code for updating the positive and negative mood would then be;

```
if bel( incPosMood(I,PM,NM,NewPM,NewNM) ) then
 delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).

if bel( incnegMood(I,PM,NM,NewPM,NewNM) ) then
 delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
```

Inserting this in what is made so far we get that an emotion such as joy would now have the code;

```
forall pleased(X,D), bel( bel(X), me(A), abovePositiveMood(D) ) do {
 %% if the emotion for the same thing exist then replace it with
 if bel( emo(joy,A,X,OtherI) ) then {
  if bel( OtherI =< D ) then delete( emo(joy,A,X,OtherI) ) + insert(
      emo(joy,A,X,D) ).
  }
  if not(bel( emo(joy,A,X,_) )) then insert( emo(joy,A,X,D) ).
  %% update mood
  if bel( incPosMood(D,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
}
```

where the mood is increased in the positive direction as joy is a positive emotion and the agent is then in a positive mood.

As mentioned in the previous chapter the mood of the agent should not stay at this value and should decay as time goes on just as the decay of the emotions, however the moods should slowly regress back to their default value which in this case is 20. This can be defined by a quadratic function where the minimum value is 20 and where the X-axis is the current value and Y-axis is the new decayed value. This means that if the current mood value is above or below 20 then the decay would slowly return the value back to 20, however it will never reach 20 so for that the decay value can be set to 20 if the current value is close enough. An example of such a function can be seen in the following prolog code;

```
decayMood(X,Y) :-
  X = 21 ->
    Y = 20 ;
    Y is 1/100*(X-40)*X+24.
```

The GOAL code for decay of the mood would then be;

```
if bel( mood(M), decayMood(M,NewM) ) then
  delete( mood(M) ) + insert( mood(NewM) ).
```

With the functionalities, realization of emotion, decay, and mood in place the order of which they are executed has importance as the decay function should only be applied to existing emotions and not include the emotions that have just been realized in the same cycle. The decay of the mood is irrelevant as placing it before the realization of emotions will not affect them as in the very first cycle the mood is the default value and the decay will not change that. Then the following cycles would be the same as if it was placed after the realization but since the decay of the mood is much the same as the decay of the emotions then it will be placed the same place. To increase the readability of the code both the decay for the emotions and mood is placed in their own module inside the emotion file and is then called before the emotions are realized.

5.2 Expressing feelings

After the agent is done with realizing new emotions the agent needs to be able to express its emotions but agents in GOAL neither has a face or a voice to use these expression types. However the entity it is connected to in the environment can be made to express emotions and with GOAL being declarative it should be easy to create a sentence from the emotions that can be written out in the environment. The only way the information can be passed from the agent to the environment is through action with the information given as parameters, but there is a potential problem with the way information is formatted by EIS.

The expressions will be implemented in a new module called "mapping" inside the `emotions.mod2g` and called from the main module.

Visual Expression

The first type of expression is the visual and as discussed before all the emotions that needs to be mapped to the expression types and the type with the 2 or 3 highest intensity of emotions should be shown. The GOAL code is fairly simple as the heavy duty will be done in the knowledge base

```
if bel( highestExpression(P,V), V > 0 ) then emotiondisplay(P).
```

The function `highestExpression` is a function which is written in Prolog and can be seen in appendix A.1. The function finds and computes the highest two emotions of each expression and finds the expression with the highest value, the value and name of that expression is returned and if the expression value is above 0 the expression is sent to the environment. It is then up to the environment to display that expression.

Verbal Expression

Next is the verbal Expression and as mentioned this is easy to implement because of the declarative language and example of how these emotions are can be written by the environment is already shown at the start of this chapter. However there are some problem with the way the parameters of an action is send to the environment that needs to be solved first. As defined in section 2.1.2 the environment can receive 4 different types of parameters, and when sending the name of the emotion it is received as an Identifier. But the agents related to an emotion can either be a single agent, e.i. ”agent1” or a tuple of agents, e.i. ”(agent1,agent2)” which means that when the environment receives an emotion with a single agent it is received as an Identifier but for multiple agents the tuple is received as a Function. The same problem is with the subject, it can either an an atom, e.i. ”money”, or it can be a term, e.i. ”money(5)”, the former will be received as an Identifier and the latter will be received as a Function.

The first problem can easily be solved by just having two different actions defined in the `actionspec` in the agent with the same name. One of them will then consist of an action with four parameters for emotions with a single agent and the second with five parameters for emotion with two agents where the additional parameter is the second agent;

```
emotionverbal(Name,Agent,Subject,Intensity)
emotionverbal(Name,Agent1,Agent2,Subject,Intensity)
```

Thereby avoiding sending a tuple.

The other problem is also fairly easy to solve as Prolog has a function to make a term into a string and thereby the term will be received as a Identifier instead, sadly GOAL does not support all Prolog function and this particular function is not supported. A similar function can be defined in the knowledge base made with the other Prolog functions that are supported by GOAL;

```
term_to_string(T,S) :-
     =..(T,[Name,X]),
     string_concat(Name,"(",S1),
     string_concat(S1,X,S2),
     string_concat(S2,")",S).
```

This however only support terms with one argument.

With these problems out of the way the agent can send the emotions to the environment but the agent needs to send each emotion to the environment each cycle as some emotions may be deleted and new emotions realized along with changes to the intensity. The resulting code is then;

```
forall bel( emo(P,A,X,I) ) do {
  if bel(atom(X)) then {
    if bel(A = (A1,A2)) then emotionverbal(P,A1,A2,X,I).
    if bel(A \= (A1,A2)) then emotionverbal(P,A,X,I).
  }
  if bel( \+ atom(X), term_to_string(X,NewX)) then {
    if bel(A = (A1,A2)) then emotionverbal(P,A1,A2,NewX,I).
    if bel(A \= (A1,A2)) then emotionverbal(P,A,NewX,I).
  }
}
```

Here it can be seen the subject is checked to be an atom or a term with Prolog function `atom`, additional, the emotion is checked to be either one or two agents and use the relevant action.

Chapter 6

Emotional Agent

With the framework done it is time to put it to the test. To test the framework an environment along with an agent has been developed called SimpleJim which has the focus of human-computer interaction where the agent is equipped with displaying emotions and intended to display most of the emotions defined in chapter 4. The idea is that an agent called Jim follows his simple life style or working, shopping, enjoying his food and going to sleep. Through the daily cycle Jim will show his emotions but without any interference he will have a happy life and a form of hedonistic life style. The interaction then lies in that a user of the system can interfere with Jims daily life by giving him food or money out of good will or to take them instead. Jim will then respond with emotions about the interference and express his gratitude or anger against the user.

6.1 SimpleJim

Since Jim need to be able to go to work, go shopping and sleep, most of these action will lead Jim to different locations. But as the focus is not on where Jim is then it doesn't need to be visible locations that can be occupied so where Jim is not important, but rather what is he doing. Jim has currently seven states he can be in each having a related action that leads to the states;

Working, stealing money, stealing food and *shopping* for when he is out of his house.
Eating and *sleeping* which are action that are performed in his home.
Home which is the neutral state for Jim as he will return to his house after each of the previous state with the action *Return*.

Since the user is to interact with Jim an additional state is needed between when he is home and when he is performing one of the actions in order to see the effect of the interaction. For example if Jim has just earned money it should be possible to take his money before he goes out to spend them. This state can simply be a *waiting* state has he will wait before performing an action, additionally the environment will not continue before both the user and Jim has performed an action. Jim will go through following procedure between each action;

$$Home->Waiting->Action->Return->Home$$

where action $Action \in work, shop, eat, sleep, stealmoney, stealfood, return.$

The user is able to perform five actions in the environment, give money or food and take money or food along with the option to do nothing. These action can be performed at any time but their timing can be a factor for what emotions are shown. If the action to take money is done when Jim is aware of it he will know the user is responsible for it and express the disapproval of the action, but if it is done when he is sleeping then he won't know who took it and can't blame anyone.

To make SimpleJim go around his daily cycle he needs some motivation and not follow a scripted cycle, and since he sleeps and eat Jim can be equipped with energy and hunger as motivation for his actions. Most of the actions that Jim performs will slowly deplete his energy and increase is hunger and the only way to replenish energy is by sleeping and to reduce hunger by eating. But in order to eat he needs food which can only be obtained through shopping or stealing. To buy food he needs money which requires that he works and to do any of these action he needs enough energy or hunger. Jim's energy is set to max 5 and will reduce to 0 and he's hunger will start at 0 and rise to 5. Some of these action has a larger demand than others such as going to work demands more energy than stealing money and the cost of each action can be seen in table 6.1. The cost of each of these action will first be realized when Jim returns to his home.

As it can be seen from the table 6.1 it is more demanding to work than steal but

Action	Energy	Hunger	Obtained/Spend
Working	-2	+2	+Money
Shopping	-1	+1	+Food/-Money
Eating	-1	Set to 0	-Food
Sleeping	Set to 5	-2	
Stealing food	0	0	+Food
Stealing Money	0	0	+Money

Figure 6.1: The cost and benefits of Jim's actions

since Jim has moral norms about stealing he will not perform these action, that is unless he does not have any other alternative. Given that the user interferes enough with Jim's daily life he can end up with no money to buy food and too hungry and tired to work that his only option is to steal food. An important part is that the environment also keeps track of time and one time unit is when both the user and Jim has performed an action. This time is needed to synchronize between the agent and the environment as they run independently of each other and without the synchronization the agent would not know when to update the emotions resulting in the emotions being updated each cycle of the agent resulting in a rapid decay of emotions.

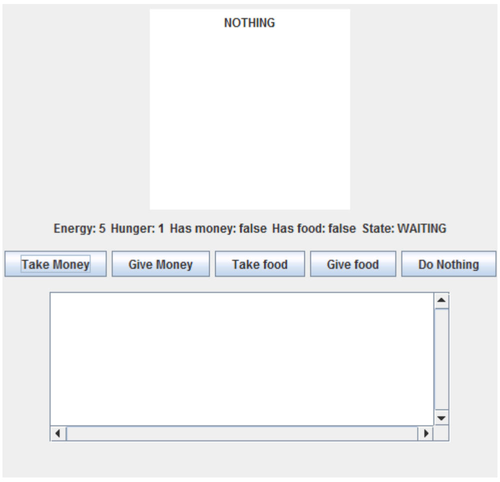
Looking at what is possible with this setup Jim should be able to express a large number of emotions.

6.1.1 Environment

As already described the environment is where Jim express his emotions and where the interaction between the user and Jim takes place. However this environment is not the main focus of the thesis so several optimizations and good practice has been neglected. The environment was only developed for testing emotions with SimpleJim and cannot be reused to other kind of agent systems, but this is expected as the focus is on the agent showing emotions. Only the concepts of the environment along with connecting the environment with EIS to make it communicate with GOAL is discussed. For more detail read appendix A.3.

The environment is constructed by four classes, `EnvironmentInterface`, `Environment`, `EnvironmentWindow` and `Agent` with related java files. `Agent` is where the entity Jim is defined and keeps track of the variables such as energy, hunger and what Jim is doing. The environment that Jim occupies is defined in `Environment` and it is here the actions that both Jim and the user performs are handled and the changes they make to the system is applied.

The display of Jim and the interaction between Jim and the user is done in `EnvironmentWindow` and can be seen in figure 6.2. The communication between GOAL and the environment is done in the `EnvironmentInterface` and it is here the EIS discussed in section 2.1.2 along with its methods is implemented. This means that the received action from GOAL is received and the percepts that Jim receives is sent from here.



Energy: 5 Hunger: 1 Has money: false Has food: false State: WAITING

Figure 6.2: Environment for SimpleJim

The first method is `public void init(Map<String, Parameter> parameters)` which is called when the MAS is started in GOAL. This method goes through the parameters in the `Map` and apply these to the environment. This method can handle the following variables

- `energy = X`: is how much energy Jim has in the range $X \in [0, 5]$
- `hunger = X`: is how hungry Jim is in the range $X \in [0, 5]$
- `money`: Jim has money
- `food`: Jim has food
- `debug`: debug is activated

`Money`, `food` and `debug` are simple boolean values and are set to true when written in the init parameter of the MAS file in GOAL and false when not present.

The agent may request percepts at any given time and each time a request is send the method `LinkedList<Percept> getAllPerceptsFromEntity(String entity)` is invoked. As discussed in section 2.1.2 these percepts can contain `Parameters` in order to apply value to percepts.

The following percepts can be received:

- `energy(X)`
- `hunger(X)`
- `food`
- `money`
- `state(X)`
- `time(X)`
- `userPerformed(X)`
- `given(X)`
- `lost(X)`

Besides percepts about Jim's state and the time of the environment, three new percepts are introduced which are `userPerformed`, `given` and `lost`. These are all part of the user interaction with the agent as the agent will receive the percept `given` then the user has given Jim anything and `lost` when the user has taken anything from Jim. The percept `userPerformed` is used when the agent can see what action the user has perform, but is only received if Jim was there to see the action. The agent can only witness an action when at home, eating or waiting. Unlike the other percepts which are send at each request, these three are only send once. Because these are not states of the agent, but information that should only be received a single time after the user has performed an action.

The last thing is to manage action from the agent which is done by the method `Percept performEntityAction(String entity, Action action)` which is invoked when the agent performs an action. The method simply matches the received action name with the know action and apply these action to the environment. The know actions are the following nine;

- `work`

- `sleep`
- `eat`
- `stealmoney`
- `stealfood`
- `wait`
- `return`
- `emotiondisplay`
- `emotionverbal`

Most of these actions has already been discussed and the last two are the action needed for the agent to display the emotions both verbal and visual to the environment. Examples of these displays can be seen in figure 6.3, where all 5 visual expression shown with color instead of a face.

6.1.2 Agent

After the environment has been created it is possible to develop an agent in GOAL that is to take control of the entity Jim in the environment. This agent should in the end be able to express emotion by using the defined framework developed in chapter 5. The agent consist of an agent file called `Jim.goal` and 3 modules file with different tasks; `emotionHandler.mod2g` that deals with a lot of the predicate and variable for emotions and `planner.mod2g` that deals with planning of the agents action, along with `emotion.mod2g` which was defined in precious chapter. All of these files can be seen in appendix A.2. This section will mostly focus on implementing the predicates and variables needed for making the agent expressing emotions. The approach will be to first make the agent able to handle the percepts from the environment and make the agent capable of executing action where after each group of emotions can be added to the system

The first thing that is executed when the MAS starts is the ***init module*** of the agent and this is where the first percepts can be received and dealt with, and since there are nothing in the beliefs base then they are simply inserted.

```
init module{
 program{
   %% handle initial percepts
   if bel( percept(energy(X))) then insert( energy(X) ).
```

```
    if bel( percept(hunger(X))) then insert( hunger(X) ).
    if bel( percept(state(X)) ) then insert( state(X) ).
    if bel( percept(food) ) then insert(food ).
    if bel( percept(money) ) then insert( money ).
    if bel( percept(time(Time)) ) then insert( time(Time) ).
  }
}
```

After that the percepts will only be updated in the ***event module*** of the agent.

```
event module{
  program{
    %% update percepts
    %% handle boolean percepts
    if bel( percept(food) ), not(bel( food )) then insert( food,
        bel(food)).
    if bel( food ), not(bel( percept(food) )) then delete( food ).
    if bel( percept(money) ), not(bel( money )) then insert( money,
        bel(money)).
    if bel( money ), not(bel( percept(money) )) then delete( money ).
    if bel( percept(time(T)), time(OldT), T \= OldT) then delete(
        time(OldT) ) + insert( time(T) ).

    %% handle always percepts
    if bel( percept(energy(E)), energy(OldE), E \= OldE) then delete(
        energy(OldE) ) + insert( energy(E) ,bel(energy(E)) ).
    if bel( percept(hunger(H)), hunger(OldH), H \= OldH) then delete(
        hunger(OldH) ) + insert( hunger(H), bel(hunger(H)) ).
    if bel( percept(state(S)), state(OldS), S \= OldS) then delete(
        state(OldS) ) + insert(state(S)).
  }
}
```

The agent is now able to handle most of the percepts, however some are omitted as they are mainly used for emotions and will be implemented later.

Next on the agenda is to make the agent plan in order to perform action but this will only be touched lightly as it is not in the scope of the thesis. The agent will make use of the module `planner.mod2g` which will produce the predicate *plan()* that will contain a list of the actions; *eat*, *sleep*, *getmoney* and *getfood*. *Getmoney* and *getfood* are action that can either be acquiring the item through honest ways or through stealing and it is up to the agent an the state of the energy and hunger to decide the specific action to perform. Besides these action the agent can *return* or *wait* but the will always be executed in between each of the previous action in the plan so there is no reason to fill the plan with these actions. The planner also checks if the actions are still feasible and if not then they are removed and the agent re-plans. The planning step will be performed the first time in the initial module after the initial percepts has been handled.

```
if true then planner.
```

Any consecutive planning will be done in the event module, but the planning step should only happen when important changes are made to the environment and not at every cycle of the agent. This is where the time predicate can be used as the time is incremented when both the user and the agent as performed an action and it is here when changes to the environment are executed. So when a new time step is detected the planing step should be executed:

```
if bel( percept(time(T)), time(OldT), T \= OldT) then{
  if true then delete( time(OldT) ) + insert( time(T) ).

  %%perform planning
  if true then planner.
}
```

With the agent able to plan it should act upon this plan but these action should only occur when the agent is in the waiting state, Additionally, all actions should only be executed when a new time step is detected. For this a predicate *doneAction* is implemented which is inserted in the belief base through the actions post-condition.

```
work{
  pre{ true }
  post{ doneAction, bel(working) }
}
```

Then each time the agent has performed an action the *doneAction* is inserted and is first removed when a new time step is detected as so;

```
if bel( percept(time(T)), time(OldT), T \= OldT) then{
  if true then delete( time(OldT) ) + insert( time(T) ).
  %% delete action
  if bel( doneAction ) then delete( doneAction ).
  if bel( state('HOME'), plan([X|Tail]) ) then delete( plan([X|Tail]) )
       + insert(plan(Tail)).

  %%perform planning
  if true then planner.
}
```

The code for performing an action is made in the ***main module***;

```
main module{
  program{
    if not( bel(doneAction) ) then {
      if bel( state('WAITING') ) then {

        if bel( plan([X|T]) ) then {
          if bel( plan([X|T]) ) then {
          if bel( X = eat ) then eat.
          if bel( X = sleep ) then sleep.
          if bel( X = getfood ) then{
            %% buy
            if bel( energy(E), E > 0, money ) then buyfood.
            %% steal
            if not(bel( money )) then stealfood.
          }
          if bel( X = getmoney ) then{
            %% buy
            if bel( energy(E), E > 1, hunger(H), H < 4) then work.
            %% steal
            if true then stealmoney.
          }
        }
      }
      if bel( state('HOME') ) then wait.
      if not(bel( state('WAITING') ; state('HOME') )) then return.
    }
  }
}
```

With this the agent is now fully able to interact with the environment, and next is to make it able to interact with the user through emotions. The approach will be to add the framework from the file `emotion.mod2g` to the agent followed by implement the basic emotions which are the only emotion required to be managed in order to show emotions.

As described in chapter 5 the `emotion.mod2g` consist of multiple modules but to use the framework the modules ***initEmotion module*** and ***emotionUpdate module*** are the only one needed and should be called from the agent file. The first module is called in the ***init module*** of the agent file

```
if true then initEmotion.
```

and the second module is called in the event module when a new time step is detected.

```
if true then emotionsUpdate.
```

Most of the predicate and variables needed for showing emotions are handled in the module file `emotionHandler.mod2g` with the only exception being emotional beliefs that is received through percepts and handled in the event module. The calls to this module will be done before the *emotionsUpdate* call to insure that all the predicates and variables are up to date before handling emotions.

```
if true then emotionHandling.
if true then emotionsUpdate.
```

Joy and fear are the first emotions to be implemented and these emotions require that the agent has desires and beliefs about the outcomes that can happen. When focusing on the outcome that the agent can cause the agent can desire to obtain food or money, replenish energy and eliminate hunger.

The ***emotionHandler module*** will then have the following code for implementing the desire for food;

```
% food
if not(bel( food)), bel( plan(X) ) then {
  if bel(member(getfood,X)) then {
     if goal( des(food,50) ) then drop( des(food,50) ) + adopt(
          des(food,75)).
     if not(goal( des(food,_) )) then adopt( des(food,75)).
  }
  if bel(\+member(getfood,X) ) then adopt(des(food,50)).
}
```

Money is implemented in the same way as above. For hunger it is;

```
% hunger
if bel( hunger(X), X > 0, NewD is (50/5*X) ) then {
  if goal( des(hunger(0),D) ) then drop( des(hunger(0),D) ) + adopt(
       des(hunger(0),NewD) ).
  if not(goal( des(hunger(0),D) )) then adopt( des(hunger(0),NewD) ).
}
```

where energy is the same.

The desire for eliminating hunger is constructed so that the desire is increased the hungrier the agent is and much the same with energy where the desire increases as the energy is depleted.

Now that the emotional desires are in place, next is the emotional beliefs. These are managed in the event module along with the percepts, when a change to the beliefs are received the emotional beliefs are insert. The changes for food or money is simply that the agent goes from not having the item to having it and for energy or hunger it when a change in the level is detected.

```
if bel( percept(food) ), not(bel( food )) then insert( food,
     bel(food)).
if bel( percept(energy(E)), energy(OldE), E \= OldE) then delete(
     energy(OldE) ) + insert( energy(E) ,bel(energy(E)) ).
```

However these are only positive outcomes so only joy is realized, but for negative outcome the interference by the user needs to be managed.

When the user interfere it can cause the agent to lose money or food but they can also be given and these emotional desires should be present in the agents mind at all time.

```
% Desires for beeing given food and money
if not(goal( des(given(food),_) )) then adopt( des(given(money),50) ).
if not(goal( des(given(money),_) )) then adopt( des(given(food),50) ).

%% update undesires for loosing food and money
if not(goal( undes(lost(food),_) )) then adopt( undes(lost(food),100)
    ).
if not(goal( undes(lost(money),_) )) then adopt(
    undes(lost(money),100) ).
```

For the related beliefs the agent will receive a percept if money or food has either been given or lost.

```
% Percept for not witnessed action
if bel( percept(lost(food)) ) then insert( bel(lost(food)) ).
if bel( percept(given(food)) ) then insert( bel(given(food)) ).
if bel( percept(lost(money)) ) then insert( bel(lost(money)) ).
if bel( percept(given(money)) ) then insert( bel(given(money)) ).
```

The agent is now able to express both joy and distress.

Next is action based emotions and these emotions requires that the agent has morals about the actions that can be performed by both the agent and the user. The agent should know who is responsible for an action and at least know if the outcome has taken place through beliefs. These ideals are such as stealing, loosing or given food/money. These ideals are never deleted by the framework and should only be inserted once in the ***init module*** of the agent. The agent has the following morals inserted in its belief base;

```
% Morals
%% The agent
ideal(working,10).
ideal(neg(stealing),100).
%% The user
ideal(neg(lost(food)),100).
ideal(neg(lost(money)),100).
ideal(given(food),50).
ideal(given(money),50).
```

The agent should be able to feel some pride so for that the positive moral attitude of going to work is inserted, but with a low value.

Next is responsibility and for actions that the agent is responsible for is inserted when the agent performs an action so the ***init module*** is updated with changes for acquiring money and food.

```
if bel( X = getfood ) then{
  %% buy
  if bel( energy(E), E > 0, money ) then buyfood.
  %% steal
  if not(bel( money )) {
    if bel( me(Me) ) then insert(resp(Me,stealing)) + then stealfood.
  }
}
if bel( X = getmoney ) then{
  %% buy
  if bel( energy(E), E > 1, hunger(H), H < 4, me(Me))
    then insert(resp(Me,working)) + work.
  %% steal
  if bel( me(Me) ) then insert(resp(Me,stealing)) + stealmoney.
}
```

For dealing with responsibility of the user, the agent receive percept from the environment if it witnessed the user performing one of four actions; took money or food and gave money or food. These percepts are dealt with in the event module along with all other percepts and an example of one of these percepts would be;

```
% Percepts about the user actions.
if bel( percept(userPerformed("TAKEFOOD")) ) then
  insert( resp(user,lost(food)) ).
```

Next is the beliefs of the consequence of action, the beliefs about lost or given food or money are already dealt with but the agents own actions, working and stealing, are not. These, however, are fairly easy to implement as they can be inserted in the actions post-condition.

```
work{
  pre{ true }
  post{ doneAction, bel(working) }
}
stealmoney{
  pre{ true }
  post{ doneAction, bel(stealing) }
}
stealfood{
  pre{ true }
  post{ doneAction, bel(stealing) }
}
```

Both the emotion pride and shame along with reproach and gratitude is now part of the agents emotions, but a slight improvement can be made. Some action by the user can lead the agent to stealing so instead of blaming itself, the agent could blame the user for being responsible for his actions. When the agent is about to steal food a simple check to see if an existing emotion exist about the user stealing its money can make the agent blame the user. This can be done for both food or money and the updated action rules for stealing money in the ***main module*** is then;

```
%% steal money
if not(bel( money )) then {
  %% if an existing emotion toward another agent that resulted in
       stealing then he is responsible
  if bel(emo(_,(_,A),lost(money),_)) then insert(resp(A,stealing)) +
       stealfood.
  %% if no other is to blame then the agent itself is to blame
  if bel( me(Me) ) then insert(resp(Me,stealing)) + stealfood.
}
```

The action rule for stealing food is altered the same way.

Now that both well-being and action based emotions are implemented the agent is also able to feel more complex emotions such as anger or gratification.

Next is hope and fear and these emotions requires that the agent expects an outcome and may have a know probability of that outcome. Since the agent has a plan it is easy to insert what the agent expects by adding the following code in `emotionHandler.mod2g`;

```
%%update expected outcome
if bel(plan(X)) then {
  % insert expected outcomes for actions in the plan
  if bel(member(getfood,X)) then insert(expect(food)).
  if bel(member(getmoney,X)) then insert(expect(money)).
  if bel(member(sleep,X)) then insert(expect(energy(5))).
  if bel(member(eat,X)) then insert(expect(hunger(0))).
  % Delete expect outcomes that are not in the plan
  forall bel( expect(E), member(E,X) ) do delete( expect(X) ).
}
```

It is also important to delete expectations that are no longer in the plan as the agent can re-plan.

In this implementation there are no probability for these expectations in order to reduce overhead but these could be done by keeping track of how many time each action has succeeded.

The agent can express hope and fear, but this is only for the agents own action and not for the users potential actions. Expecting what the user would do is hard but can be done with first calculating the probability of the users actions in order to predict the user. At first the agent has not experienced any action from the user so it does not expect any of them to happen, but if the user starts to perform actions then a probability of the actions can be computed and if the chance is high enough then the agent may start to expect the users actions. Using the time and counting the amount of time an user have performed an action can be used to find the probability, so first the agent needs to keep track of the users actions. In the ***init module*** variables about the amount of time the user has performed and action is initialized;

```
% Variables for computing probability
stolenFood(0).
stolenMoney(0).
givenFood(0).
givenMoney(0).
```

These are then updated when the agent receives percepts about the user actions and an example of the updated percepts are;

```
if bel( percept(userPerformed("TAKEFOOD")), stolenFood(OldV), NewV is
    OldV+1 ) then
  delete( stolenFood(OldV) ) + insert( resp(user,lost(food)),
      stolenFood(NewV) ).
```

With the agent keeping track of the users actions the responsibility can be computed in `emotionHandler.mod2g` using the counter and time and if the probability is above a threshold then the expectation is inserted. For stolen food the code is;

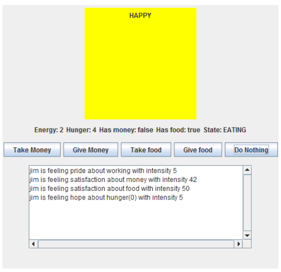
```
%% update probablilty
if bel(stolenFood(X), X > 0, time(T), P is X/T, prob(OldP,lost(food)),
    OldP \= P ) then
  delete(prob(OldP,lost(food))) + insert(prob(P,lost(food))).
if bel(stolenFood(X), X > 0, time(T), P is X/T), not(bel(
    prob(OldP,lost(food)) )) then insert(prob(P,lost(food))).

if bel( prob(X,lost(food)), X >= 0.3) then insert(expect(lost(food))).
```

In this case the agent is a little paranoid so when there is a 30% or higher chance of the user stealing food then the agent will expect the user to steal its food. The same is done for stolen money and given food or money.

With hope and fear implemented the agent is also able to feel emotions such as satisfaction and disappointment without additional implementation.

The last emotions is like and hate but all of the computation is done in `emotion.mod2g`, the only thing that is missing is an initial relation value between the agent and the user which can inserted in the belief base in the ***init module***.

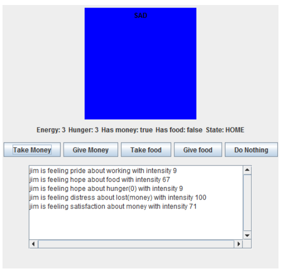
```
% Relations
relation(user,0).
```



Energy: 2 Hunger: 4 Has money: false Has food: true State: EATING

```
jim is feeling pride about working with intensity 5
jim is feeling satisfaction about money with intensity 42
jim is feeling satisfaction about food with intensity 50
jim is feeling hope about hunger(0) with intensity 5
```

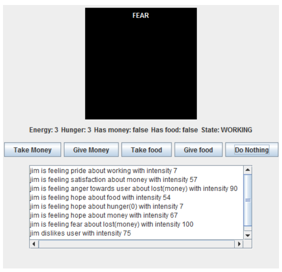
(a) Happy agent



Energy: 3 Hunger: 3 Has money: true Has food: false State: HOME

```
jim is feeling pride about working with intensity 9
jim is feeling hope about food with intensity 67
jim is feeling hope about hunger(0) with intensity 9
jim is feeling distress about lost(money) with intensity 100
jim is feeling satisfaction about money with intensity 71
```

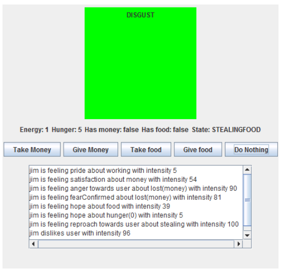
(b) Sad agent



Energy: 3 Hunger: 3 Has money: false Has food: false State: WORKING

```
jim is feeling pride about working with intensity 7
jim is feeling satisfaction about money with intensity 57
jim is feeling anger towards user about lost(money) with intensity 90
jim is feeling hope about food with intensity 54
jim is feeling hope about hunger(0) with intensity 7
jim is feeling hope about money with intensity 67
jim is feeling fear about lost(money) with intensity 100
jim dislikes user with intensity 75
```

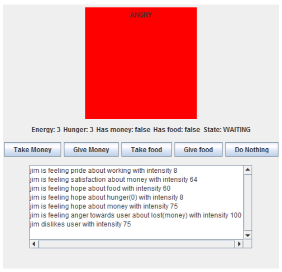
(c) Fearful agent



Energy: 1 Hunger: 5 Has money: false Has food: false State: STEALINGFOOD

```
jim is feeling pride about working with intensity 5
jim is feeling satisfaction about money with intensity 54
jim is feeling anger towards user about lost(money) with intensity 90
jim is feeling fearConfirmed about lost(money) with intensity 81
jim is feeling hope about food with intensity 39
jim is feeling hope about hunger(0) with intensity 5
jim is feeling reproach towards user about stealing with intensity 100
jim dislikes user with intensity 96
```

(d) Disgusted agent



Energy: 3 Hunger: 3 Has money: false Has food: false State: WAITING

```
jim is feeling pride about working with intensity 8
jim is feeling satisfaction about money with intensity 64
jim is feeling hope about food with intensity 60
jim is feeling hope about hunger(0) with intensity 8
jim is feeling hope about money with intensity 75
jim is feeling anger towards user about lost(money) with intensity 100
jim dislikes user with intensity 75
```

(e) Angry agent

Figure 6.3: Expression of Jim in the environment

The only emotions that the agent is not able to experience are the emotions fortune of others, this require that the agent knows what other agents desires and knows but in this case the only other agent is the user which is a human-being that simply press some buttons and his desires and beliefs are not know, almost as a form of deity. The agent is now able to experience 18 of the 22 emotions that are possible from the framework.

Chapter 7

Discussion

At the start of this paper the the following question was asked: ”How close will the agents emotion be to a humans emotions and is it able to express the same variety as a human?” Looking at SimpleJim the agent is able to show 18 of the 22 emotions of the framework where the missing 4 are fortunes of other. However these emotions can still be implemented in a case with agent-to-agent interaction where they can communicate their desires and there by resent or be happy for each other in reaching their desire. But this is still far from the potential of these emotions as most of such desires are not directly communicated between each other but derived from the context of the situations.

For example, if a person is entering a marathon without much practice we know that they desire to finish the race with out being directly told. Another example is done trough indirect communication such as facial expression, if we see a person who is disappointed in the outcome of an event we may feel pity in that person, as by the context of the facial expression and situation it can be derived that person desired a different outcome. This also means that fortunes of others emotions, the same with prospect-based emotion, does not follow a linear temporal logic, it is first when a person knows about the outcome that a emotions is realized even though the event has passed some time ago.

A lot of the proposed variables in the OCC model is not implemented in the framework as most of them are not possible in GOAL. This does influence the precision of the intensity of the emotions and can make an agent overreact in some cases compared to what a human would do in same situation. As proximity is not part of the framework the agent will not be able to different between how old the information of an outcome is. If a person is first told about a relatives death months after the event happened then the elicited emotion is not as strong compared if the the event happened few minutes ago. As stated in section 4.1 it is possible to implement however it demands a more information regarding outcomes and could be an improvement to the framework. Though the agent can experience pride or shame for itself but the lack of the variable *strength of unit* inhibits the agent from feeling pride or shame on the behalf of others or as an organization of agents. But as mentioned in section 4.1 this could be done through organization model as proposed by [Spu13] where the agent is able to create organization between each other. These organization can then be used to determine the variable *strengh of unit* to know what type of emotion to use for actions.

The global variable arousal is not part of the framework and in section 3.1 it was mentioned that there are multiple theories of emotion and some of these believes that arousal is a large part of a person emotions. This is however somewhat implemented through the mood of the agent as it is similar, since a series of positive outcome makes it more likely to experience positive emotions but they do not affect the intensity of positive emotions so this does distance the agents emotion from what a human may experience.

The OCC model was decided to be implemented with persistent emotions rather than interpret the agents state of mind to emotions which resulted in the need of a decay function to reduce the emotions relevance. With the current decay implementation all emotion decay with the same linear function and all but prospect based emotions are deleted when they reach a lower threshold. Beside the linear decay this thesis does not deal with differentiating between the emotional impact emotion has on the agent as the interpretation approach would have been able to, so emotion such as joy is assumed to have the same impact as disappointment. When an agent is angry with another agent the relation between them could have a factor of how the emotion anger is decaying so if the agents relation improves, the decay of the emotion can increase but if the relation is worsened then it may be haltered. Along with the relation the variable strength of unit can also be part of it as the anger between two agents with strong unity could decay slower. So all emotion with the same intensity will have the same lifespan even though they should have a different emotional impact and solving this problem demands complex decay function that takes variables for each emotion into account.

The same problem exist in the mood of the agent as each emotion affects the mood the same way, such as hope will affect the mood with the same effect as joy will. However, hope would probably not have the same effect as joy. Say that a person hopes five different outcomes, his mood would not be quite as positive as if he felt joy for five different outcomes, but unlike decay this problem cannot be solved with the interpretation approach. That is unless the intensity is designed from the start so the intensity of hope never reaches the same level of intensity of joy. But this is not the case for the current implementation however one way to solve it would be to apply a weight to each emotion in order to adjust how much they will impact the mood in the agent.

The agent is also able to show five facial expression based on the emotions that the agent is experiencing. So the agent is able to experience and express a large variety of emotions but there is still missing a lot before it can compete with a human being.

Amongst the expressed emotions proposed by Ekmann in [EO79] the agent is not able to experience surprise as it is not part of the OCC model.

As stated in section 4.2.5 surprise is not considered an emotion but a cognitive state and is part of the variable unexpectedness, however this does not necessarily mean that it cannot be part of the agents expression. If an outcome is considered unexpected then it is a surprise to the agent then it could give rise to a surprised expression and then transition over to the show a relevant emotions related the outcome that elicited the surprise. Sadly as unexpectedness is not part of the framework the implementation of the facial expression is still not feasible.

As it is implemented now it is only the two most intense emotion in each of the five expression categories that is used to compute what expression the agent should make, yet the distribution of the emotion across these five expression is not equal as 'happy' contains 11 different emotions where 'fear' only contains a single emotions. In order to solve this, the emotions should be weighted much the same way as discussed with mood as hope does not carry that big of a positive impact on the agent as joy does.

Since GOAL is declarative it is easy to read the emotions. However the intensity could be used to express that emotion better verbally instead of reading it from the value of the intensity directly. As stated in the OCC model each emotion has a set of tokens related to that emotion, joy has the tokens; contented, cheerful, delighted, glad, joyful and excited etc. Each of these token can be used to express joy with different intensity as each of these token are loaded differently, so if the intensity of joy low the token 'delighted' could be used an if it was high intensity the token 'ecstatic' could be used.

The thesis mention that emotions are an integrated part of the human decision making but the agent SimpleJim does not use these emotion in its decision making as stated in section 1. These emotion are only meant to be used as information about the agent state of mind, yet the framework could easily be used for one of the other categories. It was described in section 3.1.1 that appraisal structure also has a coping mechanic where the agent used these emotions to cope with the situation. In this case for the emotions can be used to decide what problem to engage first by looking at what emotion that affects the agent the most thereby moving the usage to the attention function category. Instead of using emotions as coping mechanic the agent can also use the knowledge of how emotion work to avoid or to seek certain emotions being realized and by that using it as a motivational function. An example of such a function can be seen in [MC04] where the emotions are used as a motivational factor for exploring unknown dynamic environments.

The agent SimpleJim could also be made better as currently the agent can steal money but it is redundant as the money are only used for food which Jim also is able to steal so there are no reason for stealing money. A solution would be to insert something that Jim desires but is not obtainable. This could be that Jim should pay his bills before the end of the month which will also give the possibility to have an event that Jim can expect.

It is evident that there a multiple things that can be improved and one of the first thing would be to use the interpretation approach which would have been a more suitable solution for implementing emotion as it avoids a lot of problems when dealing with intensity of emotions. The wish to avoid temporal logic has also made some things harder such as not having the proximity variable.

Chapter 8

Conclusion

Existing theories of emotion have been examined where the theory of appraisal has been chosen. A model called The OCC Model based on appraisal theory has been examined as a model for emotion which have proven most optimal for implementing in the agent-programming language GOAL. Based on this model a logical formalization has been developed to fit with GOAL and contains logic for 22 emotions ranging from both positive to negative emotions and focus on both the agent and other agents interaction. A framework has been created in GOAL with the formalization and given an agent the capabilities of experiencing the 22 emotions and is implemented with mood to enhance the effects of emotions. The framework also gives an agent the option to express these emotions both trough visual and verbal representations. Finally an agent is developed implemented with this framework and made to interact with a user trough emotional reaction to the user's actions.

Appendix A

Appendix

A.1 emotions.mod2g

```
module initEmotion{
  %% initial predicates are inserted here.
  %% should be called in the inital module of the agent.

  program[order=linearall]{

    %% initial value for emotion threshold
    if true then insert( mood(20,20) ).

    %% setup initial relations to other agents
    forall bel( agent(X) , me(A), X \== A ) do
      insert( relation(X,0) ).

    %% dummies that are never reached so that the program can run
        without complaining
    if bel( false ) then {
      if not(bel(expect(_))) then insert( expect(dummy) ).
      if not(bel(bel(_))) then insert( bel(dummy) ).
      if not(bel(prob(_,_))) then insert( prob(dummy,unknown) ).
      if not(bel(ideal(_,_))) then insert( ideal(dummy,unknown)).
      if not(goal(des(_,_))) then adopt( des(dummy,unknown)).
      if not(goal(undes(_,_))) then adopt( undes(dummy,unknown)).
    }

  }
```

```
}

module emotionsUpdate{
  %% Main module for handling emotions

  knowledge{
    % term_to_string(+Term,String)
    term_to_string(T,S) :-
      =..(T,[Name,X]),
      string_concat(Name,"(",S1),
      string_concat(S1,X,S2),
      string_concat(S2,")",S).
  }

  program[order=linearall]{

    %% decay %%
    if true then decay.

    %% Realizations of emotions &&
    if true then realization.

    %% map the emotions and finds the highest emotion along with
         updateing the emotionalstate %%
    if true then mapping.

    %% Tells the environment what it feels
    forall bel( emo(P,A,X,I) ) do {
      if bel(atom(X)) then {
        if bel(A = (A1,A2)) then emotionverbal(P,A1,A2,X,I).
        if bel(A \= (A1,A2)) then emotionverbal(P,A,X,I).
      }
      if bel( \+ atom(X), term_to_string(X,NewX)) then {
        if bel(A = (A1,A2)) then emotionverbal(P,A1,A2,NewX,I).
        if bel(A \= (A1,A2)) then emotionverbal(P,A,NewX,I).
      }
    }
  }
}

module realization {
  %% realizes emotions in the agent defined from emotional desires and
       beliefs along with
  %% other predicates and variables

  knowledge{

    % function to calculate new intensity

    %prospectInten(+Des,+Prob,?Int)
    prospectInten(Des, Prob, Int) :- Int is min(abs(Des * Prob),100).
    % averageInten(+Des,+Int1,?Int2)
    averageInten(Int1, Int2, Int3) :- Int3 is (Int1 + Int2)/2.

    incNegMood(Intensity, OldPM, OldNM, NewPM, NewNM) :-
    mood(OldPM,OldNM),
    Intensity > OldPM,
    NewPM is (Intensity+OldPM)/2,
    NewNM is (OldPM/Intensity)*OldNM.

  incPosMood(Intensity, OldPM, OldNM, NewPM, NewNM) :-
    mood(OldPM,OldNM),
    Intensity > OldNM,
    NewNM is (Intensity+OldNM)/2,
    NewPM is (OldNM/Intensity)*OldPM.

  abovePositiveMood(I) :-
    mood(P,_), I >= P.

  aboveNegativeMood(I) :-
    mood(_,N), I >= N.
}

program[order=linearall]{
  %% macroes
  #define pleased(X,D) goal( des(X,D) ).
  #define displeased(X,D) goal( undes(X,D) ).
  #define approving(X,Id) bel( ideal(X,Id), bel(X) ).
  #define disapproving(X,Id) bel( ideal(neg(X),Id), bel(X) ).
  #define deserving(A,X,D) goal( des(bel(A,X),D) ).
  #define undeserving(A,X,D) goal( undes(bel(A,X),D) ).
  #define otherDesire(A,X,D) bel(bel(des(A,X,D))).
  #define otherUndesire(A,X,D) bel(bel(undes(A,X,D))).

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%% Basic Emotions
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %%%% Prospective Consequence %%%%
  %% Hope %%
  forall pleased(X,D), not(bel( bel(X) )), bel( expect(X), me(A) ) do
      {
    if not(bel( emo(hope,A,X,_) )) then {
      if not(bel( prob(X,_) )), bel(prospectInten(D,1,I),
          abovePositiveMood(I) ) then {
        if true then insert( emo(hope,A,X,I) ).
        %% update mood
        if bel( incPosMood(I,PM,NM,NewPM,NewNM) ) then
          delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
      }
      if bel( prob(X,R), prospectInten(D,R,I), abovePositiveMood(I) )
          then {
        if true then insert( emo(hope,A,X,I) ).
        %% update mood
        if bel( incPosMood(I,PM,NM,NewPM,NewNM) ) then
          delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
      }
    }
  }
```

```
%% Fear %%
forall displeased(X,D), not(bel( bel(X) )), bel( expect(X), me(A))
     do {
  if not(bel( emo(fear,A,X,_) )) then {
    if not(bel( prob(X,_) )), bel(prospectInten(D,1,I),
         aboveNegativeMood(I) ) then {
      if true then insert( emo(fear,A,X,I) ).
      %% update mood
      if bel( incNegMood(I,PM,NM,NewPM,NewNM) ) then
        delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
    }
    if bel( prob(X,R), prospectInten(D,R,I), aboveNegativeMood(I) )
         then {
      if true then insert( emo(fear,A,X,I) ).
      %% update mood
      if bel( incNegMood(I,PM,NM,NewPM,NewNM) ) then
        delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
    }
  }
}

%%%% Actual unexpected Consequence %%%%

%% Joy %%
forall pleased(X,D), bel( bel(X), me(A), abovePositiveMood(D) ) do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(joy,A,X,OtherI) ) then {
    if bel( OtherI =< D ) then delete( emo(joy,A,X,OtherI) ) +
         insert( emo(joy,A,X,D) ).
  }
  if not(bel( emo(joy,A,X,_) )) then insert( emo(joy,A,X,D) ).
  %% update mood
  if bel( incPosMood(D,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
}

%% Distress %%
forall displeased(X,D), bel( bel(X), me(A), aboveNegativeMood(D) )
     do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(distress,A,X,OtherI) ) then {
    if bel( OtherI =< D ) then delete( emo(distress,A,X,OtherI) ) +
         insert( emo(distress,A,X,D) ).
  }
  if not(bel( emo(distress,A,X,_) )) then insert(
       emo(distress,A,X,D) ).
  %% update mood
  if bel( incNegMood(D,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
}

%%%% Action of the agent it self %%%%

%% Pride %%
```

```
forall approving(X,Id), bel( resp(A,X), me(A),
     abovePositiveMood(Id) ) do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(pride,A,X,OtherI) ) then {
    if bel( OtherI =< Id ) then delete( emo(pride,A,X,OtherI) ) +
         insert( emo(pride,A,X,Id) ).
  }
  if not(bel( emo(pride,A,X,_) )) then insert( emo(pride,A,X,Id) ).
  %% delete used variables
  if true then delete(resp(A,X)).
  %% update mood
  if bel( incPosMood(Id,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
}

%% Shame %%
forall disapproving(X,Id), bel( resp(A,X), me(A),
     aboveNegativeMood(Id) ) do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(shame,A,X,OtherI) ) then {
    if bel( OtherI =< Id ) then delete( emo(shame,A,X,OtherI) ) +
         insert( emo(shame,A,X,Id) ).
  }
  if not(bel( emo(shame,A,X,_) )) then insert( emo(shame,A,X,Id) ).
  %% update mood
  if bel( incNegMood(Id,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% delete used variables
  if true then delete(resp(A,X)).
}

%%%% Action of another agent %%%%

%% Admiration %%
forall approving(X,Id), bel( resp(A,X), me(Me), Me \= A,
     abovePositiveMood(Id) ) do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(admiration,(Me,A),X,OtherI) ) then {
    if bel( OtherI =< Id ) then delete(
         emo(admiration,(Me,A),X,OtherI) ) + insert(
         emo(admiration,(Me,A),X,Id) ).
  }
  if not(bel( emo(admiration,(Me,A),X,_) )) then insert(
       emo(admiration,(Me,A),X,Id) ).
  %%update relationship
  if bel( relation(A,Rel), averageInten(Rel,Id,NewRel) ) then
    delete( relation(A,Rel) ) + insert( relation(A,NewRel) ).
  %% delete used variables
  if true then delete(resp(A,X)).
  %% update mood
  if bel( incPosMood(Id,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
}

%% Reproach %%
```

```
forall disapproving(X,Id), bel( resp(A,X), me(Me), Me \= A,
     aboveNegativeMood(Id) ) do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(reproach,(Me,A),X,OtherI) ) then {
    if bel( OtherI =< Id ) then delete(
        emo(reproach,(Me,A),X,OtherI) ) + insert(
        emo(reproach,(Me,A),X,Id) ).
  }
  if not(bel( emo(reproach,(Me,A),X,_) )) then insert(
      emo(reproach,(Me,A),X,Id) ).
  %% update relationship
  if bel( relation(A,Rel), NegI is -1*Id,
      averageInten(Rel,NegI,NewRel) ) then
    delete( relation(A,Rel) ) + insert( relation(A,NewRel) ).
  %% update mood
  if bel( incNegMood(Id,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% delete used variables
  if true then delete(resp(A,X)).
}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Complex Emotions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%% Related Consequence and Action %%%%
%%% against the agent itself %%%

%% Gratification %%
forall bel( emo(pride,A,X,Ip), emo(joy,A,X,Ij) , me(A),
     averageInten(Ip,Ij,I), abovePositiveMood(I) ) do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(gratification,A,X,OtherI) ) then {
    if bel( OtherI =< I ) then delete( emo(gratification,A,X,OtherI)
        ) + insert( emo(gratification,A,X,I) ).
  }
  if not(bel( emo(gratification,A,X,_) )) then insert(
      emo(gratification,A,X,I) ).
  %% update mood
  if bel( incPosMood(I,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% remove eliciting emotions and redundant variables
  if true then delete( emo(pride,A,X,Ip), emo(joy,A,X,Ij)).
}

%% Remorse %%
forall bel( emo(shame,A,X,Is), emo(distress,A,X,Id), me(A),
     averageInten(Is,Id,I), aboveNegativeMood(I) ) do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(remorse,A,X,OtherI) ) then {
    if bel( OtherI =< I ) then delete( emo(remorse,A,X,OtherI) ) +
        insert( emo(remorse,A,X,I) ).
  }
  if not(bel( emo(remorse,A,X,_) )) then insert( emo(remorse,A,X,I)
      ).
  %% update mood
  if bel( incNegMood(I,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% remove eliciting emotions and redundant variables
  if true then delete( emo(shame,A,X,Is), emo(distress,A,X,Id)).
}

%%%%% against another agent %%%%%

%% Gratitude %%
forall bel( emo(admiration,(Me,A),X,Ia), emo(joy,Me,X,Ij), me(Me),
      averageInten(Ia,Ij,I), abovePositiveMood(I) ) do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(gratitude,(Me,A),X,OtherI) ) then {
    if bel( OtherI =< I ) then delete(
        emo(gratitude,(Me,A),X,OtherI) ) + insert(
        emo(gratitude,(Me,A),X,I) ).
  }
  if not(bel( emo(gratitude,(Me,A),X,_) )) then insert(
      emo(gratitude,(Me,A),X,I) ).
  %% update relation
  if bel(relation(A,Rel), averageInten(Rel,I,NewRel) ) then
    delete( relation(A,Rel) ) + insert( relation(A,NewRel) ).
  %% update mood
  if bel( incPosMood(I,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% remove eliciting emotions and redundant variables
  if true then delete( emo(admiration,(Me,A),X,Ia),
      emo(joy,Me,X,Ij)).
}

%% Anger %%
forall bel( emo(reproach,(Me,A),X,Ir), emo(distress,Me,X,Id),
      me(Me), averageInten(Ir,Id,I), aboveNegativeMood(I) ) do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(anger,(Me,A),X,OtherI) ) then {
    if bel( OtherI =< I ) then delete( emo(anger,(Me,A),X,OtherI) )
        + insert( emo(anger,(Me,A),X,I) ).
  }
  if not(bel( emo(anger,(Me,A),X,_) )) then insert(
      emo(anger,(Me,A),X,I) ).
  %% update relation
  if bel( relation(A,Rel), NegI is -1*I,
      averageInten(Rel,NegI,NewRel)) then
    delete( relation(A,Rel) ) + insert( relation(A,NewRel) ).
  %% update mood
  if bel( incNegMood(I,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% remove eliciting emotions and redundant variables
  if true then delete( emo(reproach,(Me,A),X,Ir),
      emo(distress,Me,X,Id)).
}

%%%%% Expected Outcome %%%%%
```

```
%% Satisfaction %%
forall goal(des(X,D)), bel( emo(hope,A,X,Ih), bel(X), me(A),
     averageInten(Ih,D,I), abovePositiveMood(I) ) do{
  %% if the emotion for the same thing exist then replace it with
  if bel(emo( satisfaction,A,X,OtherI) ) then {
    if bel( OtherI =< I) then delete( emo( satisfaction,A,X,OtherI)
         ) + insert( emo(satisfaction,A,X,I) ).
  }
  if not(bel( emo(satisfaction,A,X,_) )) then insert(
       emo(satisfaction,A,X,I) ).
  %% update mood
  if bel( incPosMood(I,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% remove eliciting emotions and redundant variables
  if true then delete( emo(hope,A,X,Ih) ).
  if bel(expect(X)) then delete( expect(X) ).
  if bel( emo(joy,A,X,Ij) ) then delete( emo(joy,A,X,Ij) ).
}


%% Dissapointment %%
forall goal(des(X,D)), bel( emo(hope,A,X,Ih), bel(neg(X)), me(A),
     averageInten(Ih,D,I), aboveNegativeMood(I) ) do{
  %% if the emotion for the same thing eexist then replace it with
  if bel(emo(dissapointment,A,X,OtherI) ) then {
    if bel( OtherI =< I) then delete( emo(dissapointment,A,X,OtherI)
         ) + insert( emo(dissapointment,A,X,I) ).
  }
  if not(bel( emo( dissapointment,A,X,_) )) then insert(
       emo(dissapointment,A,X,I) ).
  %% update mood
  if bel( incNegMood(I,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% remove eliciting emotions and redundant variables
  if true then delete( emo(hope,A,X,Ih) ).
  if bel(expect(X)) then delete(expect(X)).
}
%% Relief %%
forall goal(undes(X,D)), bel( emo(fear,A,X,I), bel(neg(X)), me(A),
     averageInten(If,D,I), abovePositiveMood(I) ) do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(relief,A,X,OtherI) ) then {
    if bel( OtherI =< I) then delete( emo(relief,A,X,OtherI) ) +
         insert( emo(relief,A,X,I) ).
  }
  if not(bel( emo(relief,A,X,_) )) then insert( emo(relief,A,X,I) ).
  %% update mood
  if bel( incPosMood(I,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% remove eliciting emotions and redundant variables
  if true then delete( emo(fear,A,X,I) ).
  if bel(expect(X)) then delete(expect(X)).
}

%% FearConfirmed %%
```

```
forall goal(undes(X,D)), bel( emo(fear,A,X,If), bel(X), me(A),
     averageInten(If,D,I), aboveNegativeMood(I) ) do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(fearConfirmed,A,X,OtherI) ) then {
    if bel( OtherI =< I) then delete( emo(fearConfirmed,A,X,OtherI)
        ) + insert( emo(fearConfirmed,A,X,I) ).
  }
  if not(bel( emo(fearConfirmed,A,X,_) )) then insert(
      emo(fearConfirmed,A,X,I) ).
  %% update mood
  if bel( incNegMood(I,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% remove eliciting emotions and redundant variables
  if true then delete( emo(fear,A,X,If) ).
  if bel( emo(distress,A,X,Ij) ) then delete( emo(distress,A,X,Ij) ).
  if bel(expect(X)) then delete(expect(X)).
}

%%%%% Fortune of others %%%%%

%% Happy For %%
forall deserving(A,X,D), otherDesire(A,X,OtherD), bel( bel(X),
     me(Me), averageInten(D,OtherD,I), abovePositiveMood(I) ) do {
  %% if the emotion for the same thing exist then replace it with
  if bel( emo(happyFor,(Me,A),X,OtherI) ) then {
    if bel( OtherI =< I) then delete( emo(happyFor,(Me,A),X,OtherI)
        ) + insert( emo(happyFor,(Me,A),X,I) ).
  }
  if not(bel( emo( happyFor,(Me,A),X,_) )) then insert(
      emo(happyFor,(Me,A),X,I) ).
  %% update mood
  if bel( incPosMood(I,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% remove eliciting emotions and redundant variables
  if true then drop( des(bel(A,X),D) ).
}

%% Resentment %%
forall undeserving(A,X,D), otherDesire(A,X,OtherD), bel( bel(X),
     me(Me), averageInten(D,OtherD,I), aboveNegativeMood(I) ) do {
  %% if the emotion for the same thing eexist then replace it with
  if bel( emo(resentment,(Me,A),X,OtherI) ) then {
    if bel( OtherI =< I) then delete(
        emo(resentment,(Me,A),X,OtherI) ) + insert(
        emo(resentment,(Me,A),X,I) ).
  }
  if not(bel( emo(resentment,(Me,A),X,_) )) then insert(
      emo(resentment,(Me,A),X,I) ).
  %% update mood
  if bel( incNegMood(I,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% remove eliciting emotions and redundant variables
  if true then drop( undes(bel(A,X),D) ).
}
```

```
%% Gloating %%
forall deserving(A,X,D), otherUndesire(A,X,OtherD), bel( bel(X),
    me(Me), averageInten(D,OtherD,I), abovePositiveMood(I) ) do {
  %% if the emotion for the same thing eexist then replace it with
  if bel( emo(gloating,(Me,A),X,OtherI) ) then {
    if bel( OtherI =< I) then delete( emo( gloating,(Me,A),X,OtherI)
        ) + insert( emo(gloating,(Me,A),X,I) ).
  }
  if not(bel( emo( gloating,(Me,A),X,_) )) then insert(
      emo(gloating,(Me,A),X,I) ).
  %% update mood
  if bel( incPosMood(I,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% remove eliciting emotions and redundant variables
  if true then drop( des(bel(A,X),D) ).
}

%% Sorry for %%
forall undeserving(A,X,D), otherUndesire(A,X,OtherD), bel( bel(X),
    me(Me), averageInten(D,OtherD,I), aboveNegativeMood(I) ) do {
  %% if the emotion for the same thing eexist then replace it with
  if bel( emo(sorryFor,(Me,A),X,OtherI) ) then {
    if bel( OtherI =< I) then delete( emo( sorryFor,(Me,A),X,OtherI)
        ) + insert( emo(sorryFor,(Me,A),X,I) ).
  }
  if not(bel( emo( sorryFor,(Me,A),X,_) )) then insert(
      emo(sorryFor,(Me,A),X,I) ).
  %% update mood
  if bel( incNegMood(I,PM,NM,NewPM,NewNM) ) then
    delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).
  %% remove eliciting emotions and redundant variables
  if true then drop( undes(bel(A,X),D) ).
}

%%%%% update the dislike and like of a person %%%%%%
%% add new emotion of know releations %%
forall bel( relation(A,NewI), me(Me) ) do {
  % Like
  if bel( NewI > 20 ) then{
    if bel( emo(like,Me,A,OldI), NewI \= OldI ) then
      delete( emo(like,Me,A,OldI) ) + insert( emo(like,Me,A,NewI) ).
    if not(bel( emo(like,Me,A,_) )) then insert( emo(like,Me,A,NewI)
        ).
  }
  % dislike
  if bel( NewI < -20, PosI is abs(NewI) ) then {
    if bel( emo(dislike,Me,A,OldI), PosI \= OldI ) then
      delete( emo(dislike,Me,A,OldI) ) + insert(
          emo(dislike,Me,A,PosI) ).
    if not(bel( emo(dislike,Me,A,_) )) then insert(
        emo(dislike,Me,A,PosI) ).
  }
  % delete unneeded like or dislike
  if bel(NewI < 20, NewI > -20) then{
        if bel( emo(dislike,Me,A,OldI) ) then delete(
            emo(dislike,Me,A,OldI) ).
        if bel( emo(like,Me,A,OldI) ) then delete( emo(like,Me,A,OldI) ).
      }
    }

    %% update optained emotional goals and beliefs
    %desire
    forall goal( des(X,D) ) do {
      % confirmed
      if bel( bel(X) ) then drop( des(X,D) ) + delete( bel(X) ).
      % disconfirmed
      if bel( bel(neg(X)) ) then drop( des(X,D) ) + delete( bel(neg(X))
          ).
    }
    %undesire
    forall goal( undes(X,D) ) do {
      % disconfirmed
      if bel( bel(X) ) then drop( undes(X,D) ) + delete( bel(X) ).
      % confirmed
      if bel( bel(neg(X)) ) then drop( des(X,D) ) + delete( bel(neg(X))
          ).
    }

    %% delete unused outcomes
    forall bel( bel(X) ) do delete( bel(X) ).

    %% Delete prospect based emotions if they are no longer expected
    forall not(bel( expect(X) )), bel( emo(E,A,X,I),
        member(E,[fear,hope]) ) do delete( emo(E,A,X,I) ).

  }
}

module decay{
  % Decay the intensity and threshold of emotions

  knowledge{

    % decay(+OldInt,?NewInt)
    decayEmotion(OldInt,NewInt) :- NewInt is OldInt*0.90.

    % decayMood(+X,?NewX)
    decayMood(X,Y) :-
      Y is 1/100*(X-40)*X+24.
  }

  program[order=linearall]{

    %% Decay emotion %%
    forall bel( emo(E,A,X,I), \+ member(E,[hope,fear]) ) do {
      if bel( I =< 1) then
        delete( emo(E,A,X,I) ).
      if bel( I > 1 , decayEmotion(I,NewI)) then
        delete( emo(E,A,X,I) ) +
        insert( emo(E,A,X,NewI) ).
    }

    %% Decay emotion %%
    forall bel( emo(E,A,X,I) , member(E,[hope,fear]) ) do {
      if bel( I > 1 , decayEmotion(I,NewI)) then
        delete( emo(E,A,X,I) ) +
        insert( emo(E,A,X,NewI) ).

    }

    %% Decay mood%%
    if bel( mood(PM,NM), decayMood(PM,NewPM), decayMood(NM,NewNM) ) then
      delete( mood(PM,NM) ) + insert( mood(NewPM,NewNM) ).

  }
}

module mapping{
  % Maps all the agents emotion to five type of emotions that can be
       visualized
  % happy, sad, fear, disgust and angry

  knowledge{

    % cutList2(+List1,?List2)
    cutList2([One,Two|_],[One,Two]).
    cutList2(L,L).

    % listSum(+List,?Sum).
    listSum([Item], Item).
    listSum([Item1,Item2 | Tail], Sum) :- Item is Item1 + Item2,
         listSum([Item|Tail], Sum).

    %% mapping of the 22 emotions i have implemented into 5 emotions %%
    happyEmotions([hope,joy,pride,admiration,gratitude,
            gratification,satisfaction,relief,happyFor,gloating]).
    sadEmotions([distress,fearConfirmed,dissapointment,remorse,pity]).
    fearEmotions([fear]).
    disgustEmotions([reproach,shame]).
    angryEmotions([anger,resentment]).

    % calcEmotion(+List,?X)
    calcEmotion(List,X) :-
      findall(Int,(emo(Predicate,_,_,Int),
          member(Predicate,List)),List1),
      sort(List1,List2),
      cutList2(List2,FinalList),
      listSum(FinalList,X),!.
    calcEmotion(List,0).

    % descending(_,+Elem1,+Elem2)
    descending('<', (C1,_),(C2,_)) :- C1>=C2.
    descending('>', (C1,_),(C2,_)) :- C1<C2.
    % highestExpression(?Predicate,?X)
    highestExpression(Predicate,X) :-
      happyEmotions(HappyEmotion),calcEmotion(HappyEmotion,H),
      sadEmotions(SadEmotion), calcEmotion(SadEmotion,S),
      fearEmotions(FearEmotion), calcEmotion(FearEmotion,F),
      disgustEmotions(DisgustEmotion), calcEmotion(DisgustEmotion,D),
      angryEmotions(AngryEmotion), calcEmotion(AngryEmotion,A),
      predsort(descending,[(H,happy),(S,sadness),(F,fear
          ),(D,disgust),(A,anger)],List),
      List = [(X,Predicate)|_].
  }

  program[order=linearall]{

    %% Maps the emotion to 5 facial expressions and
    %% send the highest expression to the environment
    if bel( highestExpression(P,V), V > 0 ) then emotiondisplay(P).
  }
}
```

A.2 Agent

A.2.1 Jim.goal

```
#import "planner.mod2g".
#import "emotion.mod2g".
#import "emotionHandler.mod2g".

init module{
  beliefs{
    % initial plan
    plan([]).
    % Moral
    %% The agent
    ideal(working,10).
    ideal(neg(stealing),100).
    %% The user
    ideal(neg(lost(food)),100).
    ideal(neg(lost(money)),100).
    ideal(given(food),50).
    ideal(given(money),50).

    % Variables for computing probability
    stolenFood(0).
    stolenMoney(0).
    givenFood(0).
    givenMoney(0).

    % Relations
    relation(user,0).
  }
```

```
program{

  %% handle initial percepts
  if bel( percept(energy(X))) then insert( energy(X) ).
  if bel( percept(hunger(X))) then insert( hunger(X) ).
  if bel( percept(state(X)) ) then insert( state(X) ).
  if bel( percept(food) ) then insert(food ).
  if bel( percept(money) ) then insert( money ).
  if bel( percept(time(Time)) ) then insert( time(Time) ).

  %% Perform planning
  if true then planner.
  if true then initEmotion.
  if true then emotionHandling.

}

actionspec{
  work{
    pre{ true }
    post{ doneAction, bel(working) }
  }
  stealmoney{
    pre{ true }
    post{ doneAction, bel(stealing) }
  }
  stealfood{
    pre{ true }
    post{ doneAction, bel(stealing) }
  }
  eat{
    pre{ food }
    post{ doneAction }
  }
  buyfood{
    pre{ money }
    post{ doneAction }
  }
  sleep{
    pre{ true }
    post{ doneAction }
  }
  return{
    pre{ true }
    post{ doneAction }
  }
  wait{
    pre{ true }
    post{ doneAction }
  }
  emotionverbal(P,A,X,I){
    pre{ true }
    post{ true }
  }
```

```
    emotionverbal(P,A1,A2,X,I){
      pre{ true }
      post{ true }
    }
    emotiondisplay(X){
      pre{ true }
      post{ true }
    }
  }

}

main module{
  program{
    if not( bel(doneAction) ) then {
      if bel( state('WAITING') ) then {
        if bel( plan([X|T]) ) then {
          if bel( X = eat ) then eat.
          if bel( X = sleep ) then sleep.
          if bel( X = getfood ) then{
            %% buy
            if bel( energy(E), E > 0, money ) then buyfood.
            %% steal
            if not(bel( money )) then {
              %% if an existing emotion toward another person that
                   resulted in stealing then he is responsible
              if bel(emo(_,(_,A),lost(money),_)) then
                   insert(resp(A,stealing)) + stealfood.
              %% if no other is to blame then the agent itself is to blame
              if bel( me(Me) ) then insert(resp(Me,stealing)) + stealfood.
            }
          }
          if bel( X = getmoney ) then{
            %% buy
            if bel( energy(E), E > 1, hunger(H), H < 4, me(Me)) then
                 insert(resp(Me,working)) + work.
            %% steal
            %% if an existing emotion toward another person that resulted
                 in stealing then he is responsible
            if bel(emo(_,(_,A),lost(money),_)) then
                 insert(resp(A,stealing)) + stealmoney.
            %% if no other is to blame then the agent itself is to blame
            if bel( me(Me) ) then insert(resp(Me,stealing)) + stealmoney.
          }
        }
      }
      %Inbetween actions
      if bel( state('HOME') ) then wait.
      if not(bel( state('WAITING') ; state('HOME') )) then return.
    }
  }
}

event module{
```

```
program{

  %% update percepts
  %% handle boolean percepts
  if bel( percept(food) ), not(bel( food )) then insert( food,
       bel(food)).
  if bel( food ), not(bel( percept(food) )) then delete( food ).
  if bel( percept(money) ), not(bel( money )) then insert( money,
       bel(money)).
  if bel( money ), not(bel( percept(money) )) then delete( money ).

  %% handle always percepts
  if bel( percept(energy(E)), energy(OldE), E \= OldE) then delete(
       energy(OldE) ) + insert( energy(E) ,bel(energy(E)) ).
  if bel( percept(hunger(H)), hunger(OldH), H \= OldH) then delete(
       hunger(OldH) ) + insert( hunger(H), bel(hunger(H)) ).
  if bel( percept(state(S)), state(OldS), S \= OldS) then delete(
       state(OldS) ) + insert(state(S)).

  % Emotional beliefs about given or lost
  if bel( percept(lost(food)) ) then insert( bel(lost(food)) ).
  if bel( percept(given(food)) ) then insert( bel(given(food)) ).
  if bel( percept(lost(money)) ) then insert( bel(lost(money)) ).
  if bel( percept(given(money)) ) then insert( bel(given(money)) ).

  % Percepts about the user actions.
  if bel( percept(userPerformed("TAKEFOOD")), stolenFood(OldV), NewV
       is OldV+1 ) then
    delete( stolenFood(OldV) ) + insert( resp(user,lost(food)),
        stolenFood(NewV) ).
  if bel( percept(userPerformed("TAKEMONEY")), stolenMoney(OldV),
       NewV is OldV+1 ) then
    delete( stolenMoney(OldV) ) + insert( resp(user,lost(money)),
        stolenMoney(NewV) ).
  if bel( percept(userPerformed("GIVEFOOD")), givenFood(OldV), NewV
       is OldV+1 ) then
    delete( givenFood(OldV) ) + insert( resp(user,given(food)),
        givenFood(NewV) ).
  if bel( percept(userPerformed("GIVEMONEY")), givenMoney(OldV), NewV
       is OldV+1 ) then
    delete( givenMoney(OldV) ) + insert( resp(user,given(money)),
        givenMoney(NewV) ).

  % new timestep
  if bel( percept(time(T)), time(OldT), T \= OldT) then{
    if true then delete( time(OldT) ) + insert( time(T) ).
    %% delete action
    if bel( doneAction ) then delete( doneAction ).
    if bel( state('HOME'), plan([X|Tail]) ) then
      delete( plan([X|Tail]) ) + insert(plan(Tail)).

    %%perform planning
    if true then planner.
    %% update emotions
    if true then emotionHandling.
```

```
      if true then emotionsUpdate.
    }
  }
}
```

A.2.2 planner.mod2g

```
module NewPlan{
  %% plan action based on the agents energy and hunger along with what
       is in its inventory.
  knowledge{
    notPlanned(Action, Plan) :- \+ member(Action,Plan), !.
  }

  program[order = linearall]{
    %% eating
    if bel( hunger(X), X > 0) then{
      if bel( plan([C|T]), notPlanned(eat,[C|T]) ) then delete(
          plan([C|T]) ) + insert( plan([eat,C|T]) ).
      if bel( plan([]) ) then delete(plan([])) + insert( plan([eat]) ).
    }
    %% aquireing food
    if bel( plan([eat|T]), notPlanned(getfood,[T]) ), not(bel( food ))
        then delete( plan([eat|T]) ) + insert(plan([getfood,eat|T])).
    %% aquring money
    if bel( plan([getfood|T]), hunger(H), H < 5,
        notPlanned(getmoney,[T]) ), not(bel( money )) then delete(
        plan([getfood|T]) ) + insert(plan([getmoney,getfood|T])).

    %% restoring energy
    if bel( energy(E), E < 2, hunger(X), X < 4 ) then {
      if bel( plan([C|T]), notPlanned(sleep,[C|T]) ) then
          delete(plan([C|T])) + insert( plan([sleep,C|T])).
      if bel(plan([])) then delete(plan([])) + insert( plan([sleep]) ).
    }

    %% if nothing in the plan then try to obtain wither money or food
    if bel(plan([])), not(bel( money) ) then delete(plan([])) +
        insert(plan([getmoney])).
    if bel(plan([])), not(bel( food )) then delete(plan([])) +
        insert(plan([getfood])).
  }
}

module planner{
  %% checks the plan
  program[order=linearall]{
    % checks if the next action cen be performed
    if bel( plan([X|T]) ) then {
      if bel( X = eat, expect(hunger(0))), not(bel( food )) then delete(
          plan([X|T]), expect(hunger(0)) ) + insert( plan([])).
      if bel( X = sleep, expect(energy(5)), hunger(H), H < 4 ) then
          delete( plan([X|T]), expect(energy(5)) ) + insert( plan([])).
    if bel( X = work, expect(money), hunger(H), H < 4, energy(E), E >
        2 ) then delete( plan([X|T]), expect(money) ) +
        insert(plan([])).
   }
   %% if no plan then replan
   if true then NewPlan.
  }
}
```

A.3 Environment

A.3.1 EnvironmentInterface.java

```
package SimpleJim;

import java.util.LinkedList;
import java.util.Map;
import eis.EIDefaultImpl;
import eis.exceptions.*;
import eis.iilang.*;

public class EnvironmentInterface extends EIDefaultImpl {

  private static final long serialVersionUID = 1L;
  private Environment environment;
  LinkedList<Parameter> p = null;

  // enumerator for initial variables received from GOAL
  enum InitKey {
    MONEY, FOOD, ENERGY, DEBUG, UNKNOWN, HUNGER;

    static InitKey toEnum(String string) {
      try {
        return valueOf(string.replaceAll("([A-Z])",
            "_$1").toUpperCase());
      } catch (Exception e) {
        return UNKNOWN;
      }
    }

    static String toString(InitKey key) {
      String[] strings = key.toString().split("_");
      StringBuilder sb = new StringBuilder(strings[0].toLowerCase());
      for (int i = 1; i < strings.length; i++) {
        sb.append(strings[i].substring(0, 1));
        sb.append(strings[i].substring(1).toLowerCase());
      }
      return sb.toString();
    }
  }
```

```
enum AgentAction {
  WORK, STEALMONEY, EAT, BUYFOOD, STEALFOOD, SLEEP, RETURN, WAIT,
      EMOTIONVERBAL, EMOTIONDISPLAY, UNKNOWN;

  static AgentAction toEnum(String string) {
    try {
      return valueOf(string.replaceAll("([A-Z])",
          "_$1").toUpperCase());
    } catch (Exception e) {
      return UNKNOWN;
    }
  }

  static String toString(AgentAction action) {
    String[] strings = action.toString().split("_");
    StringBuilder sb = new StringBuilder(strings[0].toLowerCase());
    for (int i = 1; i < strings.length; i++) {
      sb.append(strings[i].substring(0, 1));
      sb.append(strings[i].substring(1).toLowerCase());
    }
    return sb.toString();
  }
}

enum UserAction {
  GIVEMONEY, TAKEMONEY, GIVEFOOD, TAKEFOOD, NOTHING, UNKNOWN;

  static UserAction toEnum(String string) {
    try {
      return valueOf(string.replaceAll("([A-Z])",
          "_$1").toUpperCase());
    } catch (Exception e) {
      return UNKNOWN;
    }
  }

  static String toString(AgentAction action) {
    String[] strings = action.toString().split("_");
    StringBuilder sb = new StringBuilder(strings[0].toLowerCase());
    for (int i = 1; i < strings.length; i++) {
      sb.append(strings[i].substring(0, 1));
      sb.append(strings[i].substring(1).toLowerCase());
    }
    return sb.toString();
  }
}

public EnvironmentInterface() {
  super();
}

public static void main(String[] args) {
}

public void addAgentEntity(String entity) {
  try {
    addEntity(entity, "agent");
  } catch (EntityException e) {
    e.printStackTrace();
  }
}

public void deleteAgentEntity(String entity) {

  try {
    deleteEntity(entity);
  } catch (RelationException e) {
    e.printStackTrace();
  } catch (EntityException e) {
    e.printStackTrace();
  }
}

// Apply the initial variables received from GOAL to the environment
@Override
public void init(Map<String, Parameter> parameters) throws
     ManagementException {
  super.init(parameters);

  // defaults
  boolean food = false;
  boolean money = false;
  int energy = 5;
  int hunger = 0;
  boolean debug = false;

  for (String key : parameters.keySet()) {
    Parameter parameter = parameters.get(key);
    switch (InitKey.toEnum(key)) {

      case MONEY:
        money = true;
      case FOOD:
        food = true;
      case ENERGY:
        if (!(parameter instanceof Numeral))
          throw new ManagementException("Expected numeral");
        energy = ((Numeral) parameter).getValue().intValue();
        if (energy < 0)
          throw new ManagementException("Expected positive value");
        break;
      case HUNGER:
        if (!(parameter instanceof Numeral))
          throw new ManagementException("Expected numeral");
        hunger = ((Numeral) parameter).getValue().intValue();
        if (hunger < 0)
          throw new ManagementException("Expected positive value");
        break;
      case DEBUG:
        debug = true;
        break;
      default:
        throw new ManagementException("Unknown initialization key: " +
            key);
    }
  }

  environment = new Environment(this, money, food, energy, hunger,
      debug);

  pause();
  start();
}

@Override
public String requiredVersion() {
  return "0.3";
}

// manage percepts to agents
@Override
protected LinkedList<Percept> getAllPerceptsFromEntity(String entity)
    throws PerceiveException, NoEnvironmentException {
  LinkedList<Percept> percepts = new LinkedList<Percept>();

  Agent a = environment.agent;

  //global percepts

  // current state
  percepts.add(new Percept("state",
      new Identifier(a.currentState.toString())
  ));
  //food and money
  if(a.food){
    percepts.add(new Percept(
        "food"
    ));
  } else if (a.money)
  percepts.add(new Percept(
      "money"
  ));

  // energy
  percepts.add(new Percept(
      "energy",
      new Numeral(a.energy)
  ));

  //hunger
  percepts.add(new Percept(
      "hunger",
      new Numeral(a.hunger)
  ));
```

```
percepts.add(new Percept(
    "time",
    new Numeral(environment.getTime())
));

// percepts for obtained or stolen items along with percepts if the
     agent witnessed it.
switch(environment.lastUserAction){
case GIVEFOOD:
  percepts.add(new Percept(
      "given",
      new Identifier("food")
  ));
  if(environment.agentWitnessedAction){
    percepts.add(new Percept(
        "userPerformed",
        new Identifier(environment.lastUserAction.toString())
    ));
  }
  break;
case GIVEMONEY:
  percepts.add(new Percept(
      "given",
      new Identifier("money")
  ));
  if(environment.agentWitnessedAction){
    percepts.add(new Percept(
        "userPerformed",
        new Identifier(environment.lastUserAction.toString())
    ));
  }
  break;
case TAKEFOOD:
  percepts.add(new Percept(
      "lost",
      new Identifier("food")
  ));
  if(environment.agentWitnessedAction){
    percepts.add(new Percept(
        "userPerformed",
        new Identifier(environment.lastUserAction.toString())
    ));
  }
  break;
case TAKEMONEY:
  percepts.add(new Percept(
      "lost",
      new Identifier("money")
  ));
  if(environment.agentWitnessedAction){
    percepts.add(new Percept(
        "userPerformed",
        new Identifier(environment.lastUserAction.toString())
    ));
  }
```

```
      break;
    default:
      break;

    }

    // resets the users last action
    if(environment.lastUserAction != UserAction.NOTHING)
      environment.lastUserAction = UserAction.NOTHING;

    return percepts;
}

  @Override
  protected boolean isSupportedByEnvironment(Action action) {
    return AgentAction.toEnum(action.getName()) != AgentAction.UNKNOWN;
  }

  @Override
  protected boolean isSupportedByType(Action action, String type) {
    return AgentAction.toEnum(action.getName()) != AgentAction.UNKNOWN
         && type.equals("actor");
  }

  @Override
  protected boolean isSupportedByEntity(Action action, String entity) {
    return AgentAction.toEnum(action.getName()) != AgentAction.UNKNOWN
         && getEntities().contains(entity);
  }

  // manage action received from GOAL
  @Override
  protected Percept performEntityAction(String entity, Action action)
        throws ActException {

    AgentAction actorAction = AgentAction.toEnum(action.getName());

    if(actorAction == AgentAction.EMOTIONDISPLAY) {
      String emotion =
           ((Identifier)action.getParameters().get(0)).getValue();
      environment.agent.displayEmotion(entity, actorAction, emotion);
    }else if(actorAction == AgentAction.EMOTIONVERBAL) {

      if (action.getParameters().size() == 5){

        String predicate =
             ((Identifier)action.getParameters().get(0)).getValue();
        String agent1 =
             ((Identifier)action.getParameters().get(1)).getValue();
        String agent2 =
             ((Identifier)action.getParameters().get(2)).getValue();
        String object =
             ((Identifier)action.getParameters().get(3)).getValue();
        int intensity = ((Numeral)
             action.getParameters().get(4)).getValue().intValue();
        environment.agent.sayEmotion(entity, actorAction, predicate,
            agent1, agent2, object, intensity );

      } else if (action.getParameters().size() == 4) {

        String predicate =
            ((Identifier)action.getParameters().get(0)).getValue();
        String agent =
            ((Identifier)action.getParameters().get(1)).getValue();
        String object =
            ((Identifier)action.getParameters().get(2)).getValue();
        int intensity = ((Numeral)
            action.getParameters().get(3)).getValue().intValue();

        environment.agent.sayEmotion(entity, actorAction, predicate,
            agent, object, intensity );

      } else{
        throw new ActException(ActException.NOTSUPPORTEDBYENVIRONMENT);
      }

    }else if(actorAction != AgentAction.UNKNOWN) {
      environment.performAction(entity, actorAction);
    }else {
      throw new ActException(ActException.NOTSUPPORTEDBYENVIRONMENT);
    }

    return null;
  }

  public Environment getEnvironment() {
    return environment;
  }
}
```

A.3.2 EnvironmentWindow.java

```
package SimpleJim;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.*;
import javax.swing.*;

import SimpleJim.EnvironmentInterface.UserAction;

public class EnvironmentWindow extends JFrame implements
    WindowListener {
  private static final long serialVersionUID = 1L;
  private Environment env;
  private ViewPanel viewPanel;
private JPanel mainFrame, control, agentVar, debugPanel, emotion;
private JLabel energy, money, food, agentState, hunger;
public JLabel expression;
public JTextArea emotionText;
public JLabel debugLabel = new JLabel("debug variable: ");;

//Setup the environment with the initial settings
public EnvironmentWindow(Environment environment) {

  this.env = environment;

  setTitle("SimpleJim Environment");
  setSize(500, 500);
  setResizable(false);
  setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
  addWindowListener(this);

  viewPanel = new ViewPanel();
  mainFrame = new JPanel();
  control = new JPanel();
  agentVar = new JPanel();
  debugPanel = new JPanel();
  emotion = new JPanel();
  energy = new JLabel("Energy: " + env.agent.energy , JLabel.LEADING);
  hunger = new JLabel("Hunger: " + env.agent.hunger , JLabel.LEADING);
  money = new JLabel("Has money: " + env.agent.money ,
       JLabel.LEADING);
  food = new JLabel("Has food: " + env.agent.food , JLabel.LEADING);
  expression = new JLabel(env.agent.disEmotion.toString(),
       JLabel.LEADING);
  agentState = new JLabel("State: " + env.agent.currentState ,
       JLabel.LEADING);

  emotionText = new JTextArea();
  emotionText.setEditable(false);
  JScrollPane sp = new
       JScrollPane(emotionText,ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
       ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
  sp.setPreferredSize(new Dimension(400, 150));
  sp.setBounds(0, 0, 600, 500);
  emotion.add(sp);

  add(mainFrame);
  viewPanel.setPreferredSize(new Dimension(200,200));
  mainFrame.add(viewPanel);
  mainFrame.add(agentVar);
  mainFrame.add(control);
  mainFrame.add(emotion);

  if(env.debug){
    mainFrame.add(debugPanel);
    debugLabel = new JLabel("debug variable: ");
    debugPanel.add(debugLabel);
  }
  showButton();
  agentVar.add(energy);
  agentVar.add(hunger);
  agentVar.add(money);
  agentVar.add(food);
  agentVar.add(agentState);
  viewPanel.add(expression);

  setVisible(true);

}

class ViewPanel extends JPanel {

  @Override
  public void paintComponent(Graphics g_) {
    super.paintComponents(g_);

    Graphics2D g = (Graphics2D) g_;
    Graphics2D gg;

    env.agent.draw(g, this);
    updateAgentVar();

  }
}

public void updateAgentVar(){

  //Updates the text of the agents variables
  energy.setText("Energy: " + env.agent.energy);
  hunger.setText("Hunger: " + env.agent.hunger);
  money.setText("Has money: " + env.agent.money);
  food.setText("Has food: " + env.agent.food);
  agentState.setText("State: " + env.agent.currentState);
  expression.setText(env.agent.disEmotion.toString());
}

public void showButton(){

  // add buttons to the window
  JButton tmb = new JButton("Take Money");
  tmb.setActionCommand("TAKEMONEY");

  JButton doNothing = new JButton("Do Nothing");
  doNothing.setActionCommand("NOTHING");

  JButton gmb = new JButton("Give Money");
  gmb.setActionCommand("GIVEMONEY");

  JButton tfb = new JButton("Take food");
  tfb.setActionCommand("TAKEFOOD");

  JButton gfb = new JButton("Give food");
  gfb.setActionCommand("GIVEFOOD");
```

```
    doNothing.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        env.performUserAction(UserAction.NOTHING);
      }
    });

    tmb.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        env.performUserAction(UserAction.TAKEMONEY);
        updateAgentVar();
      }
    });
    gmb.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        env.performUserAction(UserAction.GIVEMONEY);
        updateAgentVar();
      }
    });
    tfb.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        env.performUserAction(UserAction.TAKEFOOD);
        updateAgentVar();
      }
    });
    gfb.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        env.performUserAction(UserAction.GIVEFOOD);
        updateAgentVar();
      }
    });

    control.add(tmb);
    control.add(gmb);
    control.add(tfb);
    control.add(gfb);
    control.add(doNothing);
    this.setVisible(true);


  }

  @Override
  public void windowActivated(WindowEvent e) {
  }

  @Override
  public void windowClosed(WindowEvent e) {
  }

  @Override
  public void windowClosing(WindowEvent e) {
    env.terminate();
  }

  @Override
  public void windowDeactivated(WindowEvent e) {
  }

  @Override
  public void windowDeiconified(WindowEvent e) {
  }

  @Override
  public void windowIconified(WindowEvent e) {
  }

  @Override
  public void windowOpened(WindowEvent e) {
  }

}
```

A.3.3 Environment.java

```
package SimpleJim;

import SimpleJim.EnvironmentInterface.AgentAction;
import SimpleJim.EnvironmentInterface.UserAction;
import eis.exceptions.ManagementException;

public class Environment {

  private EnvironmentInterface environmentInterface;
  public boolean debug = false;
  private EnvironmentWindow window;
  public Agent agent;
  UserAction lastUserAction = UserAction.NOTHING;
  boolean userPerformedAction = false;
  boolean agentPerformedAction = false;
  boolean agentWitnessedAction = false;
  AgentAction pendingUserAction;
  AgentAction pendingAgentAction;
  private long time = 0;

  public Environment(EnvironmentInterface environmentInterface, boolean
      money, boolean food, int energy, int hunger, boolean debug){

    this.environmentInterface = environmentInterface;
    this.debug = debug;

    String entity = "agent";
    agent = new Agent(this, money, food, energy, hunger);

    environmentInterface.addAgentEntity(entity);
    window = new EnvironmentWindow(this);

    resetState();
  }
```

```
public EnvironmentWindow getEnvWin(){
  return window;
}

private void resetState() {
  window.repaint();
}

public long getTime() {
  return time;
}

public void performUserAction(UserAction action) {

  // resets the text are for emotions
  getEnvWin().emotionText.setText("");

  //records the last user action
  lastUserAction = action;

  // Checks if the agent witnessed an user action or not
  switch(agent.currentState){
  case SHOPPING:
  case SLEEPING:
  case STEALINGFOOD:
  case STEALINGMONEY:
  case WORKING:
    switch(action){
    case GIVEFOOD:
    case GIVEMONEY:
      agentWitnessedAction = false;
      break;
    case TAKEFOOD:
      if(agent.food)
        agentWitnessedAction = false;
      break;
    case TAKEMONEY:
      if(agent.money)
        agentWitnessedAction = false;
    default:
      break;
    }
    break;
  default:
    switch(action){
    case GIVEFOOD:
    case GIVEMONEY:
      agentWitnessedAction = true;
      break;
    case TAKEFOOD:
      if(agent.food)
        agentWitnessedAction = true;
      break;
    case TAKEMONEY:
      if(agent.money)
        agentWitnessedAction = true;
    default:
      break;
    }
    break;
  }

  boolean noFood = false;

  // apply changes to the environment
  switch(action){
  case GIVEFOOD:
    agent.food = true;
    break;
  case GIVEMONEY:
    agent.money = true;
    break;
  case TAKEFOOD:
    if( agent.food && agent.currentState != Agent.State.EATING)
      agent.food = false;
    break;
  case TAKEMONEY:
    if(agent.money && agent.currentState != Agent.State.SHOPPING){
      agent.money = false;
    } else {
      noFood = true;
    }
    break;
  default:
    break;
  }

  // perform agents action
  if(agentPerformedAction) {
    agentUpdateAction(pendingAgentAction);
  }

  if (noFood){
    agent.food = false;
  }
}

// manage the agents action
public void performAction(String entity, AgentAction action) {

  pendingAgentAction = action;
  agentPerformedAction = true;

  if(agentPerformedAction && !userPerformedAction){
    return;
  } else {
    agentUpdateAction(action);
  }
```

```
}

public void agentUpdateAction(AgentAction action){

  // apply agent action cost to the agent
  if(action.equals(AgentAction.RETURN)){
    switch(agent.currentState){
    case EATING:
      if(agent.food){
        agent.food = false;
        agent.hunger = 0;
        agent.energy -= agent.energy == 0 ? 0 : 1;
      }
      break;
    case SHOPPING:
      if(agent.money){
        agent.food = true;
        agent.money = false;
        agent.hunger += agent.hunger == 0 ? 0 : 1;
        agent.energy -= agent.energy == 0 ? 0 : 1;
      }
      break;
    case SLEEPING:
      agent.energy = 5;
      agent.hunger += agent.hunger >= 4 ? (agent.hunger == 4 ? 1 : 0 )
           : 2;
      break;
    case STEALINGFOOD:
      agent.food = true;
      break;
    case STEALINGMONEY:
      agent.money = true;
      break;
    case WORKING:
      agent.money = true;
      agent.hunger += agent.hunger >= 4 ? (agent.hunger == 4 ? 1 : 0 )
           : 2;
      agent.energy -= agent.energy <= 1 ? (agent.energy == 1 ? 1 : 0 )
           : 2;
      break;
    case UNKNOWN:
      break;
    default:
      break;

    }
  }

  // update the state of the agent
  agent.processAction(action);
  userPerformedAction = false;
  agentPerformedAction = false;

  // increment time
  time++;
```

```
    window.repaint();
  }

  public void terminate() {

    environmentInterface.deleteAgentEntity("agent");

    window.dispose();
    window.setVisible(false);
    window = null;
    if (environmentInterface != null) {
      try {
        environmentInterface.kill();
      } catch (ManagementException e) {
        e.printStackTrace();
      }
    }

  }


}
```

A.3.4 Agent.java

```
package SimpleJim;

import java.awt.Color;
import java.awt.Graphics2D;
import javax.swing.JTextArea;
import SimpleJim.EnvironmentInterface.AgentAction;
import SimpleJim.EnvironmentWindow.ViewPanel;

public class Agent {

  public Environment env;
  boolean food = false;
  boolean money = false;
  int energy = 5;
  int hunger = 0;
  State currentState = State.WAITING;
  displayingEmotion disEmotion = displayingEmotion.NOTHING;

  public Agent(Environment environment, boolean money2, boolean food2,
       int energy2, int hunger2){
    this.env = environment;
    this.money = money2;
    this.energy = energy2;
    this.food = food2;
    this.hunger = hunger2;
  }
```

```
enum displayingEmotion {
  HAPPY, SAD, FEAR, ANGRY, DISGUST, NOTHING, UNKNOWN;

  static displayingEmotion toEnum(String string) {
    try {
      return valueOf(string.replaceAll("([A-Z])",
           "_$1").toUpperCase());
    } catch (Exception e) {
      return UNKNOWN;
    }
  }

  static String toString(AgentAction action) {
    String[] strings = action.toString().split("_");
    StringBuilder sb = new StringBuilder(strings[0].toLowerCase());
    for (int i = 1; i < strings.length; i++) {
      sb.append(strings[i].substring(0, 1));
      sb.append(strings[i].substring(1).toLowerCase());
    }
    return sb.toString();
  }
}

enum State {
  HOME, EATING, SLEEPING, WORKING, SHOPPING, STEALINGFOOD,
      STEALINGMONEY, WAITING, UNKNOWN;

  static State toEnum(String string) {
    try {
      return valueOf(string.replaceAll("([A-Z])",
           "_$1").toUpperCase());
    } catch (Exception e) {
      return UNKNOWN;
    }
  }

  static String toString(AgentAction action) {
    String[] strings = action.toString().split("_");
    StringBuilder sb = new StringBuilder(strings[0].toLowerCase());
    for (int i = 1; i < strings.length; i++) {
      sb.append(strings[i].substring(0, 1));
      sb.append(strings[i].substring(1).toLowerCase());
    }
    return sb.toString();
  }
}

public void processAction(AgentAction action) {

  // change the agent state
  if(currentState.equals(State.WAITING)){
    switch(action){
    case BUYFOOD:
      currentState = State.SHOPPING;
      break;
    case STEALMONEY:
      currentState = State.STEALINGMONEY;
      break;
    case STEALFOOD:
      currentState = State.STEALINGFOOD;
      break;
    case EAT:
      currentState = State.EATING;
      break;
    case SLEEP:
      currentState = State.SLEEPING;
      break;
    case WORK:
      currentState = State.WORKING;
      break;
    case UNKNOWN:
      break;
    default:
      break;
    }
  }
  else if(action.equals(AgentAction.RETURN)) {
    currentState = State.HOME;
  }
  else if(action.equals(AgentAction.WAIT)) {
    currentState = State.WAITING;
  }
}

public void draw(Graphics2D create, ViewPanel viewPanel) {
  // draw expression in the environment
  switch(disEmotion){
  case ANGRY:
    create.setColor(Color.RED);
    create.fillRect(0, 0, viewPanel.getWidth(), viewPanel.getHeight());
    break;
  case DISGUST:
    create.setColor(Color.GREEN);
    create.fillRect(0, 0, viewPanel.getWidth(), viewPanel.getHeight());
    break;
  case FEAR:
    create.setColor(Color.BLACK);
    create.fillRect(0, 0, viewPanel.getWidth(), viewPanel.getHeight());
    break;
  case HAPPY:
    create.setColor(Color.YELLOW);
    create.fillRect(0, 0, viewPanel.getWidth(), viewPanel.getHeight());
    break;
  case NOTHING:
    create.setColor(Color.WHITE);
    create.fillRect(0, 0, viewPanel.getWidth(), viewPanel.getHeight());
    break;
  case SAD:
    create.setColor(Color.BLUE);
    create.fillRect(0, 0, viewPanel.getWidth(), viewPanel.getHeight());
```

```
      break;
    case UNKNOWN:
      break;
    default:
      break;
    }
  }

  //set displaying emotion text
  public void displayEmotion(String entity, AgentAction actorAction,
       String emotion) {

    env.getEnvWin().expression.setForeground(Color.BLACK);
    switch(emotion){
    case "happy":
      disEmotion = displayingEmotion.HAPPY;
      break;
    case "anger":
      disEmotion = displayingEmotion.ANGRY;
      break;
    case "sadness":
      disEmotion = displayingEmotion.SAD;
      break;
    case "fear":
      env.getEnvWin().expression.setForeground(Color.WHITE);
      disEmotion = displayingEmotion.FEAR;
      break;
    case "disgust":
      disEmotion = displayingEmotion.DISGUST;
      break;
    default:
      disEmotion = displayingEmotion.NOTHING;
      break;
    }

    env.getEnvWin().repaint();
  }

  //Translate the received emotion into natural language with multiple
       agents
  public void sayEmotion(String entity, AgentAction actorAction,
      String predicate, String agent1, String agent2, String object,
      int intensity) {
    JTextArea ta = env.getEnvWin().emotionText;
    String newText = ta.getText() + agent1+" is feeling " + predicate +
        " towards " + agent2 + " about " + object + " with intensity "
        + intensity + "\n";
    ta.setText(newText);
  }

  //Translate the received emotion in to natural language with one agent
  public void sayEmotion(String entity, AgentAction actorAction,
      String predicate, String agent, String object, int intensity) {
    JTextArea ta = env.getEnvWin().emotionText;
    String newText;
    if(predicate.equals("like") || predicate.equals("dislike") )
     newText = ta.getText() + agent +" " + predicate +"s " + object + "
          with intensity " + intensity+ "\n";
    else
     newText = ta.getText() + agent +" is feeling " + predicate + "
          about " + object + " with intensity " + intensity+ "\n";
    ta.setText(newText);

  }

}
```

Bibliography

[AGHL06] Carole Adam, Benoit Gaudou, Andreas Herzig, and Dominique Longin. OCC's Emotions: A Formalization in a BDI Logic. In Jérôme Euzenat and John Domingue, editors, *Artificial Intelligence: Methodology, Systems, and Applications*, volume 4183 of *Lecture Notes in Computer Science*, pages 24–32. Springer Berlin Heidelberg, 2006.

[AHL09] Carole Adam, Andreas Herzig, and Dominique Longin. A logical formalization of the OCC theory of emotions. *Synthese*, 168(2):201–248, 2009.

[Bar02] Christoph Bartneck. Integrating the OCC model of Emotions in Embodied Characters. 2002.

[BHD] Tristan M Behrens, Koen V Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms.

[BHD11] TristanM. Behrens, KoenV. Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61(4):261–295, 2011.

[Cho] Anurag Choudhari. Exploring the tech behind the games industry. `http://www.geekycube.com/exploring-the-tech-behind-the-games-industry/1329`. Accessed June 26, 2014.

[DP05] João Dias and Ana Paiva. Feeling and reasoning: A computational model for emotional characters. In *Progress in artificial intelligence*, pages 127–140. Springer, 2005.

[Dyc] Max Dyckhoff. Naughty Dog - Ellie - Buddy AI in the Last of Us. `http://www.gdcvault.com/play/1020364/Ellie-Buddy-AI-in-The`. Accessed June 26, 2014.

[EO79] Paul Ekman and Harriet Oster. Facial expressions of emotion. *Annual review of psychology*, 30(1):527–554, 1979.

[GLL+11] Nadine Guiraud, Dominique Longin, Emiliano Lorini, Sylvie Pesty, and Jérémy Rivière. The Face of Emotions: A Logical Formalization of Expressive Speech Acts. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '11, pages 1031–1038, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

[GM04] Jonathan Gratch and Stacy Marsella. A domain-independent framework for modeling emotion. *Cognitive Systems Research*, 5(4):269–306, 2004.

[Hin14] Koen V. Hindriks. Programming Cognitive Agents in GOAL, March 2014.

[Lyn] David Lynch. Next-Gen: Complex AI Will Lead To 'Realistic Behaviour' & 'Simulated Emotions'. `http://www.nowgamer.com/news/1835868/nextgen_complex_ai_will_lead_to_realistic_behaviour_simulated_emotions.html`. Accessed June 26, 2014.

[MC04] Luis Macedo and Amilcar Cardoso. Exploration of unknown environments with motivational agents. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, AAMAS '04, pages 328–335. IEEE Computer Society, IEEE Computer Society, July 2004.

[ME12] Christos N Moridis and Anastasios A Economides. Affective Agents in E-Learning Platforms. pages 527–554, 2012.

[Mey06] John-Jules Ch Meyer. Reasoning about emotional agents. *International journal of intelligent systems*, 21(6):601–619, 2006.

[OCC88] Andrew. Ortony, Gearald L. Clore, and Allan Collins. *The cognitive structure of emotions.* first edition, 1988.

[Onl] AllPsych Online. Physchologi 101 Chapter 7: Motivation and Emotion. `http://allpsych.com/psychology101/emotion.html`. Accessed June 25, 2014.

[RHD$^+$] Rainer Reisenzein, Eva Hudlicka, Mehdi Dastani, Jonathan Gratch, Koen Hindriks, Emiliano Lorini, and J Meyer. Computational Modeling of Emotion: Towards Improving the Inter-and Intradisciplinary Exchange. *IEEE TRANSACTIONS ON AFFECTIVE COMPUTING.*

[RNC$^+$10] Stuart J. Russell, Peter Norvig, John F. Candy, Jitendra M. Malik, and Douglas D. Edwards. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Inc., third edition, 2010.

[SD02] Alexander Serenko and Brian Detlor. Agent toolkits: A general overview of the market and an assessment of instructor satisfaction with utilizing toolkits in the classroom, 2002 2002.

[SDM07] Bas R Steunebrink, Mehdi Dastani, and John-Jules Ch Meyer. A logic of emotions for intelligent agents. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, pages 142–147. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[SDM09] Bas R Steunebrink, Mehdi Dastani, and John-Jules Ch Meyer. The OCC model revisited. In *the 4th Workshop on Emotion and Computing*, 2009.

[SK09] Craig A Smith and Leslie D Kirby. Putting appraisal in context: Toward a relational model of appraisal and emotion. *Cognition and Emotion*, 23(7):1352–1372, 2009.

[Spu13] Johannes S Spurkeland. Interaction in Organization-Oriented Multi-Agent Systems. Master's thesis, Denmark Technical University, 2013.

[Vel97] Juan D. Velásquez. Modeling emotions and other motivations in synthetic agents. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, AAAI'97/IAAI'97, pages 10–15. AAAI Press, 1997.

[WJ95] Michael Wooldridge and Nicholas R. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2):115–152, Octoober 1995.