Description and prescription of collaboration in multi agent-systems using activities in contexts

Master Thesis in Computer Systems Engineering

The Maersk Mc-Kinney Moller Institute
for Production Technology
University of Southern Denmark

February 1, 2006

Author:
Steffen Jensen

Supervisor:
Palle Nowack

Abstract

One of today's fast-growing domains in software engineering is multi-agent systems(MAS). One of the forces in MAS is the decentralised approach of achieving common, often complex, goals utilising simple agents. At the same time, this is also a drawback because a lot of negotiation is required when agents engage in collaboration to achieve their common (complex)goals.

In this thesis the concepts; agent, activity and context, are investigated. And, it is investigated how to achieve collaboration between multiple agents by using activities in contexts, while preserving a high level of agent autonomy. An experimental platform, a soccer simulator, labelled SoccerLab has been developed, to be able to experiment with agents and especially the concepts activity and context and how they are applied during analysis, design and implementation. The experiments revealed that activities are flexible and powerful for describing and prescribing collaboration, both at a conceptual level during analysis and design and at a programming level during implementation, between agents in contexts, while preserving agent autonomy.

Contents

Abstract i

Preface xiii

Organisation of this report xv

1 Summary of thesis 1
- 1.1 Motivation . . . 1
- 1.2 Problem analysis . . . 1
  - 1.2.1 Supporting agents through contexts . . . 2
  - 1.2.2 Collaboration between agents using activities . . . 2
  - 1.2.3 Problem formulation . . . 3
- 1.3 Project constraints . . . 3
- 1.4 Work process . . . 4
  - 1.4.1 The problem solving process . . . 5
- 1.5 Results . . . 6
  - 1.5.1 SoccerLab prototype version 1 . . . 6
  - 1.5.2 The application survey . . . 6
  - 1.5.3 Characterisation of activities and contexts . . . 7
  - 1.5.4 SoccerLab prototype version 2 . . . 8
- 1.6 Hypotheses . . . 9
  - 1.6.1 Activities . . . 10
  - 1.6.2 Contexts . . . 12
  - 1.6.3 Visualisation . . . 14
- 1.7 Discussion . . . 15
- 1.8 Reflections and future work . . . 16
  - 1.8.1 Reflections on collaboration . . . 16
  - 1.8.2 Suggestions for future work . . . 18

2 Prototype version 1 21
- 2.1 Introduction . . . 21
- 2.2 Application description . . . 22
- 2.3 Architecture . . . 23
  - 2.3.1 The model component . . . 23
  - 2.3.2 The GUI component . . . 24
- 2.4 Dynamics . . . 25
  - 2.4.1 Initialisation of the soccer simulator . . . 25
  - 2.4.2 Setup of team formation . . . 25

2.4.3 The activity selection process . . . 26
2.5 Classes . . . 27
2.5.1 The model component . . . 27
2.5.2 The GUI component . . . 30
2.6 Test and evaluation . . . 31
2.6.1 Test . . . 31
2.6.2 Evaluation . . . 32
2.7 Summary . . . 33

3 Application survey **35**
3.1 The Tao of Soccer . . . 35
3.1.1 Collaboration strategies . . . 36
3.1.2 Platform features . . . 36
3.1.3 Special features . . . 38
3.2 FIFA Football 2005 . . . 38
3.2.1 Collaboration strategies . . . 38
3.2.2 Special features . . . 46
3.3 Summary . . . 46

4 Analysis of concepts **49**
4.1 Introduction . . . 49
4.1.1 Motivation . . . 50
4.2 Conceptual modelling . . . 50
4.2.1 Physical modelling . . . 50
4.3 The Agent Concept . . . 52
4.3.1 Description of the agent concept . . . 52
4.3.2 The role concept . . . 55
4.4 The activity concept . . . 56
4.4.1 Characterisation of the activity concept . . . 56
4.5 The context concept . . . 58
4.5.1 Characterisation of the context concept . . . 58
4.6 Organisation of agents . . . 60
4.6.1 Approaches for organising agents . . . 60
4.7 Summary . . . 61

5 Prototype version 2 **63**
5.1 Motivation . . . 63
5.2 Application description . . . 63
5.3 Architecture . . . 65
5.3.1 The Model component . . . 65
5.3.2 The GUI component . . . 71
5.4 Dynamics . . . 76
5.5 Evaluation . . . 76
5.5.1 Agent support . . . 77
5.5.2 Activity support . . . 78
5.5.3 Context support . . . 78
5.5.4 The graphical user interface . . . 78
5.6 Summary . . . 79

Bibliography **80**

A Preliminary application survey **85**
A.1 Application descriptions . . . 85
A.1.1 The blue team . . . 85
A.1.2 The green team . . . 86
A.1.3 The pink team . . . 88
A.1.4 The red team . . . 89
A.2 Comparisons . . . 90
A.3 Summary . . . 93

B Literature studies **95**
B.1 Object-Oriented Programming in BETA . . . 96
B.1.1 Brief summary . . . 96
B.2 TangO: Modeling In Style . . . 100
B.2.1 Brief summary . . . 101
B.3 Architecture = Abstractions over Software . . . 103
B.3.1 Brief summary . . . 103
B.4 Transverse Activities . . . 104
B.4.1 Brief summary . . . 105
B.5 Activities: Abstractions for Collective Behavior . . . 107
B.5.1 Brief summary . . . 107
B.6 Complex Associations: Abstractions in Object-Oriented Modeling 108
B.6.1 Brief summary . . . 108
B.7 Object-Oriented Modeling with Roles . . . 110
B.7.1 Brief summary . . . 110
B.8 Roles: Conceptual Abstraction Theory & Practical Language Issues 112
B.8.1 Brief summary . . . 112
B.9 Subjective behavior . . . 114
B.9.1 Brief summary . . . 114
B.10 Associations: Abstractions over Collaboration . . . 115
B.10.1 Brief summary . . . 116
B.11 Multi-criteria optimization of ball passing in simulated soccer . . 117
B.11.1 Brief summary . . . 117

C Conceptual modelling of activities and contexts **121**
C.1 Conceptual modelling using the activity concept . . . 121
C.1.1 Classification/exemplification . . . 121
C.1.2 Specialisation/generalisation . . . 122
C.1.3 Aggregation/decomposition . . . 123
C.2 Conceptual modelling using the context concept . . . 124
C.2.1 Classification/exemplification . . . 124
C.2.2 Specialisation/generalisation . . . 125
C.2.3 Aggregation/decomposition . . . 125

D FIFA 2005: special features and design **127**
D.1 Special features in FIFA 2005 . . . 127
D.1.1 The squad screen . . . 127
D.1.2 The player biography screen . . . 127
D.2 Design of observed collaboration strategies . . . 129
D.2.1 Introduction of agent vision . . . 129
D.2.2 Pass ball backwards . . . 129

D.2.3 The triangular pattern . . . 130
D.2.4 Through-pass . . . 130
D.2.5 Cross the ball to score . . . 131
D.2.6 Defencive strategies . . . 131

E Statek Report **133**
E.1 Introduction . . . 136
E.2 Motivation . . . 136
E.3 Statek - Ship simulator . . . 136
E.3.1 Pilot on board activity . . . 137
E.3.2 Agents . . . 138
E.3.3 Roles . . . 139
E.3.4 Habitats . . . 140
E.3.5 Activities . . . 142
E.3.6 Summary . . . 143
E.4 The soccer simulator . . . 143
E.4.1 Defencive positioning activity . . . 143
E.4.2 Agents . . . 144
E.4.3 Roles . . . 145
E.4.4 Habitats . . . 146
E.4.5 Activities . . . 148
E.4.6 Summary . . . 149
E.5 Candidates for a general framework . . . 150
E.5.1 Agents . . . 150
E.5.2 Roles . . . 151
E.5.3 Activity . . . 152
E.5.4 Habitat . . . 152
E.5.5 Summary . . . 153

F CD contents **155**

List of Figures

1.1 Box-diagram presenting the connection between the five stages. . 4
1.2 The problem solving process. 6
1.3 Screen shot of SoccerLab prototype version 1 7
1.4 Screen shot of SoccerLab prototype version 2. 9
1.5 The context interface inheritance structure. 9
1.6 Screen shot of the soccer field from SoccerLab prototype version 2, with visualisation of the soccer field context. 15
1.7 Screen shot of the activity info panel from SoccerLab prototype version 2 with visualisation of activities. 16
1.8 The five coordinating mechanisms presented in [Mintzberg, 1983]. 17
1.9 Examples of visualisation of the activity concept. 19

2.1 The prototype version 1 chapter in a larger perspective. 21
2.2 Screen shot of soccer simulation application. 22
2.3 Screen shot of the habitat chooser panel. 23
2.4 Class diagram of the soccer simulation's model component. . . . 24
2.5 Class diagram of the GUI component. 25
2.6 Sequence diagram of the initialisation of the soccer simulator. . . 26
2.7 Sequence diagram showing the interaction between a soccer team and it's formation. 27
2.8 A screen shot presenting the visualisation of habitats 31

3.1 The application survey chapter in a larger perspective. 35
3.2 A screen shot of The Tao of Soccer. 36
3.3 Client-server architecture of The Tao of Soccer 37
3.4 Screen shot of FIFA Football 2005 application. 39
3.5 A screen shot of FIFA Football 2005 tactics screen. 40
3.6 Outlines of the two identified possessive collaboration strategies concerning passing the ball. 43
3.7 Outlines of the two identified possessive collaboration strategies concerning scoring. 44
3.8 Outlines of the two identified non-possessive collaboration strategies concerning defending. 45

4.1 The analysis of concepts chapter in a larger perspective. 49
4.2 Referent system and model system. 51
4.3 General abstraction model. 52
4.4 Beliefs Desires Intentions model. 54

4.5 The AACR-model inspired by the AALAADIN-model. 61

5.1 The prototype version 2 chapter in a larger perspective. 63
5.2 Screen shot of SoccerLab prototype version 2. 64
5.3 General class diagram of SoccerLab prototype version 2 model component. 66
5.4 The segment of the class diagram of the model component of SoccerLab prototype version 2 concerning agents. 67
5.5 The segment of the class diagram of the model component of SoccerLab prototype version 2 concerning activities. 68
5.6 The segment of the class diagram of the model component of SoccerLab prototype version 2 concerning physical contexts. . . . 70
5.7 The segment of the class diagram of the model component of SoccerLab prototype version 2 concerning logical contexts. 71
5.8 Class diagram of SoccerLab prototype version 2 GUI component. 72
5.9 Sequence diagram exemplifying of usage of the context options panel in SoccerLab prototype version 2. 73
5.10 Screen shots of the activity panel. 74
5.11 Sequence diagram exemplifying of usage of the the context options panel in SoccerLab prototype version 2. 75

A.1 A screen shot of the blue team's soccer simulation. 86
A.2 A screen shot of the blue team's soccer simulation, when the debug information toggled on. 87
A.3 The blue team's class diagram. 87
A.4 A screen shot of the green team's soccer simulation. 88
A.5 The green team's class diagram of the soccer domain model. . . . 88
A.6 A screen shot of the pink team's soccer simulation. 89
A.7 The pink team's class diagram. 89
A.8 A screen shot of the red team's soccer simulation. 90
A.9 The red team's class diagram. 91

B.1 Screen shot of reachable pass-receiving points 118

C.1 General abstraction model. 121
C.2 Classification and exemplification. 122
C.3 Specialisation and generalisation. 123
C.4 Aggregation and decomposition. 124

D.1 A screen shot of FIFA Football 2005 squad screen. 128
D.2 A screen shot of FIFA Football 2005 player bio screen. 129

E.1 The ship simulator class diagram. 137
E.2 The part of the class diagram concerning agents. 138
E.3 The part of the class diagram concerning roles. 139
E.4 Overlapping habitats visualised. 140
E.5 The part of the class diagram concerning habitats. 141
E.6 The part of the class diagram concerning activities. 142
E.7 The soccer simulator class diagram. 144
E.8 The part of the class diagram concerning agents. 144
E.9 The part of the class diagram concerning roles. 145

E.10 The part of the diagram concerning habitats. 146
E.11 The part of the class diagram concerning high-level inheritance in habitats. 147
E.12 The part of the class diagram concerning logical habitats. 147
E.13 The part of the class diagram concerning physical habitats. . . . 148
E.14 The part of the class diagram concerning activities. 149
E.15 The general layout of an agent. 150
E.16 The general layout of a role. 151
E.17 The general layout for habitats. 152

List of Tables

1.1 Example of an activity sub-class implementation. 12

5.1 Example of an activity sub-class implementation. 68
5.2 The general context interface. 69
5.3 The physical context interface. 70
5.4 The logical context interface. 70

A.1 Overview of the four applications 92

B.1 Evaluation of: Object-Oriented Programming in BETA 96
B.2 Evaluation of: TangO: Modeling In Style 101
B.3 Evaluation of: Architecture = Abstractions over Software 103
B.4 Evaluation of: Transverse Activities 105
B.5 Evaluation of: Activities: Abstractions for Collective Behavior. . 107
B.6 Evaluation of: Complex Associations 108
B.7 Evaluation of: Object-Oriented Modeling with Roles 110
B.8 Evaluation of: Roles: Conceptual Abstraction 112
B.9 Evaluation of: Subjective behavior 114
B.10 Evaluation of: Associations: Abstractions over Collaboration . . . 116

Preface

This report is the result of Steffen Jensen's Master Thesis[1] and represents the final step before obtaining the Master's Degree in computer systems engineering[2]. The work in this thesis has been carried out on an individual basis, except for the literature study of agents, activities and contexts, documented in chapter 4, which has been co-written with Beata Hargesheimer[3]. Beata Hargesheimer has also written some of the summaries of the studied literature, presented in appendix B. It should be noticed that a CD containing the two developed prototypes has been included in appendix F.

This thesis has been written under supervision of Palle Nowack, Associate Professor, Ph.D. and it was carried out in the period February 2005 to February 2006.

The Maersk Mc-Kinney Moller Institute for Production Technology,
University of Southern Denmark, Odense.
1^{st} of February 2006.

[1] Danish: speciale
[2] Danish: cand.polyt
[3] Master student at The Maersk Mc-Kinney Moller Institute for Production Technology, University of Southern Denmark, Odense.

Organisation of this report

This page gives a brief overview of the purpose and the content of each of the chapters in this report.

Chapter 1 gives an overview of, and serves as the conclusion of this thesis. It does so by presenting and describing; the problems set-up in this thesis, the results and a discussion of these.

Chapter 2 describes the developed experimental platform for agents, activities and context, labelled SoccerLab prototype version 1.

Chapter 3 presents an application survey encompassing the two applications: The Tao of Soccer(MAS) and FIFA Football 2005(soccer game), that provided inspiration for both platform features and collaboration strategies for future version of the SoccerLab application.

Chapter 4 describes the analysis of agents, activities and contexts, based on literature studies and characterises concepts activity and context used in SoccerLab prototype version 2.

Chapter 5 documents how the concepts; agent, activity and context are supported and implemented in SoccerLab prototype version 2 and how they are visualised.

Chapter 1

Summary of thesis

This chapter presents the motivation to carry out this thesis, a problem analysis, with the resulting problem formulation, and the results of this thesis. Finally, the results are discussed and suggestions for future work are presented. This chapter serves as the conclusion of this thesis and because of that no section explicitly denoted conclusion will be found in this thesis.

1.1 Motivation

One of today's fast-growing domains in software engineering is multi-agent systems[1] (MAS), but still no unambiguous definition of what an agent is have been agreed-on. A widely accepted notion of agency, was proposed by Michael Wooldridge and Nicholas R. Jennings in [Wooldridge and Jennings, 1995]. According to Wooldridge and Jennings the term *agent* denotes a hardware or software-based computer system enjoying properties such as; autonomy, social abilities, reactivity and pro-activeness. By this notion of agents it is understood that agents operate without intervention of humans and they move around perceiving their environment while exhibiting goal-directed behaviour and collaborating with other agents.

A multi-agent system is a system composed of several agents, capable of reaching goals that are difficult to achieve by an individual system [Zambonelli and Parunak, 2002]. One of the major forces in MAS is the decentralised approach of achieving common goals of the agents. At the same time this is also a major drawback because a lot of negotiation is required when agents engage in collaboration to achieve their common goals. The motivation for this thesis is to examine how to achieve(describe and prescribe) collaboration between agents avoiding negotiation by using activities in contexts.

1.2 Problem analysis

The objective of this thesis is, to investigate a new paradigm and approach for software development, in which it is considered to model and represent both problems and solutions in terms of agents and objects engaged in activities in

[1] MAS is an abbreviation for multi-agent system(s)

contexts. The following sections presents an analysis of the problems raised by the motivation. Section 1.2.3 summarises this analysis, in form of the problem formulation.

1.2.1 Supporting agents through contexts

An example on how inhabitants can be supported by the environment or context in which they live, can be found in the TangO paradigm [May et al., 2001]. In the Tango paradigm, a paradigm for modelling digitally pervasive environments, tangible objects, physical objects with computational power, live in habitats, which are logical contexts. The interplay between tangible objects and between tangible objects and a habitat is achieved by using associations.

Habitats as abstractions over environment

The concept of habitat as an abstraction over environment has evolved since it was introduced in [May et al., 2001]. For instance [Andersen and Nowack, 2004] describe habitats as a container where inhabitants can move into and live for a period of time and then move out again. To be more specific the topic in [Andersen and Nowack, 2004] is to model moving machines(PDAs[2] and laptops), and how they can be influenced by the context in which they are present. For instance, at a train station the platforms (modelled as habitats) automatically provide information such as arrival, departures and delay of the train to which a person has a ticket, through his PDA. When the person enters a train(another habitat) he will be guided to his seat by his PDA, which also present other useful information, provided by the train habitat, like the estimated travel time and where the train stops.

Similar to the way habitats influence PDAs and laptops, agents could be influenced by the context[3] in which they are present and hence be encouraged to participate in collaboration.

Not only the habitat concept has evolved, also the concept of association as an abstraction over interplay or collaboration between artifacts has evolved as well. This is described in the following section.

1.2.2 Collaboration between agents using activities

The association concept used in the TangO paradigm [May et al., 2001] originates from the activity concept introduced as transverse activities by Bent Bruun Kristensen in [Kristensen, 1993a]. Bent Bruun Kristensen's activity concept is an abstraction over collaboration in object-oriented programming.

Activities as abstractions over collaboration

The activity concept, as presented in [Kristensen, 1993a], is founded on the conceptual framework of BETA [Madsen et al., 1993], where programming is considered as physical modelling. Because the activity concept is founded on

[2] PDA is an abbreviation for personal digital assistant.

[3] During the investigation of the habitat concept it was decided to rename it to context, this decision is further elaborated in chapter 4

the conceptual framework of BETA a clear distinction is made between class representation and object representation of the activity concept.

During the years not only has the concept of activity[4] been renamed to association, but it has also been refined in several iterations. [Kristensen, 2005] presents the latest refinement of the association concept, which is the sum of all the refinements made to the concept since it was introduced as transverse activities. In its latest version the association concept is an abstraction over collaboration where the collaborating participants are only accessed through roles. When all the participants are gathered in the association it executes its directive which prescribes the collaboration. The association concept provides a means to describe and prescribe collaboration between multiple objects in a centralised manner.

1.2.3 Problem formulation

The objective of this thesis is to investigate a new paradigm and approach for software development, in which it is considered to model and represent both problems and solutions in terms of agents and objects engaged in activities in contexts. The main focus is put on how to achieve collaboration between multiple agents using activities in contexts, and how the collaboration can be described and prescribed. As an attempt to make the abstract concepts of activity and context more concrete, to increase understandability, it should be considered how the two concepts can be visualised.
This can be decomposed into the following three problems:

- How can the concept of activity be characterised, and how can it be applied during analysis, design and implementation?
- How can the concept of context be characterised, and how can it be applied during analysis, design and implementation?
- Can visualisation increase the understandability of concepts of activity and context?

1.3 Project constraints

The primary interest is to investigate the concepts of activities and contexts to be able to characterise the two concepts and use them in experiments of modelling collaboration between multiple agents. Secondary, the MAS paradigm is investigated to gain knowledge of the conflicts that arises when centralised control is introduced into multi-agent systems.

To be able to experiment with the concepts of agent, activity and context an experimental platform has to be developed. Due to the autonomous nature of agents, acting without intervention from humans or others, a simulator is chosen. The requirements for the scenario that is to be simulated are:

- It has to have natural support of multiple entities (agents).
- The environment has to have a high level of dynamism.

[4] This concept of activity is not the same concept as the activity concept presented in this thesis.

- The scenario has to have various examples of collaboration.

A soccer[5] scenario has been chosen because of its variety of possibilities, and primary it fulfils the listed requirements.

The practical aspect of the project then involves development of a simple soccer simulation platform for experimenting with the concepts agents, activities and contexts, and how these can be represented or visualised, enabling the soccer simulation platform to be used as a demonstrator application of the concepts; agent, activity and context(AAC). As an initial approach the soccer players will be modelled using a simple agent architecture, to ensure the focus is kept on investigating the concepts of activities and contexts.

1.4 Work process

This section describes the working plan according to which the problems presented in section 1.2.3 has been resolved, with respect to the project constraints described in section 1.3.

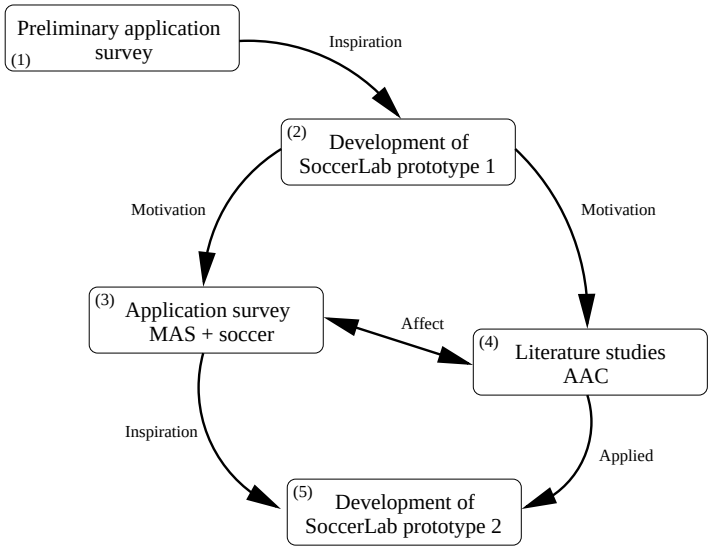


Figure 1.1: Box-diagram presenting the connection between the five stages.

Figure 1.1 depicts a box-diagram presenting how the five stages of the work plan, described in the following list, are connected.

1. Preliminary application survey of four simple soccer simulators, described in appendix A, to gather inspiration for developing the first prototype of SoccerLab(soccer simulator).

2. Development of SoccerLab prototype 1. The intention of this stage is to develop an AAC experimental platform and to get some hands-on experience with soccer simulation. This first prototype includes the first attempt

[5] Through out the report the term soccer is used for the European kind of football not to be confused with American football.

on modelling activities and contexts, that later on will be refined based on the literature studies in stage four. SoccerLab prototype version 1 is described in chapter 2.

3. Conduct an application survey of a MAS application and a soccer game to gain inspiration for both activities and platform features for the SoccerLab application. This application survey of a MAS and a soccer game is documented in chapter 3.

4. Study of literature concerning agents, activities and contexts, to be able to characterise the concepts; activity and context, and to gain more knowledge about MAS. This analysis of concepts is presented in chapter 4.

5. Development of SoccerLab prototype 2 based on the literature studies from stage four, and the application survey carried out in stage three. SoccerLab prototype version 2 is described in chapter 5.

As shown in figure 1.1 the work process has been rather sequential. However, stage three and stage four were overlapping in time, which is the reason why they have affected each other. For each of the listed stages various relevant literature have been studied. The majority of the studied literature in stage four have been categorised, summarised, and rated according to the relevance for this thesis, and the summaries are presented in appendix B.

For both software development activities in stage two and stage five, the prototyping approach [Floyd, 1984] have been used. A benefit of using the prototyping approach is that it allows for dummy implementations of complex classes which are not in focus for a given release [Barnes and Kölling, 2004]. To keep track of the different releases the project has been developed using version control.

In addition to this thesis, a report for a FUSS[6] research project, concerning design of simulations of both soccer and a maritime environments using agent, activities and contexts, has been written. This report is included in appendix E.

1.4.1 The problem solving process

Section 1.2.3 presented the general problems in this thesis. Instead of attempting to resolve the problems at a general level, a concrete example in form of a soccer simulator has been chosen, allowing to experiment with both problems and solutions at a specific level. When the problems have been resolved for the concrete example, the general parts of these solutions will also be the solutions for the general problems. Figure 1.2 presents a schematic view of the problem solving process.
The following relates this thesis to the problem solving process, presented in figure 1.2:

General problems: Presented in the problem formulation section 1.2.3.

Specific problems: Presented in the documentation of the two SoccerLab prototypes, chapter 2 and 5.

[6] Frameworks for Understanding Software-Based Systems supported by it-vest

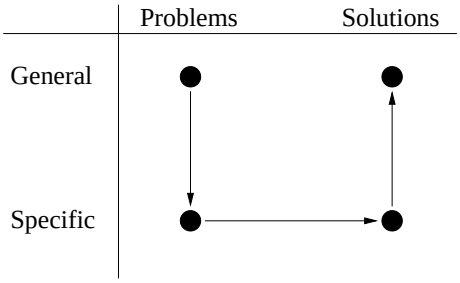


Figure 1.2: The problem solving process.

Specific solution: All the specific results obtained in this thesis are presented in section 1.5.

General solution: The general results of this thesis are presented as hypotheses in section 1.6.

Another reason for choosing a specific example such as developing a soccer simulator is that the specific result also will server as first example on how to implement collaboration using the AAC concepts.

1.5 Results

This section presents the results that have been obtained during the different stages in the same order as presented in the previous section 1.4.

1.5.1 SoccerLab prototype version 1

Based on the experience gained by conducting the preliminary application survey, presented in appendix A, the first prototype of the SoccerLab application was developed. The result is a basic soccer simulation platform suited for experimentations with the activity and habitat[7] concepts, with a simple graphical user interface. Figure 1.3 presents a screen shot of the first prototype.

Even though SoccerLab prototype version 1 only have support for simple versions of the AAC concepts, it is a well suited application for experimenting with future versions of the concepts.

1.5.2 The application survey

Two soccer related applications have been investigated during the application survey, namely The Tao of Soccer[8] and FIFA Football 2005[9].

Unfortunately no collaboration strategies were observed in The Tao of Soccer, instead it provided a lot of inspiration concerning improvements of the simulation platform. In FIFA Football 2005 by EA SPORTS^{TM} both offencive

[7] When the first prototype was implemented the habitat concept had not yet been renamed to context.

[8] http://soccer.sourceforge.net/soccer/

[9] The official FIFA Football 2005 web page: http://www.easports.com/games/fifa2005/home.jsp

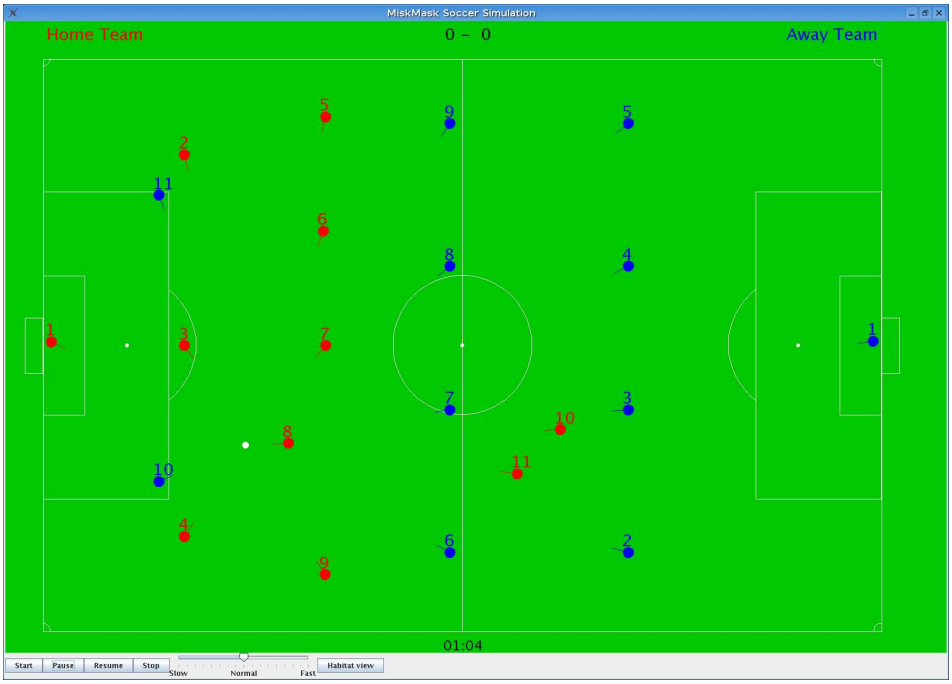


Figure 1.3: Screen shot of SoccerLab prototype version 1. The working title of the prototype was MiskMask as shown in the title bar.

and defencive collaboration strategies have been identified. Six collaboration strategies, four offencive and two defencive, have been emphasised. Two of the four offencive collaboration strategies concern playing the ball and the other two mainly concern scoring goals. FIFA 2005 also gives the user ability to define the overall team strategy by providing modifiable team-properties like; mentality, attack and defence.

Both the investigated applications have contributed with inspiration for new features in the soccer simulator. The identified features concern both collaboration strategies and platform features.

1.5.3 Characterisation of activities and contexts

Based on literature studies the two concepts activity and context have been characterised and an agent architecture has been proposed.

A BDI[10]-architecture [Rao and Georgeff, 1995] has been proposed for the agents, in combination with a modified version of the AALAADIN-model [Ferber and Gutknecht, 1998] supporting activities and contexts. The modification of the AALAADIN-model for supporting activities and contexts, and the organisation of agents supporting the role concept, is further elaborated in section 4.6.

The activity concept has been characterised as follows:

[10] Beliefs,Desires,Intentions

- Activities are abstractions over collaboration.
- An activity consist of a set of participants, a logical context, qualified by roles and a directive.
- An activity only communicates with its participants through roles.
- The directive of an activity is a sequence of actions, describing or prescribing the collaboration between its participants.
- An activity is described by a class that can be instantiated as objects with state, behaviour and identity.

Similar to the activity concept, the context concept has been characterised:

- Physical contexts are abstractions over environments, that are spatial and delimited.
- Logical contexts are abstractions over grouping of agents, and they are dimensionless and ubiquitous.
- A context is aware of the agents it comprises, and it influences the agents by providing activities.
- Contexts may overlap and are not isomorphic.
- A context is described by a class that can be instantiated as objects with state, behaviour and identity.

These flexible characterisations of the context and activity concept have been chosen to allow for experimentations, although they later on might become candidates for definitions of the concepts.

1.5.4 SoccerLab prototype version 2

Based on the application survey and the newly characterised concepts activity and context the second prototype of SoccerLab was developed. Figure 1.4 shows a screen shot of the developed prototype. As shown, to the right, in figure 1.4 an option panel has been introduced in the graphical user interface, providing more user interaction and visualisation of activities and contexts.

The newly characterised activity concept has been implemented in SoccerLab prototype version 2 for prescribing collaboration between multiple agent. After experimenting with activities in SoccerLab prototype version 2, activities have proved to be flexible and powerful for prescribing collaboration. The flexibility is achieved because activities only access their participants through roles. The powerfulness of activities is achieved because activities provide centralised control of their participants and hence makes it easy to prescribe even complex collaboration.

The support for the context concept in SoccerLab prototype version 2 has been achieved by defining two interfaces, one for physical contexts and one for logical contexts, for the context implementing classes. Both interfaces inherit methods defined in the general context interface as visualised in figure 1.5.
After experimenting with the context concept in this prototype it is suggested to change the approach of implementing contexts from interfaces to abstract

Figure 1.4: Screen shot of SoccerLab prototype version 2.

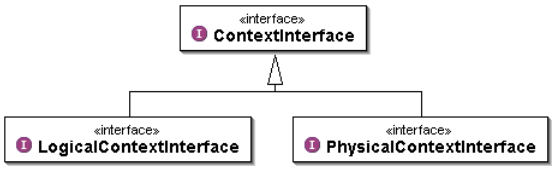


Figure 1.5: The context interface inheritance structure.

classes.

In this section the specific results of this thesis have been presented. The following section presents the results from a general perspective by presenting seven hypotheses.

1.6 Hypotheses

This section presents the result of the thesis in form of seven hypotheses, enumerated as follows:

1. The activity concept is an abstraction over collaboration between agents.

2. Activities are implemented by classes that can be instantiated as active objects.
3. Implementation of activities is facilitated by utilising an inheritance hierarchy.
4. Physical contexts are abstractions over environments supporting its comprised agents.
5. Logical contexts are abstractions over groups supporting its comprised agents.
6. Contexts are implemented as classes that are instantiated as active objects.
7. Dynamic visualisation of contexts and activities increases the understandability.

Each of the seven hypotheses are substantiated by arguments and exemplification. The hypotheses can be classified into three categories based of the topics they concern; activities section 1.6.1, contexts section 1.6.2 and visualisation section 1.6.3.

1.6.1 Activities

The activity concept can be applied both at an abstract conceptual level and at the concrete programming level. Activities are used at the abstract conceptual level during analysis and design to describe and prescribe collaboration phenomena. The activity examples identified during analysis are described by classes that at runtime are instantiated as active objects prescribing collaboration in the executing program. The following presents the results of this thesis concerning activities in a general perspective in form of three hypotheses:

Hypothesis 1: The activity concept is an abstraction over collaboration between agents

Activities model collaboration between agents at a conceptual level. Observed collaboration phenomena from the referent system or problem domain[11] are classified into the concept of activity according to their common properties. The common properties include the roles of the participant and a sequence of their interaction which is captured by the activity concept. An activity consist of a set of participants(agents) qualified by roles forming a logical context and a directive describing or prescribing the coordinated action sequence. The activity concept is described in details in section 4.4.1.

The activity concept have many similarities with the association concept presented in [Kristensen, 2005], but there are some important differences too. Associations model collaboration between objects in object orientation whereas activities model collaboration between agents. Also the perspective on the participants in an association and an activity is different. Associations consider their participants as a list, whereas the participants in an activity are treated as a logical context, meaning that relevant informational data is provided to the participants in an activity at group-level which is not the case for associations.

[11] The notions of problem domain is further elaborated in [Jacobsen et al., 1999]

Example: The activity concept can, for instance, be applied where the problem domain is a soccer scenario. In the referent system various examples of collaboration-phenomena between soccer players such as: throw-in, corner kick, ball passing etc. are identified. All of these phenomena can be classified into activities. To engage in the throw-in activity agents (players) that can play the following roles are needed; the "thrower"-role and one or more receiving-role(s). When the roles in the activity are filled the action sequence prescribed by the directive is executed as follow:

1. Receiver(s) should reposition themselves to obtain unmarked location(s).
2. The "thrower" should observe how receivers reposition.
3. The "thrower" selects a receiver and throws the ball to him.

More examples of modelling with activities at a conceptual level is presented in appendix C.1.

Hypothesis 2: Activities are implemented by classes that can be instantiated as active objects

When observed collaboration phenomena have been modelled as activities at a conceptual level, they are implemented as classes in an object oriented programming language, using specific language mechanisms and constructs. When a program containing activities is executed the activity classes are instantiated as active objects and hence they prescribe collaboration in the executing application identical to the collaboration observed in the problem domain. The implementation of activities is further described in chapter 5, which is the first attempt to implement the activity concept in software.

Example: SoccerLab prototype version 2 is implemented in Java. The soccer related activities are implemented as classes in the soccer simulator implementing the runnable interface. When the application is executed the activity-classes are instantiated as runnable object, threads. This is further elaborated in section 5.3.1.

Hypothesis 3: Implementation of activities is facilitated by utilising an inheritance hierarchy

To simplify the addition of new activities to an application, activities can advantageously be implemented in an inheritance hierarchy. All the logic required by the implementation language for handling concurrent execution, e.g. threads in Java, handling of roles etc. are implemented in the super-class and the specific activities are implemented as sub-classes. The activity sub-classes define the roles required for the specific activity and override the directive of the super-class. The benefit of using this approach is that the (sub-)class representation of activities matches the conceptual activities facilitating the understanding of the relation between activities as concepts and activities as classes. Another benefit of utilising an inheritance hierarchy for implementing activities is that the part of the implementation common to all activities is reused.

Example: In SoccerLab prototype version 2 the activity-classes are organised in an inheritance hierarchy. The activity super-class implements the runnable interface and logics for assigning participants to roles. The activity sub-classes implements the specific activities. This is described in section 5.3.1.

```
public class SimpleDefenciveActivity extends Activity {
    private Role role1,role2,role3,role4;

    public SimpleDefenciveActivity() {
        super();
        role1=new DefenciveRole("defenciveRole");
        role2=new DefenciveRole("defenciveRole");
        role3=new DefenciveRole("defenciveRole");
        role4=new DefenciveRole("defenciveRole");
        requiredRoles.add(defined roles);
    }

    protected void directive(){
        do{
            role1.runToLocation(Ball.getPosition());
            role2.runToLocation(defencive location);
            role3.runToLocation(defencive location);
            role4.runToLocation(defencive location);

            Thread.sleep(activityThreadDelay);
        }while(all the roles are filled || the collaboration isn't done);
    }

}
```

Table 1.1: Example of an activity sub-class implementation.

Table 1.1 shows a simple pseudo example of an activity sub-class. The constructor initialises the required roles, and the directive of the super-class is overridden. The directive describes defencive collaboration involving four participants, where the agent filling role1 will chase the ball and the rest of the agents will run to their defencive locations.

1.6.2 Contexts

Similar to the activity concept the context concept can be applied both at an abstract conceptual level and a concrete programming level. Contexts are used as abstraction over environment(physical context) and grouping(logical context) supporting their comprised agents. Analogous to activities, contexts are implemented as classes, implementing the context interface, that at runtime are instantiated as active objects. The general result of investigating and experimenting with the context concept is presented as the following three hypotheses:

Hypothesis 4: Physical contexts are abstractions over environments supporting its comprised agents

Physical contexts model environments supporting its comprised agents by providing activities and relevant information to them.

Environmental phenomena encouraging or supporting its inhabitants are classified into the concept of physical context. The common properties of such environments includes spatiality, physical boundaries and encouragement of specific behaviour. The physical context concept captures all of these properties.

A physical context consist of delimited area in which it is aware of the present agents. A physical context supports its comprised agents both by encouraging them to take special actions by providing activities and by providing relevant informational data. The context concept is described in details in section 4.5.1.

Example: The referent system is as soccer scenario in which the physical context concept is applied. If the soccer field is investigated in details various phenomena like; penalty area, goal area, centre circle, etc. are identified and classified into physical contexts. These physical contexts are aggregated to form a new physical context namely the soccer field context. Besides providing activities to the agents, the soccer field context provides informational data to the agents like; what is the score of the match and which player is closest to the ball. More examples of conceptual modelling with physical contexts are presented in appendix C.2.

Hypothesis 5: Logical contexts are abstractions over groups supporting its comprised agents

Logical contexts model groups supporting its comprised agents by providing activities and relevant information to them.

Group phenomena encouraging or supporting its members are classified into the concept of logical context. The majority of the properties of physical and logical contexts are identical. The differences is that physical contexts are spatial and delimited by physical boundaries whereas logical contexts are dimensionless and ubiquitous with conceptual boundaries delineated to what is appropriate where and when. Similar to physical contexts, logical contexts support their comprised agents both by encouraging them to take special actions by providing activities and by providing relevant informational data.

The powerful aspect of logical contexts is that they provide activities to a group of agents independently of the environment in which they are present, allowing the agents to engage in collaboration because they are members of the same group. Logical contexts are further explained in section 4.5.1.

Example: By investigating a soccer team in details, each soccer player can be categorised in groups depending on his responsibilities on the team. These group phenomena; defence, midfield and offence, can be classified into logical context. Each of these logical contexts provide activities that are relevant for the specific group. Additional examples of conceptual modelling using logical contexts are presented in appendix C.2.

Hypothesis 6: Contexts are implemented as classes that are instantiated as active objects

Both physical and logical contexts modelled at a conceptual level are implemented as classes using specific language mechanisms and constructs. It is a requirement that the implementing classes implement the context interface, described in chapter 5. When an application containing context classes is executed the context classes are instantiated as active context objects, providing support for contexts in the executing application.

Example: The SoccerLab prototype version 2 application has various examples of both physical and logical contexts classes. For instance, the physical context soccer field which supports the agents with soccer related activities, is an composition of two goals, two goal areas, two penalty areas and a centre circle, which are all physical contexts. Section 5.3.1 presents a detailed description of context implementation in SoccerLab prototype version 2.

1.6.3 Visualisation

As an attempt of increasing the understandability of activities and contexts, dynamic visualisation has been included in the graphical user interface of SoccerLab prototype version 2. The result is presented in the form of the following hypothesis:

Hypothesis 7: Dynamic visualisation of contexts and activities increases the understandability

Based on the experimentations with visualisation of contexts and activities, described in section 5.3.2, it is concluded that dynamic visualisation of the concepts facilitate the understanding of them and how they are applied.

The dynamic visualisation of contexts provides the possibility of identifying which parts of the simulated environment that are influenced by the contexts, at runtime, and at the same time view relevant information provided by the context. Similar, the dynamic visualisation of the activity concept provides insight of what an activity is and what is required for it to execute, which is difficult to achieve from a static drawing on a piece of paper.

Example: In SoccerLab prototype version 2 simple approaches have been chosen for visualising both contexts and activities.
Physical contexts are visualised by cyan coloured transparent shapes on top of the soccer field, as presented at figure 1.6, and additionally presenting the following information:

- The name of the context.
- The number of contained agents.
- The number of contained activities.
- Whether the ball is present or not.

The context's informations are continuously updated and presented to the user in a dynamic manner allowing the user to keep track of the various information. Logical context are currently visualised by a textual description present in the console of the GUI, due to the fact that they are dimensionless.

When an activity is executing it is visualised by the way the agents collaborate accompanied by a textual description in the activity info panel, as depicted at figure 1.7 presenting detailed information of the activity.

In general the visualisation of contexts and activities in SoccerLab prototype version 2 is a good tool for demonstrating the concepts "in action".

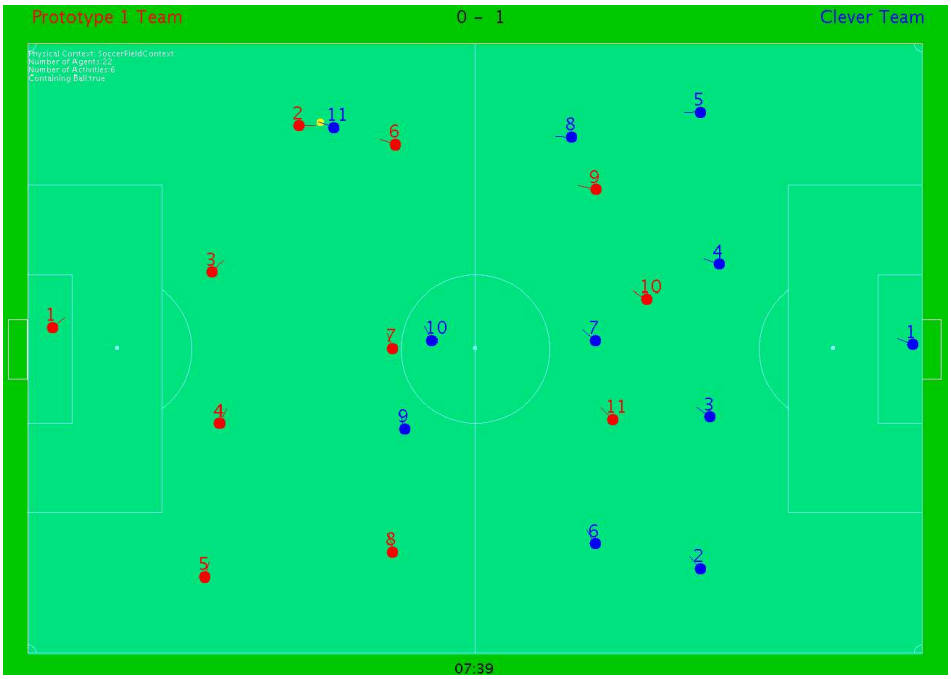


Figure 1.6: Screen shot of the soccer field from SoccerLab prototype version 2, with visualisation of the soccer field context.

1.7 Discussion

This section presents a discussion of the developed product, SoccerLab prototype version 2 and the obtained results.

At the specific level, a soccer simulator, SoccerLab prototype version 2, has been developed to serve as the experimental platform for working with the AAC concepts. SoccerLab prototype 2 has been developed using the prototyping approach, briefly described in section 1.4, which in this case has been a good approach because it has provided the flexibility to experiment with the AAC concepts.

SoccerLab prototype version 2 has support for agents provided by a simple agent architecture, described in section 2.5.1 and section 5.3.1, satisfying the notion of agency, presented in section 1.1 and further elaborated in section 4.3.1.

At the general level the problems have been solved using the approach presented in section 1.4.1. A flexible characteristic of the activity concept is presented in section 1.5.3. Section 1.6.1, describes and exemplifies how activities are applied during analysis,design and implementation. Based on experiments it is concluded that activities are flexible and powerful for describing and prescribing collaboration between agents. Even though activities prescribe and describe collaboration in a centralised manner, they preserve a high level of agent autonomy because it is up to the agents to decide when to engage in an activity and when to leave it again. As argued, in section 1.6.1, activities share some

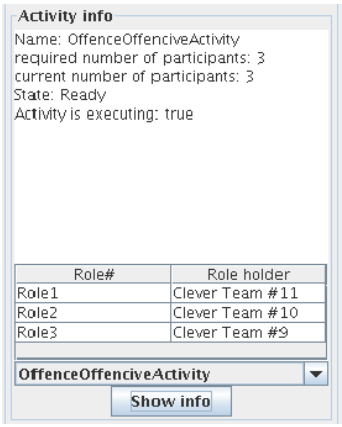


Figure 1.7: Screen shot of the activity info panel from SoccerLab prototype version 2 with visualisation of activities.

properties with associations but there are differences too. To the knowledge of the author no concrete example of implementing associations exists which have been achieved with activities in this thesis.

The concept of context have been characterised, the characterisation is presented in section 1.5.3, and section 1.6.2 argues and exemplifies how contexts are applied both during analysis and design at a conceptual level, and at programming level during implementation.

Section 1.6.3 argues why and exemplifies how dynamic visualisation of activities and contexts increase the understandability of the concept. One could argue that existing static notation for executing activities, see figure 1.9(a) can be used, but a dynamic representation of a dynamic phenomenon gives a much better impression and understanding of the phenomenon.

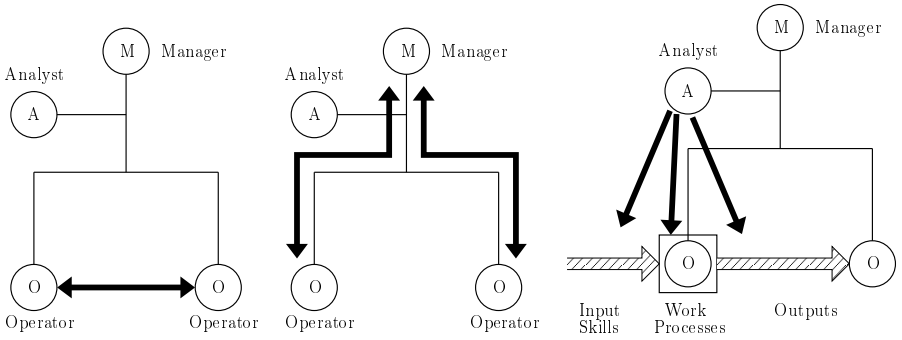
1.8 Reflections and future work

This section presents; reflections on coordination of collaboration both between agents and between humans, and suggestions for future work.

1.8.1 Reflections on collaboration

When entities, agents as well as humans, engage in collaboration it set-up some requirements for coordination of the individual entity's actions.

According to Henry Mintzberg [Mintzberg, 1983] there are five coordinating mechanisms that seems to explain the fundamental ways in which organisations coordinate their work; mutual adjustment, direct supervision, standardisation of work process, standardisation of work outputs, and standardisation of worker skills. The five coordinating mechanisms are illustrated in figure 1.8.



(a) Mutual adjustment. (b) Direct supervision. (c) Standardisation.

Figure 1.8: The five coordinating mechanisms presented in [Mintzberg, 1983].

Coordinating mechanism in multi-agent systems

In general multi-agent systems, such as presented in [Appel and Nielsen, 2005], all the coordination is obtained through negotiation between the involved agents, using mutual adjustment as coordinating mechanism, see figure 1.8(a).

Coordinating mechanisms in SoccerLab

In SoccerLab prototype version 2, collaboration between agents is achieved by using activities. An activity is considered as a combination of several of the coordinating mechanisms presented in figure 1.8. An activity tells its participants what to do, which is an example of direct supervision, see figure 1.8(b), but at the same time the directive of the activity can be considered as standardisation of work process, illustrated in figure 1.8(c). When agents decide to engage in an activity, they are qualified by roles or skills and hence activities can also be considered as standardisation of skills, also illustrated in figure 1.8(c).

Coordinating mechanisms in real soccer

In real soccer all of the five coordinating mechanisms are represented. The manager or captain of a team direct players using direct supervision. In dead ball situations various examples of standardisation of work processes are identified. An example of standardisation of outputs, illustrated in figure 1.8(c), is; when a right-wing player has the ball, the attackers in front of the goal expects the output from the right-wing player to be a cross-ball. When the players play different field-positions such as; right-wing and attacker, it is examples of standardisation of skills. Least but not last, creativity when the players are passing the ball to each other, often making soccer entertaining to watch, is an example of mutual adjustment, based on quick decisions.

In future work it would be interesting to investigate further, how all of the five coordinating mechanisms could be applied in MAS, instead of the current narrow-minded perspective of only using mutual adjustment as coordinat-

ing mechanism. The following section presents additional suggestions for future work.

1.8.2 Suggestions for future work

Even though the developed experimental platform SoccerLab prototype version 2, demonstrates the essential usage and benefits of the AAC concepts, various aspects can be improved. Both proposed ideas from analysis of concepts chapter and the application survey chapter haven't yet been implemented, but also ideas for future applications of the SoccerLab simulator have emerged during the final phase of this thesis though they require additional research.

Description of an activity's purpose: The current implementation of activities are both flexible and powerful. With the current implementation of activities humans can easily decide which activity to choose, based on the name of the activity, but this is not the case for agents. An improvement that would make it easier for the agents to choose the correct activity in given situation would be to introduce some kind of formal description of activities that agents understand, consisting of purpose and requirements for the specific activity.

Support for "half-baked" activities: Currently activities can only describe or prescribe collaboration between participants qualified by a set of required roles, but one could imagine collaboration between participants where some of them are required and others just participate if they are available. E.g. in soccer a corner-kick possesses those properties, it only requires one player for kicking the ball, but there are no requirements stating how many teammates that should be located in front of the opponent's goal. The number of players in front of the opponents goal may vary from none to ten, depending on the situation or more likely a decision made by the coach or the team captain.

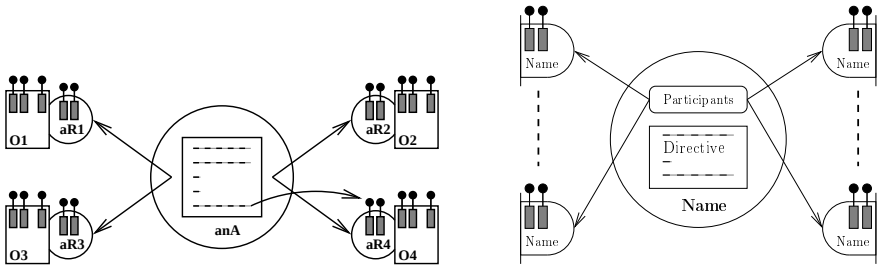
This kind of collaboration can be handled by "half-baked" activities[12]. A "half-baked" activity is in many ways similar to the activity concept presented in this thesis, the major difference is that "half-baked" activities qualify its participant both by essential or required roles and a set of possible roles, and the activity will execute as long as the essential roles are filled.

Improve visual representation of activities: In SoccerLab prototype version 2 visualisation of activities is proposed as textual, but dynamic descriptions of important information of the activity in the GUI.

Inspired by a notation for executing activities[13], figure 1.9(a), proposed in [Kristensen and May, 1996], a more graphical visualisation, compared to the existing textual one, is proposed, at figure 1.9(b).

[12] Confer conversation with professor Peter Bøgh Andersen, Department of Information and Media Studies, University of Aarhus

[13] The activities presented in [Kristensen and May, 1996] are abstractions for collective behaviour in object orientation. Not to be confused with the activities presented in this thesis as abstractions over collaboration.



(a) Notation for an executing activity as proposed in [Kristensen and May, 1996].

(b) Proposal for visualisation of activities inspired by the activity notation from figure 1.9(a).

Figure 1.9: Examples of visualisation of the activity concept.

The circle in the middle of figure 1.9(b) represents the activity at runtime with the name, directive and participants(logical context) of the activity. Additionally the circle could present the informations that the textual display from the current visualisation presents. It might not be necessary with a textual description of the number of required roles and the number of filled roles, because each of the "legs" of the activity represent a role required by the activity. When a specific activity is chosen for visualisation, the filled roles are drawn with solid lines whereas unfilled roles can be drawn with dot-and-dash or shaded lines. The filled role should also present the name or an identification of the participant playing the specific role.

Develop an AAC application framework: One of the most obvious steps for the future is to develop an application framework for working with agents, activities and context, and this would of course require some studies of framework theory. Additionally experiments with the AAC concepts should be conducted for other scenarios than soccer, for instance a ship simulator like the one described in appendix E. Currently the concepts in SoccerLab prototype version 2 i not fully mature to be applied in an application framework, especially the agent concept needs some refinement.

Chapter 2

Prototype version 1

This chapter documents the development of the first prototype. The purpose of developing this prototype, is to get a simple experimental platform suited for experiments with the AAC concepts. During this stage the context concept was labelled habitat, for further elaboration on renaming the habitat concept to context refer to section 4.1. The development of this prototype is founded on

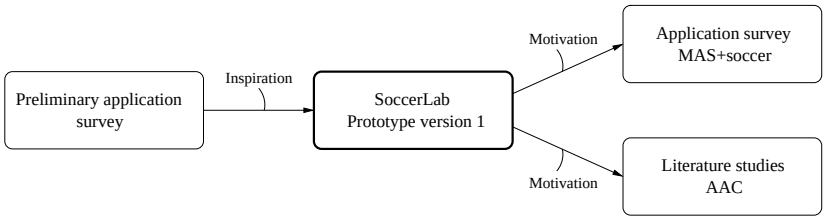


Figure 2.1: The prototype version 1 chapter in a larger perspective.

inspiration gained from a preliminary application survey, presented in appendix A, of four simple soccer simulations. The prototype described in this chapter has motivated another application survey and the literature studies of the AAC concepts, as visualised in figure 2.1, both presented in forthcoming chapters. During this stage the developed prototype had not yet been labelled SoccerLab. The working title of the prototype during this stage was MiskMask or simply Prototype version 1.

2.1 Introduction

The motivation for developing this first prototype is to develop a basic soccer simulation platform suited for experimentations with the habitat and activity concepts. At this early stage of the project the habitat and activity concepts have not been paid noticeable attention and because of that, the realisation of these two concepts in the prototype will be rather simple.

The habitat concept for this prototype will be restricted to be a limited area, but without restrictions to the shape and size of it.

The restrictions for activity concept in prototype version 1 is that only soccer players will take the initiative to execute an activity, albeit it is not decided yet whether this is the correct approach or not.

The main guidelines for this prototype will be: simplicity, flexibility, understandability and extendability to ensure that the basic soccer simulator is suitable for experimentation in future prototypes.

2.2 Application description

The desire for simplicity is clearly reflected in the GUI of the developed prototype version 1. The graphical user interface is depicted at figure 2.2.

Figure 2.2: Screen shot of soccer simulation application.

The user interface contains:

1. The start button is used for starting the simulation. If the simulation is running this button restarts the simulation.
2. The pause button pauses the simulation.
3. The resume button resumes the simulation.
4. The stop button stops the simulation.
5. The speed slider adjusts the simulation speed, from zero to four times normal speed.
6. The habitat view button toggles visibility of the habitat panel (figure 2.3). If the simulation is not started this button has no effect.

7. The game clock. When 90 minutes has passed, the game is over.
8. The score of the match
9. The team name. For demonstration purposes the teams are currently named: “Home Team” and “Away Team”.

Figure 2.3: Screen shot of the habitat chooser panel.

When the habitat view button is pressed, the habitat panel becomes visible. The habitat panel is shown at figure 2.3. This panel allows the spectator to enable drawing of the active habitats. The habitat panel can be divided into three sections.

1. Choose one or both teams for which their habitats should be drawn. If neither of the teams are chosen no habitats will be drawn.
2. Choose the kind of habitat(s) to be drawn. If neither of the habitats are chosen, no habitats will be drawn.
3. Choose the player group(s) for which their habitat should be drawn. If none of the groups are chosen, no habitats will be drawn.

2.3 Architecture

A client-server architecture has been chosen for the application, in which the client consist of the application GUI, and the server consist of the model component or simulation engine. The reason for choosing the client-server architecture is mainly because of the desire for e.g. extendability. Later on the simulator could be extended to support multiple users in a distributed manner. In the following sections a more detailed description of the GUI- and model component is given.

2.3.1 The model component

A class diagram of the model component for this first prototype is depicted in figure 2.4. The most important parts of the class diagram besides the habitat class and the activity class is the SoccerSimulation class and the association between the Player class and the Activity class.

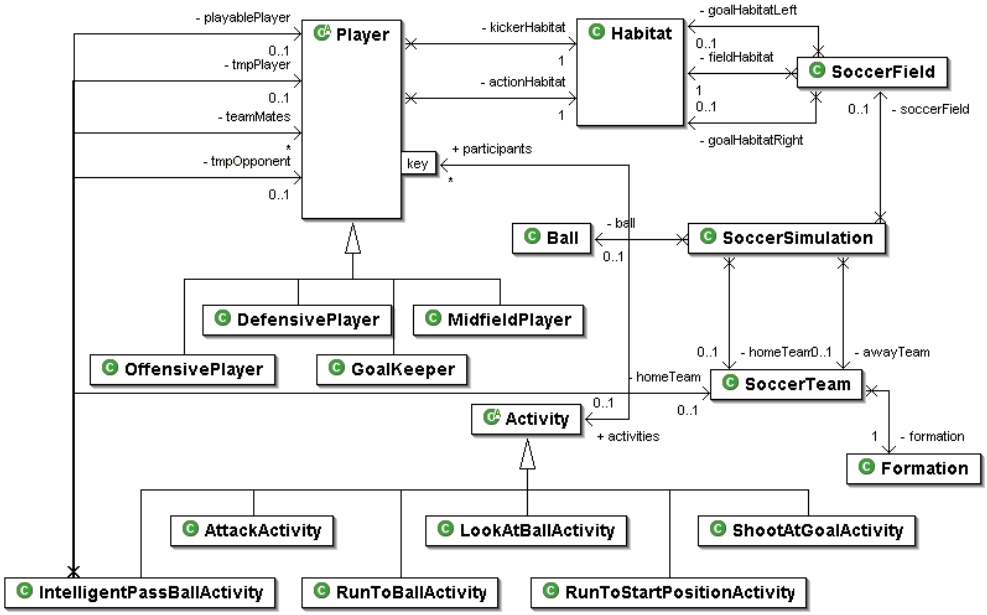


Figure 2.4: Class diagram of the soccer simulation's model component.

The habitat class is implemented as a static factory [Bloch, 2001]. The static factory provides the different kinds of habitats. Currently the habitats only differ in shape.

As a first approach it has been chosen that only the player can execute activities. To allow the players to execute activities in a uniform manner, the command pattern has been implemented [Gamma et al., 1995]. This makes the introduction of new activities very simple and highly flexible. To introduce a new activity to the soccer simulator the only thing needed is to implement a new sub class of the abstract activity class.

In the model component the SoccerSimulation class binds everything together and initialises the simulation. When the simulation is started from the GUI, the request is handled by the SoccerSimulation class. This also apply for the other control options present in the GUI.

To make the simulation more realistic all distances in the model component are measured in meters and all velocities are scaled to realistic values.

2.3.2 The GUI component

The GUI component only holds three classes, see figure 2.5. The main class of the GUI component is the SoccerSimulationGUI class. This class also holds the main method to start the application. Basically the SoccerSimulationGUI class has the responsibility to draw the graphical user interface and to send the users requests, start, stop, pause, etc. to the soccer simulator model. To keep the simulation as smooth as possible the simulation window is updated with a frame rate of approximately 30 frames pr second. This gives the GUI around 30 milliseconds to read all information in the model component that

should be presented to the user. The DrawableSocccerField class is a helper

Figure 2.5: Class diagram of the GUI component.

class for the main GUI class. Besides drawing of the field this class also converts model coordinates to GUI coordinates. This conversion is necessary because of the draw conventions in Java and the position of the origin of the coordinate system of the soccer field in the model component.

The HabitatPanel class has the responsibility of handling user interaction, concerning drawing of habitats.

2.4 Dynamics

To describe the dynamics of the soccer simulation prototype version 1, three of the more interesting areas from the model component has been chosen, for a more detailed description.

2.4.1 Initialisation of the soccer simulator

When the application is started the first thing that will happen is that the soccer simulator is initialised. Figure 2.6 shows a sequence diagram of the initialisation method in SoccerSimulation class.
First the soccer field is instantiated and initialised, next the ball is instantiated and finally the two teams are instantiated and initialised which include setup of the teams' formation. Now the soccer simulator is initialised and waiting for the user to start a simulation from the graphical user interface. The objects created during the initialisation will exist until the simulation is stopped or the application is terminated.

2.4.2 Setup of team formation

The sequence diagram in figure 2.6 shows among others that the team formation is set for each of the teams. Figure 2.7 gives a more detailed view on the setFormation method invocation.
The sequence diagram in figure 2.7 shows that when a soccer team object's setFormation method is invoked it passes the method invocation to the soccer team's formation object which has the responsibility to calculate the positions of the players of the team. When the player positions have been calculated the players of the team are updated with their new position.

The reason setting the formation in this manner is to be able to change formation dynamically during matches. This feature is unfortunately not supported in this prototype, which is mainly because a change in formation doesn't affect the player kind. To put it in other words, if a midfield player is moved to the defence, due to a change in formation, he will still act as midfield player and not as defencive player as expected. This should be changed before it makes sense to enable the feature of dynamically changing formation at runtime.

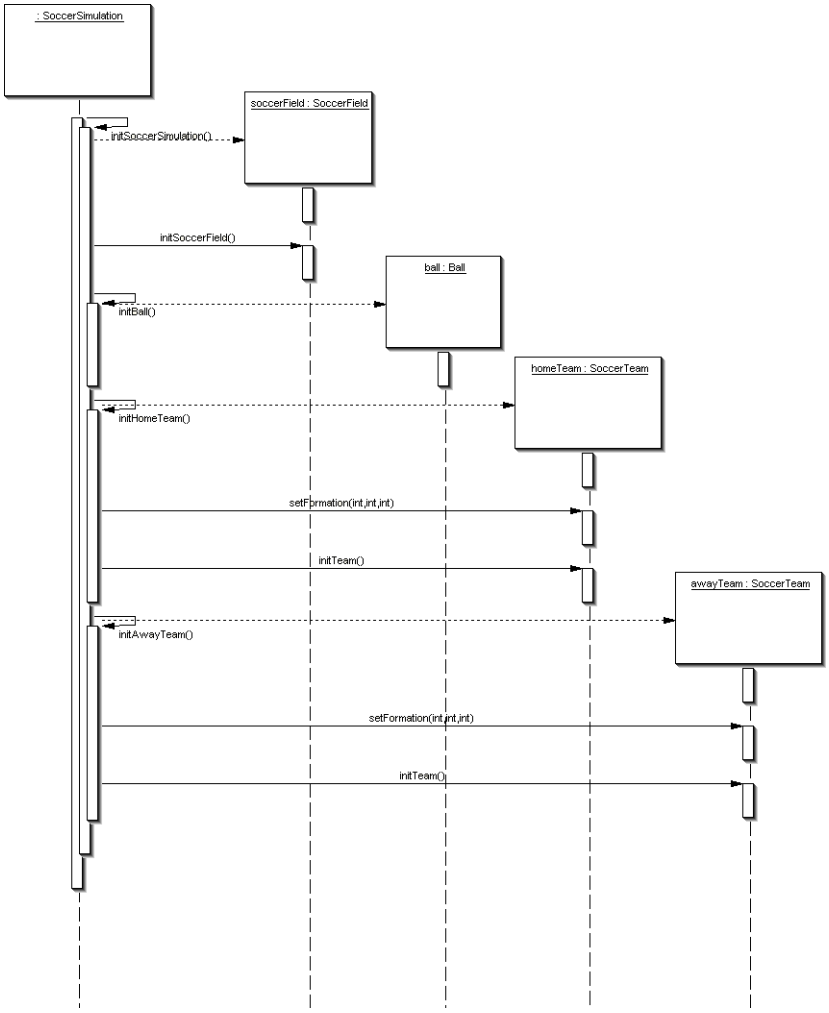


Figure 2.6: Sequence diagram of the initialisation of the soccer simulator.

2.4.3 The activity selection process

The last of the three emphasised areas of the soccer simulation concerning dynamics, and maybe most interesting, is how an player selects which activity to execute. Algorithm 1 shows an example of the activity selection part of a midfield player's run loop.

For each step in the midfield players' run loop this activity selection process is executed, which allows a player to react quickly to sudden changes like; getting in ball possession, tackling or simply intercepting the ball if it's within reach, which all are frequently occurring events in soccer.

All of the player kinds have similar activity selection processes with only minor deviations, for instance, the offencive players will try to score, if they are close to the opponents goal, instead of passing the ball and the goalkeeper almost never leaves the goal area.

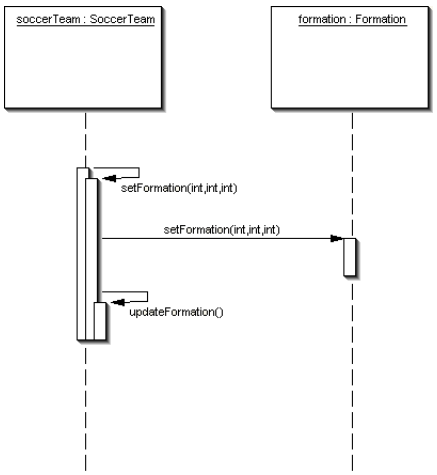


Figure 2.7: Sequence diagram showing the interaction between a soccer team and it's formation.

This way of handling activities ensures that an agent is only engaged in an activity for a short period of time and thereby prevents the agents from being stuck in one activity when they are supposed to engaged in another one.

2.5 Classes

In this section a brief description of each of the classes in the model component and the GUI component is presented.

2.5.1 The model component

The model component contains the following classes, presented in groups of activities, habitats, players(agents) and miscellaneous:

***Activity*:** Abstract super class defining the interface for a command in this command pattern implementation, say, the execute method. The reason for choosing an abstract class implementation instead of an interface implementation is to achieve an inheritance structure for all the specific activities.

AttackActivity: A sub class of *Activity*. This activity is executed, when a team becomes in ball possession. The result is that all players on the team, except the goal keeper, will run to a more offencive field position. This activity is a bit different from the rest of the activities, because normally an activity is executed, when a player is in ball possession, i.e. the player in possession of the ball has the initiative to execute an activity. This activity will be executed, if just one of the players team mates is in ball possession.

IntelligentPassBallActivity: A sub class of *Activity*. This class is called "intelligent" because it is a new and more offencive version of the PassBal-

Algorithm 1: Activity selection for midfield players

```
if the ball is inside the action habitat then
    if the ball is inside the kick habitat then
        execute passBallActivity;
    end
    else
        execute runToBallActivity;
    end
else if our team has the ball then
    execute attackActivity;
else
    if start position can be reached in one step then
        stay and defend;
    else
        execute runToStartPositionActivity;
    end
end
```

lActivity class. This activity will try to pass the ball to a more or equally offencive positioned and unmarked team mate. If none of the team mates fulfil the criteria of being playable, or the player executing the activity is the most offencive positioned player on the team, the ball will be passed to a randomly chosen team mate.

KickBallActivity (*Deprecated*): A sub class of *Activity*. This was the first attempt on an activity to let players kick the ball around. When the ball is inside a player's kick habitat he will kick the ball to a random location with a random velocity. This activity is currently not in use, but it is kept in this prototype for testing purposes.

LookAtBallActivity: A sub class of *Activity*. This activity calculates the direction of the player "nose". All the players execute this activity as the last thing in their run method. It can be questioned, whether this is an activity or not, because it is common for all the players, and thereby the responsibility could be moved to the *Player* class.

PassBallActivity (*Deprecated*): A sub class of *Activity*. This is the old and simple version of IntelligentPassBallActivity. This activity simply chooses randomly among the players team mates, and passes the ball to the chosen one.

RunToBallActivity: A sub class of *Activity*. When a player executes this activity, he will start running towards the ball. When this activity was first introduced, it was the default activity for all the players, which resulted in some sort of flocking behaviour, and within 30 seconds all the players were gathered around the ball and following it around the field.

RunToStartPositionActivity: A sub class of *Activity*. This activity will reset a players current field position. E.g. if a defencive player has been chasing the ball because it was inside his action habitat,then, when the ball leaves

the action habitat the player will execute this activity and run to his default position. This activity is also used, when the teams switch halves. This is done by calculating the new starting positions at half time and during the halftime break setting all the players to execute this activity.

ShootAtGoalActivity: A sub class of *Activity*. This activity is executed if a player is inside the opponents penalty area. When this activity is executed the player will kick the ball to a random position between the poles and behind the goal line. The speed of the ball is determined by a random value between $90\,km/h$ and $120\,km/h$.

Habitat: This class enables support for the habitat concept. This class is implemented as a static factory [Bloch, 2001], this allows for instantiating habitats in circular and rectangular shapes with the same functionality.

***Player*:** Abstract super class that holds all the basic fields and methods of the different player types. A player has two habitats: a kick habitat and an action habitat. The kick habitat is a circular area of 1 meter in diameter. When the ball is inside this habitat the player is allowed to kick or pass the ball. The shape and size of action habitat should be overridden with appropriate values in the subclasses. Currently the shape of the action habitat is rectangular but it could be any shape provided by the Habitat class. When the ball is inside the action habitat the player is supposed to take special action like running to the ball or pay more attention to the opponents.

The *Player* class severs as the implementing class of the agent architecture supported in this prototype. The *Player*(agent)-class implements the runnable interface which allows the agent to act in an autonomous and pro-active fashion. For each iteration in an agents run-loop it will perceive the environment, which is achieved by an if-else structure in combination with the agent's associated habitats. Depending on how the agent is situated, the if-else structure will determine which one of the activities that is appropriate to execute and hence make the agent act and provide agent reactivity.

DefensivePlayer: A sub class of *Player*. The responsibility of this class is, as the name implies, to handle the defencive players. The reason for distinguishing between different player types is that it should be possible to introduce strengths and weaknesses for each player type according to their field position.

GoalKeeper: A sub class of *Player*. This type of player has the responsibility of stopping the other team from scoring. Hence he will stay at the goal and when the ball enters the penalty area, he will try to kick it away or pass it to one of his team mates.

MidfieldPlayer: A sub class of *Player*. This player type is supposed to help both the defencive and offencive part of the team.

OffensivePlayer: A sub class of *Player*. The most offencive of the player types. If an offencive player is inside the opponents penalty field, he will try to shoot to score.

Ball: This class holds the ball implementation. The ball is implemented as a tread, to allow it to move concurrently with the soccer players. When a soccer player wants to kick the ball, he calls a method on the ball object that sets the ball's desired destination and a desired velocity. Then the ball calculates it's own trajectory, which is a straight line from the ball current position to the newly set desired position. The ball thread's run method will now handle the movement of the ball in step sizes according to the desired velocity, if none of the other players intervene.

Formation: The Formation class is associated to the SoccerTeam class. This is done to decouple the formation specific responsibility from the SoccerTeam class and increase the flexibility of the application. A Formation object is always created, when a SoccerTeam class is instantiated. After the SoccerTeam class is instantiated, the setFormation method can be invoked with a desired formation. The desired formation is given in normal soccer terms e.g. 4-4-2, that is to say four defencive, four midfield and two attacking players. The Formation class will automatically position the players equally over the field.

SoccerField: The soccer field is implemented by three rectangular habitats, a large habitat representing the field and two habitats, one for each end, representing the goals. All distances in the soccer field model are measured in meters. The coordinates in the soccer field are handled by a two dimensional coordinate system with origin in the centre of the field. To ensure a proper proportioned soccer field all distances have been collected from FIFA's web page, [Federation Internationale de Football Association].

SoccerSimulation: The SoccerSimulation class has the responsibility of initialising the simulation platform. Besides that, it is also the simulation engine which handles all the requests from the GUI component like starting, stopping and pausing the soccer simulation.

SoccerTeam: This class has the responsibility to managing a team and instantiate the desired number of players, normally 11 players. This class also holds the team name and colour.

2.5.2 The GUI component

The classes of the GUI component are:

DrawableSoccerField: This class has the responsibility of drawing all the required lines in a soccer field defined in FIFA's Laws of the Game [Federation Internationale de Football Association]. The class also handles the conversion of field coordinates from the model component to GUI coordinates. This is required because of the drawing conventions in Java, where the origin is placed in the upper left corner.

HabitatPanel: This class handles the habitat panel where the user can choose which of the habitats that should be drawn at run time, as described in section 2.2. A screen shot of the habitat panel is shown at figure 2.3. Figure 2.8 shows a screen shot of a running simulation where it has been chosen to draw the action habitats for the offencive players and the goalkeeper

for both of the teams. The transparent magenta areas in the screen shot is the first attempt to visualise habitats.

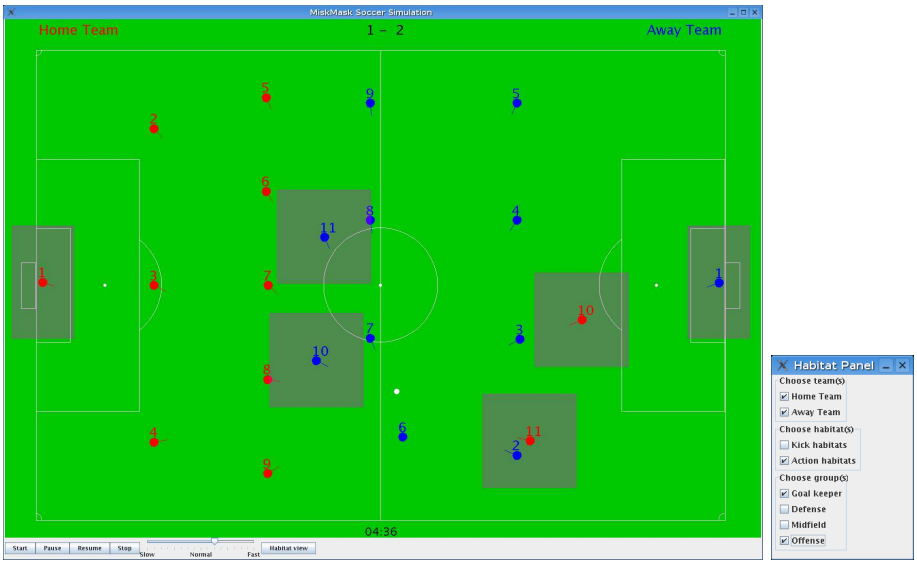


Figure 2.8: A screen shot of the running soccer simulator when the visualisation of habitats is enabled.

SoccerSimulationGUI: This is the main class of the GUI component, and it also holds the main method of the entire application. The GUI implemented in this class is depicted at figure 2.2. The green background to which the soccer field is drawn is the graphics2D object of a Canvas. The graphics2D object has been chosen to utilise the openGL pipeline in Java. See appendix F for instructions of how to enable the openGL pipeline in Java applications, which by default is disabled.

2.6 Test and evaluation

This section will describe the test of the prototype version 1 and give an evaluation of the prototype with suggestions for improvements.

2.6.1 Test

Due to prototyping working process, new features have been tested as soon as a new prototype release was available. After the final release of the prototype version 1, a full time test of the running application has been done. The application was tested by running the simulation for 90 minutes, the length of a normal soccer match, to verify that the application wouldn't crash unexpectedly. The simulated match ended after 90 minutes as expected, and no unexpected behaviour of the application was discovered. The pause/resume features were

tested in both first and second half of the match and they didn't have other influence on the simulation, besides the intended. At half time the players correctly switched halves, also as intended.

2.6.2 Evaluation

In general this prototype satisfies the requirements of simplicity, flexibility and extendability, which is important because it will function as the basic platform for experimenting with the habitat and activity concepts. Even though the test of the soccer simulator was very successful, there are some shortcomings, that should be improved for the next prototype. The soccer simulator can be divided into five parts, GUI, simulation engine, players, habitats and activities. In the following sections a brief description of each of the five parts are given.

Graphical user interface

Improvements of the GUI would mainly concern user interaction, i.e. the user should have more possibilities to interact with the running simulation, like changing team formation at run time, substitute players and alter player stats. It would also be desirable to improve the visualisation of habitats and activities to bring more information to the user in a simple way.

Simulation engine

The simulation engine currently covers the SoccerSimulation class in the model component. To make the application more understandable to other developers this part of the soccer simulation might need some refactoring. Another approach for improving the simulation engine could be to re-design and re-implement the simulation engine. By using this approach the understandability could be increased, and a more maintainable simulation engine could be achieved. An important improvement for the simulation engine is to introduce more soccer rules and to make the simulation more realistic.

Players

Currently the player implementation is rather simple. In a future version the player model should be improved with more attributes like stamina, agility, defencive abilities, offencive abilities, etc. A more physical model of the players would be desirable too, so that instead of running at a constant velocity, the speed of the players would be compensated when they are accelerating or decelerating. Later on it might be beneficial to look into the multi agent paradigm to gain autonomy and cooperation between the players by using agent principles.

Habitats

The first and maybe simplest improvement of habitat implementation is to introduce support for polygon shaped habitats. The next improvements are far from simple and will require some consideration, even whether the improvements should be included in the prototype or not. The improvements include:

- Making the habitats able to influence the players behaviour.

- Allow the habitats to execute activities.
- Distinguish static and dynamically spawned habitats and give them different privileges.

Activities

An improvement for the activity concept could be to categorise different kinds of activities. Currently all the activities are mixed in a single layered inheritance structure. It might be a more clever approach to distinguish between the activities based on the number of players it takes to execute them. At this stage it is too early to make a conclusive statement on which approach to follow, because it requires more thorough investigation of the subject.

Independent of which approach is chosen, it is for certain that there should be introduced new and more complex activities. The introduction of new activities will hopefully also make the simulation more realistic. Examples on new activities could be: corner kick, free kick or some activity involving collaboration between team mates.

2.7 Summary

The main purpose of this phase of the project was to implement a basic soccer simulation platform suited for experimentations with the activity and habitat concepts, which have been achieved.

The graphical user interface has been kept very simple and understandable, as intended. The GUI supports simple visualisation of habitats. The habitats are visualised as 2D magenta coloured shapes, so the user is able to see whether a player is supposed to react on the ball's current position or not.

The application has been implemented with a client-server architecture, where the client is the GUI component and model component is the server.

The model component handles the entire simulation. The main class of the model component is called SoccerSimulation, and has the responsibility of initialising and executing the simulation and providing information to be read from the GUI.

In this prototype version 1 it is only the players that are allowed to execute activities. This is achieved by implementing the command pattern to ensure flexibility and extendability and to give the player a uniform way of executing the activities.

The model component also holds support for the habitat concept. In this prototype the habitat concept is simplified to be a limited area, which is mainly used to define the soccer field and to define the areas in which the players are allowed to kick the ball.

The working process of this prototype version 1 has been based on prototyping. This working process has been very suitable because of the flexibility it provides. The prototyping working process will also become handy later on when this prototype is to be extended with better support for the habitat and activity concepts. Other desirable extensions worth mentioning are: give the user more options to control the simulation at runtime, better models of the players i.e. stamina, agility, etc. All together it can be said that prototype version 1 works as it is supposed to, without any unexpected crashes and it can be used

as soccer simulation platform for experimenting with habitats and activities as desired.

Chapter 3

Application survey

This chapter describes an application survey that have been carried out to gain inspiration and ideas for improving the SoccerLab simulator. This survey includes two soccer applications namely, The Tao of Soccer and FIFA Football 2005. This application survey was motivated by the development of SoccerLab

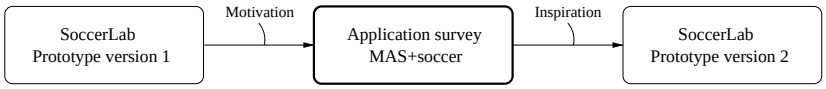


Figure 3.1: The application survey chapter in a larger perspective.

prototype version 1, as visualised in figure 3.1, and it has contributed with inspiration for SoccerLab prototype version 2, presented in chapter 5.

The main focus of the survey will be to identify collaboration strategies between agents or players in the two applications. Another aspect concerns special features in the application which not necessarily have to involve soccer phenomena, but could be of a more technical kind, to improve the simulation platform.

3.1 The Tao of Soccer

The Tao of Soccer (TOS) application is both a game and a simulator. TOS is a freeware application[1]. Because TOS is both a game and a simulation it can be considered as an application for both entertainment and educational purpose. TOS is a branch of the RoboCup AI soccer simulator[2]. TOS was developed, by Yu Zhang[3], to allow humans to play virtual soccer against an AI multi-agent soccer team, and to investigate different strategies for multi-agent cooperation.

The Tao of Soccer allows the user to implement his or her own virtual soccer team and set it up against other virtual soccer teams. Another possibility is to join a team and control a player with the keyboard. It could also be the case that all the players on both teams were controlled by humans which turns

[1] It can be downloaded at http://soccer.sourceforge.net/soccer/
[2] Further information can be found at: http://www.robocup.org/
[3] Yu Zhang's homepage: http://www.cs.ubc.ca/spider/yzhang/

the simulation into a multi player game instead of a simulation. At figure 3.2 a screen shot of the TOS GUI is depicted.

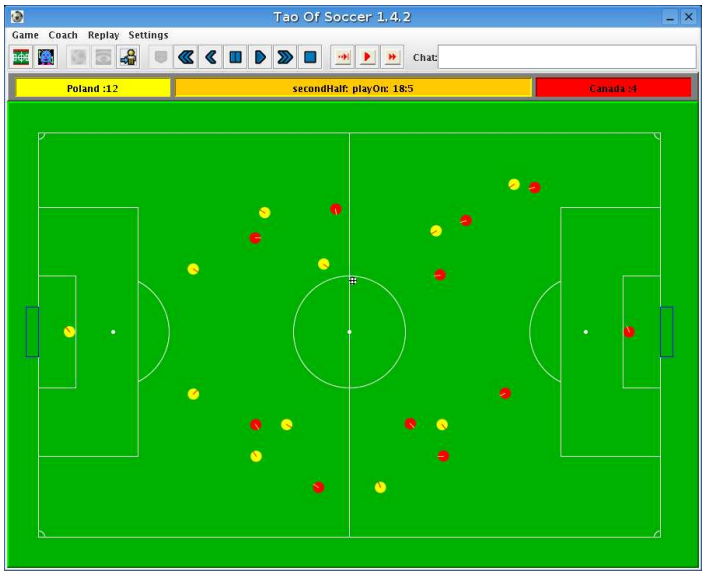


Figure 3.2: A screen shot of The Tao of Soccer.

At the end of this section the described features of The Tao of Soccer are discussed.

3.1.1 Collaboration strategies

There are no predefined player collaboration strategies in the tao of soccer, because it's the user's responsibility to implement the team and it's strategy which afterwards can be tested against another soccer team implementation. The simulator contains two simple AI teams for demonstration purposes. The AI for these two teams is so simple that no collaboration strategies can be found, because they are practically none existing. The most collaborative action for these two team is when one player passes the ball to another player. It is unfortunate that the simulator isn't provided with a more clever AI team, when the main focus of the application survey concerns collaboration strategies. The TOS application is well documented [Zhang, 2005], and it brings inspiration for improvements of the SoccerLab prototype version 1 platform.

3.1.2 Platform features

The tao of soccer has a lot of nice platform related features. The most interesting of those is briefly described in the following section.

Client-server architecture

The tao of soccer is implemented with a client-server architecture, as presented in figure 3.3. The server can be run both as a local server or a network server

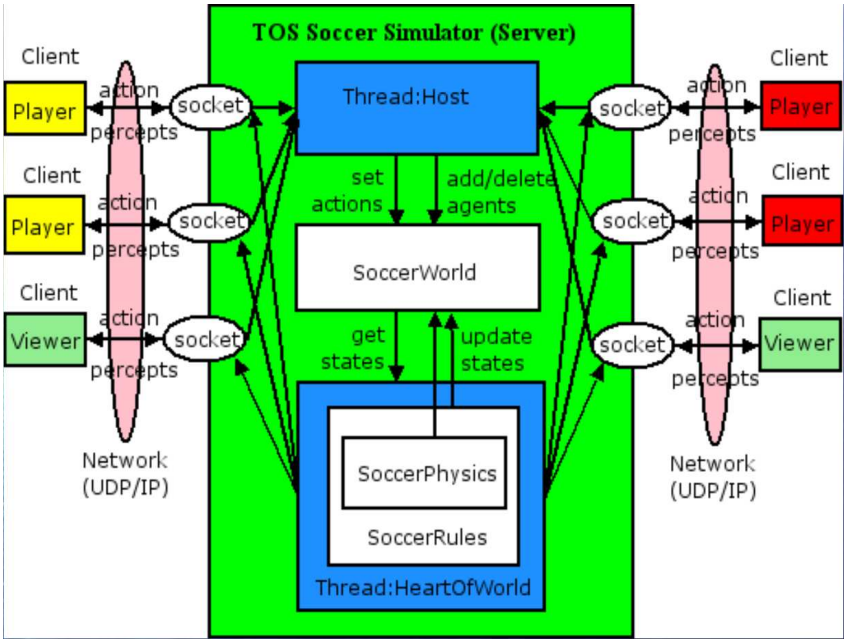


Figure 3.3: Diagram describing the architecture of The Tao of Soccer, as presented in [Zhang, 2005].

allowing the user to set up matches over the Internet. The server can be divided into three parts:

1. The host provides the basic server functionality, such as handling requests from clients when adding or removing players from a team.

2. The soccer world is the model of the world in which the simulation takes place.

3. The simulation engine, referred to as HeartOfWorld at figure 3.3, executes the simulation and keeps the soccer world updated when, for instance, the players have moved or kicked the ball. The simulation part is also in charge of sending soccer world state to the client(s) in order to update player position on the client side.

The three client modes

The GUI client for TOS is called SoccerMaster. This client can be run in three different modes: view mode, play mode and replay mode. Figure 3.3 only shows two different client modes, this is because there are no reasons to distinguish between user controlled and AI controlled agents from a server perspective.

- When the client is run in view or simulation mode, the user can log into and watch a simulated match between two AI teams. When the client is run in this mode it is very similar to the SoccerLab simulator just with some extra features.

- When the TOS client is in play mode the user is allowed to use the keyboard to control a soccer player on an AI team, and compete against another team, which can either be controlled by AI or another user.
- The last of the three modes in the TOS client is the replay mode, it can play back a match recorded in either play or view mode. When a match is recorded it is stored at the client side.

In general it is a great idea to be able to run the simulation in different modes depending on the user's intension of running the application. It might be too early to transform the current SoccerLab prototype into a user controlled game, but it is a feature that should be kept in mind. A multi purpose client or user interface for the SoccerLab simulator is a good candidate for an improvement that should be introduced in a future prototype version.

3.1.3 Special features

Besides the earlier mentioned features, a special feature in the TOS application is the possibility of hot switching between a 2D and a 3D visualisation of a simulation. The intension of the 3D view is only a matter of appearance and it doesn't influence the simulation in any way, (this conclusion is based on the fact the ball never leaves the ground and benefit from the third dimension). When the view is switched from 2D to 3D the player representation changes from small dots into small animated humans. Unfortunately the representation of the animated humans is very simple and thereby this feature doesn't contribute to the realism of the simulation, but it is still a good feature and a good alternative to the dot-representation of the players in the 2D view.

3.2 FIFA Football 2005

FIFA Football 2005[4], developed by EA SPORTS^{TM}, is currently one of the most popular soccer games in the world. FIFA 2005 is a commercial product and it's purpose is purely entertainment. Figure 3.4 shows a screen shot of the application simulating a match between Real Madrid CF and Liverpool FC. Because FIFA 2005 is a commercial application it is difficult to find any documentation, describing how the game has been implemented and how player collaboration is achieved. Due to that fact, the following sections is only based on the author's observations during matches between two computer controlled teams. I.e. for this application survey FIFA 2005 is used as a simulation tool, instead of a game with user interaction, as it is principally intended.

3.2.1 Collaboration strategies

At a first glimpse it can be difficult to get the full overview of both teams during a match because it isn't possible to choose a camera angle that covers the entire field. Instead the application holds at feature labelled radar, which gives a bird's eye view of the field showing how the players are positioned. A screen shot of the running application is depicted at figure 3.4, with the radar view is located in

[4] The official FIFA Football 2005 web page: http://www.easports.com/games/fifa2005/home.jsp

Figure 3.4: Screen shot of FIFA Football 2005 application.

the bottom of the picture. The collaboration strategies described in the following sections will all be based on observations of both the normal view, camera angle, and the radar view.

Tactic preferences

In FIFA 2005 the user has some team options to influence the team's collaboration strategy. The options will influence the team differently depending on whether the team is under user control or computer control. If the team is controlled by a user it would be more correct to say that the team options affect the team's positioning strategy instead of it's collaboration strategy, because as soon a player becomes in ball possession he is controlled by the user and thereby the user decides how the player should act. For this survey both teams are computer controlled so in this case the team options will affect the individual teams' collaboration strategies.

Figure 3.5 shows a screen shot of the team tactics screen in FIFA 2005. The tactics screen allows the user to modify the general team setup. For this survey the most interesting options provided by the tactics screen are Mentality, Attacking and Defence, because they all affect the team's collaboration strategy. The following gives a thorough description of each of the three options.

Mentality: The team mentality option influence the team's general field position and it's players' preferred passing direction. The team's mentality can be set to following three values:

- Defence – This option will make the players obtain a defencive field

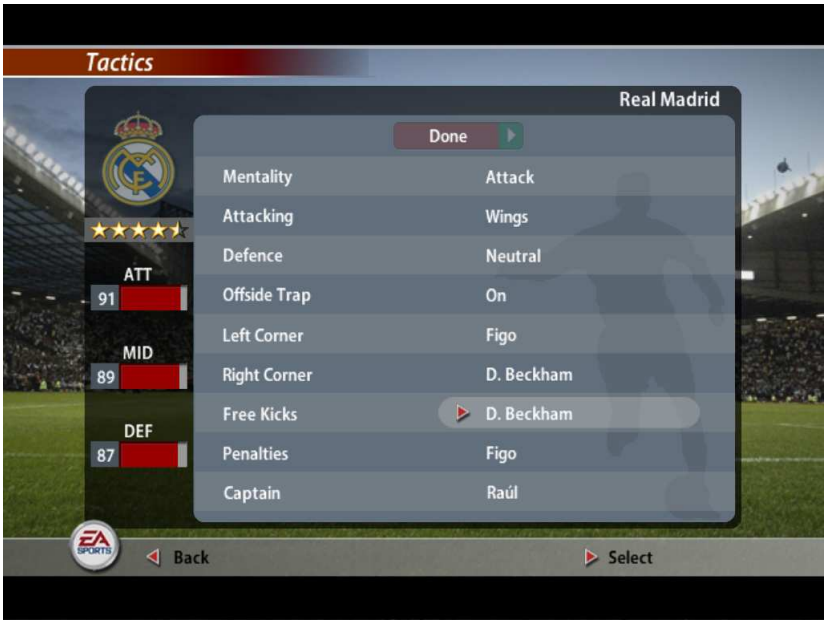


Figure 3.5: A screen shot of FIFA Football 2005 tactics screen.

position as well as the majority of their passes will be directed backwards.

- Neutral – The players are positioned neutrally in the field and they don't have a preferred passing direction. Instead the passing direction is determined by the specific situation. In most cases an equal amount of forward and backwards passes can be observed when the team mentality is set to neutral.
- Attack – The Attack option will put team in an offencive field position and the players rarely passes the ball backwards.

Attacking: This option gives the user the possibility of choosing the team's attack strategy when it is in ball possession. The three values are as follows:

- Long Pass – This strategy is also known as kick and rush. The basic principle is that defenders and mid-fielders will kick long balls to the attackers, to quickly setup an attack formation and thereby get an scoring opportunity.
- Wings – When choosing this attack strategy the players, mostly the mid-fielders, will play the ball over the wings and then make a cross to the players, usually attackers, in front of the goal.
- Possession – The last strategy is based on passing the ball to stay in ball possession as long as possible. One could argue that this strategy is more "safe" because as long as the team is in ball possession the opponents will not score.

Defence: This option applies to the team when it is not in ball possession. The following three values are available for the defencive strategy:

- Contain – This strategy will try to keep the defencive line intact by giving the mid-field players more defencive responsibility, i.e. the mid-field players have to support the defence to gain the player majority when the team is under attack. When the team attacks all the defence players will stay at the middle-line to avoid situations like, for instance, counter-attacks.
- Neutral – The neutral defencive strategy is more flexible than the contain strategy. For instance, the defencive players are allowed to act more individually like running a little forward to mark an opponent. This strategy also allows a defencive player to help the offence and mid-field players when the team is attacking.
- Pressing – The pressing strategy is obtained by letting the defencive wings mark an opponent each, while the mid-field players will try to tackle the opponent in ball possession. This leaves the centre defence observing and ready to take action if the mid-field players fail to stop the attack. When the team attacks only the centre defence will stay at the middle-line while the wings will participate in the attack.

Due to the fact that each of the three options, Mentality, Attacking and Defence, have three sub-options, the user has 27 different possibilities of choosing the team's collaboration strategy. Because FIFA 2005 is a game, the intension of the tree earlier mentioned options is to allow the user to define how the players should position them self when they are not under user control, instead of defining the team's collaboration strategy. For this application survey, as earlier mentioned, FIFA 2005 is used as a simulator and in this case the Mentality-, Defence- and Offence option will not only define player position but they will also affect the team's collaboration strategy, because of the "missing" user interaction, which is the reason for describing these three options as collaboration influencing options.

Guidelines for understanding the strategy outlines

Some of the collaboration strategies found in FIFA 2005 are sketched as outlines in the following two sections. In each of the outlines both a defencive team and a offencive team are situated. It should be emphasised that the notation used in all of the outlines is defined by the following six guidelines:

1. The offence is always attacking from the bottom to the top.
2. The large numbers prefixed by a "#" is the player number. The player number is used for distinguishing the player kinds on each team. The player number is associated to a player by using the following regulations:
   - The goalkeeper always has number 1.
   - Defencive players have numbers between 2 and 5.
   - The mid-field players' number range is from 6 to 9.
   - Offencive players have the numbers 10 and 11.
3. The purpose of the small numbers is to refer to a specific part of the outline.

4. The dot-and-dash line arrows, represents the path that the player will follow when he starts running. Dotted lines with arrowheads in both directions, indicates that the player is moving a little and not just looking passively at the ball.

5. The solid line arrows, represent the path of the ball when it is passed.

6. Each of the player are marked with an arrow, representing his current orientation.

Possessive collaboration strategies

This section will describe some of the possessive collaboration strategies in FIFA 2005, in this case possessive defines that the team is attacking or in ball possession. An outline for each of the four most interesting possessive collaboration strategies is presented at figure 3.6 and 3.7.

Pass the ball backwards is the first of the emphasised collaboration strategies. Figure 3.6(a) illustrates an example of this strategy, where the offence makes a temporary break in their attack to pass the ball backwards to a team mate and thereby stays in ball possession.

1. Player `#9` dribbles the ball up the field and when player `#3` approaches, player `#9` stops and turns around with the ball.

2. Player `#9` then passes the ball to his team mate player `#8`, who has positioned himself in a playable position behind player `#9`.

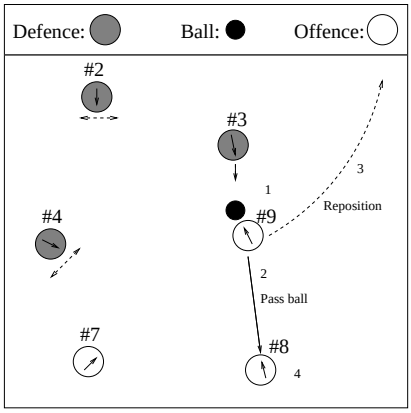
3. Player `#9` then re-positions himself, according to the shown path, to be playable again.

4. Player `#8` now have the choices of either playing `#7` or `#9` to continue the attack.

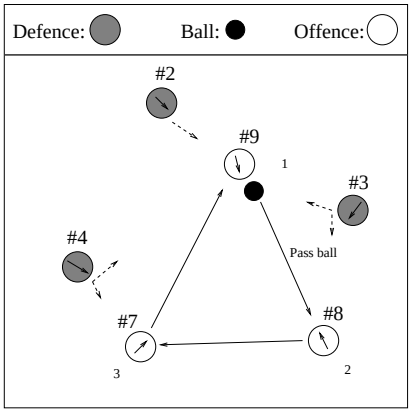
The attacking team in FIFA 2005 often uses this collaboration strategy, especially if the Attacking option, described in section 3.2.1 is set to Wings. In this case the idea of playing the ball backwards, like illustrated in figure 3.6(a), is also to tempt the defence, `#3`, to move further up the field and thereby creating additional space for the offence, `#9`, to establish an attack over one of the wings.

The triangular pattern collaboration strategy is emphasised as the second collaboration strategy, figure 3.6(b) depicts an outline for this strategy. At figure 3.6(b) the players are situated in almost the same way as at figure 3.6(a). The main difference of the two strategies is the offence's way of handling the situation. The following describes the strategy of passing the ball in triangular patterns:

1. Player `#9` is put under pressure by player `#2` and `#3`, so he passes the ball to team mate `#8`.



(a) Collaboration strategy 1: Pass the ball backwards.



(b) Collaboration strategy 2: Triangular pattern.

Figure 3.6: Outlines of the two identified possessive collaboration strategies concerning passing the ball.

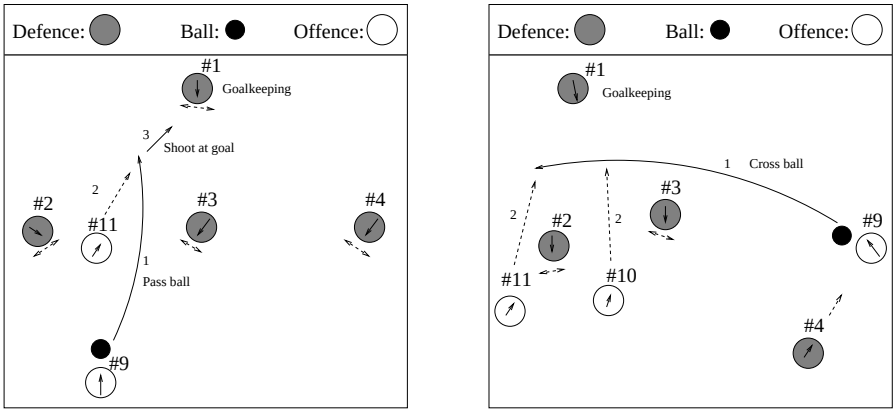
2. Player `#3` will start chasing the ball to maintain the pressure on the offence, as a result of that action player `#3` will semi-mark player `#9` at the same time. When player `#8` receives the ball he passes it to player `#7` due to the fact that player `#9` is semi-marked.

3. When player `#7` receives the ball, he will pass it on to player `#9`, to avoid a tackle from player `#4`. The ball has now been passed in a triangular pattern.

When the ball is back at player `#9`, the pattern can be repeated if the situation hasn't changed. For instance, before player `#7` passes the ball (at point 3.), player `#3` has three choices: he can continue chasing the ball, stay and try to mark player `#8` or retreat to his initial position to maintain the defencive line, where the latter two might be the most possible solutions. If player `#3` chooses to stay and mark player `#8`, it might create an opening for player `#9`, who then can continue the attack along the wing of the field. If player `#3` instead chooses to run to his initial position, the pattern is likely to be repeated.

Through-pass is the most frequently observed of the possessive collaboration strategies in FIFA Football 2005, particularly in the case where a team scores. To put it in other words, roughly 70% of the scored goals during this survey were conducted by using this strategy. The outline at figure 3.7(a) is described in the following enumeration:

1. Player `#9` is dribbling the ball up the field. Player `#9` "sees" the opening in the defence and passes a fast ball in front of player `#11`. Because FIFA football 2005 runs in 3D the pass can be either a fast pass at the ground or lob ball over the defence.



(a) Collaboration strategy 3: Pass the ball to score.

(b) Collaboration strategy 4: Cross the ball to score.

Figure 3.7: Outlines of the two identified possessive collaboration strategies concerning scoring.

2. Player `#11` runs to the ball. The defencive players, `#2` and `#3`, will also run to the ball to either kick it away or tackle player `#11` to prevent him from scoring.

3. If player `#11` gets the ball he will shoot at goal and try to score.

During the application survey it was observed, that the attacker in most cases will get the ball and if he chooses to shoot in the far side of the goal he is also likely to score. This observation indicates that the application holds some sort of patterns to guide the positioning and actions of the players when they collaborate. This strategy can also be observed in other parts of the field, but will of course not always result in a goal.

Cross the ball to score is the last of the emphasised collaboration strategies. An outline of this very classic situation is depicted at figure 3.7(b). The strategy can be explained as follows:

1. Player `#9` kicks a cross behind the defence to his team mates in front of the goal.

2. The defence, player `#2` and `#3`, will try to mark the offence, player `#10` and `#11`. As soon player `#9` has kicked the ball, player `#10` and `#11` will start running towards the ball to get it, and thereby getting the chance of scoring a goal.

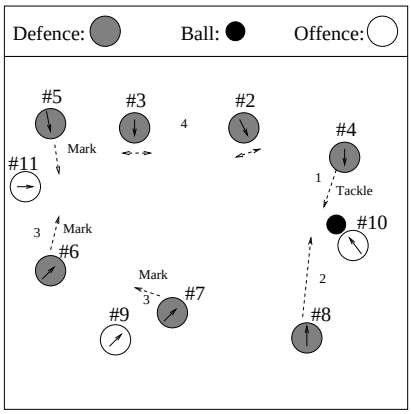
In the FIFA 2005 application the cross can be made both as a high ball or a low ball, due to the 3D environment.

Even though these four emphasised strategies are categorised as possessive, they also give a simple description of how the defence is organised in these

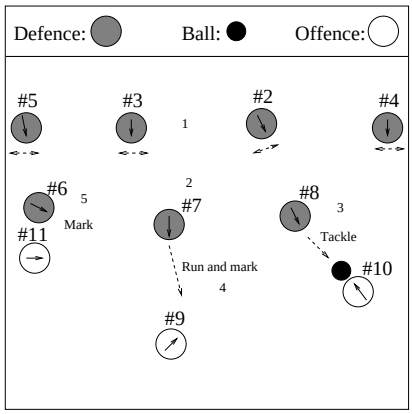
specific situations. The next section will give a more thorough description of the non-possessive collaboration strategies in FIFA 2005.

Non-possessive collaboration strategies

This section will describe two of the non-possessive collaboration strategies observed in FIFA 2005. A collaboration strategy is defined as non-possessive if the team is not in ball possession while it takes place. An outline of the two observed strategies is presented at figure 3.8.



(a) Collaboration strategy 5: Pressing defence.



(b) Collaboration strategy 6: Containing defence.

Figure 3.8: Outlines of the two identified non-possessive collaboration strategies concerning defending.

Pressing defence. The outline at figure 3.8(a) gives several examples of defencive collaboration strategies to mark the opponents. The following enumeration gives a more detailed description of the specific situations:

1. Player `#4` tries to tackle player `#10` to stop the attack.
2. Player `#8` runs back home towards player `#10` to support the defence in stopping the attack.
3. Player `#7` will try to mark player `#9`. At the same time player `#6` and player `#5` will mark player `#11`, to minimise player `#10`'s chance of making a successful pass to one of his team mates.
4. Player `#2` and player `#3` of the centre defence are constantly observing the play to see whether it is possible to intercept a loose pass from the offence's players.

The overall strategy for the defending team is in this case to put pressure on the offence. This is carried out by having the wings of the defence "attacking" or marking the opponents. The centre defence has a more passive role but they

are constantly aware of the play and they will try to intercept the ball whenever it is possible. To gain the player majority the mid-field players, of the defending team, have defencive duties too. This kind of defencive strategy can be observed in FIFA 2005 if the Defence option, described in section 3.2.1, is set to Pressing.

Containing defence Figure 3.8(b) presents an outline of another defencive collaboration strategy. A detailed description of the strategy is given in the following:

1. The defencive line has obtained it's defencive position. From this position the defencive players will wait and observe when to take action.
2. The mid-field players of the defence have obtained defencive positions too.
3. Player `#8` will try to put the player in ball possession, player `#10`, under pressure and hopefully tackle him.
4. Player `#7` will run to player `#9` to mark him.
5. Player `#6` is already marking player `#11` and he will continue doing that until the ball is kicked out of danger.

The intension of this collaboration strategy is to prevent the opponents from scoring by containing the defencive line while the mid-field players have the responsibility of conquering the ball. In FIFA 2005 this strategy can be achieved by setting the Defence option described in section 3.2.1 to Contain.

The reason for only depicting two non-possessive collaboration strategies is that they in general give a good description of the activities involved in defending. Of course defencive activities can take place in other part of the field, but this is just variations of those presented in the outlines of figure 3.8.

3.2.2 Special features

In addition to the collaboration strategies, various special features, that could be desirable for future version of SoccerLab, have been observed. The most interesting of the observed special features are described in appendix D.1.

A special feature that will be applicable, in the SoccerLab application is a feature similar to the player biography screen, described in D.1. The player biography screen gives a slight look into how the players maybe are handled in FIFA 2005 by presenting 24 values defining the individual players' skills. The idea of the players having different skills seems very useful, and should be introduced in a future version of SoccerLab. It will be too complex to introduce all 24 skills at once, instead a more realistic approach would be to choose five or six skills for the first approach. For further details of the player biography screen see appendix D.1.2.

3.3 Summary

Two soccer related applications have been investigated during this application survey, The Tao of Soccer and FIFA Football 2005.

The Tao of Soccer

Unfortunately the AI of the teams provided with The Tao of Soccer isn't very advanced so the survey of this application hasn't contributed any ideas of player collaboration strategies to the soccer simulator. Instead The Tao of Soccer has provided a lot of inspiration concerning the simulation platform such as; letting the simulator be run in three modes, simulation-, play- and replay mode. The approach of using a client-server architecture allows for different client implementations, and at the same time it is possible to run a simulation over the Internet.

The Tao of Soccer holds a special feature which is an experimental 3D visualisation of the simulation, however it might be more beneficial to introduce a 3D simulation model instead of a 3D visualisation of the simulation in the soccer simulator.

FIFA Football 2005

FIFA Football 2005 by EA SPORTS^{TM}, is a commercial soccer game. In this survey FIFA Football 2005 has been used as a simulator, to observe player collaboration strategies.Both offencive and defencive collaboration strategies have been identified in FIFA 2005. The application also allows the user to influence the players collaboration strategies at team level, through the use of the three value, Mentality, Attacking and Defence, in the team tactics screen. Each of the three values can be set to three different options, which result in 27 possibilities of defining the team strategy. Six collaboration strategies, four offencive and two defencive, have been emphasised in the survey of this application. Two of the four offencive collaboration strategies concern playing the ball and the other two mainly concern scoring goals.

Besides the collaboration strategies FIFA Football 2005 also has a special feature called the player biography which shows that each player's abilities are defined by 24 different skills. It is desirable to have a similar approach in the soccer simulator for defining the individual player skills, but it might be sufficient to have five or six different skills instead of 24. In general the FIFA Football 2005 application is a nice attempt to make a realistic soccer game, which also might be the reason for it's popularity.

The applications in general

Both of the investigated applications in this survey have contributed with inspiration for new features for the soccer simulator. The identified features that should be introduce in future version of SoccerLab, concerning both collaboration strategies and platform features, are summarised in the following list:

- Multi-mode SoccerLab client with support for:
  - View/simulation mode – standard simulation.
  - Play mode – allowing user controlled players in a simulated soccer team.
  - Replay mode – analysing tool.
- The six presented collaboration strategies.

- The ability to define the overall team strategy, similar to the approach found in FIFA 2005 using the options like; Mentality, Attacking and Defence.
- Player biography with approximately five skills.

Chapter 4

Analysis of concepts

This chapter presents an analysis of the concepts; agent, activity and context, based on literature studies. In appendix B summaries of the majority of the studied literature for this chapter is presented. This analysis was motivated by

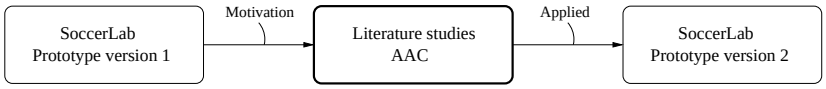


Figure 4.1: The analysis of concepts chapter in a larger perspective.

the development of SoccerLab prototype version 1 and the result of it is applied in SoccerLab prototype version 2, as visualised in figure 4.1.

The following summarises the organisation of this chapter: First an introduction including the motivation for this work is given. In the later sections the concepts agent, context and activity are discussed, basing on the knowledge that studied literature have brought to this project. Finally the result of this work is summarised.

4.1 Introduction

During the investigation the concepts; agent, habitat and activity, it was decided to rename the habitat concept to context. The reason for renaming the concept is simply because of the semantic of the word habitat. If we lookup the word habitat in a dictionary[1] it is defined as: *The area or environment where an organism or ecological community normally lives or occurs.* This is of course correct if we wanted to use the concept just as an abstraction over environment, but during the investigation it was disclosed that it would be beneficial to use the concept both as an abstraction over environment and grouping of agents, this is elaborated in section 4.5.1. If we lookup the word context in a dictionary, it is defined as: *The circumstances in which an event occurs; a setting*, which is in accordance with how we conceive of the (context)concept.

[1] The definitions of habitat and context are from www.dictionary.com

Besides renaming the habitat concept, during the investigation of the AAC concepts, we quickly realised that it was required to study other concepts too, in addition to three main concepts. This lead to the study of conceptual modelling, the role concept and organisation of agents.

4.1.1 Motivation

SoccerLab prototype version 1, described in chapter 2, only supports immature ad hoc defined versions of the AAC concepts. Among others, the activity concept supported in prototype 1 is an abstraction over individual actions and not an abstraction over collaboration as intended. Also the contexts(prior to this known as habitats) are simplified to being circular or rectangular areas, and last but not least the agents in the first SoccerLab prototype could be refined as well.

Our motivation for this work is to investigate concepts that are abstractions over collaboration and environment to be able to characterise the concepts activity and context. Also, literature concerning MAS theory will be studied to make us able to propose a suitable agent architecture.

4.2 Conceptual modelling

Various perspectives on programming exist, and all of them reflect the conceptual framework on which they are based. Conceptual modelling uses concepts of phenomena, concept and mappings of abstraction such as specialisation and aggregation.

BETA [Madsen et al., 1993] is an example of an object-oriented programming language from the Scandinavian school of object-orientation. The BETA programming language is provided with a conceptual framework which is a means to be used when modelling phenomena and concepts from the real world.

A corner stone in the conceptual framework of BETA is that software development, analysis, design and implementation, is considered as modelling activities at different abstraction levels.

4.2.1 Physical modelling

In [Madsen et al., 1993] the perspective on object-oriented programming is defined as follows:

> *A program execution is regarded as a physical model simulating the behaviour of either a real or imaginary part of the world.*

The definition of the object-oriented programming implies that an executing program is composed of some physical material. In this case the material is objects which are considered as computerised material.

Referent system and model system

The conceptual framework of BETA distinguish between; the real or imaginary part of the world being modelled called the *referent system* and the program execution or physical model which is called the *model system*. Figure 4.2 presents the programming process that involves identification of concepts and phenomena

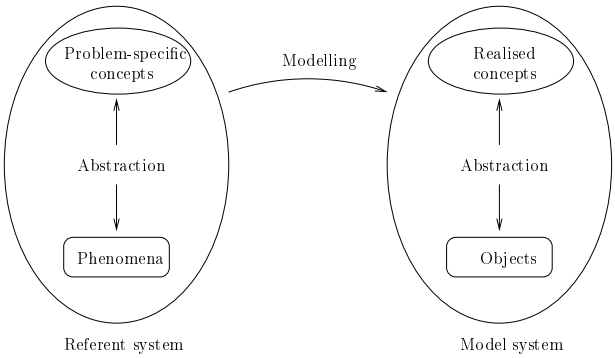


Figure 4.2: Referent system and model system.

in the referent system and representation of these in model system.
In the conceptual framework of BETA phenomenon and concept are defined as follows:

A phenomenon is a thing that has definite, individual existence in reality or in the mind; anything real in itself.

A concept is a generalised idea of a collection of phenomena, based on knowledge of common properties of instances in the collection.

The abstraction process relates phenomena and concepts and it occurs in both the referent system and the model system.

Abstraction process

The abstraction process in the conceptual framework of BETA is described as the process of producing and using knowledge and it consists of:

1. Identification of phenomena and their properties.

2. Classification of phenomena into concepts.

3. Generalisation and composition of concepts.

Figure 4.3, as depicted in [Kristensen, 2003], presents the basic elements of abstraction; classification, generalisation and aggregation, which can be seen as mappings between concepts and phenomena.
Phenomena possessing similar properties can be classified into concepts. These concepts can be either specialised/generalised or decomposed/aggregated and a concept can be exemplified by any phenomenon in its extension.

As part of this work we have experimented with conceptual modelling of activities and contexts, by utilising the general abstraction model, figure 4.3, in a soccer scenario. The result of the experiment is presented in appendix C.

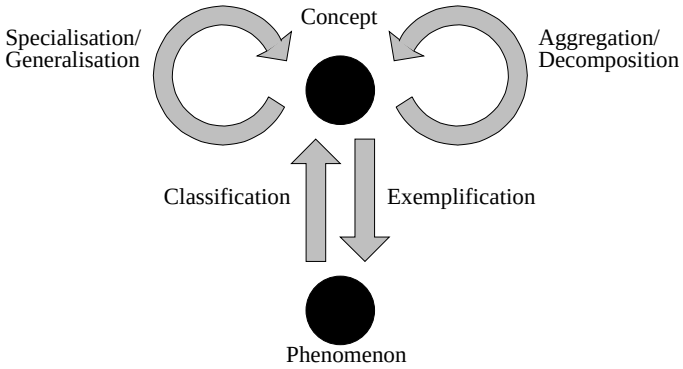


Figure 4.3: General abstraction model.

4.3 The Agent Concept

We have chosen the agent concept for modelling soccer players. A soccer player is seen as an abstraction over a set of phenomena in the referent system.

The following sections describe how we understand this concept and what internal structure we have chosen for its representation. Our understanding of agents is based on the literature studies of both agent-oriented and object-oriented research areas.

4.3.1 Description of the agent concept

The notion of agency was formalised by Wooldridge and Jennings [Wooldridge and Jennings, 1995] which is based on extensive review of research done in this area. Their conclusion is that the term *agent* is used to denote a hardware or (more often) software-based computer system that enjoys following properties:

Autonomy - agents operate without intervention of human or others, and have some kind of control over their actions and internal state.

Social ability - agents interact with other agents and possibly humans.

Reactivity - agents perceive their environment, and respond in a timely fashion to changes that occur.

Pro-activeness - agents do not simply act in response to their environment, they are able exhibit goal-directed behaviour by taking the initiative.

Some researchers go even further and conceive agents to be either conceptualised or implemented using concepts that are more usually applied to humans, like *mentalistic notions*, such as knowledge, belief, desire, intention or obligation [Wooldridge and Jennings, 1995], [Wooldridge et al., 1999]. In this manner, agents are seen as an *increase of abstraction* compared to objects [Wooldridge et al., 1999], [Parunak, 1997], where agents are given its own thread of control and its own internal goals and no longer need external invocation.

Agent Architecture

Agents should organise themselves and adapt dynamically to changing circumstances without top-down control. Therefore some researchers propose complex agents to emulate human intelligence, reintroducing many of the problems of complex system design and implementation, that motivated the increase of localisation and introduction of agent concept.

We agree with these researchers, who, inspired by world of nature, propose *thin agents* instead, and claim that even simple behaviour of the individual agents can generate significantly complex behaviour of the system.

One of the requirements of an autonomous agent is the ability to perform *means-end reasoning*[2], which has been a major research issue in artificial intelligence. A large majority of the MAS community uses a beliefs, desires and intentions[3] agent model, which we have chosen to focus on.

BDI architecture

We adopt beliefs, desires and intentions agent model as described by the conceptual framework proposed by [Rao and Georgeff, 1995]. This approach enables agents to break down ultimate goals, so they can commit themselves to achieve a subset of their entire goal universe. An agent still has to perform means-end reasoning and determine the plan to fulfil the selected goals, as stated in [Meneguzzi et al., 2004]. After Rao and Georgeff we adopt the meaning of the three mental states in the BDI agent model:

Beliefs: The agents knowledge about the world, which is highly subjective. Beliefs can be divided into environmental, social, relational and personal.

Desires: Objectives to be accomplished, which do not have to be consistent with each other and do not have to be believed to be achievable.

Intensions: Plans currently under execution.

As seen in Figure 4.4, the BDI kernel occupies a central position and its execution is as follows:

- The BDI-kernel observes desires, that could be affected by the environment or by the agent itself.
- A desire is selected.
- The most suitable plan, according to the agent's beliefs, is selected.
- The selected plan is placed in the intention structure.
- The plan is executed.
- The execution of the intention may change the environment, which can establish new goals and influence beliefs.

[2]Means-end reasoning bases on selecting such courses of actions that ultimately achieve goals of the agent.

[3]We also use abbreviation BDI for Beliefs, Desires and Intentions throughout the report

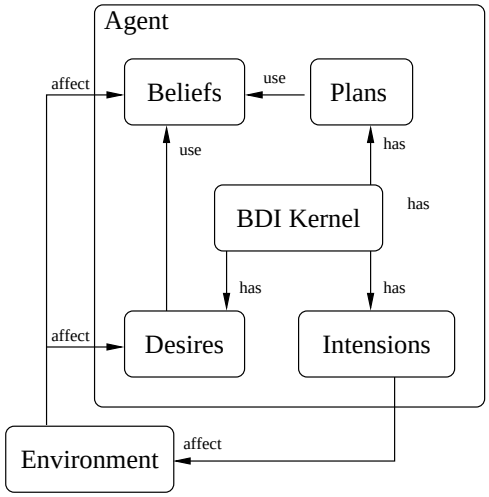


Figure 4.4: Beliefs Desires Intentions model.

If an agent do not commit itself to participate in an activity, its desires can contain a mixture of single-agent oriented goals and roles proposals made by activities through contexts. The BDI kernel selects a desire, where collaboration-oriented ones will be more likely chosen. Lets assume that a specific role was selected, the role concept is presented in section 4.3.2. Then capabilities of the agent are checked, and a specific plan for taking on the role is selected, placed in intentions and executed. Now, the activity influences the agent's desires, taking the control over the agent. The desire for finishing once chosen activity is higher than for changing it into another.

Selected extensions. Inspired by studied literature, we would like to propose several extensions to the original BDI model. First of all we would like to present the concept of *plan*, which is a recipe representing procedural knowledge of an agent. Plans are specifications for achieving certain goals or doing tasks on the way to achieve goals and the conditions under which the plan is applicable [Padgham and Lambrix, 2000]. The main advantage of introducing the plans structure is to reduce complexity of creating complete plans at runtime.

We propose to include the *degrees of belief* as an extension to the BDI model. Rao's and Georgeff's model explicitly acknowledges that an agent's knowledge about the world is incomplete. By using beliefs, it does not attempt to introduce information about how likely agent's imagination is to be the actual world in which an agent operates. We introduce a new property of belief modelling, describing how certain an agent is of its belief, similarly to [Parsons and Giorgini, 1999].

Rejected extension proposals. The research on MAS covers a variety of areas. Throughout the years there have been many attempts to extend BDI theory with new concepts. Padgham and Lambrix in [Padgham and Lambrix, 2000] propose to include *capabilities*, which are used to make advantage in choosing

plans that could be used to fulfil given desire. Capabilities cluster plans into groups, where each group of plans can fulfil specific goal. The motivation of capabilities structure is in contradiction with our model, where the capabilities of the agent will be specified by a basic set of plans for simple actions and roles that the agent is currently playing. Usage of roles allows agents to appear as they have different capabilities, depending on the perspective from which it is accessed.

As we mentioned above, degrees of beliefs are included into our model. We do not consider to include various kinds of intentions and desires as introduced in [Meneguzzi et al., 2004]. Furthermore, we hope to archive collaboration through use of activities and we do not consider to introduce plans exchange as a means for reaching cooperativity, as proposed in [Ancona and Mascardi, 2003].

4.3.2 The role concept

As mentioned previously, agents take on roles. The understanding of roles in agent-oriented analysis and design is similar to the view considered in object-oriented analysis, except that phenomena that take on roles are called agents, not objects or tangible objects (from [May et al., 2001]). In the following section we describe the concept of role.

Roles

In MAS the role is seen as a basic concept that builds the structure of organisation of agents. It is an abstract representation of an agent function, service or identification within a group [Ferber and Gutknecht, 1998] [Wooldridge et al., 1999]. Roles and interactions between them are typically expected to be stable and change slowly. The role is connected with obligations and responsibilities, but might also have requirements about specific abilities needed to take on role and may provide new abilities.

The motivation for roles, as seen in [Kristensen, 1995] , is to allow special perspectives on a phenomenon. This perspective is used by other objects in the model as a selective way of knowing and accessing the object. The perspective is a set of properties of the phenomenon modelled by a set of methods.

As seen in [Kristensen and Osterbye, 1996] the role introduces new properties into an object, properties that are present only because of the perspective. They cannot modify properties of an object, but only extend it with new properties. Role properties must be accepted by the object and recognisable by other objects.

Notion of roles seems to be similar in both object-oriented and agent-oriented perspectives, the most important properties of roles include:

- An entity can play different roles simultaneously which supports modelling of changing perspectives.
- An entity can change its role.
- An entity can play the same role several times simultaneously, meaning that an entity might be acting different specialisations of the same role at a time
- A single role can be played by different entities.

Roles is treated as a language construct [Kristensen, 1995] , and, more specifically, as a specialisation of concept [Kristensen and Osterbye, 1996] . Roleification is seen both as a kind of specialisation and classification. It differs however from specialisation, because no property of an object can be modified by the role and the class of an object is not a natural generalisation of the role, even if the role may seem as a natural specialisation of that class. It differs from classification, because it is dynamic, an object can be dynamically reclassified into new roles. The phenomenon will belong to the role extension only for some part of its life.

The availability of methods of the subject (object with its current roles) depends on classification of the subject - if the subject is only known by the object, then only methods of the object are known; if the subject is known by its role, then both the methods of the object and of the role are known/available. Methods of the role can utilise methods of the object in their definitions, but not vice versa. Other objects can access an entity through its roles to obtain different interfaces and behaviours from the object, which supports subjective behaviour, as suggested in [Kristensen, 2001] .

We acknowledge and adopt the role concept as it is described in this section.

4.4 The activity concept

To be able to describe and prescribe collaboration between agents, we have decided to use the activity concept.

The following sections describe how we understand the activity concept and how it can be used to describe and prescribe collaboration between agents, both at a conceptual level and at a language level. Our understanding of the activity concept is founded upon literature studies and practical experience gained during the development of first prototype of the SoccerLab simulator, see chapter 2.

4.4.1 Characterisation of the activity concept

The studied literature concerning the activity concept encompass five articles. In all the studied articles an activity is described by a class that can be instantiated as objects with state, behaviour and identity. The activity concept has evolved over time, and the following gives a brief overview of the most important steps in the evolution of the concept.

Inspiration from the studied literature

The activity concept emerged as *transverse activities* in [Kristensen, 1993a] , which is one of the first approaches to model collaboration from a conceptual perspective.

A transverse activity is presented as a means to model non object-centric actions between several objects which is not part of the existing methodologies for object-oriented analysis, design and implementation. A transverse activity must at least include a listing of participant and a sequence of action forming the combined directive of the activity. We acknowledge that activities have knowledge of their participants, but we think of the participants as a logical

context, see section 4.5 for further elaboration of logical contexts, instead of the more general term listing.

In [Kristensen and May, 1996] the concept of transverse activities was renamed to *activities* and the concept is seen as a modelling and language mechanism that may be used to create abstractions relating objects together and describing their interplay. Another addition to the activity concept is that the activity communicates with the participants through *roles*.

The addition to the activity concept of communicating with participants through roles, makes the activity concept even more flexible and introduces the possibility to describe and prescribe collaboration between even more abstract entities than objects, in our case agents.

The concept of *association* introduced in [May et al., 2001] models interaction of artifacts. Tangible objects and habitats (also introduced in this article) are the only artifacts considered. Associations make it possible for tangible objects and habitats to interact in different ways at different times where this interaction is neither fixed nor predetermined by the properties of these two artifacts.

We don't consider an activity as an artifact for modelling collaboration between agents and contexts, our understanding of the relation between activities and contexts is described later in section 4.5. Besides the relation between a habitat and its inhabitants, the notion of associations presented in the article is very similar to our notion of activities.

The association concept is further elaborated in [Kristensen, 2003], where it is seen as abstraction over collaboration. There are a lot of similarities between associations and activities, as presented in [Kristensen and May, 1996] , i.e. the participants are qualified by roles. Additionally, a new powerful aspect is introduced in the association concept which is an event-based mechanism that can be used for asynchronous broadcast of information to potential receivers among the association's participants. The event-based communication, is an attempt to achieve interleaved execution using existing language constructs.

We recognise this event-based communication between participants of the activity, in our case this can be achieved by utilising the logical context of the activity which contain all the participants of the activity.

The latest attempt to define an association is given in [Kristensen, 2005], where the association concept from [Kristensen, 2003] is further refined. The focus in the article is put on explaining how the collaboration can be explicitly described through the directive of the association.
The following summarises the characteristics of the activity concept:

- Activities are abstractions over collaboration.
- An activity consist of a set of participants, a logical context, qualified by roles and a directive.
- An activity only communicates with its participants through roles.
- The directive of an activity is a sequence of actions, describing or prescribing the collaboration between its participants.

- An activity is described by a class that can be instantiated as objects with state, behaviour and identity.

Additionally, we allow activities with only one participant, even though it could be questioned whether it is collaboration or not when only one participant is required.

We allow activities to prescribe collaboration between different kinds of entities as long they fulfil the required roles. When nothing else is mentioned we think of the participants as agents.

In all the studied articles the notion of activities builds on the same conceptual understanding. Appendix C.1, describes our experiments with conceptual modelling of activities in a soccer scenario.

4.5 The context concept

To be able to model environments in which agents act and engage in activities, we have decided to refine and use the habitat concept introduced in [May et al., 2001]. We have also decided to rename the concept from habitat to *context*, as explained in section 4.1.

4.5.1 Characterisation of the context concept

The concept of context is an abstraction over environment in which agents can move around and collaborate with other agents. In [May et al., 2001], a habitat is defined as a logical context in which tangible objects interact and exist. This is similar to our understanding of contexts if, of course, tangible objects are replaced by agents.

The concept of habitat is also an abstraction over environment where the concept captures the three spaces physical, informational and conceptual. The physical space is spatial and limited by physical boundaries. Informational space stores information and may possess computational abilities too. The conceptual space may influence its inhabitants to act in an appropriate manner at a given time, the conceptual boundaries are delineated to what is appropriate where and when. The interplay between these three spaces concerning habitats is further elaborated in [Andersen and Nowack, 2002].

The three spaces covered by the habitat concept also indicates that the context concept should be considered from different perspectives.

Physical contexts vs. logical contexts

Another property of habitats is that they are aware of their inhabitants, which is implied by the following quote, from [May et al., 2001], which exemplifies the usage of habitats:

> ***Scenario: The Living Room***... *The context in this case is The Living Room... The Living Room also has tangible qualities: it is aware of the tangible objects that are situated within and can interact with them.*

We acknowledge and adopt the idea of inhabitant awareness of habitats, so that contexts are aware of agents present at any time.

Similar to [Pedersen and Kristensen, 2002] we think of both physical and logical contexts as temporal, they may emerge or disappear, but physical contexts are spatial and delimited whereas the logical contexts are dimensionless and ubiquitous. Another dissimilarity between physical and logical contexts is their individual purposes in a modelling perspective, where physical contexts model location, logical contexts model grouping of agents. E.g. a soccer team can be considered as a logical context, and hence the boundaries of a logical context are identical to those of the conceptual space in [May et al., 2001].

It should be considered whether logical context should be renamed to social context, inspired by [May and Kristensen, 2004], because of its purpose of grouping agents, but for now we will settle with logical context.

Besides the earlier mentioned differences physical and logical contexts offer the same functionality to agents which indicates that both of them might be specialisations of a general context concept.

The context concept in general

In general we think of contexts as described by classes that can be instantiated as objects with state, behaviour and identity. Contexts may overlap and they are not isomorphic [May et al., 2001].

In [May and Kristensen, 2004] a more elaborated and detailed description of the understanding of the habitat concept is given:

- Specifying some kind of locality, an area that is delimited by some kind of boundary.
- Comprising inhabitants.
- Providing support to its inhabitants in the form of opportunities and services that allow its inhabitants to interact and achieve their various goals.

This is almost analogous to how we conceive of contexts, if *comprising inhabitants* doesn't mean that a context always contain at least one agent. It is not a necessity that a context is inhabited at all times. E.g. in a soccer field it isn't a requirement that there are players in, for instance, the goal area all the time (assuming that the goal keeper has run up the field).

Another definition of the notion of habitat is presented in [Andersen and Nowack, 2004], and is as follows:

> *A habitat is a container. Inhabitants can move into a habitat, live there for a while, and move out again. The action possibilities (and as a consequence the information needs) of the inhabitants depend upon the habitat they currently inhabit.*

This definition is in full agreement with our notion of context. Especially that inhabitants' action possibilities are dependent upon the habitat they inhabit. A context provides activities, in which agents may engage, to support the actions of the agents present in that specific context.
The following list summarises the characteristics of the context concept:

- Physical contexts are abstractions over environments, that are spatial and delimited.

- Logical contexts are abstractions over grouping of agents, and they are dimensionless and ubiquitous.
- A context is aware of the agents it comprises, and influence the agents by providing activities.
- Contexts may overlap and are not isomorphic.
- A context is described by a class that can be instantiated as objects with state, behaviour and identity.

Appendix C.1, describes our experiments with conceptual modelling of contexts in a soccer scenario.

4.6 Organisation of agents

Multi-agent systems capture the agents flexible, autonomous and problem solving behaviour, variety of interactions and organisational structures into one entity [Jennings and Wooldridge, 1995], [Parunak, 1997]. The MAS approach is inspired by biology and therefore it is suggested that agents should be able to form populations that organise themselves and adapt dynamically to changing circumstances without external control.

4.6.1 Approaches for organising agents

In the early MAS research, it was suggested that both complex structures and complex behaviours could emerge naturally in agent societies as a result of actions of simple agents collaborating in the similar way that we see in nature.

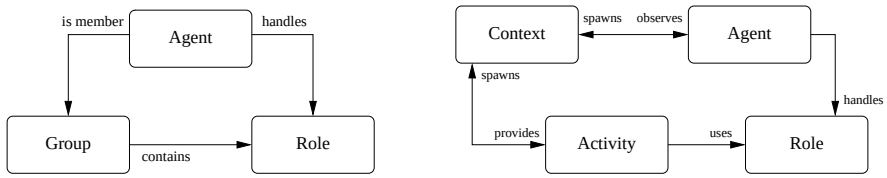
Ferber states in [Ferber and Gutknecht, 1998] that only very few works have attempted to develop models of MAS using organisational approaches, and practically no effort has been made to incorporate organisational terms and concepts into MAS, even though organisation has been treated and presented as a major issue of these systems. It also suggests that this lack of interest in organisations comes from the general trend that defines MAS as mere aggregations of agents interacting together, where agents are treated as autonomous entities that behaves individually, almost selfishly.

Even though research in MAS is widespread certain characteristic features that describe most of the systems of agents can be defined. Inspired by [Zambonelli and Parunak, 2002] and [Jennings and Wooldridge, 1995], we selected the following to be essential to understand and design the system of agents for the purpose of this project:

Situatedness: Agents execute in the context of an environment, perceive it and can both influence and be influenced by it.

Locality in control: Agents represent autonomous loci of control, although there is a need of coordinating activities with other agents.

Locality of interactions: Agents interact with each other, and the system should be modelled as clusters of locally interacting agents, where inter-cluster interactions should be modelled explicitly.



(a) The AALAADIN model.

(b) Agent-Activity-Context-Role model.

Figure 4.5: The AACR-model inspired by the AALAADIN-model.

We propose AACR-model, figure 4.5(b), which is highly inspired by and a modified version of the AALAADIN-model, figure 4.5(a) [Ferber and Gutknecht, 1998]. We think of activities as central component capturing roles interactions. Agents engage in activities by taking on roles and we consider agents to be capable of handling several roles simultaneously, if these roles require only exclusive sets of resources. When agents take on roles, they acquire new abilities, that they would not have obtained otherwise.

Activities control agents through roles and they are not aware who exactly is playing the role at a specific point of time. Agents are grouped basing both on their physical and logical context. Logical contexts can be formed for agents engaged in the same activity. Agents can inhabit several contexts and they can communicate with other agents only through common context. Contexts also communicate with agents providing various information. This if further elaborated in section 4.5.

Activities, roles provided by them and corresponding logical contexts are very temporal in nature, emerge and disappear dynamically.

Agents are treated as autonomous in that sense that the ultimate decision about the choice of activity is made independently by an agent. After they commit themselves, however, to perform specific role in chosen activity, activity influences their desires, and indirectly influences their choices of actions. As stated in [Parunak, 1997] it is important property of an effective organisation to limit information processing, which is done by use of contexts.

4.7 Summary

The purpose of this work was to investigate various concept to be able to propose an agent architecture and to characterise the two concepts activity and context, which have been accomplished.

Proposal of agent architecture

We have proposed a BDI[4]-architecture for the agents. The BDI-architecture is a widely used agent architecture that allows the agents to break down ultimate goals, so they can commit themselves to achieve a subset of their entire goal universe.

[4] Beliefs,Desires,Intentions

For organisation of the agents we have proposed a modified version of the AALAADIN-model, that supports the concepts of activity and context. The reason for using this approach is that the AALAADIN-model doesn't impose any restrictions on the agent architecture and at the same time provides support for the role concept, which is required by the activity concept.

Characteristics of the activity concept

Based on literature studies and experience gained during development of SoccerLab prototype version 1, we have characterised the activity concept as follows:

- Activities are abstractions over collaboration.
- An activity consist of a set of participants, a logical context, qualified by roles and a directive.
- An activity only communicates with its participants through roles.
- The directive of an activity is a sequence of actions, describing or prescribing the collaboration between its participants.
- An activity is described by a class that can be instantiated as objects with state, behaviour and identity.

We have chosen this flexible characterisation of the activity concept to allow for experimentations, although it later on might become a candidate for a definition of the activity concept.

Characteristics of the context concept

Similar to the activity concept, we have made characteristic of the context concept. We have characterised the context concept as follows:

- Physical contexts are abstractions over environments, that are spatial and delimited.
- Logical contexts are abstractions over grouping of agents, and they are dimensionless and ubiquitous.
- A context is aware of the agents it comprises, and influence the agents by providing activities.
- Contexts may overlap and are not isomorphic.
- A context is described by a class that can be instantiated as objects with state, behaviour and identity.

Additionally, we have conducted two experiments of conceptual modelling of activities and contexts in a soccer scenario. These results are presented in appendix C.

Chapter 5

Prototype version 2

This chapter documents the development of SoccerLab prototype version 2. The purpose of this prototype is to introduce support for the AAC concepts. By utilising the AAC concepts, new collaboration strategies are introduced in the soccer simulator. The development of this prototype is based on the literature

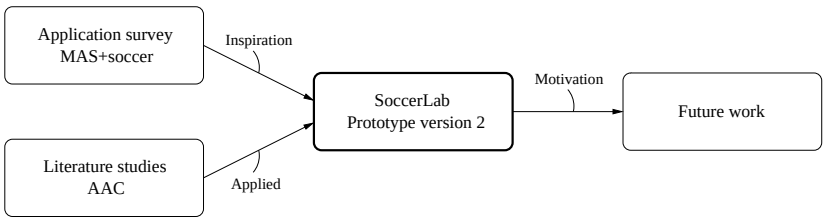


Figure 5.1: The prototype version 2 chapter in a larger perspective.

studies in chapter 4, the inspiration gained from application survey in chapter 3, as visualised in figure 5.1.

5.1 Motivation

The motivation for developing the SoccerLab prototype version 2 is to evolve the first prototype of the soccer simulator to support the AAC concepts defined in chapter 4. Besides supporting the AAC concepts it is desirable to introduce more user interaction in the application to illustrate how the concepts are used in the soccer simulator.

The development of this prototype is also motivated by the application survey, described in chapter 3, which suggest a list of both platform features and collaboration strategies that would be desirable to have in a soccer simulation application like SoccerLab.

5.2 Application description

The graphical user interface of the application has been modified in the SoccerLab prototype version 2. A screen shot of the executing application is presented

at figure 5.2. The following enumeration describes the corresponding parts of

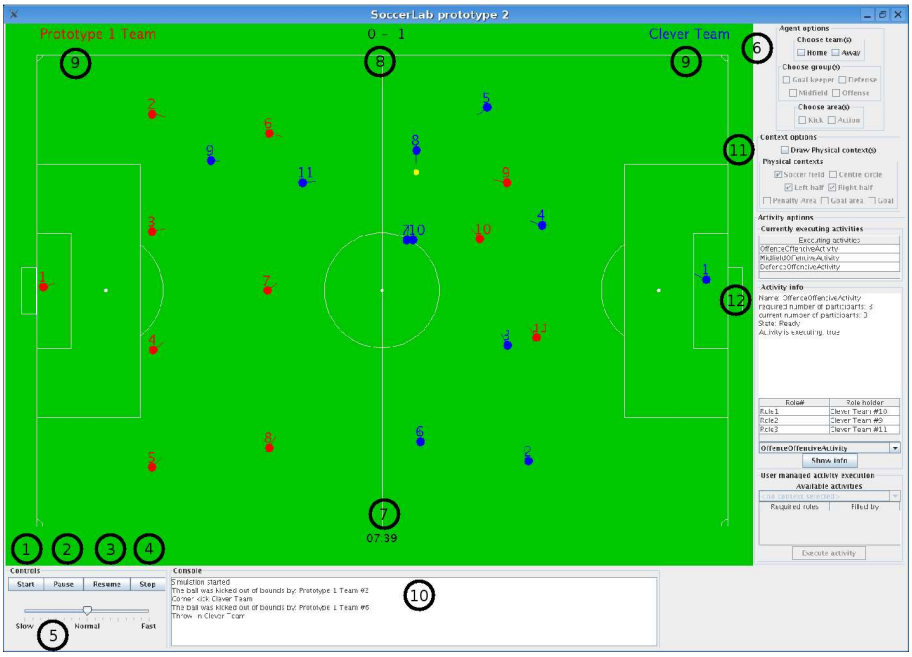


Figure 5.2: Screen shot of SoccerLab prototype version 2.

the SoccerLab GUI presented at figure 5.2:

1.-5. Have the same functionality as in the first prototype, described in section 2.2 page 22, they have just been slightly repositioned.

6. The agent option panel which is similar to the habitat panel in prototype 1, figure 2.3 page 22, has just been made a permanent part of the graphical user interface for SoccerLab prototype version 2. A more detailed description of this part of the GUI is given in section 5.3.2.

7.-9 Also have the same functionality as in prototype 1; game clock, score and team names, respectively.

10. Console for presenting text feedback to the user.

11. The context option panel, which can enable drawing of contexts. A more detailed description of the context option panel is given in section 5.3.2.

12. The activity option panel including both the activity info panel and the activity execution panel. A more through description of this part of the GUI is presented in section 5.3.2.

To tests the new version of the SoccerLab simulator the left side team, prototype 1 team, is using the implementation from prototype 1 whereas the right side team, clever team is using the new prototype 2 implementation of activities and contexts. By simulating a match between these two teams, the final score of the match hopefully will give an indication of whether the second prototype is better than the first prototype or not.

5.3 Architecture

In the first prototype, chapter 2, it was decided to use a client-server architecture. This decision was substantiated by the result of the application survey, chapter 3, which also brought inspiration for making a multi-mode client for the SoccerLab prototype version 2, allowing for instance, normal simulation of soccer matches and replay of already simulated matches. This SoccerLab prototype doesn't support the multi-mode client feature, but uses a single-mode client similar to the solution of SoccerLab prototype version 1, where the graphical user interface is considered to be the client and the model component is the server.

The following sections give a more detailed description of the model component and GUI component of the SoccerLab prototype 2 application.

5.3.1 The Model component

Figure 5.3 depicts the general class diagram of the model component of SoccerLab prototype version 2. The model component consist of both modified and non-modified classes from prototype version 1, and new classes have been introduced to support the AAC concepts defined in chapter 4.

The following sections describe in details the segments of the class diagram concerning agents, activities, contexts and which modification that have been made to the existing classes from prototype version 1.

Agent

Due to the limited time available for developing this second SoccerLab prototype, it was decided to keep the agent architecture applied in SoccerLab prototype version 1, instead of improving the agents by introducing a BDI-architecture as suggested in chapter 4. The reason for choosing this limitation is that several examples on how to implement the BDI-architecture is described in various literature and thereby doesn't contribute with new knowledge to this project. E.g. [Appel and Nielsen, 2005] describes such an implementation where the BDI-architecture is combined with the AGR-model. Figure 5.4 present the segment of the class diagram of the SoccerLab prototype version 2 concerning agents and roles.

Because the left team, Prototype 1 Team, consist of players with the prototype 1 implementation the classes DefencivePlayer, MidfieldPlayer and OffencivePlayer are kept in the Player (agent) inheritance structure. Both teams use the goalkeeper implementation, GoalKeeper which extends the super-class Player. The class NewStylePlayer, which is also a sub-class of the abstract Player class, has the responsibility to handle the prototype version 2 agents with roles and how they engage in activities.

The new agents can participate in activities if they can play one of the roles required by an activity. Because it has been decided to keep the agent architecture from prototype version 1, the role implementation doesn't reflect the role concept described in chapter 4. In this prototype roles are considered to be transparent and they are only used for qualifying the participants of an activity. This is achieved by giving the agents a set of instantiated roles that they currently hold, then it can be checked if an agent can play a specific role simply by

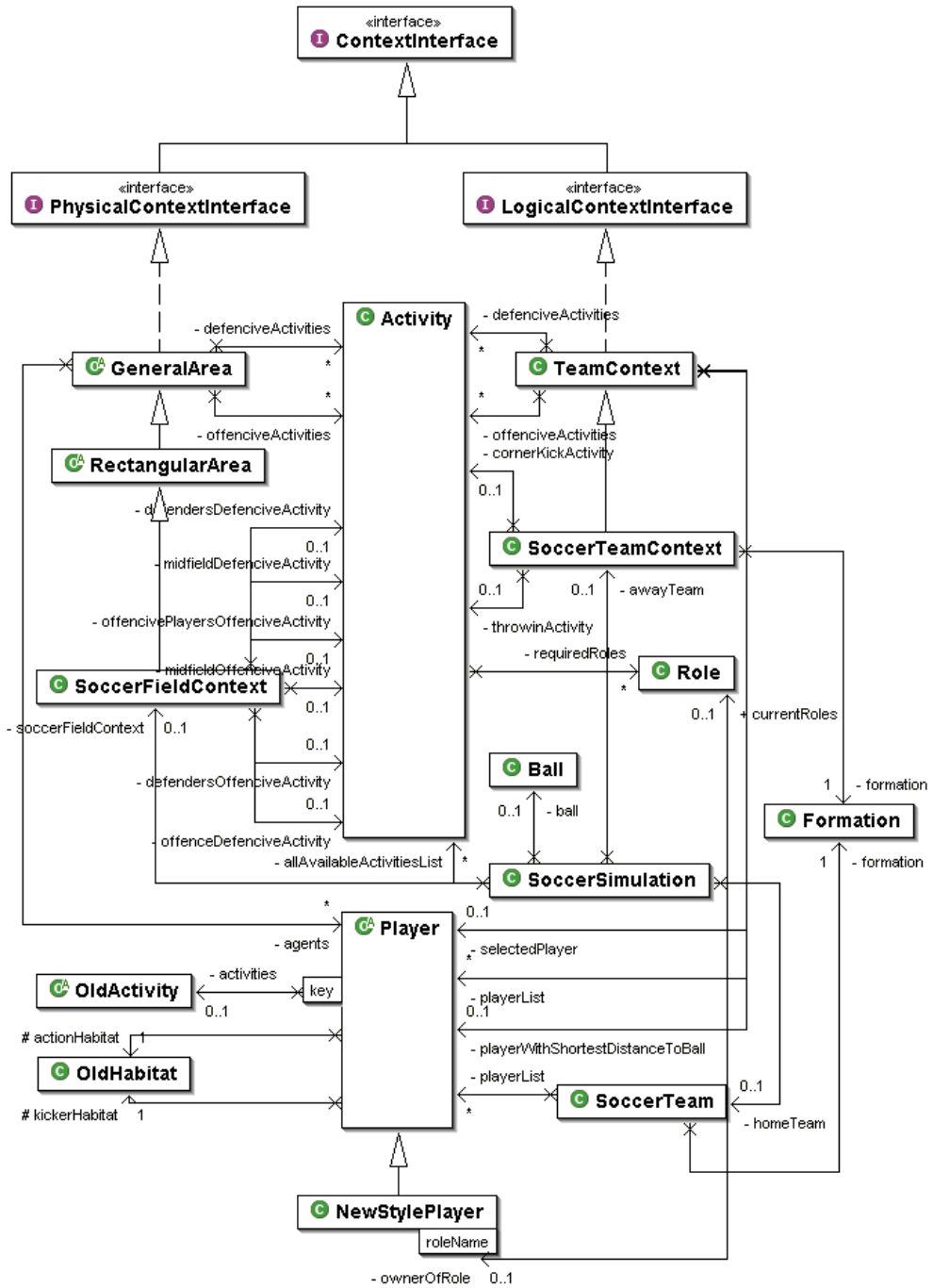


Figure 5.3: General class diagram of SoccerLab prototype version 2 model component.

checking whether it has the role in the set of current roles or not. A transparent

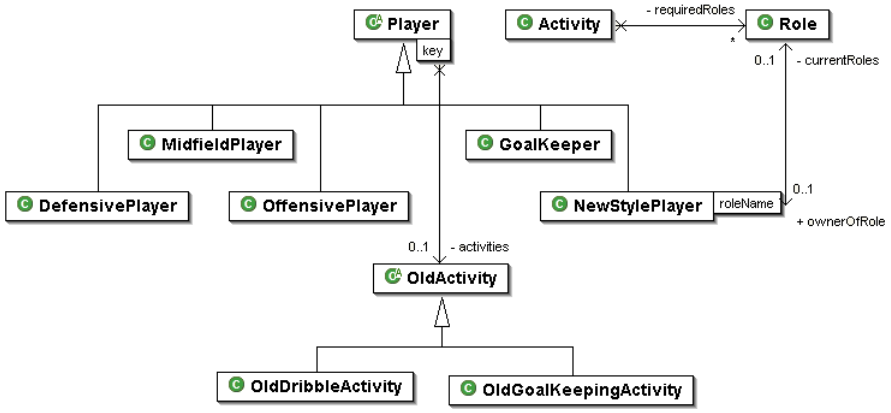


Figure 5.4: The segment of the class diagram of the model component of SoccerLab prototype version 2 concerning agents.

role is understood as a role that provides full access to its owner which mean that the role neither provides additional properties nor restrict the abilities of the role owner. The reason for choosing this simple role implementation is mainly because of the simple agent architecture. The simple agent architecture doesn't encourage the use of roles and because of that it has been of higher priority to focus on the implementation of activities.

The class OldActivity holds the activity implementation from prototype 1, because the understanding of the activity concept has changed during the project. These old activities are not considered to be activities any more, they are more like capabilities for the individual agent executed when no collaboration is needed or when the directive of an activity tells an agent to execute a specific capability. The reason for naming the super class OldActivity is to reflect the change from prototype 1 to prototype 2. In the class diagram in figure 5.4 it should be noticed that the Player class(agent) has been decoupled from the (new)Activity class, which is different from prototype 1(OldActivity).

Activity

After finishing the development of SoccerLab prototype version 1, it was concluded that the platform among others was well suited for experimenting with activities. This statement was quickly proved wrong when experimenting with more complex activities involving more than one participant, which among other motivated the literature study described in chapter 4. The literature study lead to the definition of the activities implemented in this prototype.

The activities implemented in SoccerLab prototype version 2 is conceptually in accordance with the activities defined in section 4.4 with only a minor flaw which is that the participants are not treated as a logical context when they engage in an activity.

In chapter 4 examples on how activities can be used to describe collaboration in the referent system are given. The following describes how the activities are used to prescribe collaboration between agents in software or the model system.

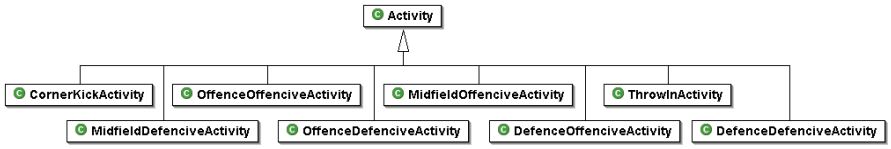


Figure 5.5: The segment of the class diagram of the model component of SoccerLab prototype version 2 concerning activities.

Activities are implemented as threads, figure 5.5 depict the segment of the SoccerLab prototype version 2 concerning activities. The common parts for all the activities, such as implementing the run loop for the thread and assigning roles to participants, are implemented in the Activity super-class. Each of the specialised activities are implemented as sub-classes that override the directive of the general activity.

Table 5.1 presents a simple pseudo example of an activity sub-class implementation.

```
public class ExampleActivity extends Activity {
    private Role role1,role2,role3,role4;

    public ExampleActivity() {
        super();
        role1=new DefenciveRole("defenciveRole");
        role2=new DefenciveRole("defenciveRole");
        role3=new DefenciveRole("defenciveRole");
        role4=new DefenciveRole("defenciveRole");
        requiredRoles.add(all four roles);
    }

    protected void directive(){
        do{
            for(Role role:getRequiredRoles()){
                if(isRoleClosestToBall(role)){
                    role.runToLocation(Ball.getPosition());
                }
                else{
                    role.runToLocation(desired location);
                }
            }
            Thread.sleep(activityThreadDelay);
        }while(all the roles are filled || the collaboration isn't done);
    }

}
```

Table 5.1: Example of an activity sub-class implementation.

The constructor in table 5.1, initialises and adds the roles to the list of required roles. When the required number of participants, in this case four, have joined the activity(all the required roles are filled) the directive of the activity is executed. The directive in table 5.1 is an example on how to prescribe collaboration between four participants. In this case the participant closest to the ball is directed to chase the ball and the last three participants are told to obtain a more defencive or offencive position depending on the purpose of the activity. The activity will continue to guide its participants until one or more of them have

to leave the activity due to a change in their environment, e.g. the team gets in ball possession or looses ball possession. The example activity in table 5.1 are highly similar to the DefenceDefenciveActivity from figure 5.5.

The reason why activities are such a powerful tool for prescribing collaboration is that an activity only accesses its participants through roles, which makes it indifferent to the participants as long as they can fulfil one of the required roles. In the long term perspective this means that activities can describe and prescribe collaboration between different species if they just can fulfil the required roles. Activities are easy to use when prescribing collaboration between agent and as a result of that activities such as throw-in and corner-kick has been added to soccer simulation application.

Context

The support for contexts in SoccerLab prototype version 2, has been achieved by defining a context interface, table 5.2. The interface in table 5.2 reveals that contexts distinguish the activities it provides in offencive and defencive activities. Of course this should not be the case, but this solution has been chosen because, currently, there are no other way to distinguish between the purposes of activities. Also the prototype doesn't have any examples of temporal contexts, but it is because no example has been found and not because it is not possible. Besides these two exceptions the contexts implemented in SoccerLab prototype version 2 is in conceptual accordance with the definition of context from chapter 4. A context is represented as a class that implements the context

```
public interface ContextInterface extends Runnable {
    public List<Activity> getOffenciveActivities();
    public List<Activity> getDefenciveActivities();
    public List<Player> getContainedAgents();
    public int getNumberofContainedAgents();
    public void addOffenciveActivity(Activity activity);
    public void addDefenciveActivity(Activity activity);
}
```

Table 5.2: The general context interface.

interface. Contexts are implemented as threads which is also defined by the context interface. This makes it possible for the context to continuously keep track of for instance how many agents that are currently present.

To reflect the physical and logical context of the context concept two subinterfaces have been defined, the interfaces for physical and logical contexts are presented as table 5.3 and table 5.4, respectively. The physical context interface, table 5.3, extends the general context interface and defines three methods, to get the physical context's location, width and height. The logical context interface, table 5.4, also extends the general context interface but it doesn't define any methods. This implies that the understanding of context should be modified so that all contexts are logical and then a physical context is simply a specialisation of a logical context, for handling the spatiality of the context.

The following gives a more detailed description of how contexts have been used in the SoccerLab prototype version 2.

```
public interface PhysicalContextInterface extends ContextInterface{
    public SoccerLabPoint getLocation();
    public double getWidth();
    public double getHeight();
}
```

Table 5.3: The physical context interface.

```
public interface LogicalContextInterface extends ContextInterface{

}
```

Table 5.4: The logical context interface.

Physical contexts: Figure 5.6 presents in details the segment of the class diagram of model component that concerns physical contexts.

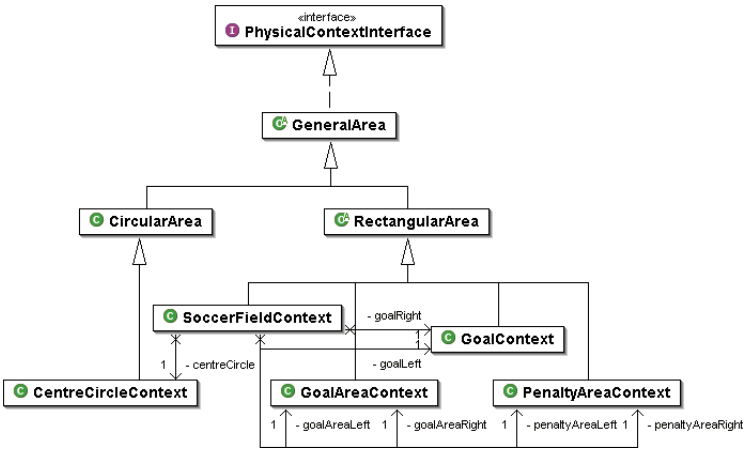


Figure 5.6: The segment of the class diagram of the model component of SoccerLab prototype version 2 concerning physical contexts.

The abstract class GeneralArea implements the physical context interface, and thereby also the general context interface. This class implements all the methods defined in the interfaces and the run loop of a physical context. For each step, approximately 10 times pr second, the run loop in a context(GeneralArea) counts the current number of agents present and determine whether the ball is present in the specific context or not. The two classes CircularArea and RectangularArea are both sub-classes of GeneralArea, the only difference in these two class is the shape of the context they implement.

The sub-class of CircularArea and the sub-classes of RectangularArea are used for composing SoccerFieldContext, which itself is a sub-class of RectangularArea. This means that when the application is running the soccer field consist of eight physical contexts, including the soccer field itself. This approach allows for getting information of in which physical context the ball is and how many agents that are currently present in each of the physical contexts.

Logical contexts: Figure 5.7 presents in details the segments of the model component concerning logical contexts.

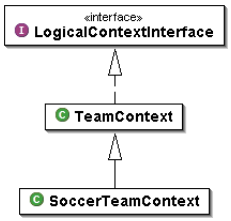


Figure 5.7: The segment of the class diagram of the model component of SoccerLab prototype version 2 concerning logical contexts.

The logical context inheritance structure, figure 5.7 is similar to the inheritance structure of physical contexts, figure 5.6. The logical contexts are intended to group agents which is the reason why the LogicalContextInterface, is implemented by the abstract super-class TeamContext. The TeamContext implements the run loop of logical contexts similar to the one for physical context.

The reason for having the TeamContext class to provide the flexibility for implement additional agent contexts besides the soccer team context. For instance, later on it could be desirable to implement a soccer team as a composition of the logical contexts defence, midfield and offence, similar to how the soccer field is composed from other physical context. This approach is not tested in this prototype, but it should be considered in later prototype version.

The class SoccerTeamContext is a sub-class of TeamContext. The SoccerTeamContext class is similar to the SoccerTeam class from prototype 1, it just have some additional functionality which makes it a logical context, inherited from the TeamContext class. It should be noticed that it is only the right team, clever team figure 5.2, that uses this new soccer team implementation to be able to compare the two prototype to each other, the left team, prototype 1 team figure 5.2 uses the soccer team implementation from prototype 1.

Both the physical context interface and the logical context interface are implemented by abstract classes, implying that these two interface could be replaced by abstract classes. This approach should be investigated further in a later prototype.

5.3.2 The GUI component

Additional features have been added to the graphical user interface in SoccerLab prototype version 2, this is reflected in the GUI component where a few additional classes have been introduced. Figure 5.8 depicts the class diagram of the GUI component. An option panel have been added in the right side of the graphical user interface, see Figure 5.2. The following describes the usage of the option panel and hence how to enable visualisation of contexts and activities.

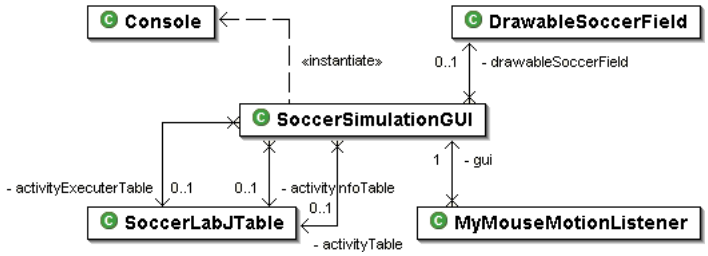


Figure 5.8: Class diagram of SoccerLab prototype version 2 GUI component.

The agent option panel

The agent option panel, item 6 section 5.2, corresponds to the habitat panel in the first prototype, and have the same functionality as described in section 2.2.

Due to the definition of a context in chapter 4, the understanding of contexts(formerly known as habitats) has changed and for that reason the agent option panel no longer enables visualisation of habitat but the areas in which the agents are supposed to take special actions, like kicking/passing the ball or try to intercept the ball.

The context option panel

An example of usage of the context option panel, item 11 section 5.2, is presented at figure 5.9 as a modified sequence diagram. To left in figure 5.9 the user's interactions time line is shown, similar to standard sequence diagrams. Instead of representing method invocations on different objects the horizontal arrows represent the user's action and the visual feedback from the GUI. The following gives a brief description of the modified sequence diagram at figure 5.9.

- The upper screen shot shows the initial layout of the context option panel when the application is started.
- The second screen shot shows the context option panel when the "Draw physical context(s)" has been enabled(checked).
- The third screen shot shows how the centre circle context has been selected.
- The lower screen shot shows a section of the soccer field when drawing of the centre circle has been chosen. In the upper left corner of the visualised context, the information of the context is presented: the context's name, the number of contained agents, the number of activities provided by this context and, at last, whether the context contains the ball or not.

The context option panel gives the possibility of visualising any of the physical contexts in SoccerLab prototype version 2. It should be noticed that to enable visualisation of any of the penalty areas, goal areas or goals either left half or right half has to be selected.

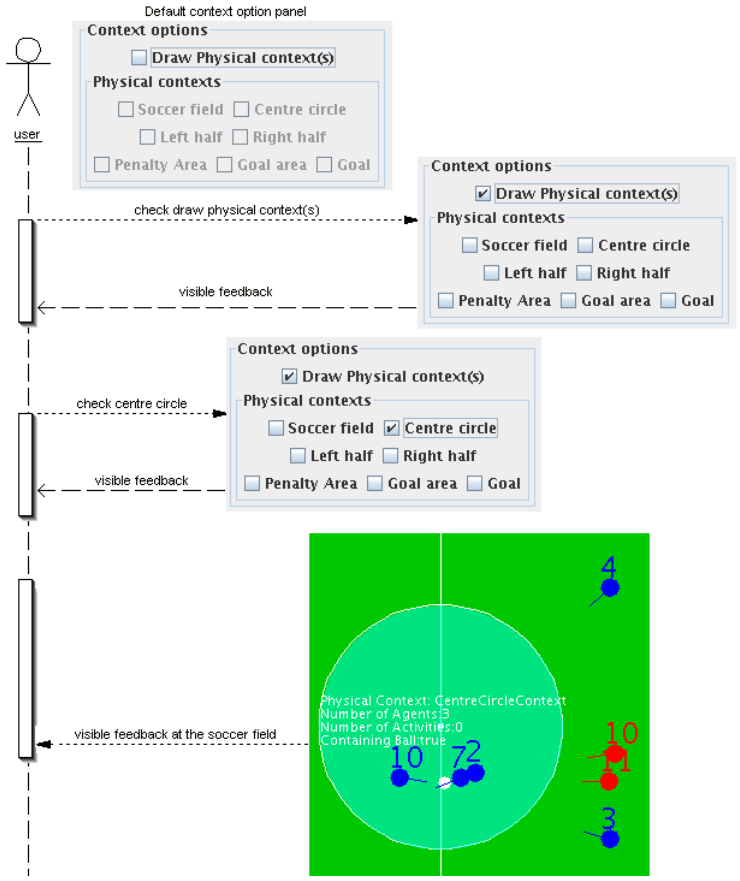
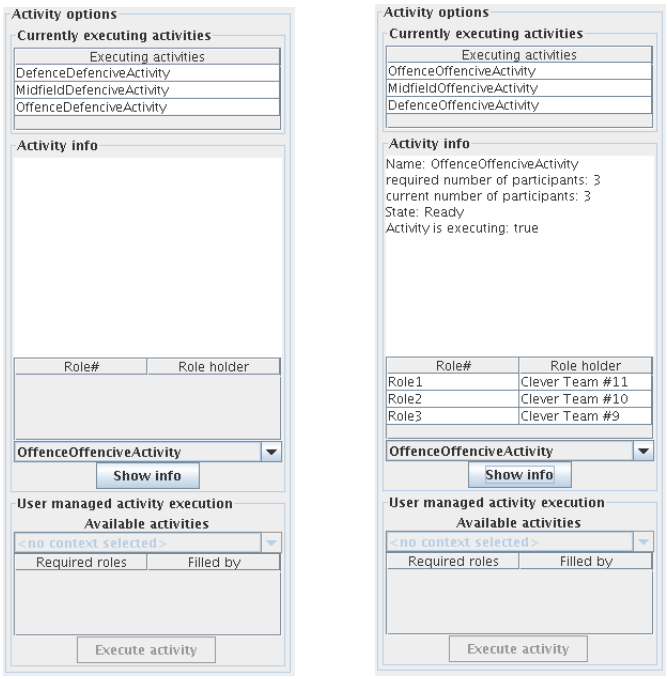


Figure 5.9: Sequence diagram exemplifying of usage of the context options panel in SoccerLab prototype version 2.

The activity option panel

The activity option panel can be divided into three sub-panels: a currently executing activity panel, an activity info panel and an activity executer panel. Figure 5.10 depicts the activity option panel in both its initial and active state when the application is running. The following gives a more detailed presentation of the activity panel.

The activity info panel: Figure 5.10(a) depicts the initial state of the activity option panel when the simulation is started, where the currently executing activity panel lists the executing activities.
Figure 5.10(b) shows how activity information is visualised when an activity has been selected. The activity information can be displayed by either selecting one of the currently executing activities from the currently executing activity panel or by selecting an activity from the combo box associated with activity info panel by clicking the show info button. When an activity has been selected, the activity info is presented as text informing the user of the name of the activity,



(a) The activity panel when it isn't showing activity information.

(b) The activity panel when it is showing activity information.

Figure 5.10: Screen shots of the activity panel.

the required number of participants, the current number of participants, the activity state, described later on, and whether the activity's directive is executing or not.

Additionally a table presents the type of the required roles if the left column and the participant filling the role in the right column.

The activity executer panel: At the bottom of the activity option panel the user managed activity execution panel[1] is positioned. By default this panel is disabled. Figure 5.11 depicts a modified sequence diagram, similar to the one in figure 5.9, exemplifying how to enable and use the activity executer panel, which is also described in the following.

- The upper left screen shot shows how to enable the activity executer panel by right clicking a context in the soccer field context, in this case the soccer field context itself. This will bring up a pop-up menu and by selecting Manage activities the activity executer panel will become enabled, at the same time the simulation is paused.

- The upper right screen shot shows the enabled activity panel. The combo

[1] Refered to as the activity executer panel.

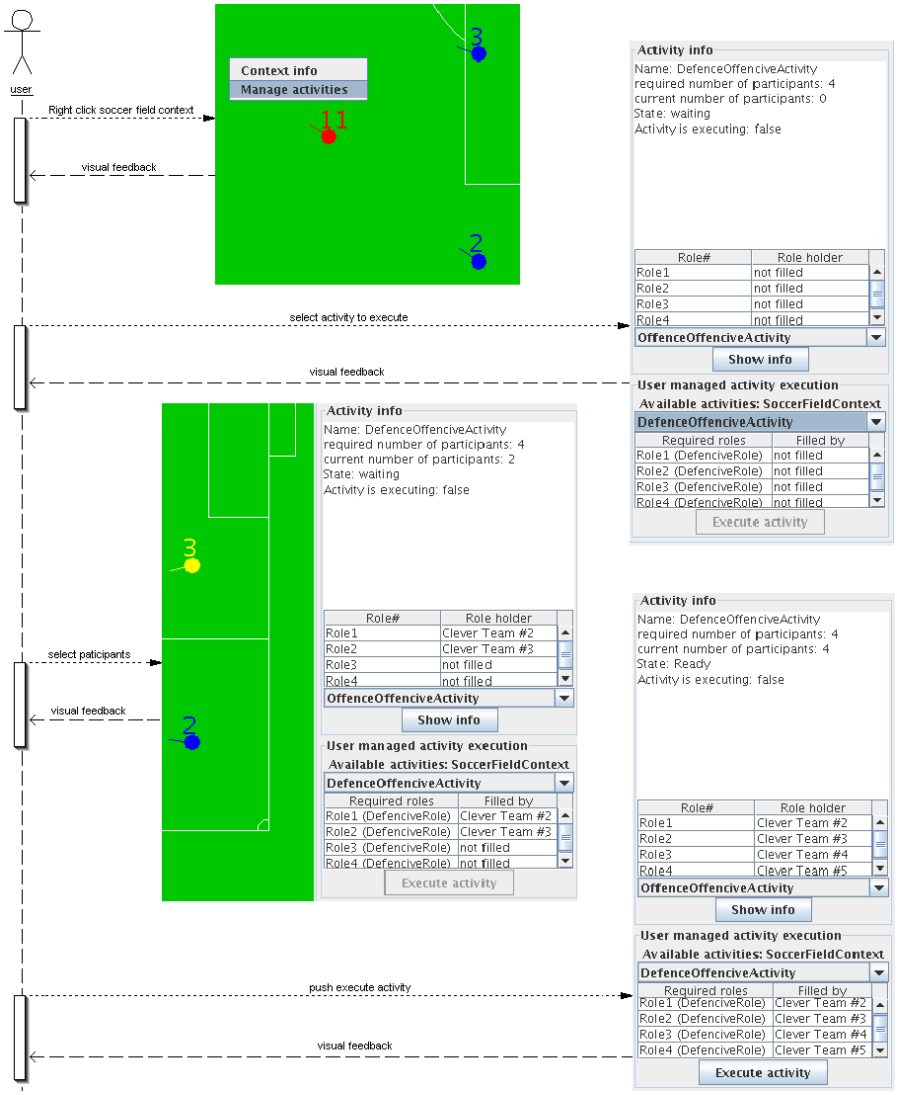


Figure 5.11: Sequence diagram exemplifying of usage of the the context options panel in SoccerLab prototype version 2.

box in user managed activity execution panel allows the user to select between all the activities provided by the selected context(the right clicked context from the previous item). In this the soccer field context is selected which is shown just above the combo box, and the DefenceOffenciveActivity chosen which is intended for defencive players when the team is in ball possession. It should be noticed that the activity info panel displays the information of the selected activity.

- The lower left screen shot shows how the roles required by the activity can be filled by using the mouse. The table below the combo box shows which

roles that are require and which participant is filling the role. A participant can be added to a role by left clicking on the desired participant and next left clicking on an empty role in the "Filled by" column in the table. If a participant that can not fill the role is selected then he is simply ignored and will not be added in the "Filled by" column. While the activity's roles are filled the activity info panel is updated, in this case it shows that only two of four roles are currently filled.

- The last screen shot, to the lower right, shows that all the participants have been gathered and the activity is ready to be executed, this can be seen because the Execute activity button has been enabled, also the state in the text field has switched to Ready, but the activity is not executing yet. When the Execute activity button is pressed, the simulation is resumed and the activity executer panel will become disabled again.

When an activity is executed there is no guarantee for how long it will execute because if the activity from the example is executed, if the team isn't in ball possession it will terminate immediately.

5.4 Dynamics

Compared to the old prototype not much of the dynamics have changed. For instance, the initialisation of SoccerLab prototype version 2 is almost identical to the initialisation of prototype version 1, sequence diagram figure 2.6. The only difference is that in prototype version 2 the away team is instantiated as a newly implemented soccer team context instead of as the old soccer team. Even though the away team has changed from being a soccer team into being a soccer team context the setup of team formation are identical.

Because of the new implementation of activities in prototype version 2, the agents' ways of selecting activities has changed. In SoccerLab prototype version 2 all the agents, of course only the agents using the new agent architecture, use the same strategy for selecting an activity, whereas the prototype 1 agents used slightly different selection strategies depending on the agent type, defencive player, offencive player, etc. Algorithm 2 describes how the prototype version 2 agents select activities.

The following summarises the strategy of algorithm 2; first the agent determines whether it is required to take immediate action such as: shooting the ball, passing the ball or run to the ball. If this is not the case it will determine if there are any changes to how it perceive the environment, this is currently simplified to: is my team still in ball possession or is my team still not in ball possession. If there are any changes the agent will leave the activity in which it is currently participating, to be able to choose another more suitable activity. At last the agent will act as directed from the activity if it is engaged in one, else it will find an activity to join.

5.5 Evaluation

Several improvements have been applied to the SoccerLab application to form the SoccerLab prototype version 2. Among others, especially the support for

Algorithm 2: Activity selection for SoccerLab prototype version 2 players

```
if the ball is inside the action area then
    if the ball is inside the kick area then
        if close enough to goal then
            execute shootAtGoalActivity;
        else
            execute passBallActivity;
        end
    else
        execute runToBallActivity;
    end
else if any changes then
    break current activity;
end
if participating in an activity then
    continue execution of this activity;
else
    find new activity to join;
end
```

the concept of activity and context should be emphasised. The graphical user interface has been improved too, to provide more user interaction.

To be able to test the improvements from the first prototype to the second prototype, the home team, prototype 1 team, uses the old implementation of activities. When soccer matches are simulated at normal speed the away team, clever team, wins 95 % of the matches, as expected, which indicates that the prototype 2 implementation is better than the old implementation. If the soccer matches are simulated at full speed(four times normal speed) the result is completely reversed so that prototype 1 team wins close to 100 % of the matches. This strange phenomenon occur due to the handling of thread delays. When the simulation speed is increased the agents' thread delay is equally decreased, giving the agents more action per second, whereas the activities' thread delay is constant. As a result of the thread handling the agents in prototype 1 team are given more actions per second which is not the case with the agents in clever team as long as they are participating in activities.

5.5.1 Agent support

A shortcoming in SoccerLab prototype version 2 is the simple agent architecture. It is encouraged that the next prototype version implements a more advanced agent architecture similar to the one presented in section 4.3, combining the BDI-architecture and the modified AALAADIN-model.

Even though the agent architecture implemented in SoccerLab prototype 2 is simple it has proven to be well suited for the simulated soccer players. The agents can engage in activities depending the roles they hold. The roles in SoccerLab prototype 2 should be improved, for instance a role could define a set of required properties, then when an agent has a desire for joining a given activity it could check the properties required by the role and if he fulfils the

properties he can play that role and thereby participate in the activity.

5.5.2 Activity support

The activities implemented in SoccerLab prototype 2 are in conceptual accordance with the activity concept characterised in section 4.4. The concept of activity seems as a powerful concept for describing and prescribing collaboration between multiple agents. Another powerful aspect of activities is that they only access their participants through roles. This provides a high level of flexibility, for instance if the agent architecture is changed as suggested in the previous section the activities are not affected because they only require that the participants can play the roles they specify.

After implementing the activities of SoccerLab prototype 2, it is suggested that some kind of mechanism is added to the activities which can be used to determine the purpose of a specific activity to ease the task for the agents to choose a suitable activity. In the case of soccer simulation the purpose of an activity could be determined by properties describing whether it should be used for offencive or defencive strategies, whether it requires one of the participants to possess the ball or maybe just the team should be in ball possession, etc.

The following lists the benefits gained by implementing the activities characterised in chapter 4.

- It is easy to add new activities which is simply achieved by sub-classing.
- An activity is a flexible tool for prescribing collaboration between agents because it only accesses its participants through roles.
- Complex collaboration can easily be prescribed because of the centralised control of the participants using the activity's directive.

5.5.3 Context support

Both physical and logical contexts have been implemented in this prototype. In the current version contexts influence its inhabitants by providing activities in which they may engage.

5.5.4 The graphical user interface

The graphical user interface has been improved with different kinds of user interactions supporting the AAC concepts. Where the focus has been on visualising the activity and context concepts. The visualisation of contexts and activities gives the user a good overview of which activities are currently executing and where the contexts are located.

The agent panel could be improved to display agent information. This feature is currently only available by right clicking a specific agent and selecting Player info in the emerging pop-up menu, which will produce a text output presenting the agent information in the GUI console.

In general SoccerLab prototype version 2 seems as a well suited platform for continuing the experiments with the concepts of: agent, activity and context.

5.6 Summary

The purpose of implementing SoccerLab prototype version 2 was mainly to add better support for the improved AAC concepts, compared to prototype version 1. Additionally it was to introduce more user interaction and visualisation of the concepts activity and context.

Due to prioritising the suggested agent architecture has not been implemented in this prototype version. Instead the existing agent architecture from prototype 2 has been modified to support simple roles and to be able to engage in activities.

Activities has been implemented is Soccerlab prototype version 2 for prescribing collaboration between multiple agents. New activities are simply added to the SoccerLab application by sub-classing the general Activity class. After experimenting with activities in SoccerLab prototype version 2 they have proved to be flexible and powerful for prescribing collaboration. The flexibility is achieved because activities only access their participants through roles. The activities are powerful because they provide centralised control of their participants and hence it is easy to prescribe complex collaboration between them.

The support for the context concept in SoccerLab prototype version 2 has been achieved by defining two interfaces, one for physical contexts and one for logical contexts, for the context implementing classes. After experimenting with the context concept in the prototype it is suggested to change the representation of contexts from interfaces to abstract classes. The definition of the context interfaces also implied that a physical context simply is a specialisation of a logical context. This issue requires further investigation. In this prototype contexts influence their inhabitants by providing activities, in which they may engage, and information data.

The graphical user interface has been improved with an option panel allowing the user to interact with the three concepts agent, activity and context. The user in also allowed to enable visualisation of context and activities. Contexts are visualised by cyan coloured shapes on top of the soccer field and additionally presenting information of the specific context such as the number of contained agents and activities. Activities are visualised in the option panel, which presents the currently executing activities and if a specific activity is chosen more detailed information of the activity is presented such as which roles are filled by which agents, how many participants are required for the activity to execute and whether the activity is executing its directive or not.

In general SoccerLab prototype version 2 is a good platform for experimenting with activities and contexts and a good tool for demonstrating the AAC concepts “in action”.

Bibliography

D. Ancona and V. Mascardi. Coo-BDI: Extending the BDI Model with Cooperativity. To appear in: *Proceedings of the First Declarative Agent Languages and Technologies Workshop (DALT)*, Melbourne, Australia, 2003.

Peter Bøgh Andersen and Palle Nowack. Tangible objects: connecting informational and physical space. In *Virtual space: spatiality in virtual inhabited 3D worlds*, pages 190–210, London, UK, 2002. Springer-Verlag. ISBN 1-85233-516-5.

Peter Bøgh Andersen and Palle Nowack. Modelling moving machines. In *Virtual Applications: Aplications with Virtual Inhabited 3D Worlds*, pages 189–211, London, UK, 2004. Springer-Verlag.

Jakob Appel and Kurt Nielsen. Applied multi-agent systems principles to cooperating robots in tomorrow's agricultural environment. Master's thesis, The Maersk McKinney Moller Institute for Production Technology, University of Southern Denmark, Odense, Denmark, June 2005.

David J. Barnes and Michael Kölling. *Objects First with Java - A Practical Introduction using BlueJ*. Prentice-Hall, second edition, 2004. ISBN 0-13-124933-9.

K. Beck and W. Cunningham. A laboratory for teaching object oriented thinking. In *OOPSLA '89: Conference proceedings on Object-oriented programming systems, languages and applications*, pages 1–6, New York, NY, USA, 1989. ACM Press. ISBN 0-89791-333-7.

Joshua Bloch. *Effective Java programming language guide*. Sun Microsystems, Inc., Mountain View, CA, USA, 2001. ISBN 0-201-31005-8.

Federation Internationale de Football Association. Laws of the game 2004. Hitzigweg 11, 8030 Zurich, Switzerland, 2004. URL `http://www.fifa.com/documents/static/regulations/LOTG2004_e.pdf`.

Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98)*, pages 128–135, Paris, France, 1998. URL `citeseer.ist.psu.edu/ferber98metamodel.html`.

Christiane Floyd. A systematic look at prototyping. In R. Budde, K. Kuhlenkamp, Lars Mathiassen, and H. Zullighoven, editors, *Approaches to Prototyping*. Springer Verlag, 1984.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. aw-pcs. Addison-Wesley, 1995.

Martin Greber, Vadim kyrylov, and David S. Bergman. Multi-criteria optimization of ball passing in the simulated soccer. To appear in Journal of Multi-Criteria Decision Analysis, 2005.

Eyðun Eli Jacobsen, Bent Bruun Kristensen, and Palle Nowack. Architecture = abstractions over software. In *TOOLS '99: Proceedings of the 32nd International Conference on Technology of Object-Oriented Languages*, page 89, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0462-0.

N. R. Jennings and M. Wooldridge. Applying agent technology. *Applied Artificial Intelligence*, 9(4):351–361, 1995. URL `citeseer.ist.psu.edu/jennings98applying.html`.

B. B. Kristensen. Subjective behaviour. *International Journal of Computer Systems Science and Engineering*, 16(1):13–24, January 2001. ISSN 0267-6192.

Bent Bruun Kristensen. Transverse activities: Abstractions in object-oriented programming. In *Proceedings of the First JSSST International Symposium on Object Technologies for Advanced Software*, pages 279–296, London, UK, 1993a. Springer-Verlag. ISBN 3-540-57342-9.

Bent Bruun Kristensen. Transverse classes and objects in object-oriented analysis, design, and implementation. In *JOOP*, 1993b.

Bent Bruun Kristensen. Complex associations: abstractions in object-oriented modeling. In *OOPSLA '94: Proceedings of the ninth annual conference on Object-oriented programming systems, language, and applications*, pages 272–286, New York, NY, USA, 1994. ACM Press. ISBN 0-89791-688-3.

Bent Bruun Kristensen. Object-oriented modelling with roles. In *Proceedings of the 2nd International Conference on Object Oriented Information Systems (OOIS'95)*, pages 57–71, Dublin, Ireland, 1995.

Bent Bruun Kristensen. Associations: Abstractions over collaboration. In *Proceedings of the 2003 IEEE International Conference on Systems, Man and Cybernetics*, volume 5, pages 4104 – 4113, Washington, D.C., USA, 2003. ISBN 0-7803-7952-7.

Bent Bruun Kristensen. Associative programming and modeling: Abstractions over collaboration. Unpublished, 2005.

Bent Bruun Kristensen and Daniel C. M. May. Activities: Abstractions for collective behavior. In Pierre Cointe, editor, *ECOOP*, volume 1098 of *Lecture Notes in Computer Science*, pages 472–501. Springer, 1996. ISBN 3-540-61439-7.

Bent Bruun Kristensen and Kasper Osterbye. Roles: Conceptual abstraction theory and practical language issues. *Theory and Practice of Object Systems*, 2(3):143–160, 1996.

Ole Lehrmann Madsen, Birger Møller Pedersen, and Kristen Nygaard. *Object-oriented programming in the BETA programming language*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993. ISBN 0-201-62430-3. URL `http://www.daimi.au.dk/~beta/books.html#betabook`.

Daniel C. M. May, Bent Bruun Kristensen, and Palle Nowack. Tango: Modeling in style. In *Proceedings of the Second International Conference on Generative Systems in the Electronic Arts (Second Iteration - Emergence)*, Melbourne, Australia, 2001.

Daniel Chien-Meng May and Bent Bruun Kristensen. Habitats for the digitally pervasive world. In *Virtual Applications: Aplications with Virtual Inhabited 3D Worlds*, pages 141–158, London, UK, 2004. Springer-Verlag.

Felipe Rech Meneguzzi, Avelino Francisco Zorzo, and Michael da Costa Móra. Mapping mental states into propositional planning. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*. ACM Press, 2004. URL `citeseer.ist.psu.edu/article/meneguzzi04mapping.html`.

H. Mintzberg. *Structure in fives: Designing effective organizations*. Prentice-Hall, Englewood Cliffs, N.J., 1983. ISBN 0-13-854191-4.

Lin Padgham and Patrick Lambrix. Agent capabilities extending BDI theory. In *AAAIIAAI*, pages 68–73, 2000. URL `citeseer.ist.psu.edu625805.html`.

S. Parsons and P. Giorgini. An approach to using degrees of belief in bdi agents, 1999. URL `citeseer.ist.psu.edu/parsons99approach.html`.

H. Parunak. Go to the ant: Engineering principles from natural multi-agent systems, 1997. URL `citeseer.ist.psu.edu/parunak99go.html`.

Kasper Hallenborg Pedersen and Bent Bruun Kristensen. Pervasive computing: Mapping tango model onto jini technology. In *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002)*, Orlando, Florida, 2002.

A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995. URL `citeseer.ist.psu.edu/rao95bdi.html`.

James Rumbaugh. Relations as semantic constructs in an object-oriented language. In *OOPSLA '87: Conference proceedings on Object-oriented programming systems, languages and applications*, pages 466–481, New York, NY, USA, 1987. ACM Press. ISBN 0-89791-247-0.

Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995. URL `citeseer.ist.psu.edu/article/wooldridge95intelligent.html`.

Michael Wooldridge, Nicholas R. Jennings, and David Kinny. A methodology for agent-oriented analysis and design. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference*

on Autonomous Agents (Agents'99), pages 69–76, Seattle, WA, USA, 1999. ACM Press. URL `citeseer.ist.psu.edu/wooldridge99methodology.html`.

Franco Zambonelli and H. Van Dyke Parunak. From design to intention: signs of a revolution. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 455–456, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-480-0.

Yu Zhang. The tao of soccer, a tutorial presented at sfu surrey, March 2005. URL `http://e-graviton.com/tos/tos.ppt`.

Appendix A

Preliminary application survey

This appendix describes the preliminary application survey. During the course SD02 in the fall of 2004, four soccer simulations have been implemented. As an initial step four existing soccer simulations will be analysed. The reason for choosing this approach is to able to reuse and extend clever ideas and design solutions, instead of starting from scratch. In the following section a short description of the four existing soccer simulations is given.

A.1 Application descriptions

The description of the four soccer simulations will be based on the important parts of their class diagrams and a runtime screen shot of each application.

A.1.1 The blue team

This application seems to run just fine, and the GUI is very simple, but highly understandable, see figure A.1. Each player is represented as a dot with a "nose" to indicate which way the player is moving/looking.
To the left in the GUI a control panel is placed to allow some user interaction at runtime. The first three buttons from the top are self explaining and handles execution of the simulation.
The two middle buttons give the user the ability to change team formation at runtime. This feature can be used to test different team formations, to see which one is best suited for this simulation to score goals.
The lower of the last two buttons in the bottom of the control panel terminates the application, the other toggles debug informations on and off. The debug information is in this case rather useful not just for the developers. When the debug information is enabled various useful debug info is shown at runtime. A specially nice feature in debug mode is the square around each player to show their particular zone of interest. A screen shot of the application with the debug information enabled is shown at figure A.2.

The important parts of the blue team's class diagram, at figure A.3, shows that there has been chosen some reasonable design choices. The best part of the

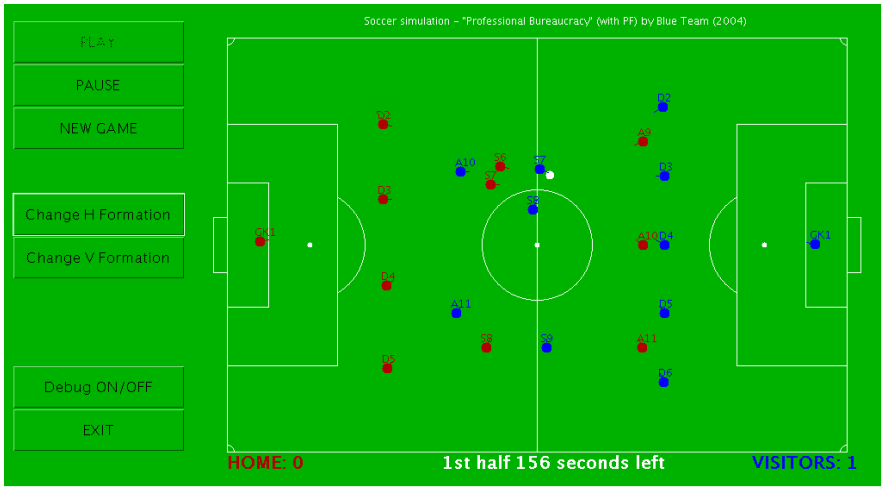


Figure A.1: A screen shot of the blue team's soccer simulation.

design is, that the role pattern has been applied to handle players, so common properties for players are implemented in an abstract player class and specialisations of attackers, defence and goalies are achieved by inheritance, this eases the understandability of the application. A good idea which should be noticed is in the `Team` class where the first couple of instance variables corresponds to a player formation for the team. This might not be the only way to handle the player formations, but it's an example on how it could be done, and is worth to keep in mind.

A.1.2 The green team

The green teams soccer simulation runs fine and again the GUI, at figure A.4, is kept simple and understandable. A feature in the GUI is that the soccer players look like animated humans, this feature is achieved by using three small soccer player images which then are flipped like a looping slide show. In the upper left corner of the GUI three buttons are placed to control the application, play, options and exit. When the options button is pressed a new window pops up and shows a list of players for each team. In the bottom of the GUI a picture of some spectators is placed they don't move and have no influence on the simulations, so the only reason they are there is to create some associations to a soccer stadium.
The Green team's class diagram for the domain model, which in this case is the important part, is actually one large inheritance diagram so everything in the soccer domain model is at some level a specialisation of the `SSComponent` class. From a understandability perspective this design is a bit confusing because it is a little strange that both a field and a player can be used as a component. On the other hand it is easy understandable that a midfield player is a specialisation of a participant. Two classes which draw some attention are `SSReferee` and `SSCoach`. The referee class is not yet implemented and thereby he isn't graphically represented in the simulation either. The benefit of having a referee

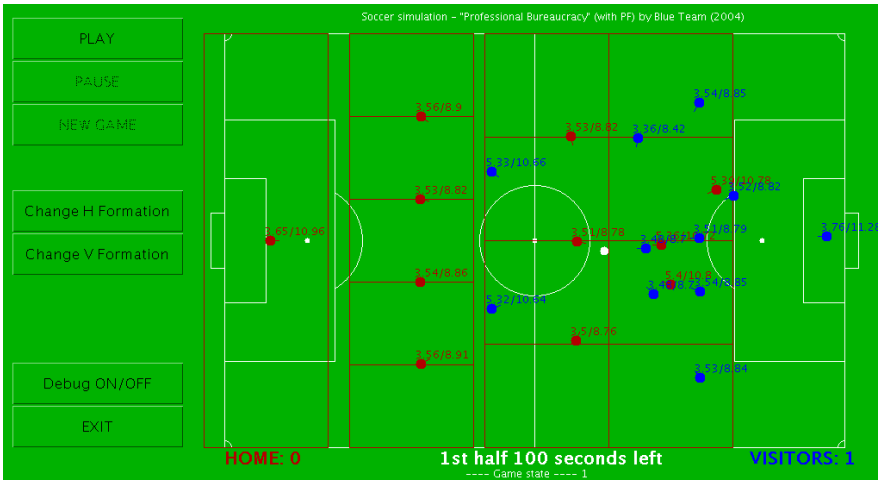


Figure A.2: A screen shot of the blue team's soccer simulation, when the debug information toggled on.

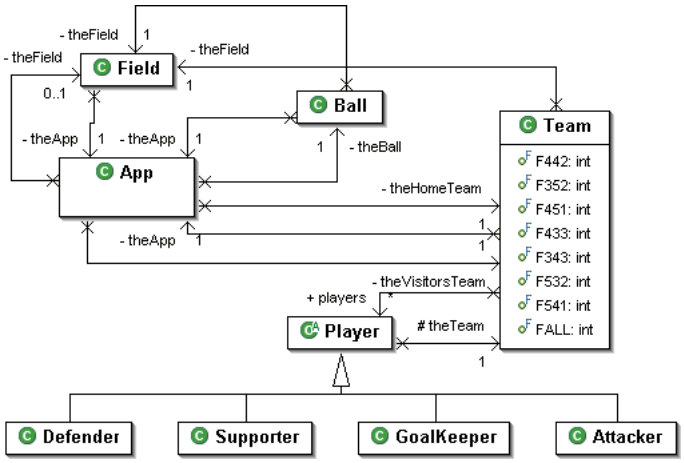


Figure A.3: The blue team's class diagram.

class/structure is that it immediately leads one to think about soccer rules, so the idea of having a referee is actually good,even though it in this case haven't been implemented yet. The coach class is also good to have for abstraction purposes, because in this case the class has the responsibility for formation selection, like a coach in the real life has. This mapping from a coach to a software coach would be even more beneficial if instead of using just a coach class for formation selection, a full coach class structure could be implemented to handle several coach relevant tasks.

Figure A.4: A screen shot of the green team's soccer simulation.

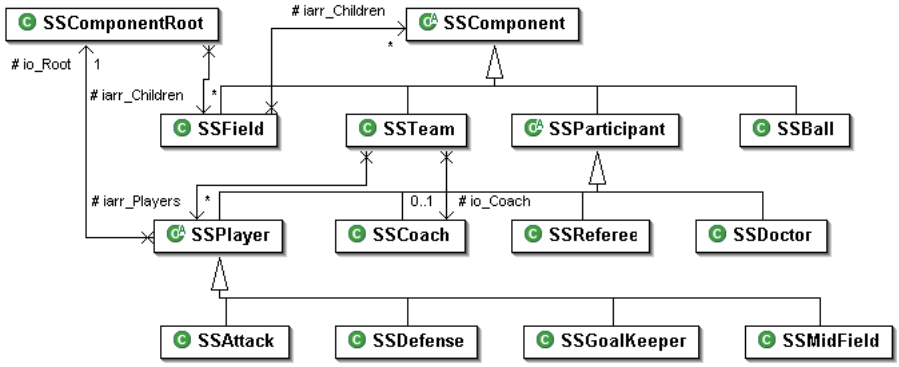


Figure A.5: The green team's class diagram of the soccer domain model.

A.1.3 The pink team

The pink teams soccer simulation, doesn't seem to be fully implemented. The players don't move at all, they just kick the ball around, before they pass it to the other team. The only things, that brings your mind on soccer, are the field, eleven players on each team and a ball. A screen shot of the GUI is shown at figure A.6.

The pink team's class diagram is shown at figure A.7. This class diagram

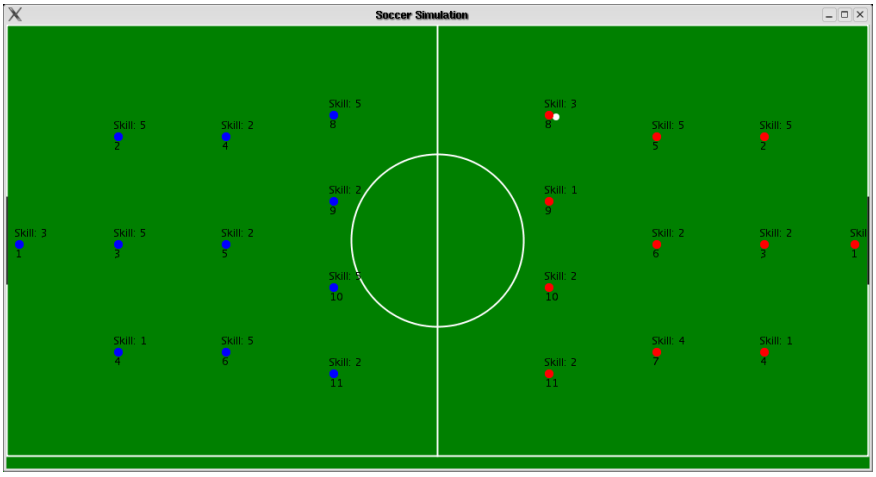


Figure A.6: A screen shot of the pink team's soccer simulation.

doesn't contain any inheritance structure like two previous ones did. Instead the different kinds of players are handled by a single class logically named `Player`. Two classes `Contingencies` and `PlayerMind` are associated to the player class. The `Contingencies` class handles random events like injuries. It might be a good idea to bring some randomness into the simulation because it gives a better reflection of the real life soccer game. The `PlayerMind` class in this case is just a dummy class with no functionality, but the perspectives are highly relevant because it can be used for implementing player intelligence or cooperativeness with the other team mates.

A.1.4 The red team

There are no goalies in the red team's soccer simulation, but besides that it runs fine, maybe a bit fast but one could argue it equals action. At figure A.8 a

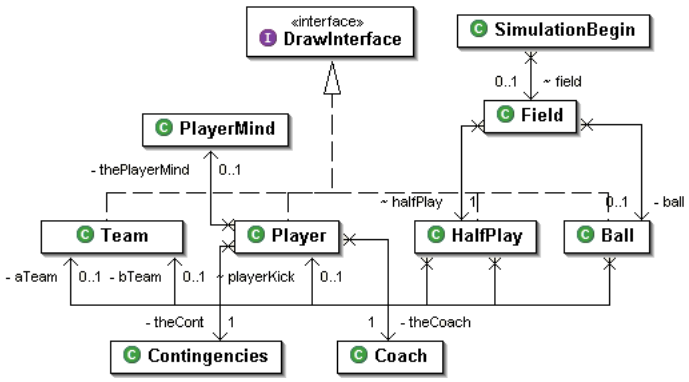


Figure A.7: The pink team's class diagram.

screen shot of the red teams soccer simulation is presented. The main difference from the red team's GUI and the previous ones, is that the field is positioned vertical and not horizontal. The rotation of the field doesn't influence the understandability of the simulation, but it brings memories from early 90's soccer games like *Kick Off* and *Sensible Soccer*.

The red teams class diagram mainly consist of two inheritance structures, as

Figure A.8: A screen shot of the red team's soccer simulation.

shown at figure A.9. The upper structure handles moving objects like players and the ball. The player specialisation is further specialised in three subclasses to handle different kind of players. It seems, a little strange, that the players and a ball can replace each other, just because they are moving objects. The greatest achievement of placing a ball and a player in the same inheritance structure is to simplify handling of movements in the simulation. An other way to achieve simple movement handling could be to define and implement an interface and thereby achieve similar functionality.

The lower inheritance structure handles team strategy and formations. This is a nice alternative for handling team formations compared to the approach described in section A.1.1. A great advantage of handling team formations and team strategies in this manner is, that it is highly extend-able. The only thing needed for adding new team formations, is to extend the `Team` class with a new sub class containing the new formation and strategy.

A.2 Comparisons

Since all four applications are soccer simulations, an obvious result is that coinciding designs have been chosen. To summarise the important parts of the four applications table A.1 has been made.

The similarity of the GUIs in the four applications are very high, this is because they all are kept very simple. The simplicity is good and increases the understandability. One of the applications uses small images to represent players, this

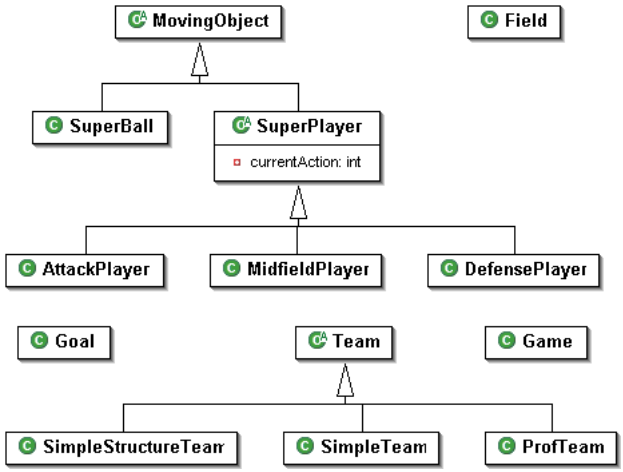


Figure A.9: The red team's class diagram.

approach works fine, but it isn't in any way better than the other applications that just uses dots for player. One might even argue that the dot representation is better because it doesn't draw as much attention from the simulation as the small images might do. In some of the applications spectators are placed along the field, but as long they don't influence the simulation there is no reason for representing them in the GUI.
In all of the applications except one some sort of control panel is implemented in the GUI. In two of those applications it is possible to change team formation at runtime from the control panel, this is a nice feature which enables the possibility to experiment with different formation without restarting the application.

The debug information which has been implemented in the blue teams GUI is an outstanding feature. The idea of showing the square of interest for each player is very good, because it helps understanding why a given player doesn't act as expected because the ball isn't in his area of interest.
The similarities of applications are not just showing in the GUIs, they are also present in the class diagrams. The primary classes that appear in each of the applications are; player, ball, field and point or position. The only differences in these classes from one application to another are minor derivations in the class implementation and naming, but in general the responsibility for each of them are the same.
In the previous sections only the player classes have been mentioned, because different design solutions have been chosen, but the responsibility to handle a player is the same for all four applications.
Some of the classes besides the player class, almost seems mandatory for a soccer simulation domain model, namely the field and the ball class. As expected these two classes exist in all the class diagrams. Just by looking at the class names it implies their responsibilities and importance for the domain model. The ball classes are quite similar and take care of ball size, velocity, colour and position. The field classes are also great examples on the similarities among the different

The teams	Uniqueness	Pros	Cons
The blue team	The debug information which shows the area of interest for each player.	The GUI is kept simple and highly understandable. The players are graphically represented as dots with noses. Different kinds of players are by the role pattern.	The static handling of team formation as instance variables.
The green team	The players are graphically represented as animated humans.	The GUI is kept simple and highly understandable. The idea to introduce a referee class, which handle soccer rules.	The inheritance structure which allow both a ball and a goalkeeper to be used as the same component.
The pink team	The team has thought about modulating contingencies and player minds, but any of these haven't been implemented.	The GUI is very simple.	The players don't move, but they kick the ball around and unfortunately they often pass the ball to their opponents. There are no possibilities for controlling the application at run time.
The red team	Handling of team formations and strategies.	The team class where new formations and strategies can be added as subclasses.	There are no goalies. Using a super class for moving objects when an interface might be more appropriate.

Table A.1: Overview of the four applications

applications, they have the responsibility of defining field size and colour.
The most trivial of the common classes are the point or position classes. This

type of class is a necessity for the mathematical part of simulation. Examples on situations where such a position class is useful are; to calculate the distance between a player and his opponents, to calculate the distance to the ball or to calculate distance to the goal. In general the position class i useful whenever a distance has to be calculated or when a distance between two players in a player formation is defined.

A.3 Summary

All of the four soccer simulations, that have been implemented during the course SD02 in the fall of 2004, have both strengths and weaknesses. The best solution would actually be to merge the applications in a proper and suitable way.
The following list summarises the most important parts of a soccer simulation application based on the analysis.

- Simple GUI with a control panel for run time interaction.
- Some kind of debug view with the ability to "explain" whether a player is acting as expected or not.
- Applying design patterns to the domain model for understandability, flexibility and re-usability purposes.

In all of the application the GUIs are kept very simple and understandable. A nice GUI feature is a control panel which enables the possibility of changing team formations and strategies at runtime. The easiest way to create a graphical representation of the players is, to let a player be represented by a dot with a nose for direction indication.
A very nice GUI feature that would be desirable to have is a debug view of the application with the ability to show a given player's area of interest, player speed, player stamina etc.
The class diagrams for all four applications contain valuable design choices.
A good strategy for handling players is to have an inheritance structure with an abstract player class, and then by sub-classing specify the different kind of players. The good part choosing this strategy is that it increases understandability, and it is highly flexible and extend-able too, because new player types can be introduced without changing the existing code base.
Handling of team formation and strategy has been implemented in various ways, the most suitable solution is an inheritance structure like the one for the players. The argumentation is the same as for the player structure, but instead of adding player types as subclasses, team formations and strategies can be applied as subclasses for the abstract team super class.

Appendix B

Literature studies

This appendix concerns the articles read and summarised during the analysis of the AAC concepts, presented in chapter 4. Both during this and the other stages in this thesis additional relevant articles have been read, but these have not been summarised and because of that they not presented in this appendix[1]. The following lists the summarised articles in random, the articles suffixed with a * are summarised by Beata Hargesheimer:

- Object-Oriented Programming in the BETA Programming Language
- TangO: Modeling In Style *
- Architecture = Abstractions over Software
- Transverse Activities: Abstractions in Object-Oriented Programming *
- Activities: Abstractions for Collective Behavior
- Complex Associations: Abstractions in Object-Oriented Modeling *
- Object-Oriented Modeling with Roles *
- Roles: Conceptual Abstraction Theory & Practical Language Issues
- Subjective behavior
- Associations: Abstractions over Collaboration *

Evaluation of the summarised articles

All of the summarised articles have been categorised into the following categories:

- Conceptual framework – inspired by the conceptual framework of BETA [Madsen et al., 1993].
- Agent concept – if the article contributes with inspiration or understanding of agents.

[1] Additionally, an article concerning ball passing in soccer simulation have been summarised, which is also included in this appendix, appendix B.11

- Activity concept – if the article contributes inspiration or understanding of activities.
- Context concept – if the article contributes inspiration or understanding of contexts.

The summarised articles have been evaluated and graded; Low, Medium or High, with respect to the relevance of this thesis. The evaluations were done with focus only on this thesis, that means an article evaluated to "Low" relevance for this thesis is not necessarily uninteresting or a bad article.

B.1 Object-Oriented Programming in the BETA Programming Language

The chapters read are: 1,2 and 18, and the book [Madsen et al., 1993] can be downloaded at http://www.daimi.au.dk/ beta/books.html. Table B.1, presents the evaluation of Object-Oriented Programming in the BETA Programming Language.

Conceptual framework	Agent concept	Activity concept	Context concept	Relevance
✓				High

Table B.1: Evaluation of: Object-Oriented Programming in the BETA Programming Language

B.1.1 Brief summary

The first two chapters, chapter 1 and chapter 2, mainly concern object-orientation i general, whereas chapter 18 describes the conceptual framework associated with the BETA programming language.

Introduction – Chapter 1

The three main benefits of object-orientation are considered to be: *real world apprehension, stability of design* and *re-usability* of both designs and implementations.

Real world apprehension: It is important when working with object-oriented programming, to make the programs reflect that part of the reality they are going to treat as far as possible. Another important aspect of program development is to understand, describe and communicate about phenomena and concepts of the application domain.

Stability of design: Object-oriented programming provides a natural framework for modelling the application domain. The idea is to make a *physical model* of that part of the real world that is treated. This approach allows for changing or adding functions later on without the need of changing the underlying model.

Re-usability: Object-oriented programming languages have strong constructs supporting incremental program modification (inheritance). This allows the programmer to define a new component as an incremental extension of and existing one, thus preserving the relations between the two components.

Object-oriented programming and BETA: BETA is a modern language in the Simula tradition. A key sentence when talking about BETA is: "To program is to understand". It is of utmost importance that developed programs reveals an in-depth understanding of the application domain, to ensure that they will fit into the organisations that are to use them.
The model underlying object-orientation is by it's very nature informal, although part of the model has to be formalised to execute on a computer. BETA may be seen as a formal notation for describing those parts of the application domain, but it is important to keep in mind that the underlying conceptual framework is just as important as the language itself. In fact, BETA attempts to go beyond language mechanisms and contribute a conceptual framework for object-orientation.
In BETA the notion of **physical modelling** is introduced, to capture modelling of parts of the world based upon the identification of phenomena and concepts, and the notion of the **physical model**, to denote the models that are created in this process.

Introduction to basic concepts – Chapter 2

This chapter gives a brief and informally introduction to some of the basic concepts underlying object-oriented programming. It also gives a short introduction to other perspectives on programming, to describe how object-orientated programming relates to these.

Perspectives on programming. In the following three simple definitions of different perspectives, procedural-, functional- and constraint programming, are given:

Procedural programming. A program execution is regarded as a (partially ordered) sequence of procedure calls manipulating variables.

Functional programming. A program is regarded as mathematical function, describing a relation between input and output.

Constraint programming. A program is regarded as a set of equations describing relations between input and output.

Despite the different perspectives on programming, an ideal programming language should integrate all of them, because they are suited for different purposes.

Object-oriented programming may be defined as follows:

Object-Oriented programming. A program execution is regarded as a physical model, simulating the behaviour of either a real or imaginary part of the world.

In a computerised physical model, objects are the material used for representing physical phenomena from the application domain. Objects are characterised by attributes, which represent or model a property of the phenomenon being modelled. The object's attributes may vary in time, so an object must be able to represent this variation. When an attribute changes value, it is said that the object changes state.

Concepts and abstraction. In the real world people create concepts to capture the complexity of the world, this is done by making abstractions. A concept is a generalised idea of a collection of phenomena, based on knowledge of common properties of instances in the collection. Concepts can be characterised as follows:

- The *extension* of a concept is the collection of phenomena covered by the concept.
- The *intension* of a concept is a collection of properties that in some way characterise the phenomena in the extension of the concept.
- The *designation* of a concept is the collection of names by which the concept is known.

In many situation it is useful to consider a phenomena or concept as a *composition* of other phenomena and concepts. An example of composition is to consider a *whole* as constructed from *parts*. The notion of a *whole/part* composition is important for understanding and organising complex phenomena and concepts.

Conceptual framework – Chapter 18

This chapter introduces concepts such as information processes and systems and discuss abstraction, concepts, classification and composition. The conceptual framework provides a means to be used when modelling phenomena and concepts from the real world. The approach presented is that analysis, design and implementation are programming or modelling activities, but at different abstraction levels.

Physical modelling. The object-oriented perspective has been defined as follows:

> A program execution is regarded as a physical model, simulating the behaviour of either a real or imaginary part of the world.

The key word in this definition is physical. The term physical model is used to distinguish these models from, for instance, mathematical models. Elements of the physical model represent phenomena from the application domain. Objects are the computerised material used to construct computerised physical models. The real or imaginary part of the world being modelled, is called the referent system and the program execution (physical model) is called the model system. Abstraction in the referent system is the process of perceiving and structuring knowledge about phenomena and concepts with particular emphasis on the problem domain. Abstraction in the model system is the process of building structures that supports the computerised model.

Concepts and abstraction. Abstraction is a powerful tool available to the human intellect for understanding complex phenomena. Abstractions are used for grouping similar phenomena or concepts. They arise from recognition of the similarities while ignoring the differences.
The following definitions are given for the terms phenomenon and concept:

A phenomenon is a thing that has definite, individual existence in reality or in the mind; anything real in itself.

A concept is a generalised idea of a collection of phenomena, based on knowledge of common properties of instances in the collection.

Traditionally a concept is characterised by the following terms:

- The *extension* of a concept refers to the collection of phenomena that the concept somehow covers.
- The *intension* of a concept is a collection of properties that in some way characterise the phenomena in the extension of the concept.
- The *designation* of a concept is the collection of names (or pictures) by which the concept is known.

The vague term "in some way" in the definition of the intension is used because it is depending on the view in which the concept is seen. The Aristotelian view defines the intension of the concept as:

Aristotelian view The Intension of a concept is a collection of properties that may be divided into two groups: the defining properties that all phenomena in the extension must have and the characteristic properties that the phenomena may or may not have. the nature of the properties is such that it is objectively determinable whether or not a phenomenon has a certain property.

In a prototypical view the intension of the concept is defined as follows:

Prototypical view: The Intension of a concept consists of examples of properties that phenomena may have, together with a collection of typical phenomena covered by the concept, called prototypes.

The major difference between the two views is that the extension of a prototypical concept is not uniquely determined by the intension. It requires human judgement to decide whether or not a given phenomenon belongs to the concept. Programming languages like BETA is mainly useful for representing Aristotelian concepts.

The abstraction process. In both the referent and the model systems, concept structures are created. The process of producing and using knowledge may be split into three levels:

1. *The level of empirical concreteness.* At this level reality or individual phenomena are conceived as the are. Similarities between different phenomena aren't realised, nor is a systematic understanding of the individual phenomena obtained.

2. *The level of abstraction.* To understand the referent system the phenomena has to be analysed and concepts developed. In the programming process this corresponds to designing the classes and their attributes and to organise the classes into inheritance structures.

3. *The level of thought concreteness.* The understanding corresponding to the abstract level is further developed to obtain an understanding of the totality of the referent system. One may also be able to explain why things happen and to predict what will happen.

The following are three fundamental means of organisation for apprehending the real world:

1. *Identification of phenomena and their properties.* In perceiving the real world people identify phenomena and their properties. The result of this is a number of singular phenomena characterised by a selected set of properties.

2. *Classification.* Classification is the means by which we form and distinguish between different classes of phenomena. That is to form concepts.

3. *Composition.* A phenomenon may be understood as a composition of other phenomena.

Classification and composition. Classification and composition are means to organise complexity in terms of hierarchies. Classification distinguish between the classification of phenomena and of concepts.

Clustering Is a mean to focus on similarities between a number of phenomena and to ignore differences. To cluster, is to form a concept that covers a collection of similar phenomena

Generalisation Is a mean to focus on similarities between a number of concepts and to ignore differences. To generalise, is to form a concept that covers a number of more special concepts.

Classification is supported in BETA.
Composition is a means to organise phenomena and concepts in terms of components. One important form of composition is the structuring into whole and parts. Localisation is a means for describing/organising that the existence of phenomena/concepts are restricted to the context of a given phenomenon. The local component phenomena/concepts are dependent upon the composite phenomenon.

Relations. Relation is a common form of abstraction used for organising knowledge. There are three kinds of relations: *One-to-one*,*One-to-many* and *Many-to-many*.

B.2 TangO: Modeling In Style

This section presents a short summary of the article [May et al., 2001]. The paper introduces three new concepts: tangible objects (physical objects with

computational power), habitats (logical contexts for tangible objects) and associations (as a mean for modelling the interactions between habitats, tangible objects and human beings). Table B.2 presents the evaluation of this article.

Conceptual framework	Agent concept	Activity concept	Context concept	Relevance
✓	✓		✓	High

Table B.2: Evaluation of : TangO: Modeling In Style

B.2.1 Brief summary

Ubiquitous and pervasive world

First of all I would like to present the notions of pervasiveness and ubiquitousness.

Ubiquitous - notion introduced by Mark Weiser, researcher at Xerox Parc, in 1988. "Something that is ubiquitous is everywhere or seems to be everywhere at the same time." Collins Cobuild. Something is said to be ubiquitous if is satisfies following properties: appears to be invisible, obedient and/or intelligent, have context-aware behaviour, computationally unnoticeable, unobtrusive to use.

Pervasive - "Something, especially a quality, effect, or smell, that is pervasive, is felt throughout the space or thing." Collins Cobuild. Notion of pervasive computing was introduced by IBM. Pervasiveness include: decentralization, connectivity, diversification.

The changing world. The introduction of the ubiquitous and pervasive computing has/will change the world, and therefore will change our perception of the world, and therefore some physical models will be influenced. The question is what to model and how to model the new emerging reality. Th epaper presents the approach to understand and respond to this challenge. TangO, an abbreviation for Tangible Objects, is a conceptual framework aimed at producing real software for this digitally pervasive world.

Three spaces

The authors identify three spaces - physical, informational and computational, in which humans interact with various artefacts.

Physical space. Interaction with artefacts in their physical form of thing/atoms (we can touch, feel, move, throw them) in physical environment.

Informational space. Interaction with artefacts in their informational form of bits (we can store, copy, modify, erase them). These artefacts can have computational abilities and/or autonomy of acting.

Conceptual space. Interaction with artefacts in their conceptual form of ideas. Authors claim that concepts shape the way we understand the world and how we will act towards the world.

If we would like to model artefacts that are present in more that one space, like PUC devices, we must consider relationships between those spaces - interplay between them and their mutual dependences. Physical space becomes more and more computerized. Informational space is spreading from desktops to everyday life equipment and/or even human bodies. Design of proper interfaces to those new appliances is the challenge. Artefacts have to be made task-sensitive, context-aware, user-centred but seamless and invisible to use. Pleasure of use, artistic appearance, social issues and conceptual correctness are important issues. Therefore conceptual terms influence those artefacts, and the conceptual artefacts are shaped by the physical and informational artefacts used.

Tangible object

The paper introduces the concept of tangible objects, that are physical objects with addition of computational power, that support conceptual, artistic, social and functional activities. These objects exist in habitats and they interact with each other through associations. Other important notion connected with tangible objects is an emergent behaviour, because of the dynamic nature of associations.

The tangible object is an artefact that has been modelled and designed in all three spaces. It can belong more to one or more of them than to other(s). Tangible objects learn and evolve to satisfy the dynamic needs of the environment. They must support the bottom-up growth of the system. They work in emergent behaviours.

The habitat

Tangible objects exist in a logical context, called habitat. The habitat is aware of the tangible objects inside it and can interact with them. The concept of the habitat was introduced to correspond to the world, where the behaviour of human beings depends on the logical contexts they find their selves in. Tangible objects need to behave and interact in different ways, depending on the context they inhabit. Moreover, habitats can be understood from the view of physical, informational and conceptual space.

Authors describe also other features of habitats. Habitats may overlap, can be composed of other habitats, are not isomorphic and they affect each other and depend on each other.

The association

The interactions between artefacts and artefacts and human beings are possible because of the associations between tangible objects and the associations between tangible objects and habitats. The paper introduces the concept of association to capture the interaction between artefacts, that is dynamic, interactive and collaborative. The nature of pervasive world implies that concepts used to model it must support collaborative and interactive behaviours, in dynamically varying contexts.

B.3 Architecture = Abstractions over Software

This article [Jacobsen et al., 1999] concerns how design of software architecture can be seen as a abstraction over the software domain. Table B.3 presents the

Conceptual framework	Agent concept	Activity concept	Context concept	Relevance
✓				Medium.

Table B.3: Evaluation of: Architecture = Abstractions over Software

evaluation of this article.

B.3.1 Brief summary

In every software system, some architecture is present. The problem is that architecture is only implicitly available. The architecture of some software is a model of the software. The thesis in this article is that software architecture is abstractions over the software domain.
It is important to distinguish between organising and abstracting. When organising a certain domain, the elements in the domain is somehow ordered in groups. To abstract from a certain domain is to create concepts in another domain. The new domain holds information about the former domain.

Abstraction

Domains and views. In the analysis phase the problem domain model and usage domain model are constructed from the problem domain and the usage domain through cooperation of between user and developer. In the design phase the developer in the development domain uses the usage domain model to refine the problem domain model into a system domain model.
The architecture model is an abstraction over the system domain model. The architecture model forms the developer's conception of the architecture of the system. The model focuses for example on structure and interaction embedded in the system domain model. The purpose of the architecture model is to understand the system domain model.

Abstraction and modelling. The idea is to distinguish between the referent system and model system, like the conceptual framework of BETA [Madsen et al., 1993, chapter 18]. Then this view can be applied to different domains in the software development process. In this perspective the software domain can be considered as the referent system and the architecture domain forms the model system.

Architecture

When discussing software architecture there is a distinction between the static description of the program and the dynamic execution of it. The architectural information of a program can be obtained by abstracting over either the description or the execution. The description is a static representation of the program.

The dynamic execution can be seen as snapshots with some state and a transition that takes the execution from one snapshot to the next. For each snapshot the architecture can be described. Several types of information can be obtained by applying different perspectives for the abstraction. The same architectural abstraction can be observed from both the description and the execution, but will be expressed differently.

Characterisation: Process of architecture design

- *Architecture design is innovative construction, not just modelling or transformation.* Design of architecture is not just modelling: There are no existing phenomena in the software domain to model. Design of architecture is not just transformation: There is no existing model available.
- *Architecture design implies abstraction not just organisation.* In this case Organisation means arranging and relating subparts of wholes. Abstraction includes organisation in the sense that chunk of substance have to be identified and delimited, but abstraction also involves classification and abstraction.

Characterisation: Notation of architecture design

- *Architecture notation can be informal.* Any notation supports thinking of and description of solutions and model, but it also introduces limitations. To be innovative expressive freedom is needed, for example by use of informal notation.
- *Architecture notation can be several notations.* During innovation of a model several notations can be applied simultaneously, though this often will lead to inconsistency, which has to be resolved.

Characterisation: Principles of architecture design

- *Architecture must be developed explicitly.* The elements included in methodologies for architecture design are still preliminary. The result of the design is not described explicitly.
- *Architecture must be supported by tools.* Principles are usually supported by tools. Tools rely on notation. Because there is no unifying notation and the notation is partially informal, the tools for architecture design are not very powerful.

B.4 Transverse Activities: Abstractions in Object-Oriented Programming

This section presents one of the first articles dealing with modelling of collaboration [Kristensen, 1993a]. The article introduces a concept of transverse activities, relates to the problems discussed by the researchers these days, concludes about the solutions proposed by others and provides new solutions with use of transverse activities.

It describes the modelling of the transverse activities in terms of classification, specialization and aggregation, and the possible ways of implementation of the transverse activities using newly introduced [Kristensen, 1993b] and existing language constructs. Table B.4 present the evaluation of this article.

Conceptual framework	Agent concept	Activity concept	Context concept	Relevance
✓		✓		High

Table B.4: Evaluation of: Transverse Activities: Abstractions in Object-Oriented Programming.

B.4.1 Brief summary

Transverse activity

A transverse activity is an activity executed by several objects in some combination, and it is described separately from the descriptions of the participating objects. The description of the transverse activity must at least include a listing of the components participating in the activity and a listing of the sequence of actions making up the combined directive of the activity. Such an activity cannot exist without components and the components take part in the activity. The activity has an action sequence composed of method activations of the components and possible other transverse activities.

Modelling

In the "Scandinavian school of object-orientation" the view of programming is modelling, which implies, for example, that language mechanisms support conceptual understanding in general. That means that new language constructs may be designed and introduced to the language to support conceptual modelling.

Conceptual framework includes phenomena abstracted into concepts, classification, specialization and aggregation of concepts, and views on the characterization of the phenomena in the extension of a concept. The modelling view on object-orientation implies that objects can be active, i.e. a description of an object may include an action part that models a life cycle of the phenomenon described by the object. Abstraction mechanisms supporting specialization and aggregation must handle active objects.

Implementation

The paper describes a possible way for simulation of the transverse activities with existing language constructs and its crucial limitation. At the programming language level transverse activities can be simulated by the usual mechanisms for classes and objects only as long as we restrict the conceptual understanding to actions sequence, so that objects involved in transverse activities and other transverse activities are not executed concurrently to the transverse activity. This is of course obvious limitation, and therefore we would like to prevent it. The intention is to use slightly modified classes and objects, i.e. with references,

methods, and an individual action sequence (similar to Runnable objects in Java) for both transverse activities and their participants. However the concurrent execution of action part of an object together with execution of methods requested by activities the object is participating in should be supported additionally.

Interactions of the specific activity are described in an activity-object's action part, so that the life-cycle of an active object (component) is described both in its own action part and in various transverse activities. The author discusses various ways of how activities could invoke methods on its participants or initialize other activities. One of the possibilities is to use textbfinterleaved execution, introduced in [Kristensen, 1993b], when the participant executes its action part and transverse activity requests it to execute other of its methods, and both participant and transverse activities are modelled with use of **transverse classes** from [Kristensen, 1993b]. If this would take place in usual active objects, the execution of method by an activity can imply a synchronization with object and possibly require an explicit accept from the object.

Aggregation mechanisms Activity-objects of activity-classes can be aggregated into other activity-objects of other activity-classes. It has important implications for the participants and the directive of aggregated classes and objects.

The directive of an activity-object may be part of the directive of aggregated activity in various ways, such as making the object local or global and activating the object from aggregated directive, inserting the object directly into the aggregated directive, so that its directive will be executed sequentially with aggregated directive in specific point of the later, or with use of special language constructs enabling interleaved or concurrent execution of aggregated activity-objects, if provided by the language in use.

The participants of activity-object may be part of participants of an aggregated activity in many ways, for example the activity-object can be declared as global or local object and its participants are then accessible by usual remote access, or the activity object can be inserted directly into directive of the aggregated activity and the participants of activity are only implicitly accessible from the directive of aggregated activity and only when the inserted object is executed.

Specialization mechanisms Activity-class can be specialized from another activity-class. The specialization has implications for the participants and the directive of the specialized activity objects.

The activity participants may be refined by adding more participants or refining the description of existing participant, for example by rebinding object representing participant to a subclass of its class.

The activity directive may be refined by refining the transverse activities being part of the action sequence or by extending the action sequence, e.g. by introducing a multiple inner mechanisms to support combined execution of directives.

B.5 Activities: Abstractions for Collective Behavior

The article [Kristensen and May, 1996] introduces the concept of an activity as an abstraction mechanism to model the interplay between object, in object-orientation. The main focus in the article concerns modelling of collaboration

Conceptual framework	Agent concept	Activity concept	Context concept	Relevance
✓		✓		High

Table B.5: Evaluation of: Activities: Abstractions for Collective Behavior.

between multiple objects using an activity abstraction mechanism. The conceptual framework of activities with multiple participants are describe concurrently with the elaboration and exemplification on how the activity concept should be understood. Table B.5 presents the evaluation of this article.

B.5.1 Brief summary

Conventional object-oriented modelling lacks support for representing the interaction between objects in a conceptually intuitive way.
The article presents the activity abstraction mechanism, that seeks to capture the interplay between groups of objects throughout at different points in time. An activity can be defined as an interplay between entities over a given time, and is said to be temporal in nature. The activity is a modelling and language mechanism that may be used to create abstraction relating other objects together – describing not merely their participation in the relationship, but their interplay.

Designing with activities

In the article the is a distinction between part-activities and sub-activities. Descendant activities – activities specialised from another (super)activity – are called sub-activities or just activities. Activities that can aggregated to form larger activities (whole-activities) are called part-activities. The aggregation and specialisation can, with a few exceptions, be seen as an adaption of corresponding mechanisms of BETA [Madsen et al., 1993].
The term collective activity denotes the abstraction of the actual sequence of behaviour taken by the participants. The directive is the behaviour description part of the collective activity. A specialisation of an activity refines the directive of the (super)activity.

Simulation of activities

In uni-sequential or single threaded execution, it is a need to integrate the participants behaviour with the progress of the corresponding activity. In the support of activities in the uni-sequential case there is a distinction between the initiating activities and reactive activities.
In initiating activities the participants may be seen as passive object controlled

and activated by the collective activity. In this scenario the activity will have the initiative towards the participants, in a continued guiding manner.
In reactive activities the participants may be seen as active, having the initiative towards the collective activity, calling methods of the activity. In turns, these calls may provide feedback to the participants to direct their actions.

In multi-sequential execution the action part of an activity object describes the interaction of the activity with other entities. The collective activity is seen as a supplementary part of the life cycle of it's participating entities – the life cycle of such an entity is described both in it's own action part and in the various collective activities in which it is participating.

Experiments

The activity abstraction has been compared to the Mediator pattern[Gamma et al., 1995]. The result is that the Mediator pattern appears to be similar to the characteristics of the collective behaviour of the activities. While the overall architectural techniques are similar, motivation and focus for collective behaviour is different.
The emphasise of the Mediator pattern is to promote loose coupling and encapsulate object interaction. The activity abstraction mechanism also seeks to give force to these goals, but the emphasise is to represent and give force to the properties of interaction – collective and otherwise.

B.6 Complex Associations: Abstractions in Object-Oriented Modeling

The following section presents [Kristensen, 1994]. In the article, the author treats relations between components as phenomena and tries to model them with use of objects, not only by associations or references.

Conceptual framework	Agent concept	Activity concept	Context concept	Relevance
✓				Low

Table B.6: Evaluation of: Complex Associations: Abstractions in Object-Oriented Modeling

Table B.6 presents the evaluation of this article.

B.6.1 Brief summary

Associations

Relations between objects are important for understanding the organization of and the cooperation between objects and therefore may be supported in object-oriented analysis and design. Activities and relations are existing phenomena, that have natural conceptual identity of their own, outside any other individual component.

Author distinguishes two types of associations: **an implicit association**, that describes a relation between an object and objects local to this enclosing object; and **complex association**, that describes an explicit relation between local objects in different enclosing objects. Such associations are in the main focus of the article. They are described by class and their objects have the usual properties, such as attributes and methods.

Modelling with use of associations

In traditional object-orientated modelling the focus is on identification of concepts and their mutual relations and then describing these with use of classes and associations between them. Traditionally it is described in one, flat model, so that associations between all the classes are presented in one diagram, which does not reflect exactly the way we understand such complex systems, and could be highly unreadable.

The author proposes to use more structured way for describing the knowledge about interdependencies between concepts. The solution is to describe several levels that correspond to abstraction levels. At the top level we see mostly several components and we can name abstract association between them. In the next level, we identify and describe sub-components inside enclosing components. The associations between specific sub-components from different enclosing components can exist only because of the association between enclosing components, and they are local to the association on the top level. The domains of those associations are local to the domains of top level association.

Language mechanisms

Author's main point is that the complex structures described previously seem not only to be practical ways to illustrate a model, but are the ways to think bout it. The concept of locality was proposed, and it is available through supporting language mechanisms for both the concepts and associations and at many levels. The description of a local component is meaningful only in connection with the description of the enclosing component. The same hold for associations, the description of local association is meaningless if not connected with the description of complex association.

Author introduces slightly modified classes and objects as a means for describing implicit and complex associations. The modification includes that the association class is declared for specific class domains, that correspond to classes of objects that are related by the association or classes of roles that these objects play in the association. The notation supports one-to-many and many-to-many associations. The proposed language mechanisms support both the static relational structure and dynamic access to the objects of the complex structure.

The author has restricted himself in several points. Only mechanisms for binary associations are defined and only in the form of one-to-one and one-to-many. Concerning associations, they are discussed as passive. Author points the [Kristensen, 1993a] for the discussion of activities. Furthermore the abstraction process in terms of exemplification, specialization and aggregation is not discussed as including the attributes or methods of objects.

B.7 Object-Oriented Modeling with Roles

The article [Kristensen, 1995] elaborates on the concept of role. It presents the concept and its characteristics, uses roles in modelling, and tries to simulate roles by existing concepts. This section presents both the summary of the article and conclusions and relations to my project. Table B.7 presents the evaluation of this article.

Conceptual framework	Agent concept	Activity concept	Context concept	Relevance
✓	✓			Medium

Table B.7: Evaluation of: Object-Oriented Modeling with Roles

B.7.1 Brief summary

Roles

Objects are often described as isolated entities. They however relate to each other, interact with other objects and they play certain roles in those relations and interactions. The perspectives that objects have about each others are based on the way they treat each other, and they define the roles that they play towards each other. The paper focuses on roles as a mean of refining the understanding of an object. The concept of role is intuitive and important in the modelling of real world phenomena.

The overall motivation for roles is to allow special perspective on a phenomenon. This perspective is used by other objects as a restricted way of knowing and accessing the object. The perspective is a set of selected properties of phenomenon, modelled by a set of methods, and it is modelled by the role. Therefore the purpose of the roles is to give restricted views on the complex objects.

Perspectives on an object depend on other objects and may change over time. This means that the set of properties of an object is dynamic. Perspectives are modelled by roles, which of course include set of methods, but also the state when they are instantiated in addition to the objects. The roles may change, they may exist simultaneously and there may be other important relations between roles.

The author introduces the concepts of intrinsic object (the one to which the role is allocated) and intrinsic methods (methods of such an object). The instantiations of roles are called role instances and the methods of roles are named intrinsic methods. Moreover it is pointed out, that there exist roles of roles.

Characteristics of roles

The characteristics of the role include:

- Visibility. The visibility of an object can be restricted to the methods of a role (including object methods, but excluding methods of other roles). This provides also possibility for multiple classification of an object.

- Dependency. The role depends on the object, and its methods can be defined in terms of objects' methods, but not the other way.
- Identity. The object and its roles are seen from outside as one indivisible entity.
- Dynamicity. Roles can be dynamically added to and removed from an object.
- Multiplicity. In general several instances of a role may exist for an object at the same time.
- Abstractivity. Roles as concept can be classified and organized in generalization and aggregation hierarchies.

Simulation of role

Author tries to determine how the roles could be simulated with existing concepts, such as specialization, aggregation and association. He discusses problems with those approaches in relation to the proposed characteristics of the role concept.

Role abstraction

The abstraction processes: classification, specialization and aggregation can be performed on roles. The specialization and aggregation introduce relations between methods of the roles, which define the availability of the methods.

Specialization. Specializations of roles are roles. Some methods of a specialized role can be inherited from more general role, some can be modified (the description is preserved, but the prescription changes) and there can be additional added methods in the specialized role.

Aggregation. Roles can be aggregated from other roles, for instance part-roles, which supply various additional methods, that can be applied to form methods of the whole-role. If the method of part-role is also a method for the whole-role, it is called hereditary. Hidden methods are available only in part-roles. There can exists new methods in whole-role, and they are called emerging methods.

Associations

Author models interactions between objects with roles by associations. Associations have properties and methods, as normal objects. They support the limitation of visibility of roles methods according to specific interaction. There could exist roles for associations. The idea of association is, however, not in the scope of the article.

B.8 Roles: Conceptual Abstraction Theory & Practical Language Issues

The article [Kristensen and Osterbye, 1996] treats the notion roles in both a theoretical and a practical perspective. The theoretical perspective describes a role in a conceptual abstraction point of view, whereas the practical perspective concerns programming language mechanisms.

Conceptual framework	Agent concept	Activity concept	Context concept	Relevance
✓	✓			High

Table B.8: Evaluation of: Roles: Conceptual Abstraction Theory & Practical Language Issues

Table B.8 presents the evaluation of it's relevance to this thesis in general. The article gives a thorough description of how the role concept can be seen from a conceptual abstraction perspective and how in can be integrated in the conceptual framework.
Even though it is only briefly mentioned in the article, it holds useful information of the relation between the agent concept and the role concept. If an agent is assumed to be a specialisation of an intrinsic object, the mutual relation between an agent and a role is similar to the relation, described in the article, between an intrinsic object and a role. Due to that fact, this article i highly relevant for the project in general.

B.8.1 Brief summary

The notion of a role of an object subsumes a set of properties of the object. A role is defined as follows:

Role: A role of an object is a set of properties which are important for an object to be able to behave in a certain way expected by a set of other objects.

This means that other objects need to see the object in a certain restricted way. A role must be played by some one, and all involved parties are aware of that, in the article the "someone" is referred to as the intrinsic object.

Conceptual programming

The theoretical discussion of roles builds on conceptual programming, where programming is regarded as a modelling process of some referent system, where the phenomena and abstractions from the referent system are expressed in a programming language supporting abstractions, which are based on a general understanding of phenomena and concepts.
The notion of a role is a specialisation of the general notion of a concept. What distinguishes roles from concepts in general is the question of individuality. It is a mandatory property for roles that they are roles for something.
In the article the authors distinguish between roles and role instances. A role is a

description and models an abstraction. A role instance has state and behaviour and models a perspective on a phenomena.

Conceptual abstraction. It is investigated what kind abstraction process "roleification" is when dealing with roles in a conceptual perspective. The conclusion can be summarised as follows: Roleification has major similarities with specialisation and classification, but important distinctive differences.
Roles have been established as a specialised kind of concepts, and according to the conceptual framework, there are a number of abstraction processes which can be applied to concepts in general, and therefore to roles as well:

- A subject (object with one or more roles) is classified by any of its roles.
- Exemplification is possible for roles.
- Roles may be specialised.
- Roles may be generalised.
- Roles can be aggregated from roles – with restrictions.
- Roles can be decomposed into roles

Role properties. The relations between the properties of the intrinsic object and the role instance is described as follows:

Intrinsic properties: The properties of the intrinsic object are accessible through the role instance. This corresponds to inherited properties in specialisation and hereditary properties in aggregation.

Extrinsic properties: The properties of the role instance are properties of the subject. This corresponds to additional properties in specialisation and emerging/hidden properties in aggregation.

Property-role: Properties may be seen as special concepts. This enables the possibility of having roles for properties, property-roles. This means that property of the role instance is a role for a property of the intrinsic object, and that the properties form a combined property.

The relation between the properties of an intrinsic object and the properties of a role are intrinsic, extrinsic and property-role.

Language constructs

A role is quite similar to a class, except that the role is intended as a role for something. This means that roles cannot be instantiated by themselves, but must be bound to an existing object.
To be able to change roles dynamically it is a need to be able to bind, unbind and transfer roles. A role instance can only be transfered between objects of the type the role is declared for.
If the role concept isn't directly supported by the implementation language the decorator pattern described in [Gamma et al., 1995], can be used instead. The decorator pattern is not intended to implement the role concept, and as such it doesn't reflect the conceptual issues.

B.9 Subjective behavior

The article [Kristensen, 2001] defines the notion of subjective behaviour and support of subjective behaviour by means of object-oriented language mechanisms is investigated.

Conceptual framework	Agent concept	Activity concept	Context concept	Relevance
✓	✓		✓	Medium

Table B.9: Evaluation of: Subjective behavior

Table B.9 presents the evaluation of this article.

B.9.1 Brief summary

In object-oriented modelling an object reacts objectively to an invocation of a method in the sense that it is given by the denotation of the method invocation which description for the method is chosen. Subjective behaviour of an object means that it is not objectively given which description is chosen.
The motivation for subjective behaviour is to support more autonomous and evolutionary objects for modelling and design.

Modelling

In the context of object-orientated modelling the following definition of subjective behaviour is given:

Definition: A denotation of a method invocation for a given object is said to be subjective if different method interpretations can result.

Definition: A mechanism supports subjective behaviour if it can be used for a subjective method invocation. A subject is any entity that offers subjective behaviour.

Forces of real world behaviour Various "forces" can affect an object's behaviour in real world: A sender force, a context force and a state force.

- The sender of a message can affect how an object responds – it can be achieved by the role mechanism
- The context in which the message is send affect how an object responds – the object measures the state of the context which then can influence its reaction.
- The state of an object affects its responds – object should be "alive" and then its state, internal or external, could change during its life-cycle.

Mechanisms

To achieve subjective behaviour of objects at method-level, a single method can have multiple implementations.

Example: Method `m` of class `C` has several implementations, namely `im1`,`im2` and `im3`. Upon invocation of method `m` of object `c` it is entirely up to object `c` to decide which of the implementations to be chosen as response to the invocation – the invoking object has no influence on which implementation is selected. In this case it is assumed that the type of invoking object determines the choice.

Objectifying behaviour is another approach to support subjective behaviour. The state pattern[Gamma et al., 1995] is an example on a pattern to support subjective behaviour by objectifying the behaviour of identified states. It is important to notice that that state specific behaviour is localised and delegated to separate objects, using the state pattern. The organisation obtained by the state pattern is similar to the organisation obtained by delegation and polymorphism.

Other approaches to support subjective behaviour based on polymorphism.

Timing polymorphism is a mechanism for supporting communication between two distributed real-time objects involving timing constraints. The client object specifies time constraints on the communication, and the server objects can then dynamically select among multiple execution bodies.

Performance polymorphism refers to the concept of maintaining and selecting among multiple implementations of a method that carry out the same task and differ only in their performance measures(execution time, memory space, system configuration, result precision, etc,).

Summary

The following list summarises the results presented in the paper:

- Subjective behaviour has been defined in general terms, independent of object oriented mechanisms.
- Simulation of subjective behaviour by using the state pattern.
- Language mechanisms like the role, multiple implementation, polymorphism are means by which conceptual concepts including subjective behaviour can be supported.

B.10 Associations: Abstractions over Collaboration

This article [Kristensen, 2003] focuses on associations as abstraction over collaborations. This concept is motivated and explored. Table B.10 present the evaluation of this article.

Conceptual framework	Agent concept	Activity concept	Context concept	Relevance
✓		✓		High

Table B.10: Evaluation of: Associations: Abstractions over Collaboration

B.10.1 Brief summary

Lack of support for collaboration in existing notations and diagrams

Description and prescription of collaboration between participants is poorly supported by existing notations and diagrams, which are mainly based on object-centric modelling and programming. There exists the relation abstraction described in [Rumbaugh, 1987], but it is an abstraction over structural aspects only, interaction aspects are not covered by relations. The notion of abstraction, introduced in this paper, is seen as an abstraction over both structural and interaction aspects of collaborations. Additionally, an association supports both description and prescription at both abstract modelling and concrete programming levels. The article explores one of the possible mechanisms for expressing the actual interaction in association - event-based mechanism.

Collaboration

The approach presented in the paper is inspired by the change from more traditional systems toward pervasive and ubiquitous ones. Authors discuss tangible objects existing in habitats, collaborating with each other and moving between habitats. The notion of association is a mean of capturing, describing and prescribing planned or spontaneous collaborations between tangible objects.

Modelling

Object-centric modelling. The notion of association is already present in object-oriented modelling and they are implemented by means of references. Classical object-oriented modelling and programming claims that object support encapsulation ans are self-contained, but on the other hand "no object is an island" [Beck and Cunningham, 1989]. Authors of the paper claim that the focus here is on structure (instead of function) and on methods (instead of processes), and this attempt forms essential problems because it emphasizes an object-centric point of view.

Non-object-centric modelling. Various approaches for non object-oriented programming exist and some of them are briefly described and evaluated in the article. These include relations from [Rumbaugh, 1987], associations in UML, sequence and collaboration diagrams from UML, complex associations [Kristensen, 1994], subject oriented programming and subjective behavior, activities from [Kristensen, 1993a] and [Kristensen and May, 1996] and roles from [Kristensen, 1995] and [Kristensen and Osterbye, 1996].

Programming. In object oriented programming languages references are used to support relations between objects. The reference is statically bound to the

class, and the value of the reference changes dynamically, and it is qualified by class, which determined what types of objects might be referenced. The reference is used for methods invocations, possible in various places in various methods, which makes the use of reference separated from the reference.

Association

The association introduced in the article is an alternative to object-centric modelling. The association is a description from which instances might be dynamically created, deleted and associated with objects. It is qualified by a class, which determines what kind of objects can be associated with instance of association. The concept is used to capture the collaboration between participants that have specific roles in relation. It sequences rules for interaction among participating objects. Association is seen as a relation between roles.

The authors present the conceptual abstraction for the association. The properties of an association concept include the roles (together with qualifications of roles), the properties for the roles and the sequencing rules for interaction. Concerning the specialization, the properties of an association may be also properties of specialized association, they can be refined or there can be new properties added in the specialized association. In the aggregation, the property of part association may be hidden, may exist in whole association or a property of whole association may be aggregated from number of properties from part associations.

Communication

Participant act and observe with use of event-based mechanisms. It is asynchronous broadcast of information to potential receivers, which are determined by instances of associations. The information in this case is a concepts itself, has type, and actual piece of information is an instance of this concept.

B.11 Multi-criteria optimization of ball passing in the simulated soccer

This article, [Greber et al., 2005], propose a solution to the task of determining to what teammate the ball should be passed, and hence to what point the ball should be sent. The solution is based on a set of three criteria, tactical gain and two time balances, which should be balanced while selecting an optimal point on the field for passing the ball to. They claim that the advantage of this set of criteria is that direct and leading passes can be treated the same way by a uniform method.

B.11.1 Brief summary

The algorithm presented in [Greber et al., 2005] propose a solution on how to find an "optimal" pass-receiving point when the ball is passes from a player to an teammate using multi-criteria optimisation. The article consist of three major parts, defining the criteria, generating the prospective pass-receiving points and choosing the optimal pass-receiving point.

Defining the criteria

To find the point to which the ball should be sent, three criteria, C_1, C_2 and C_3, have be define:

C_1: Tactical gain; this criterion evaluates the tactical gain of a prospective pass-receiving point. The best tactical point to which the ball can be passed is defined to be the centre of the opponents goal, and the worst is in the centre of the own goal.

C_2: Time Balance; this criterion evaluates whether the prospective pass-receiving point lies inside the reach-circle of a teammate or not. If the pass-receiving point is outside the reach-circle, the point should be discarded immediately.

C_3: Time Balance; this criterion evaluates whether an opponent can reach the pass-receiving point before a teammate or not. If an opponent can reach the pass-receiving point before a teammate, the point should be discarded immediately.

Generating the prospective pass-receiving points.

Figure B.1 shows a screen shot of RoboCup soccer simulation, for which the algorithm has been developed, with the reachable prospective pass-receiving points.

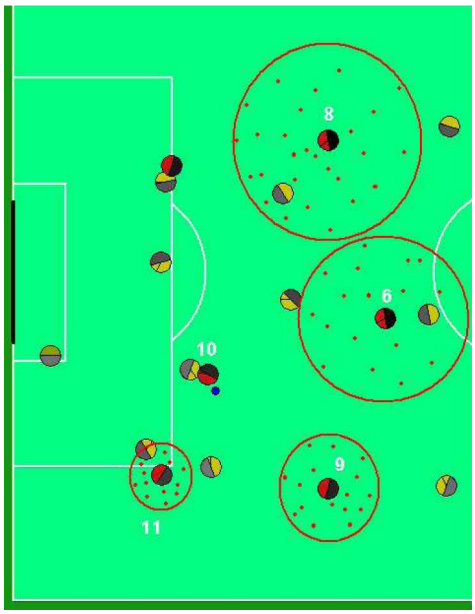


Figure B.1: A screen shot presented in [Greber et al., 2005] illustrating the prospective pass-receiving points which are within reach of a teammate.

The circles around each of the teammates are the individual player's reach. The radius of the reach circle is dependent on the individual player's running speed,

the distance between the ball and the ball's travelling speed. Greater distance between teammate and ball equals greater reach of the teammate, if the running speeds for all the player are equal.
An algorithm for generating the prospective pass-receiving points, the dots inside the circles at figure B.1, is presented in [Greber et al., 2005].

Choosing the optimal pass-receiving point

Now that desired number of prospective pass-receiving point has been generated, each of the points are evaluated with regards to the three defined criteria. If a point fulfil all of the three criteria, a 3-tuple consisting of a value for each of the three criteria is made and the point is saved, otherwise the point is discarded. All the dominated 3-tuples can then be eliminated form the solution set of 3-tuples.
Finally, the criteria will be used one-by-one to remove the worst solution, according to the individual criterion, until a single solution remains.
If for instance criterion C_1 is twice as important as the two other, this criterion can be allow to remove two "wrong" solutions each time the remaining two only removes one.

Appendix C

Conceptual modelling of activities and contexts

This chapter describes our experiments on modelling collaboration and environments/groups, where the referent system is a soccer scenario, at the conceptual level by using the newly characterised concepts activity and context.

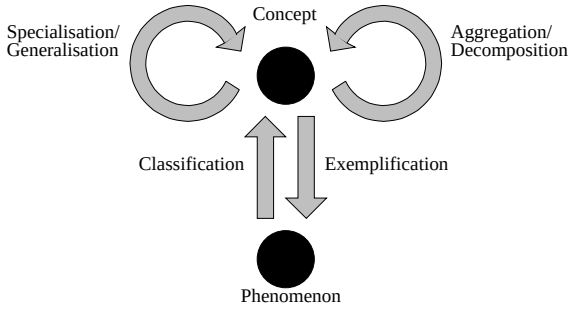


Figure C.1: General abstraction model.

Figure C.1 depicts the general abstraction model, as depicted in [Kristensen, 2003], for conceptual modelling, which we will apply in a soccer scenario.

C.1 Conceptual modelling using the activity concept

We can apply the principles of conceptual abstraction and modelling, classification, specialisation and aggregation, to the activity concept.

C.1.1 Classification/exemplification

Classification relates concrete phenomena to abstract concepts according to their common properties. Properties of the activity include the roles of participants and their properties and the directive which models the sequence of their interaction.

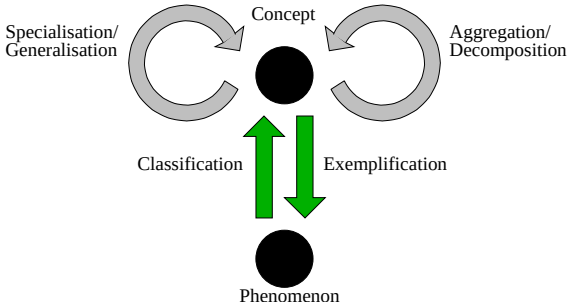


Figure C.2: Classification and exemplification.

Figure C.2 highlights the relevant part, classification and exemplification of concepts, of the general abstraction model. The following presents an example of classification and exemplification of activities in the referent system:

Example: In the referent system we identify various examples of collaboration(phenomena) between soccer players such as: throw-in, corner kick, ball passing, etc. All of these phenomena we classify into the concept of activity. Now that the collaboration phenomena have been classified into activity concepts respectively, we try to identify the roles and directive of one of the activities. For instance, the throw-in activity models the situation where an agent (player) from the opponent team has kicked the ball the touch-line of the soccer field. To engage in the throw-in activity agents (players) that can play the following roles are needed, an agent for playing the "thrower"-role and one or more agents for playing the receiving-role(s). When the roles in the activity are filled the directive of the activity is executed. In the case of throw-in the directive of the activity could specify a sequence of actions as follows:

1. Receiver(s) should reposition themselves to obtain unmarked location(s).
2. The "thrower" should observe how receivers reposition.
3. The "thrower" selects a receiver and throws the ball to him.

C.1.2 Specialisation/generalisation

Activities can be both specialised and generalised. We distinguish the following relations between the properties of a sub-activity and a super-activity:

- The properties of an activity also apply to its sub-activities.
- Additional properties may be introduced in sub-activities.
- Properties of an activity may be specialised in its sub-activities.

Figure C.3 highlights the relevant part, classification and exemplification of concepts, of the general abstraction model.

The following presents an example of specialisation and generalisation of activities in the referent system:

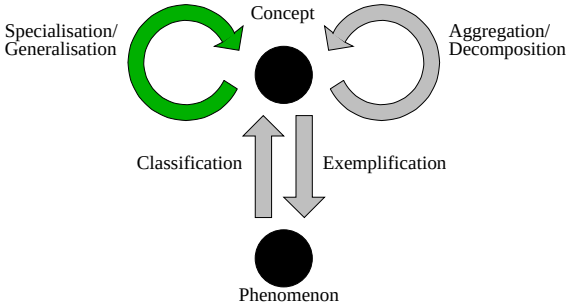


Figure C.3: Specialisation and generalisation.

Example: In the referent system we have classified collaboration between soccer players as activities. The activities may be organised into sub-activities according to their common properties. E.g. the activities throw-in and corner-kick are both specialisations of the more general dead-ball activity which again is a specialisation of the activity concept.

We model collaboration as activity and we organise specific activities into specialisation-structure. For example, both of the specialised activities, throw-in activity and corner-kick activity, inherit a role for resuming the game from the more general dead-ball activity. The throw-in activity then adds additional properties to the "resuming"-role, allowing the agent playing this role to pick up the ball using his hands. Furthermore one of the receiver-roles could be specialised to ensure that a participant has special properties. E.g. The corner-kick activity could refine one of its receiving roles to be good at heading the ball. Both of the specialised activities require a refinement of the directive to handle the situation when the ball is kicked outside the field, which is not the case for all specific dead-ball activities, i.a. free-kick, penalty-kick.

C.1.3 Aggregation/decomposition

Complex activities can be seen as aggregation of more simple activities. We distinguish the following relations between properties of whole-activities and part-activities.

- An emerging property of an (whole-) activity may be aggregated from a number of properties of some part-activities, but is not explicitly present in any of the part-activities.
- A property of a part-activity may become a hereditary property of the whole-activity.
- A property of a part-activity may be concealed and not be a property of the whole-activity.

Figure C.4 highlights the relevant part, aggregation and decomposition of concepts, of the general abstraction model.

The following presents an example of aggregation and decomposition of activities in the referent system:

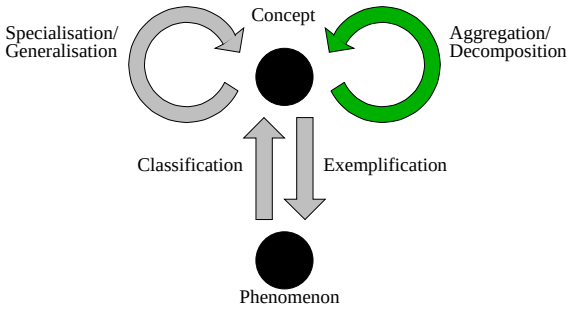


Figure C.4: Aggregation and decomposition.

Example: The identified collaboration phenomena may vary in complexity depending on among other things, the number of player involved. This is also reflected in the activity into which the phenomena are classified. In the referent system the through-pass activity can be seen as an aggregation that can be decomposed into kick-ball activity and run-to-ball activity.

When modelling collaboration such as through-pass as a through-pass activity, we aggregate it from more simple activities. The two part-activities, kick-ball and run-to-ball, only require one role to be filled each to execute whereas the whole-activity through-pass requires two filled roles, namely the two hereditary roles from kick-ball and run-to-ball, respectively. The directive of the through-pass activity(whole) is aggregated from the directives of the part-activities, and could be as follows:

1. The agent playing the kick-ball-role should kick the ball to a location (in front of the receiving agent).
2. The agent playing the run-to-ball-role should run to the location that the ball has been kicked.

C.2 Conceptual modelling using the context concept

We can apply the principles of conceptual abstraction and modelling to contexts. I.e. applying the principles of classification, generalisation and aggregation.

C.2.1 Classification/exemplification

Common properties of contexts (physical and logical) are among others: their temporal nature, they provide activities for agents, they are delimited by boundaries physical or conceptual respectively.

The following presents an example of classification and exemplification of contexts in the referent system (see figure C.2 for a highlight of the relevant part of the general abstraction model):

Example: In the referent system we identify phenomena like large grass areas with specific chalk patterns representing a soccer field. Even though the

chalk patterns vary is size we classify these phenomena into physical contexts. From time to time groups of people may enter the soccer field (or physical context) to play a game of soccer. These groups(i.e. soccer teams) of people we identify as phenomena that we classify as logical contexts. In both cases the concepts can be exemplified, for instance; the home ground of Real Madrid CF (example of soccer team and logical context) is called Santiago Bernabéu (example of soccer field/stadium and physical context).

We model soccer fields as physical contexts, because a soccer field is basically an area delimited by the outer lines of the chalk pattern encouraging people to play soccer.

In the same manner we model a soccer team as a logical context, due to the fact that a soccer team can be considered as a ubiquitous and dimensionless entity only delimited by "what is appropriate to do, and when to do it" and additionally by some agreement of being a member of the team.

C.2.2 Specialisation/generalisation

Contexts can in certain cases be considered specialisations of a more general context, to form a "is a" relationship. We distinguish the following relations between properties of specialised contexts and a general context.

- The properties of a context also apply to its sub-contexts.
- Additional properties may be introduced in sub-contexts.
- Properties of a context may be specialised in sub-contexts.

The following presents an example of specialisation and generalisation of contexts in the referent system (see figure C.3 for a highlight of the relevant part of the general abstraction model):

Example: In general soccer fields consist of several different areas, we identify these phenomena and classify them into concepts such as: goal area context, penalty area context, centre circle context and corner arc context, respectively. All of these areas can be considered as specialised versions of a general area context. All of the specialised contexts share some common properties, but they may also define new properties. For instance, if a foul is committed normally the opponents are awarded free-kick, but if the foul is committed inside the penalty area they are awarded a penalty-kick.

We model a general area as a physical context with all the common properties of an area. The general area context can be specialised into sub-contexts like goal area context, penalty area context, etc. These sub-contexts are physical contexts too, and they have the same properties as the general area context but some of the properties may be specialised and some of the sub-contexts may introduce additional properties.

C.2.3 Aggregation/decomposition

A contexts may be aggregated from various contexts in a whole/part relationship with the following properties:

- An emerging property of a whole-context may be aggregated from a number of properties of some part-contexts, but is not explicitly present in any of the part-context.
- A property of a part-context may become a hereditary property of the whole-context.
- A property of a part-context may be concealed and not be a property of the whole-context.

The following presents an example of aggregation and decomposition of contexts in the referent system (see figure C.4 for a highlight of the relevant part of the general abstraction model):

Example: As earlier mentioned we identify various phenomena in the referent system and classify them into: goal area context, penalty area context, centre circle context and corner arc context (all physical contexts). A soccer field (physical)context can be aggregated from: two goal area contexts, two penalty area contexts, one centre circle context and four corner arc contexts. We model the soccer field (whole) as a physical context by aggregating the earlier mentioned contexts goal areas (parts), penalty areas (parts), etc. also modelled as physical contexts.

If we take a closer look at a soccer team (logical)context we identify phenomena from the responsibilities of the different members of the team that form the concepts of: defence context, midfield context and offence context, respectively. I.e. the soccer team concept can be decomposed into smaller groups (logical contexts). We model the logical context soccer team (whole) as a aggregation of the logical contexts: defence context (part),midfield context (part) and offence context (part).

Appendix D

Special features and design of observed collaboration strategies in FIFA 2005

This appendix section present some of the special feature observed in FIFA 2005 and some experiments of designing the observed collaboration strategies for the SoccerLab application.

D.1 Special features in FIFA 2005

FIFA 2005 holds a lot of special features, which also may be the main reason why it is one of the most popular soccer games. This section will only concern a small subset of all the special features contained in the application, and mostly the ones which are relevant to the SoccerLab simulator.

D.1.1 The squad screen

One of the special features worth mentioning is the squad settings screen. A screen-shot of the squad screen is presented at figure D.1. The squad screen allows the user to set the team formation to one of the many predefined formations, e.g. 4-4-2, 4-3-3, 5-4-1, etc.
The user can also modify the starting-line to suite the chosen formation, and choose which of the players that should be substitutes and who that should be reserves. To get a even more detailed view of the team the user can choose the see the player biography for each of the players. A more detailed description of the player biography is given in the following section.

D.1.2 The player biography screen

Another of the special features worth mentioning is the player biography, which every individual player has, figure D.2 presents two screen-shots of a player biography. The idea of the player biography in FIFA 2005 is that the user is allowed to modify an individual player's predefined skills. The rules for modifying a player's skills are very simple, each player has limited number of skill-points

Figure D.1: A screen shot of FIFA Football 2005 squad screen.

depending on the corresponding real life player's skills. The user is then allowed remove skill-points from one category an add them to another, but the user doesn't have the ability to introduce additional skill-points exceeding the limit of total skill-points for the specific player. When a player's skills have been modified, the virtual player will no longer reflect the real life player as intended by the developers of FIFA 2005.

Each player in FIFA Football 2005 has 24 different skill that can be modified. The first 12 skills are presented at figure D.2(a) and the last 12 at figure D.2(b). Looking at the 24 skills actually gives a slight view on how the developers of FIFA 2005 consider a soccer player. The 24 skills representing each player gives a lot of possibilities for controlling the players in various situations. E.g the HEADNG value at figure D.2(a) defines how successful the player is when he is heading the ball. If this principle of the players having skills was to be applied in the soccer simulator prototype version 1 it would too complex to introduce all of the 24 skills at once, but some of the obvious skills like; stamina, acceleration and shooting power from figure D.2(a) and tackling and dribbling abilities from figure D.2(b), could become candidates for new extensions for the next prototype version.

(a) Screen shot of first half of the FIFA 2005 player bio screen.

(b) Screen shot of second half of the FIFA 2005 player bio screen.

Figure D.2: A screen shot of FIFA Football 2005 player bio screen.

D.2 Design of observe collaboration strategies identified in FIFA 2005

This section will mainly focus on adapting the identified collaboration strategies in FIFA Football 2005 to the SoccerLab simulator by using the concepts: agent, activity and context.
In the following it should be understood that activities are written in *italic* font and contexts are written in Sans-serif font.
Before the proposals of how the identified collaboration strategies can be introduced in the SoccerLab simulator are given, a new context is presented.

D.2.1 Introduction of agent vision

To simulate agent/player vision all the players are equipped with an additional context called view context. The responsibility of the view context is to keep track of the nearest team mate and opponent, in front of, behind, to the left and to the right of the player. It is not a necessity that each of the four directions are updated equally often, in fact to make a more realistic simulation of a real life player the behind position shouldn't be updated as frequently as the "in front" position, to simulate that the player has to turn his head to get information on what is happening behind him.

D.2.2 Pass ball backwards

The "pass ball backwards" collaboration strategy, as presented in section 3.2.1, page 42, can be introduced in the soccer simulator by using the AAC concepts in the following manner.
When the view context, of player in ball possession, detects an approaching opponent it should provide an activity called *pass ball backwards* . This activity is said to be "half-baked" which means that it requires the player it self to be present, which he obviously is, and that a playable team mate is position in the proximity behind him.

To determine whether there is a playable player behind him or not, the player asks his view context for the team mate behind him, if the distance to the opponent behind him is lager than the distance to the team mate, the team mate is playable and the *pass ball backwards* activity can be executed. In most cases this will result in a safe pass, but because the positions in the view context aren't updated equally often the situation behind the player may have changed and an opponent will occasionally have a chance of intercepting the ball.

D.2.3 The triangular pattern

The triangular pattern strategy can implemented by the AAC concepts, by using the same principles as for the *pass ball backwards* activity.
When the view context detects that the opponents are approaching and that a team mate is present behind the player, it will provide the *triangular pattern* activity as well as the *pass ball backwards* activity, the player can then choose one of them by some probability factor.
The requirements for choosing the *triangular pattern* activity are that the player in ball possession has a playable player behind him, and that the playable player behind him has a playable player to the left or right of him, depending on the field position. Dependent on the positioning of the players the *triangular pattern* activity sequentially calls three *pass ball* activities with the desired directions. Before each pass in the triangle the passing player should determine if the receiving player is still playable, if not the collaborative should be stopped to avoid loosing the ball.
In the general case it shouldn't be a requirement that the first pass has to be backwards, but instead the activity should require that a chain of two playable players, besides the one in ball possession, can be made. By using this approach the usage of the *triangular pattern* activity can be more flexible than if it is required that the first pass always should backwards.

D.2.4 Through-pass

The through-pass strategy can be introduced in the soccer simulator by using the AAC concepts in the following way.
When the "in front" part of the view context of the player in ball possession detects a team mate at the edge of the opponent's penalty area, the ball possessing player's view context will provide an activity called *through-pass*. If the *through-pass* activity is executed, the ball possessing player will pass the ball in front of the team mate straight ahead and past the opponents. Next the *through-pass* activity makes the team mate run towards the ball, and it's now up to the team mate to choose a suitable activity, provide by the soccer field context, to score a goal.
The *through-pass* activity can also be used for other purposes than scoring, because it is the team mate's responsibility to choose a suitable activity when or if he reaches the ball. Due to that fact it might be better to come up with a more general name for this activity later on.

D.2.5 Cross the ball to score

The "cross the ball to score" strategy is in fact just a special case of passing the ball, but for now this fact will not be taken into account. The "cross the ball to score" strategy can be achieved by using the AAC concepts in the following manner.
When the ball possessing player's view context detects that the current field position is close enough to both the touchline and the opponent's goal-line it will provide the *cross the ball to score* activity. When the *cross the ball to score* activity is chosen for execution the ball possessing player will make a long pass in front of the opponent's goal and behind their defencive line. The *cross the ball to score* activity will then inform the offencive team mates, and maybe some mid-field players too, to run towards the ball in front of the goal. If a team mate receives the ball it is his responsibility to choose a proper activity provided by the soccer field context to score a goal.

D.2.6 Defencive strategies

The two defencive strategies observed in FIFA Football 2005 and presented in section 3.2.1, page 45, have very few differences. The main differences between the two strategies are the field position of the defencive team and whether the defencive wings will "attack" the opponents or hold the defencive line.
Due to the high degree of similarity, the two strategies are considered as a single strategy and then introduced in the SoccerLab simulator using the AAC concepts.
A soccer team can be considered as a social context. When the team context detects that the team is no longer in ball possession it will provide a *defencive* activity to the players in the team. When the *defencive* activity is executed the players will run to their defencive positions and start their defencive activities which follows two simple rules:

1. If a player is far away from the ball he will try to mark an opponent.
2. If the player is close to the ball he will try to tackle the opponent to stop the attack.

An important aspect worth mentioning is that the mid-field players as much as possible should participate in the defencive duties, to give the defencive team the player majority which in some cases could ease the task of defending.

Obtaining two strategies from one activity.

To replicate the two defencive strategies identified in FIFA 2005, the teams in the soccer simulator could be extended with a value stating whether a team's defencive strategy is pressing or containing, similar to the approach in FIFA 2005 described in section 3.2.1, page 38.
To permit the teams defencive activity to be influenced by the choice of defencive strategy, pressing or containing, the role concept can be introduced to the players. In this case three roles should be considered: left wing, right wing and centre.
If the team's strategy is set to containing, the defencive players' roles will actually be ignored and hence the line of defencive players will be contained as

well as possible, which is the approach in SoccerLab prototype version 1. The containing strategy will instead influence the players to take more defencive field positions.
If the team strategy is set to pressing instead of containing, the players' roles will come into play. The defencive players holding the role of either left wing or right wing, will be more aggressive to put pressure on the opponents where the defencive players holding the centre role will be more hesitating like in the containing strategy.

Appendix E

Statek Report

A FUSS research project supported by it-vest. The report describes the design of the two activities pilot on board and defencive positioning.

Interplay between Agents, Habitats and Activities

The design document of "Pilot onboard" activity (Ship Simulator)
and "Positioning" activity (Soccer Simulator)

Authors:
Steffen Jensen
Beata Hargesheimer

E.1 Introduction

During the spring of 2005 two simulators, a ship simulator and a soccer simulator, have been independently developed. The ship simulator has been developed as part of a FUSS research project and the soccer simulator has been developed as a master thesis project at the Mærsk Mc-Kinney Moller Institute for Production Technology, University of Southern Denmark.

Because of existence of many conceptual similarities a research part of both projects has been conducted together, starting from the summer 2005.

This report mainly focuses on the design of two new activities, one for each of the simulators: pilot on board for the ship simulator and defencive positioning for the soccer simulator. The last part of the report gives a proposal on how the common parts of the two designed activities could be generalised to form an application framework.

E.2 Motivation

The motivation for this project is to investigate how "habitat-thinking" can be realised in software. The method is to formulate habitat-concepts in such a way that skilled programmer with an object-oriented tool can use the concepts for producing software.

The long term goal is to develop an application framework for simulation using the concepts agent, habitat and activity (AHA). The short term goals are to develop a ship simulator and a soccer simulator suited for experimenting with the AHA concepts.

E.3 Statek - Ship simulator

The maritime scenario has been chosen for experimentations with concepts activity and habitat, because the maritime operations have many good examples of habitats, and the activities are relatively well-defined in that domain. The maritime operations also holds clear roles like captain, pilot and sailors which can be modelled by the concepts like agent and role.

As a part of the investigation of how "habitat-thinking" can be realised in software, it is desired to introduce a collaborative activity in the ship simulator called pilot on board (POB). The following sections give a detailed description of the design of both the ship simulator and POB activity.

Figure E.1 presents the model of the ship simulator. We can distinguish agents, that are autonomous units in the simulator. Agents can take on various roles, but the number of roles that can be taken simultaneously was limited to one for now. Agents move around in habitats, that are used to represent a part of physical environment. Habitats observe their inhabitants and provide activities to them. Agents can choose between various activities provided by habitats they are inside at the moment. In this manner habitats can indirectly control the

agents' actions.

The main focus in this project is how to define concepts of activity and habitat and how to organise the interplay between them and agents.

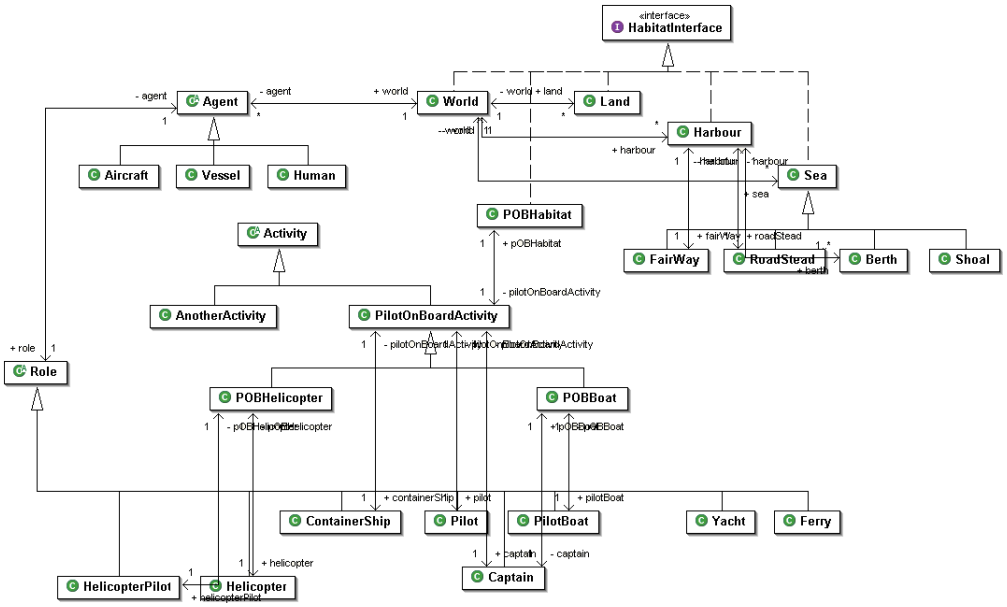


Figure E.1: The ship simulator class diagram.

E.3.1 Pilot on board activity

It is required that merchant vessels acquire a pilot to guide them in larger harbours, but it is not required for ferries and yachts. The pilot guides the captain, but it is the captain that makes decisions. Pilot cooperates with VTS.

Each harbour commands a number of pilot and itself plans their deployment. Pilot can transported to and from ships in boats or helicopters.

The sequence of actions is as follows. The captain reports his approach to the VTS and he is informed by the VTS about the time and place of pilot's embarkation. Details about the manner of pilot's transportation and his identity can be sent later. All details can be changed by VTS at any time.

If no pilot is available at the suggested time and place or the berth is not available, then a ship should anchor outside the harbour, preferably at road-stead. The pilot, however, can only be embarked when the ship continues sailing towards the harbour. Pilot's embarkation cannot take place during any dangerous situation, such as overtaking. The pilot leaves the ship by the gangway when the ship is moored in the harbour.

E.3.2 Agents

In the ship simulator all the moving objects are modulated as agents. The segment, concerning agents, of the class diagram is depicted at figure E.2.

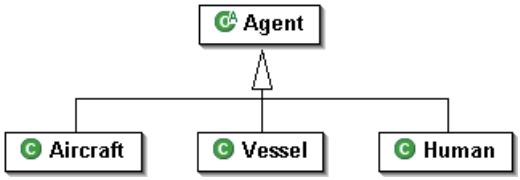


Figure E.2: The part of the class diagram concerning agents.

An inheritance structure has been chosen to handle agents. The inheritance structure consist of an abstract super class which handles the common responsibility of the agents and three sub classes, or specialisations, one for each three agent types represented in the ship simulator.

Agent classes

The agent inheritance structure, currently, consist of four classes, this section will give a more thorough description of each of the classes.

Agent: Abstract super class that holds all the common fields and methods of the different agent types. An agent is an autonomous unit with the ability to act concurrently with other agents. Due to that fact the agents in the ship simulator should be implemented as threads to allow the agents to act in an independent and concurrent manner. It is desirable that agents can take on different roles, mainly because it offers a high level of re-usability because the same agent can be used for different purposes just by changing his role. This kind of flexibility is especially desirable in the ship simulator even though the agents may have different purposes they also share a lot of common features. This class provides the functionality to allow agents to take on roles and change roles because it should be handled uniformly for all of the agent types. For this version it is agreed-on that an agent only can hold one role at a time.

Aircraft: A sub class of agent. This class has the responsibility of "adding" all the aircraft specific functionality to the general agent. The main reason of introducing this specific agent type in the ship simulator is to be able handle the helicopter in the pilot on board activity when the pilot is embarking merchant ships by helicopter.

Human: A sub class of agent. This class holds specific information of the agent type human. Of course humans are, in general, more complex than agents, but in this case a human is seen as a specialisation of an agent just to distinguish between the three agent types.

Vessel: A sub class of agent. This class handles vessel specific responsibility in the ship simulator, and presence of the class is almost obvious when simulating a maritime scenario.

In general it could be stated that vessels and aircrafts can't be considered as agents because they are reactive objects controlled by other agents (humans), even so this design has been chosen due to an earlier project proposal (Ship simulator design proposal by Peter Bøgh Andersen).

E.3.3 Roles

The role concept has been introduced in the ship simulator to have a flexible way for handling agents which only have small differences. Using the role concept also enforces a high level of re-usability, e.g. agents like captain and pilot almost have the same abilities which encourages to use the same agent type but with different roles.

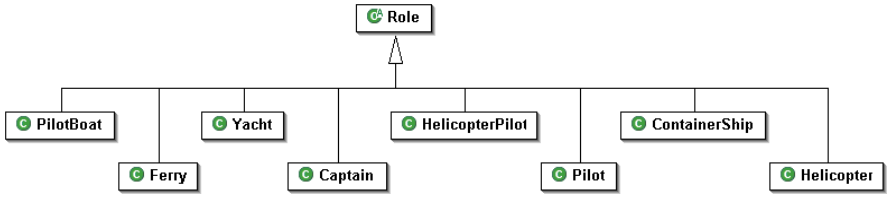


Figure E.3: The part of the class diagram concerning roles.

Role classes

Figure E.3 shows the segment of the class diagram concerning roles. The role inheritance structure consists of the abstract super class Role and the sub classes which holds the concrete roles. The following will give a more thorough description of the classes shown in the class diagram at figure E.3. Because of the great similarities of the concrete roles, for each of the three agent types, only one role for each of the agent types will be described.

Role: Abstract super class that holds the common fields and methods of the roles. For this version of the ship simulator it has been chosen that roles only will be used to distinguish different kinds of agent types. To achieve this, each of the sub classes have to set some preconditions defined in the super class. For instance, one of the preconditions could be the allowed agent type for this role, which should ensure that an aircraft agent can't take on a vessel role.

Helicopter: A sub class of role. This role can only be hold by an aircraft agent and for this reason that should be defined as a precondition for this role.

Pilot: A sub class of role. This role is intended for the agent type human and can not be hold by either vessel agents or aircraft agents.

ContainerShip: A sub class of role. The ContainerShip role is an example of a vessel role.

For this version of the ship simulator a very simple role structure has been chosen i.e. the concrete role only defines some preconditions required of an agent

to take the role. In a later version one could imagine that a role also would hold information on what "extra" actions an agent could take if he holds this role.

E.3.4 Habitats

Habitats are used model either physical or logical contexts. Figure E.5 presents the part of the ship simulator model concerning habitats. In the ship simulator it was decided to treat various physical areas, such as sea and land, as habitats. The land could be further divided into islands and mainland.

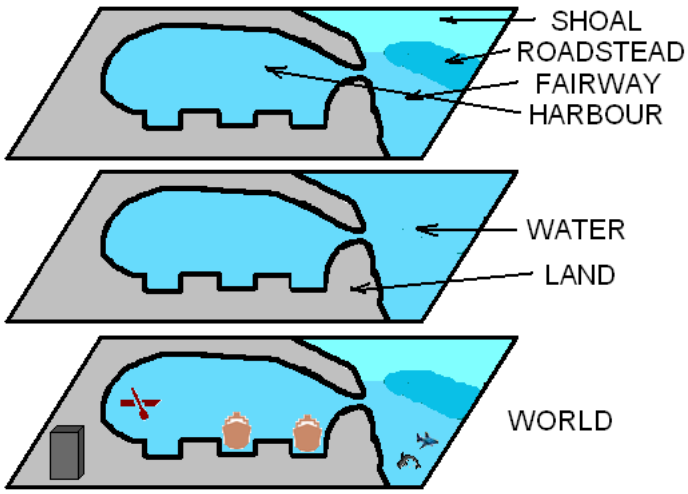


Figure E.4: Overlapping habitats visualised.

We distinguish following parts of sea: fairways (deep enough to enable merchant ships to sail), road-stead (dedicated for anchoring), shoals (only yachts are allows to sail due to water depth) and berth (for disembarkation and unload of merchant ships). They are modelled as specialisations of sea.

Additionally, we decided to model harbour habitat. This habitat is composed of a road-stead, a fairway and berths, and it is a part of the world.

There exists also a habitat that covers the area where pilot on board activity takes place, called POBHabitat. This habitat will be spawned by a harbour habitat when a merchant ship reports its approach to the specific harbour. There will exist one instance of POBHabitat for each merchant ship approaching the harbour.

Habitat classes

Figure E.5 shows the part of the class diagram concerning habitats. This section will elaborate about the structure proposed for habitats and will describe the classes presented on the figure E.5.

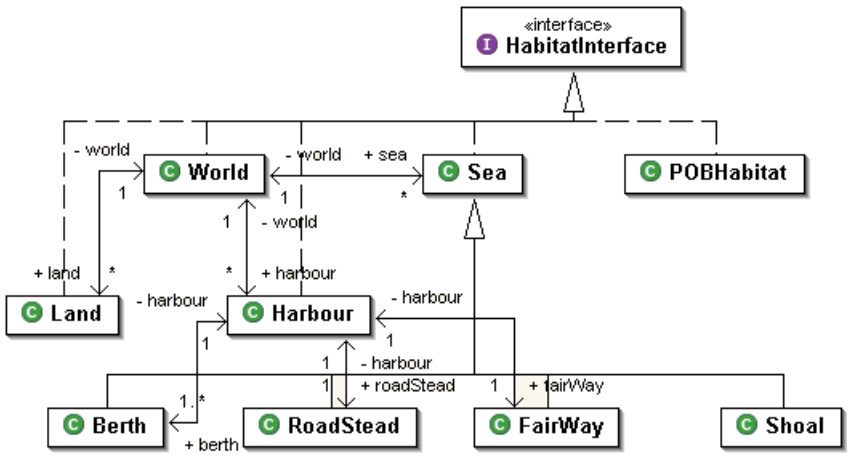


Figure E.5: The part of the class diagram concerning habitats.

Two level inheritance structure was proposed for habitats. First of all, all habitats implement the HabitatInterface interface. This was introduced to provide common way of handling their functionality, such as observation of their inhabitants. The implementation, however differs for all habitats. Each habitat generally will be implemented as thread, so that they can act independently and concurrently.

HabitatInerface: A common interface for all habitats. Habitat will be implemented as thread.

World: A class implementing the HabitatInterface interface. As a habitat, the world observes agents present in it and can indirectly influence them though manipulation on their available activities. The world consists of a sea and land-areas. This class is treated as a root for all other habitats and will provide basic functionality (as very simple activities) for agents. The World class does not correspond to any particular world or part of world in reality. An imagination of marine view of the world is the referent system for the ship simulator.

Land: A class implementing the HabitatInterface interface. This class has the responsibility for modelling the dry land, so that it e.g. limits the types of agents that can be present in it to human, vehicle and aircraft agents.

Sea: A class implementing the HabitatInterface interface. This class has the responsibility for modelling the wet land, limiting the types of agents that can be present in it to human, vessels and aircraft agents. Models the common properties of various sea parts. It is further specialised into Berth, RoadStead, FairWay and Shoal.

Berth: A sub class of Sea class. This class holds specific information of the sea part – berth. Vessels can finish their trip at berth and next unload the cargo. There are several berth places in the harbour.

RoadStead: A sub class of Sea class. This class holds specific information of the sea part – road-stead. All vessels can anchor at road-stead.

FairWay: A sub class of Sea class. This class holds specific information of the sea part – fairway. All vessels can sail (and eventually anchor) on fairways.

Shoal: A sub class of Sea class. This class holds specific information of the sea part – shoal. Only yacht can sail (or eventually anchor) on shoals.

Harbour: A class implementing the HabitatInterface interface. A harbour is composed of many berth places, a road-stead and a fairway.

POBHabitat: A class implementing the HabitatInterface interface. In is spawned around the area where pilot on board activity takes place.

E.3.5 Activities

Activities are used to model collaboration between agents. A group of agents has a common goal – to moor the ship on the berth quickly and safely in the case of a Pilot On-board activity. Each activity has a number of participants of specific roles. Activity can be active if and only if there are agents ready to be participants in the activity. Some of the participants are constant for the activity, and the others are only described by their type.

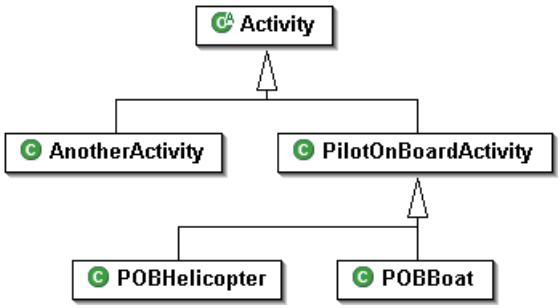


Figure E.6: The part of the class diagram concerning activities.

Activity classes

Figure E.6 shows the part of the class diagram for the activities. This section will describe the classes in more detailed manner.

Activity: An abstract class that holds all the members and common methods for the specific activities.

PilotOnBoardActivity: A sub class of Activity. It models the situation when to merchant vessel is guided by a pilot in a harbour. It starts when a ship declares its approach to VTS and ends when the ship is moored at the berth and the pilot has disembarked through a gangway. It requires participants of VTS role, Pilot role, MerchantShip role, and Captain role. It takes place in POBHabitat. It holds common members and methods for its specialisations.

POBHelicopter: A specialisation of PilotOnBoardActivity. It models to the situation that the pilot is transported to the merchant vessel by a helicopter. It requires a human participant that hold role of a HeliPilot and aircraft participant that holds role of Helicopter. It includes methods specific for pilot embarkation with use of a helicopter.

POBBoat: A specialisation of PilotOnBoardActivity. It models to the situation that the pilot is transported to the merchant vessel by a boat. It requires a human participant that hold role of a PilotCaptain and vessel participant that holds role of PilotBoat. It includes methods specific for pilot embarkation with use of a boat.

E.3.6 Summary

To introduce and design the pilot on board activity in the ship simulator it was necessary to re-design parts of the model component too. The new model component mainly consists of four parts: agents, roles, habitats, activities. All moving objects are modelled by agents. For simplicity reasons an agent only can hold one role at a once. In this version the roles are very simple, but complex enough to distinguish the agents from each other.

The world in the ship simulator is simply constructed by composing physical habitats. The habitat concept has been designed as an interface so that it can provide the desired flexibility. Aside from the habitat keeping track of it's inhabitants, it also provides them with the activities it offers. It is the agents responsibility to choose in which of the activities, offered by the habitat, it want to engage. Due to that fact agents can also choose to ignore an activity, like, for instance, the pilot on board. However it is very unlikely that an agent refuses to participate in an activity that is essential for it to achieve its goal(s). E.g. if a merchant vessel approaches a harbour and refuses to participate in the pilot on board activity, it is not allow to enter the harbour, and can't for that reason not moor at the berth and unload it's cargo.

E.4 The soccer simulator

To experiment with software realisation of the habitat and activity concepts, the soccer scenario has been chosen. This choice has been made because it is easy to identify a lot of activities in soccer that involves habitats. It is also obvious to apply the agent concept to model the soccer players.

The following sections will describe the design of a collaborative activity called defencive positioning activity.

E.4.1 Defencive positioning activity

The defencive positioning activity occurs when the team is not in the ball possession. It influences all the players in the team, they reorganise themselves according to both the overall situation and the close neighbourhood. The choice of the action depends on many factors, like ball position, positions of team mates and opponents, player properties. The activity is started when the team looses

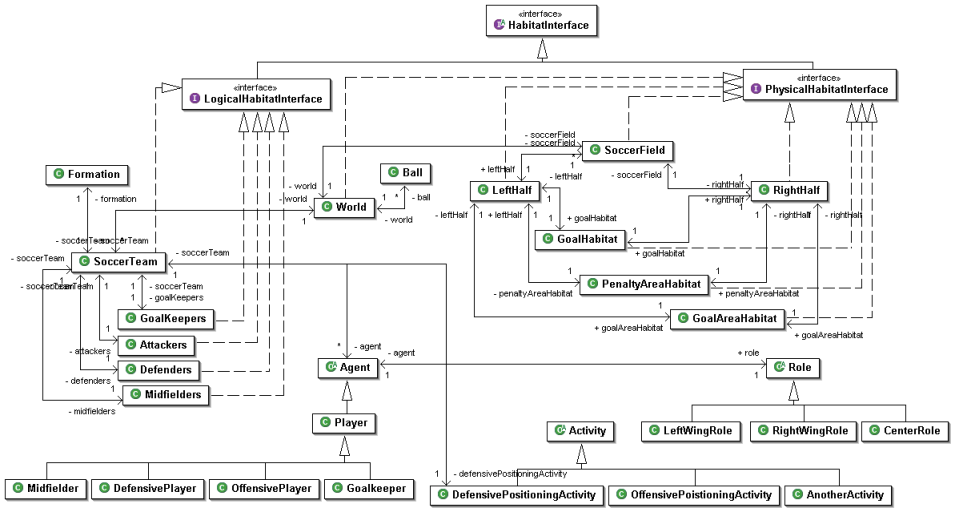


Figure E.7: The soccer simulator class diagram.

the ball. Team members reorganise and try to prevent the opponents from scoring. Defenders move back towards the own goal and cover opponent attackers. If the ball is still positioned on the other half, both the attackers and mid-fielders will cooperate to overtake the ball, else the mid-fielders will move back to help defenders, whereas only some attackers will move back and position themselves in such the way that will enable quick counter attack.

E.4.2 Agents

In the soccer simulator only the players are considered to be agents. Figure E.8 shows the segment of the soccer simulator class diagram concerning agents. As Shown in figure E.8 the agent inheritance structure has two levels of specialisation. The middle level, currently, only consists of a player class. This design approach has been chosen such that non-player agents can be introduced later on, for instance, a referee. At the lowest level the player is specialised into four different kinds of players, based on field position, this is done to be able to differentiate player properties according to their field position.

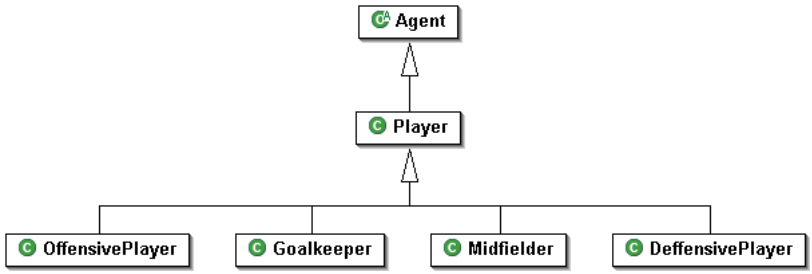


Figure E.8: The part of the class diagram concerning agents.

Agent classes

The classes in the agent inheritance structure are as follows:

Agent: Abstract super class that holds all the common fields and methods of the different agent types. The Agents in the soccer simulator should be implemented as threads to allow the agents to act in an independent and concurrent manner. It is desirable that agents can take on different roles, in this case the roles could be used to organise the players on the soccer field. This class provides the functionality to take on roles and change roles, because the handling of roles should be the same for all the agents. For simplicity reasons it is agreed-on that an agent only can hold one role at the same time.

Player: A sub class of agent. This class is only introduced to ensure extendibility so the simulator easily can be extended with other agent types later on.

Goalkeeper: A sub class of player. This class is almost self explaining it has the responsibility to set the properties specific to the goal keeper.

DefensivePlayer: A sub class of player. The DefensivePlayer class has the responsibility to set the properties specific to the defencive players.

MidfieldPlayer: A sub class of player. The responsibility of the midfield players is to help the team in both offencive and defencive situations.

OffensivePlayer: A sub class of player. The offencive of the four player kinds. When an offencive player is inside the opponents penalty area an offencive property could be the he will try to shoot at goal.

E.4.3 Roles

A simple inheritance structure has been chosen to handle the roles. Figure E.9 shows the role inheritance structure from the soccer simulator class diagram. The role concept for this version of the soccer simulator is introduced to ease

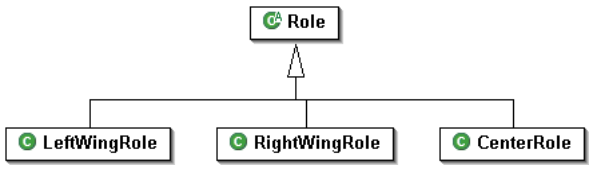


Figure E.9: The part of the class diagram concerning roles.

the positioning of the different kinds of players in the soccer field.

Role classes

Role: Abstract super class that holds the common fields and methods of the roles. For this version of the soccer simulator the role concept will be very simple. The roles will only hold some sort precondition which have to be fulfilled before an agent can take one of the roles.

LeftWingRole: A sub class of role. As the name implies this role should be hold by players in the left side of the field.

CenterRole: A sub class of role. This role is intended for the players at the centre part of the field.

RightWingRole: A sub class of role. This role is the opposite of LeftWingRole.

This is the first attempt on introducing roles in the soccer simulator. Later the role concept could be very useful for positioning the players in advanced formations on the soccer field. E.g. if a player could hold more than one role at once, it would be possible to organise the midfield players in a diamond shape just by having four midfielder: a left wing, a right wing and two centres where one of the centres have an offencive role too and the other centre a defencive role in addition to their centre roles.

E.4.4 Habitats

Habitats are used to model either physical or logical contexts and it was decided to clearly distinguish those two in the soccer simulator. Figure E.10 presents the part of the soccer simulator model concerning habitats. In the soccer simulator it was decided to treat various physical areas, such as goal area and penalty area, as physical habitats. Logical habitats are used for grouping the agents, so that can be more easily observed and manipulated.

Habitats are used to model either physical or logical contexts, that contain a group of inhabitants – agents. Habitats should be implemented as threads, and their main functionality will be to observe their inhabitants. Agents can be present both physically in physical habitats, such as a penalty area in a soccer field, and logically in a logical habitat, such as defenders. Therefore an agent can be present in several habitats simultaneously and all of those habitats will treat such agent as their inhabitant. Figure E.10 presents the part of the soccer simulator model concerning habitats.

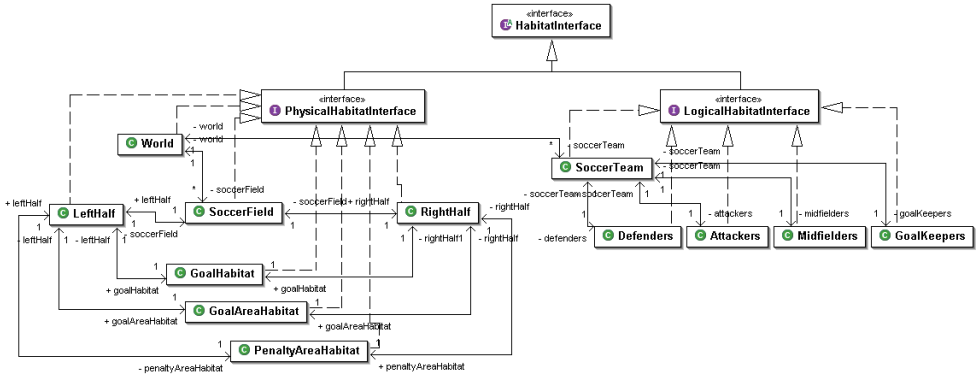


Figure E.10: The part of the diagram concerning habitats.

Figure E.10 shows the part of the class diagram concerning habitats. The following section will describe subsections of this diagram.

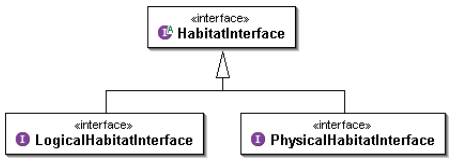


Figure E.11: The part of the class diagram concerning high-level inheritance in habitats.

It was decided to distinguish between the logical and physical habitat, so two interfaces implementing the HabitatInterface interface were created. This is presented in Figure E.11.

First of all logical habitats will be described. They are presented in figure E.12. Concerning the number of inhabitants, the largest logical habitat modelled in the soccer simulator is the soccer team. It is composed of a number of smaller logical habitats, corresponding to the group of attackers, mid-fielders, defenders and goalkeepers. Each group can hold a number of agents.

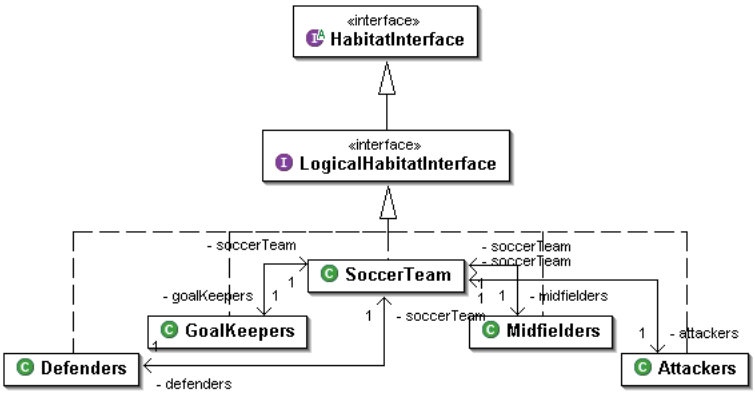


Figure E.12: The part of the class diagram concerning logical habitats.

We distinguish several physical habitats, that implement PhysicalHabitatInterface interface. It was decided to model a soccer field as a physical habitat. It is composed of several smaller habitats, such as field halves, penalty areas, goal areas and goals. The dependencies between those habitats is presented in Figure E.13.

Habitat classes

This section will describe selected classes from the inheritance structure presented before.

HabitatInterface: A general interface. Habitats will be implemented as threads.

PhysicalHabitatInterface: An interface extending the HabitatInterface. Models physical environments.

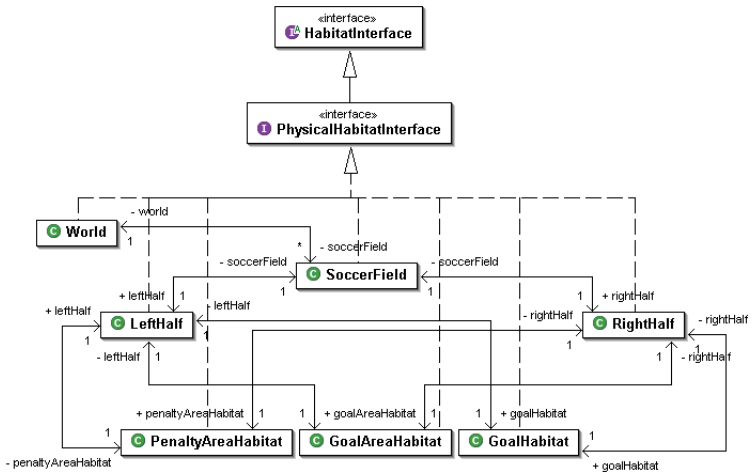


Figure E.13: The part of the class diagram concerning physical habitats.

World: A class implementing PhysicalHabitatInterface. Its thread will observe all the agents present in the world, which can contain several soccer fields.

SoccerField: A class implementing PhysicalHabitatInterface. Its thread will observe all the agents present in this soccer field. A SoccerField habitat is composed of several smaller habitats, that also observe their inhabitants.

LogicalHabitatInterface: An interface extending the HabitatInterface. Models logical contexts. It is used for grouping and its thread will observe all the agents present in the group.

SoccerTeam: A class implementing LogicalHabitatInterface. Corresponds to the group of 11 players. This class was introduced to control that group as a whole unit. A SoccerTeam is composed of several other logical habitats, such as Mid-fielders, Attackers, Defenders and Goalkeepers, also introduced for controlling groups of players.

E.4.5 Activities

Activities are used to model collaboration between agents. In this iteration of the soccer simulator, it is required to design and implement a defencive positioning activity, in which a group of agents has a common goal – to defend their own goal, so that they will not loose goal. This activity will be mandatory for some agents and optional for the others, depending on the dynamic situation on the field. This activity is provided by the team habitat, that corresponds to the collection of all players.

Activity classes

Figure E.14 shows the structure for the activities in the soccer simulator. Three classes are presented on the diagram and described below.

Activity: An abstract class.

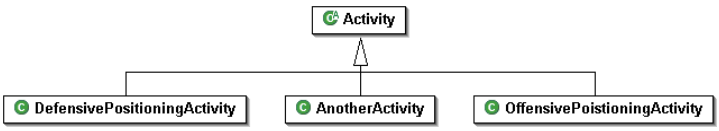


Figure E.14: The part of the class diagram concerning activities.

DefensivePositioningActivity: A sub class of Activity class. Includes functionality specific that models the situation when the players do not have the ball.

OffensivePositioningActivity: A sub class of Activity class. Includes functionality specific that models the offencive positioning in the real soccer game. Not to be implemented in this iteration of the simulator.

AnotherActivity: A sub class of Activity class. Corresponds to any other collaborative activity.

E.4.6 Summary

As it was mentioned previously, the soccer simulator project started before this project. The model of that simulator included activities and habitats, but in slightly different meaning. It was the agent that had a constant list of activities for the whole game and just chosen the one that best fit into the situation. There were only physical habitats of various shapes, that were used to represent various areas around the player.

Most of the activities were single player. The defencive positioning activity was implemented with that design during autumn 2005. Several difficulties were observed, such as complexity of manipulation with groups of players, such as mid-fielders, defenders or attackers. The lack of roles caused that all the mid-field players were indistinguishable, and therefore it was difficult to implement distinct actions depending on what position on the field do they occupy.

The new model component mainly consists of four parts: agents, roles, habitats, activities. All moving objects are modelled as agents. For simplicity reasons an agent only can hold one role at a once. In this version it was decided to implement roles for left wing, centre and rightwing player.

The world in the soccer simulator is constructed by composing physical and logical habitats. The habitat concept has been designed as an interface so that it can provide the desired flexibility. Habitats keep track of their inhabitants and provide activities. An agent can ignore those suggestions, but it is very unlikely if it wants to achieve a common goal. E.g. if a team defends and an agent refuses to cover an attacker, it is likely that the former will shoot a goal. The agent, however, has that choice, and can have reasons for it, such as high tiredness.

E.5 Candidates for a general framework

After designing the two activities, pilot on board and defencive positioning, it has become clear that both applications have some common parts. This fact encourage that those parts can be generalised for form a framework for working with the concepts: agent, habitat and activity. At this stage it is too early to talk about a fully formed framework and for this reason this section will describe the candidates, found in the two applications, that should be considered to form a framework for working with agents, habitats and activities.

E.5.1 Agents

Both applications use the agent concept. In both cases the agent's responsibilities are very similar figure E.15 shows a modular layout of the most important parts of an agent.

The basic agent parameters, will hold all information which is essential to the agent like physical properties and methods that are common to all agents such as moving.

When the agents move around in habitats, that provide activities, the agents should of course have some functionality so they can choose in which of the activities they want to engage, this is handled by the activity handling module.

The agents in both applications use the role concept (the role concept is further described in the next section). The responsibility of role handling module in the agent is to provide functionality so that the agent will be able to take on any given role, if it is possible, or to change his current role into another role.

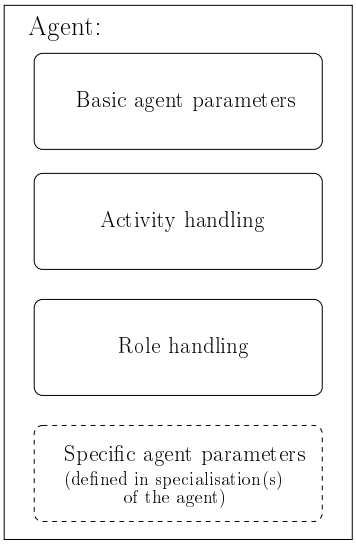


Figure E.15: The general layout of an agent.

The last of the modules contained in the general agent layout is the specific

agent parameters. This is an imaginary module, this is represented as a dotted line, the reason for showing it is to illustrate that it is possible to make specialisations of an agent which may override some of the basic agent parameters and additionally provide more functionality to the specific agent.

The agent layout depicted in figure E.15 is well suited for the two simulators for which the two new activities have been design. To ensure that this agent layout is suited for applications in general too, it is necessary to study more literature on how agent could/should be designed and which drawback and benefits different agent architectures provide. For now on it is agreed on to use the agent layout presented in figure E.15.

E.5.2 Roles

It has been chosen to introduce the role concept in both applications. This new approach of handling agents, has shown to be very useful when designing the two new activities, and at the same time it allows for a high level of re-usability.

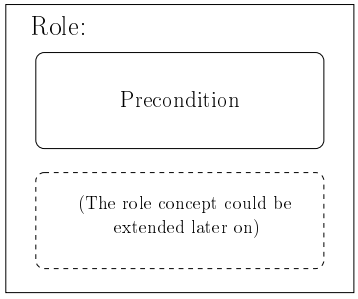


Figure E.16: The general layout of a role.

It is agreed on the roles should be kept very simple in the beginning, the general, but simple, role layout is depicted at figure E.16. Currently the role layout only has a precondition module. This module defines all the requirements to take on a specific role. It is decided that an agent only can hold one role at the same time. This decision doesn't exclude the possibility that one of the preconditions of a role could be that an agent should hold another unique role before he can take on this new role.

As the role layout indicates the role concept should be extended later on, e.g. a specific role could provide specific abilities to the agent holding this role. Another extension to the role concept could be that it should be possible for an agent to hold several roles, that are not mutual exclusive, at once. To achieve this it has to be decided whether the agent or the role has the ability to merge several roles.

Besides the two mentioned extensions, there might be a lot literature describing designs of roles and/or other useful extension for the role concept, so it is obvious to find and studied some literature before extending the design of roles.

E.5.3 Activity

Activities are provided by habitats, therefore, for each activity there exists a corresponding habitat, that both physically and logically covers the area, where the activity takes place, and the group of agents collaborating in the activity. Habitats can add or remove possible activities for each agent present in them, so that they can indirectly enforce specific actions. The ultimate decision about what activity to choose is however with the agent.

Activities are "half-baked", which means that some of the roles in the activity have instantiated participants, while others only contain descriptions of possible participants. An activity can be activated if all the required roles were filled by agents, and should respond properly when an agent deserts.

E.5.4 Habitat

Habitats are used to model either physical or logical contexts, that contain a group of inhabitants – agents. Habitats should be implemented as threads, and their main functionality will be to observe their inhabitants.

It was decided to choose one habitat to be a root of all others (this is the World class in both simulators). All other habitats are either physically or logically contained in that habitat, and they can overlap with each other. It was decided that each of the habitats will provide a set of activities available for its inhabitants. The possibilities that the specific agent has, will be unified by the root habitat, and the prohibition is prior to the permission. The agent route planning situation will take place in root habitat.

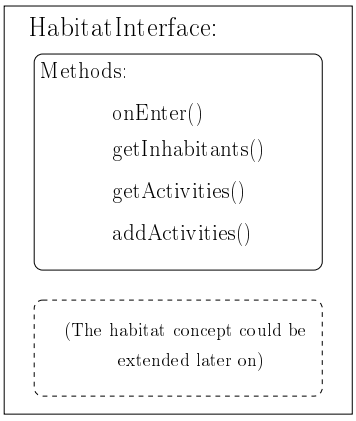


Figure E.17: The general layout for habitats.

Figure E.17 shows a modular layout of the most important parts of a habitat. The getInhabitants() method retrieves the collection of agents, that are present in the habitat at a moment. The getActivities() method is used for retrieving the collection of activities available in specific habitat, whereas the addActivities() method is supposed to add new activities to the habitat. OnEnter() method can

be used to enforce some kind of behaviour when an agent enters the habitat. As it is suggested in Figure E.17, the habitat concept can be extended later on.

E.5.5 Summary

After designing the activities for the two simulator application it has been easy to identify the parts of the designs that could be considered to be candidates for a general framework, namely: agents, habitats, activities and roles.

For all of the four concepts very general designs have been chosen to have a high degree of flexibility. The general design can be refined later on, but it will require some literature studies, especially for the concepts agent and role.

During the design phase it was discover that it is beneficial to distinguish between two kinds of habitats namely logical and physical ones. The greatest difference between the two kinds of habitats is that the logical habitat isn't bounded by physical limits. Both logical and physical habitats provides activities to their inhabitants.

Activities are considered to be "half-baked" and used for modelling collaborative behaviour between agents. An activity can be activated if all required roles are filled and it should respond properly when an agent decides to desert.

The most important aspect when forming the general framework is to define the interplay between the three main concepts habitat, activity and agent. For now it is decided that habitats provide activities to the agents and thereby indirectly influence their behaviour. Agents are free to choose among all the activities afforded by the habitats they inhabit. Habitats can be predefined or spawned dynamically by agents or activities.

Appendix F

CD contents

The attached CD contains the following.

/bin	The two developed SoccerLab prototypes.
/report	Contains the report in PostScript and PDF
/src	The source code for the two developed prototypes

To enable opengl support when running one of the prototypes, the application is started as follows:
`java -Dsun.java2d.opengl=True -jar <filename.jar>`

In case of CD malfunction, the content of this CD will also be available at:
`http://appendixf.mrhunt.dk`
during February 2006