159547

EŞ EVRİMSEL TASARLAMA

119690

Bora İ. Kumova

Haziran, 2002
İZMİR

119690

CO-EVOLUTIONARY PLANNING

A Thesis Submitted to the Graduate School of Natural and Applied Sciences of

Dokuz Eylül University

In Partial Fulfilment of the Requirements for the Degree of Doctor of Philosophy in Computer Engineering, Computer Engineering Programme

By

Bora İ. Kumova

June, 2002

İZMİR

EŞ EVRİMSEL TASARLAMA

Dokuz Eylül Üniversitesi

Fen Bilimleri Enstitüsü

Doktora Tezi

Bilgisayar Mühendisliği Bölümü, Bilgisayar Mühendisliği Anabilim Dalı

Bora İ. Kumova

Haziran, 2002
İzmir

Ph.D. THESIS EXAMINATION RESULT FORM

We certify that we have read the thesis, entitled **"CO-EVOLUTIONARY PLANNING"** completed by **BORA İ. KUMOVA** under supervision of **PROF.DR. ALP KUT** and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

[illegible]

Prof.Dr.Alp **KUT**

Dokuz Eylül University
Department of Computer Eng.
Supervisor

[illegible]

Prof.Dr.Tatyana **YAKHNO**

Dokuz Eylül University
Department of Computer Eng.
Thesis Committee Member

[illegible]

Assist.Prof.Dr.Ender **BULGUN**

Dokuz Eylül University
Department of Textile Eng.
Jury Member

[illegible]

Assoc.Prof.Dr.Yalçın **ÇEBİ**

Dokuz Eylül University
Department of Computer Eng.
Thesis Committee Member

[illegible]

Prof.Dr.Erden **BAŞAR**

Eastern Mediterranean University
Department of Computer Eng.
Jury Member

Approved by the
Graduate School of Natural and Applied Sciences

[illegible]

Prof.Dr.Cahit Helvacı
Director

ACKNOWLEDGEMENTS

To Leyâl ...

I am grateful to my advisors for supporting my ideas, especially to my co-advisor Akira Imada for sharing with me his ideas and his profound expertise in evolutionary computation; without his help I could not have designed the various issues of the algorithm that efficient.

Bora İ. Kumova

ABSTRACT

Planning is known to be NP-complete. Nevertheless, practical solutions can be found, if the application domain is restricted to a specific area. Therefore, planning components of multi-agent systems are designed usually for a specific domain. However, these solutions are relative domain-dependent, in the sense that besides the heuristics even the planning algorithms are domain-oriented. To address this problem, a domain-independent algorithmic methodology is proposed, in which the multi-agent concepts interaction, variety, scalability, focus, and diversity are mapped onto related constructs of a co-evolutionary algorithm. The algorithm solves the planning problem by seeking for global alternative plans in the search space that is opened by all possible combinations of the plan steps of the involved agents. Thereafter, the agents negotiate on the alternative plans or restart the co-evolutionary algorithm.

ÖZET

Tasarlama NP-bütün (çözümü kesin olmayan) bir olgudur. Bununla birlikte, belli uygulama anlanları için kısıtlı çözümler önerilebilir. Bu nedenden, çok aracı dizgelerin tasarlama birimleri genelde belli bir uygulama alanına yönelik düzenlenirler. Ancak, alan bilgisinden de öte, kullanılan tasarlama algoritmaları dahi alana yönelik olduğundan bu çözümler oldukça o alana bağlı olurlar. Bu çalışmada, böyle belli bir alana yönelik bir çözüm yerine, alandan bağımsız bir genel yaklaşım anlatılıyor. Bu yaklaşım, etkileşim, çeşitlik, çoğaltma, odaklama, ve türlülük gibi çok aracı dizge kavramlarını eş evrimsel bir algoritmanın ilgili yapılarına aktaran ve uygulama alanlarından bağımsız olan algoritmik bir yöntem önerilmektedir. Algoritma bu sorunu, katılan aracıların tasarı adımlarının tüm olası birleşimlerinden açılan arama uzayında seçimli genel tasarıları arayarak çözüm bulur. Bundan sonra, aracılar seçimli tasarılar üzerinde ya pazarlık yaparlar ya da eş evrimsel algoritmayı yeniden başlatırlar.

CONTENTS

Figures

Tables

CHAPTER ONE
INTRODUCTION

First, we describe the emergence of the thesis topic, give a motivation for the reader, discuss the general objectives of related research areas, and sketch the structure of the thesis.

Emergence Of The Thesis Topic

Before we start to describe the work, it is worthwhile to give some background information to the progress of the dissertation, since the topic emerged over time. Initially, the thesis was entitled "Multi–agent–based Distributed Database Management", with the objective to explore the problems concerned with distributed database management of heterogeneous data and to find a multi–agent–based solution approach. After a while, it became clear that the goal and the related research areas were too broad for this project. Therefore, we focused on one of the crucial points of distributed database management, the design of a communication protocol that can satisfy the desired requirements [Kumova; 2000a], [Kumova; 2001a]. On the other hand, we had to combine the data–oriented communication for distributed database management with intelligent communication for multi–agents, which turned our focus to agent communication languages [Kumova; 2000d], [Kumova; 2001b]. Since communication and behaviour mutually influence each other, finally our research concentrated on communication, co–operation, competition, and co–ordination structures for multi–agent systems, which can be considered as the external tools of an individual to express intelligent behaviour. However, it is known that intelligence is actually due to some cognitive processes, such as planning, learning, adaptation, and decision making within these processes. Therefore, as a combined solution approach for co–operation, competition, and co–

ordination of multi–agent systems, finally co–evolutionary planning emerged as the thesis topic.

Besides that, we improved the AgentTeam framework, which was designed and partially prototype implemented in a MSc thesis [Kumova; 1998c], from the software engineering perspective [Kumova; 1999c], [Kumova; 2000b], [Kumova; 2000e], [Kumova; 2000f], [Kumova; 2000h], [Kumova; 2000i]. Since our long–term goal is to improve AgentTeam, co–evolutionary planning is another contribution to the framework.

Motivation

Distributed problem solving in multi–agent systems is still a challenging task. It is itself a multi–dimensional problem involving various research disciplines. Even the restriction to planning is still NP–complete. Therefore, planning is usually further restricted to relative constraint domains, such games. Since, a game is usually complete with respect to its rules, planning can successfully be applied. Complex planing components were design for relative complex games, such as chess, which can beat human experts. However, those solutions are inherently domain–dependent and can hardly be adopted for planning in other problem domains. This is even worst in case of multi–agent systems for information processing systems, where various heuristics must be included in order to get some reasonable results. Because of inherent utilisation of domain–dependent concepts, analytical approaches are hardly comparable.

Other approaches are based on natural adaptation processes and can be grouped under the umbrella evolutionary computation. The idea is, given an environment, an initial state, a goal state, and some evaluate criteria that guide sample solutions towards the goal state, the algorithm is capable to explore the whole search space for optimal solutions. The advantage is that the evaluation criteria can include relative abstract domain information. Therefore, these approaches are sometimes called soft computing. Co–evolution is a variation of evolutionary computation that simulates mutual evolution of species of the same environment.

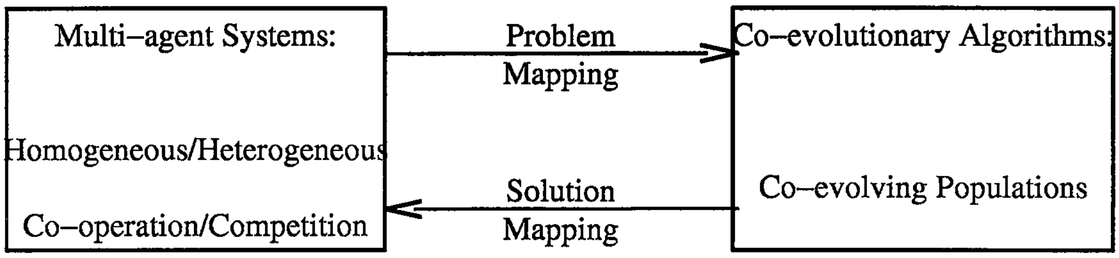


Figure 1.1: The Idea Of Planning Heterogeneous Co–operation And Competition Structures

We propose a co–evolutionary algorithm to solve planning problems of multi–agent systems. For this purpose required co–operation and competition relationships need to be mapped onto the co–evolutionary algorithm (*Figure 1.1*). The planning problem is solved with the co–evolutionary algorithm by exploring the huge search space opened by all possible plan steps of all agents. The resulting plans are return to the agents for approval.

As benchmark we have chosen Soccer, to have a relative complex domain. For instance, according a naiv comparison of the planning complexity for chess and Soccer, plan generation for Soccer is more complex, since all eleven agents can move simultaneously and with continues variable distances, if the moves of all eleven players are planed centralised by one player. Whereas, in chess, only one figure can move at a time.

Related Research Areas

Just to summarise the related research ares:

- Algorithmic complexity
- Game theory
- Artificial intelligence
- Co–evolutionary computation
- Planning
- Communication, co–operation, competition, control

- Homogeneous and heterogeneous structures
- Multi–agent systems
- Software engineering

Structure Of The Dissertation

The second chapter gives a general introduction to the planning problem and conventional planning approaches.

In the third chapter, we guide the reader through the history of evolutionary computation and the common techniques, with an emphasis on co–evolutionary approaches.

In the fourth chapter, the Co–evolutionary Planning (CEVOP) approach is introdu ced as a generic component of the AgentTeam framework.

In the fifth chapter, the revised version of the AgentTeam framework is presented.

In the sixth chapter, the SoccerTeam prototype implementation is discussed, which was implemented as a benchmark for the CEVOP methodology.

The major contribution of this dissertation is the CEVOP methodology. Readers interested only in the CEVOP methodology are referred to chapter four, the CEVOP part of chapter six, and the related appendices.

CHAPTER TWO
PLANNING STRATEGIES

This chapter provides a discussion on the cognitive process planning and on some related concepts that are relevant to this work, such as learning, planning, rule base, rule, plan step, plan, and goal. Further, the basics of analytical planning approaches are summarised, in order to give an introduction to the conventional planning techniques.

2.1 A Definition For Plan

For simplicity, we restrict our study to agents whose knowledge base (KB) consist of rules and where each rule may constitute a primitive plan step. Principally, all rules available for an agent R can be involved in some plans. Thus, the set of all available rules is the super–set of all possible plans PP. However, an agent may not be aware of some possible plans, since an exhaustive search to discover all possible plans mostly cannot be performed. Therefore, the set of all possible plans PP is usually a super–set of all already known plans P. This relationship is expressed in the equation: $R \supseteq PP \supseteq P$ (*Figure 2.1*). The relationship $PP \supseteq P$ is discussed below in the context of the frame problem of artificial intelligence.

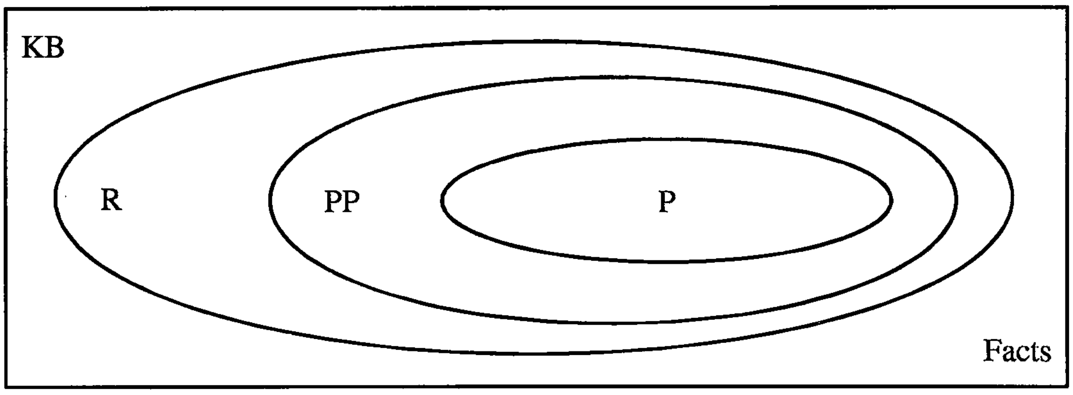


Figure 2.1: Relationship Between Plans And Rules Of A Knowledge Base

2.2 Definitions For Planning

Planning is a cognitive process that can be simulated in general as a search process. From the perspective of engineering we will emphasise here the rule–based approach for planning.

Planning As A Search Process

Planning is the process of exploring the search space opened by all possible plan steps and building alternative implementation sequences for later application. The reason why planning is usually set equal with searching in the literature is that in a simulation environment the exploration of a specific plan step is set equal to its application [Rich et al.; 1991]. However, in the implementation of a plan, a plan step cannot always be undone or can cause serious difficulties, such as unrecoverable damage or additional cost for the recovery of damage. This effect can happen in virtual environments, such as in software agents, and more costly in real–world environments, such as in robotics. Examples are deleting unrecoverable data by a software agent or breaking a tool by a robot. In a simulation environment however, undesired steps can be undone.

Engineering–oriented Definitions For Planning

Planning is a problem–solving technique that involves determining a course (or sequence) of actions that take a system from an initial state to a desired or goal state [Schalkoff; 1990]. In each sequence one or more rules are applied on the facts of the knowledge base. Any modification of a fact changes its state, thus the state of the knowledge base changes, hence the state of the agent will change. Therefore, even while planning an agent may change its behavioural state.

Since planning can be represented with searching, it inherits all the properties of searching. For instance, problem decomposition applies to planning as it does to searching. Hence, a plan may be sub–divided into several sub–plans (*Figure 2.2*). However, generating sub–plans can become problematic, when the goals of the sub–plans are conflicting, for instance in case of interactions between the sub–plans.

It is important to note that the number of ordering of n rules is n!, and therefore the potential computational complexity in plan generation is nontrivial [Schalkoff; 1990].

Another difficulty may occur in the development of a plan implied by the fact that planning involves a series of local actions with the objective of achieving some global goal [Schalkoff; 1990], which is implied by the local and global frame of the problem.

Cycles in a plan are a further difficulty for the planning component to detect and escape or to avoid within the planning process. Where, the planer may copy the related solution techniques into the plan as well, in order to resolve a cycle during the plan implementation.

A further definition of the planning process is state–oriented. Planning is formulated as a logical inference problem, using situation calculus. A planning problem is represented in situation calculus by logical sentences that describe the main parts of a problem: [Russell et al.; 1995]

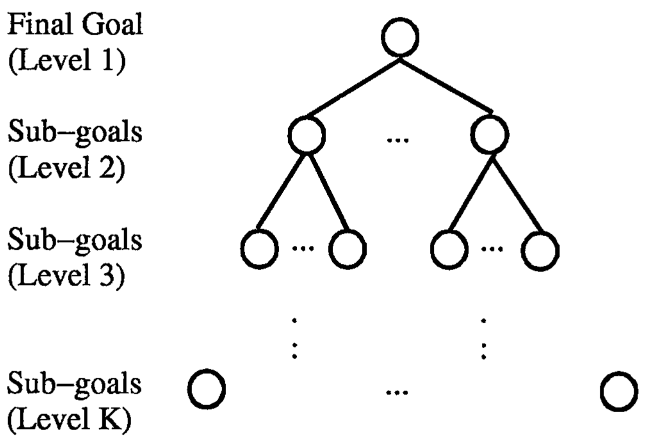


Figure 2.2: Hierarchical Plan Execution

- Initial state: An arbitrary logical sentence about a situation.
- Goal state: A logical query asking for suitable situations.
- Operators: A set of descriptions of actions.

Abstract Components Of A Planning System

Common to all planning systems are some components, which are described below briefly: [Rich et al.; 1991]

- Selection of applicable rules: One approach is, first to calculate the difference between the current state and the target state, thereafter to identify those rules that are relevant for the reduction of the difference.
- Application of a rule: By maintaining lists for the applicability of rules.
- Determining a solution: By comparing the current state with the target state.
- Detection of dead–ends: Cycles or irrelevant paths can be detected by examining the behaviour of the difference between the current state and the target state.
- Restoration of a plan: If the decomposed sub–plans do only approximate the solution, then the sub–plans are modified accordingly.
- Improvement of the application of a plan by sub–dividing the goal into succeeding sub–goals and/or alternative plans.

From this specification one can conclude that the plan generation does not always result with a successful plan. In general this is due to the NP–complete [Baral et al.; 1999] nature of planning. Various plan generation algorithms and planing strategies, which use besides the above basic components further planning techniques, are discussed in the literature, such as state propagation, the triangle table, hierarchical planning, parallel planning [Schalkoff; 1990], partial–order planning, hierarchical decomposition, sharing, and approximation, planning and execution monitoring, conditional planning and re–planning [Russell et al.; 1995].

It should be pointed out that we are not interested in the details of those analytical approaches for planning. In the next chapter we will see that they find in the average only sub–optimal solutions.

The Frame Problem Of Artificial Intelligence Affects Planning

The basis of the frame problem of artificial intelligence is that it is unreasonable to enumerate how a set of facts changes as actions or events occur [McCarth et al.; 1969], [Schalkoff; 1990]. In real implementations the difference implied by the relationship $PP \supseteq P$ can be large and even increase with any further reasoning over the knowledge base. This effect is due to the frame problem. Since it is unreasonable for the agent to make an exhaustive search in each planning process, the actual set of plans P will mostly be smaller than all possible plans PP of the agent.

2.3 Planning As A Learning Process

All cognitive processes are usually interrelated with each other. For instance, believe–desire–intention relationships or learning–planning relationships. In general, planning always results in learning, since the result of a planning process, namely the plan, is a new concept and is available for further cognitive processes, it can be considered as learning. However, the reverse relationship is not true. For instance, discovering unknown properties of an object in the environment is learning that does not necessarily requires a planning process (*Figure 2.3*).

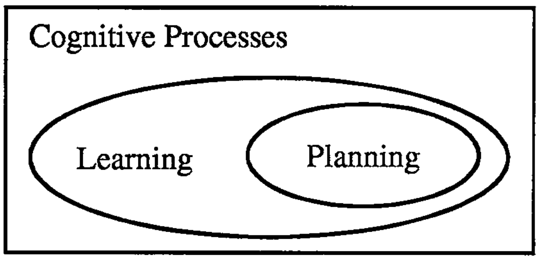


Figure 2.3: Relationship Between The Cognitive Processes Learning And Planning

The difference becomes apparent, when we compare offline with online planning. In case of offline planing, the planning process may be set equal with searching, only if after backtracking from undesired paths the previous state can fully be recovered. In case of online planning on the other hand, the search may continue with the immediate implementation of each plan step, until a not recoverable plan step is reached. This will cause the search to enter first a simulation environment, continue the search there with simulated implementation of each plan step, until a decision on this part of the plan can be made that it may be implemented without causing damage.

2.4 Relationships Between Plans Of Different Knowledge Bases

If the knowledge bases of the agents of a multi–agent system are identical ($KB_i = ... = KB_n$), then we will refer to this with knowledge base homogeneity, otherwise knowledge base heterogeneity ($KB_i \neq ... \neq KB_n$), or in short homogeneous and heterogeneous agents, respectively. In both cases each agent may plan independently from the others or co–operatively with the others.

Homogeneous agents can co–operate in any plan, since their knowledge bases are equal. Heterogeneous agents may co–operate, if their plans are at least partially equal ($\Sigma P_i \cap \Sigma P_j \neq \emptyset$), which guarantees that at least a plan step, that is identical for both, can be found and be declared as a mutual goal.

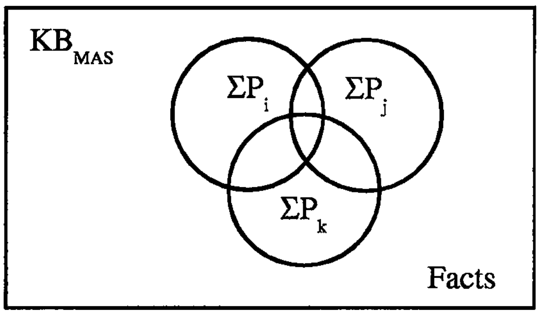


Figure 2.4: Relationships Between The Plans Of Three Heterogeneous Agents

The number of equal rules decreases dramatically with increasing number of heterogeneous agents with increased knowledge bases differences (*Figure 2.4*). This kind of analysis of the knowledge base may appear too abstract for analytical planning approaches. But in the following chapters we will show that exactly this abstraction level will be the operational level for our co–evolutionary planning approach.

2.5 Planning In Multi–agent Systems

The above basics for planning systems imply centralised planning. Extensions to distributed and/or parallel panning were the first approaches towards planning components for autonomous systems. Which are however the functional aspects of a system. From data–oriented point of view, important extensions for planning systems were introduced with the blackboard approach.

Blackboard Models

A blackboard is a data structure for storing and sharing inherently heterogeneous data. It serves the system as a co–ordination mechanism. Systems that utilise a blackboard consist usually of loosely coupled, independent and distributed components. Where, the blackboard is used by the components to put some data on the board for solving a problem of the whole system. This idea for co–operative planning or problem solving was a predecessor for multi–agent systems. With the significant difference that the problem is solved centralised on a board usually without negotiation between the components. In a naive way, it can be interpreted as putting each component's desired goal state on the board and solve the problem there by finding a solution for all goals (*Figure 2.5*).

Various systems had adopted this approach, especially to solve problems on hierarchical heterogeneous data structures, such as simulation of a distributed knowledge–based problem solving system operating on an abstract version of a vehicle monitoring system [Luke; 1998]; military mission planning [Pearson; 1985]; errand–planning on the blackboard with general features of plan behaviour, details of plan behaviour, levels of abstraction, multi–directional processing, alternative executive decisions [Hayes–Roth et al.; 1988].

For domains where the integration of inherently heterogeneous data structures appear too complicated and/or known communication facilities are insufficient, agents may put their data structures on a blackboard and solve the problem there. For example, co–ordinating multiple agents on a blackboard in a real–time strategy game [Cavazza et al.; 2001]; a blackboard architecture for building a hybrid case based reasoning system for fire field modelling, for the integration of qualitative spatial reasoning knowledge from domain experts [Petridis et al.; 2001].

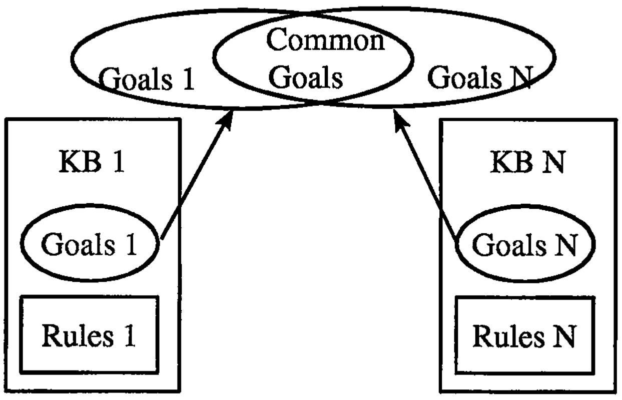


Figure 2.5: Multi–agent Planning By Seeking For Common Goals

Planning Concepts Of Intelligent Agent

The initial interpretation to solve common goals centralised on a blackboard (*Figure 2.5*), was re–interpreted and adopted for multi–agent systems, in analogy to the human model, as a distributed problem solving approach. The planning components of multi–agent systems employ usually more sophisticated planning techniques than the initial knowledge–based systems or expert systems. Further, a planning component, as a specific problem solving task, is more integrated with the other components for cognitive processes of intelligent agents. This is one reason why discussions on planning in agent systems can no more be separated from the other components. Another reason is, since most implementations of intelligent agents run their cognitive processes in interpreter mode, this capability provides a simulation environment where plan steps can be undone. Hence, planning can be set in most cases equal to searching in the plan space.

Planning in multi–agent systems requires communication, co–operation, and control structures to be considered in the planning process as well as within the plan for its later implementation. Planning alternatives, with respect to the grade of distribution, can be summarised as follows: [Ferber; 1999]

- Centralised planning
- Distributed planning with centralised co–ordination

- Distributed planning

An early example for distributed planning with centralised co–ordination is the global partial planning proposal [Durfee et al.; 1991], where the individual partial plans of the agents must be merged to construct a global one for all.

A general approach for distributed planning is for instance, the co–operative problem solving approach, expressed as a theory in a quantified multi–modal logic, consists of the four stages: [Wooldridge et al.; 1999]

- Recognition, in which an agents identifies the potential for co–operation.
- Team formation, in which the agent solicits assistance.
- Plan formation, in which the newly formed collective attempts to construct an agreed joint plan.
- Execution, in which the members of the collective play out the roles they have negotiated.

Other multi–agent planning approaches are automated planning for open architectures [Reiher et al.; 2000], where the adaptation service for end–to–end network connections is based on an automated planning approach; an integrated planning and scheduling approach combining sub–tasking and virtual clustering of agents with a modified Contract Net protocol [Maturana et al.; 1996]; the planning component of the RETSINA information gathering framework interleaves planning and execution actions to solve the problem of partial domain knowledge for the purpose of information fusion [Paolucci et al. 2000]; the proposal of the multi–agent planning system IMPECTing SHOP (Simple Hierarchical Ordered Planner) that interacts with external information sources [Dix et al.; 2000].

The following are the major advantages of planning components of multi–agent systems:

- Distributed planning: Enables joint planning of autonomous agents, utilisation of synergy effects, and parallel computation.
- Negotiation: Enables adaptation and learning.

In the next chapters we will see how evolutionary computation may improve the efficiency and success of a planning process further.

CHAPTER THREE
CO–EVOLUTIONARY COMPUTATION

Any cognitive process can be reduced to a search process that seeks for a solution to a given problem. Therefore, simulations of cognitive processes are usually implemented by search procedures, which traverse the search space that is opened by all possible mental states of the individual. Searching, even in a small rule–base can lead to an extremely large search space, which results from combinatorial explosion. In order to improve the computational efficiency, various types of constraints are introduced. For instance, restriction to a specific domain or application of heuristics. Heuristics have the capability to recognise uninteresting search paths before they have been explored entirely and to redirect the search to more promising paths. However, they are difficult to apply, since the designer must decide where and when to apply domain–dependent and domain–independent heuristics. Even the same heuristic may need to be applied in different terms, depending on the current context, meaning different sub–spaces of the search space.

In these terms, the most generic heuristic is to explore a search space, independently from any further heuristic at random positions towards the optimum value. The search space of a sigle attribute can be represented by a function in two dimensions. Optimum values in two dimensions can be found by successively examining all values of the function. Accordingly, the search space of a system of multiple attributes can be represented by a multi–dimensional search space. While searching for optimum values in a multi–dimensional search space, each attribute's value will probably be examined multiple times, until its optimum contribution to all attribute combinations can be found (*Figure 3.1*).

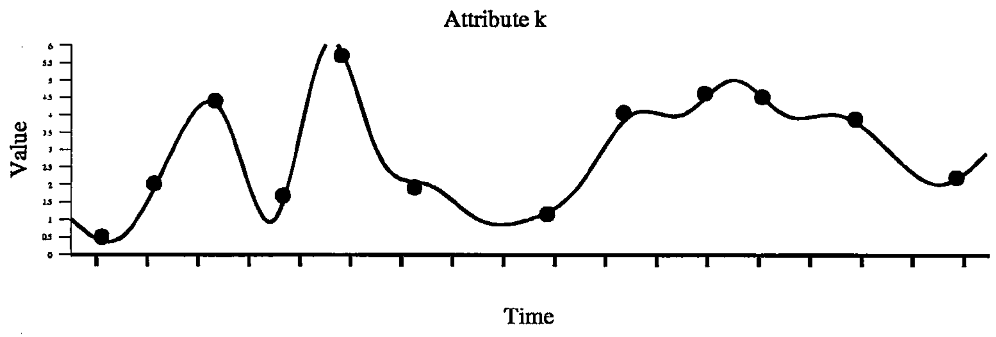


Figure 3.1: Random Exploration Of The Two–dimensional Search Space Of A Single Attribute Within A System Of Multiple Attributes

Seeking for optimum values by successively examining all values of all attributes however is too exhaustive. For this purpose, evolutionary computation has proven to be a relative stable approach that always can converge to one optimum solution, since it explores the search space at random values and redirects the search towards the most promising values. even if it is not guaranteed to be *the* optimum solution.

Evolutionary computation is utilised in machine learning as a generic technique to seek for desired solutions in the search space of a problem and thereby learn the solution as well as its path. The co–evolutionary approach is an improvement of the algorithms that has its origin again in nature. Where different populations naturally evolve by co–operating and/or competing with each other in the same environment.

2.1 Evolutionary Computation

Randomness is almost always the underlying mathematics of the balances of our nature. Each sub–system of the "super–system", nature, can work harmonically with the others, due to its adaptation mechanisms. If a system has no information about how to adapt to the changing environment, then the most successful approach is randomly to modify its properties and iteratively to evaluate feedback, until it adapts. Therefore, randomness, was discovered early in the history of computation as a powerful mathematics for the simulation of adaptation.

History Of Evolutionary Computation

Already in the early years of artificial intelligence, when researchers were establishing this field by introducing various techniques to support machine intelligence, one of them was the evolutionary approach [Box; 1957], [Fraser; 1957], [Friedman; 1959]. The idea was, within the concept of machine learning, to improve the search in large problem spaces by randomly exploring it with evolutionary approaches, where Breadth First Search, Depth First Search, and their extensions were too exhaustive. However, no successful implementations of evolutionary algorithms are known from those years. Instead, methods based on simplified random search, such as Monte Carlo, Hill Climbing, and Simulated Annealing, were more popular.

Nevertheless, a few pioneers started to reported progress in evolutionary computation in some domains. Such as function optimisation with evolution strategies [Rechenberg; 1973], evolution of finite state machines through evolutionary programming [Fogel et al.; 1966], a class of adaptive systems we now call Genetic Algorithms [Holland; 1975], [De Jong; 1975] and evolving programmes with Genetic Programming [Koza; 1989].

The fundamental principles on which all of the computational models of evolution are based can best be summarized by nineteenth century naturalist Charles Darwin, who was fascinated by the origin of the complex forms of life existing in nature. In his introduction to "The Origin of Species", [Darwin; 1859] makes the following observation:

As many more individuals of each species are born than can possibly survive; and as, consequently, there is a frequently recurring struggle for existence, it follows that any being, if it vary however slightly in any manner profitable to itself, under the complex and sometimes varying conditions of life, will have a better chance of surviving, and thus be naturally selected. From the strong principle of inheritance, any selected variety will tend to propagate its new and modified form.

To adapt these principles to the solution of a target problem of interest, we construct an evolutionary simulation in which the individuals are alternative solutions to the target problem. The frequently recurring struggle for existence that we see in nature is inherent in our model due to the limited computer resources we can devote to our simulation. This is expressed both in restrictions on the number of alternative problem solutions we store in computer memory, and the computational resources available for processing these solutions. Variation between individuals is a result of making random changes to the population of evolving solutions, and from recombining pieces of old solutions to produce new solutions. Darwin's process of natural selection can be modeled by imposing a selection distribution on the population of solutions such that the better ones have a higher probability of being recombined into new solutions and thereby preserving the attributes that made them viable. Alternatively, we can ensure that the poorer solutions have a higher probability of being eliminated from the population. To determine how good or bad a particular solution is, the evolutionary algorithm applies the solution to the target problem within the context of a domain model, and evaluates its fitness through the use of appropriate metrics.

Basic Concepts Of Evolutionary Computation

In summary, the functional principles of Darwinian evolution are

- Organisms have a finite lifetime. Therefore, propagation is necessary for the continuation of the species
- Offspring vary to some degree from their parents
- The organisms exist in an environment in which survival is a struggle with difficulties, where the variations among them will enable some better to adopt to these difficulties
- Through natural selection, the better–adopted organisms will tend to live longer and produce more offspring
- Offspring are likely to inherit beneficial characteristics from their parents, enabling some of the species to become increasingly well adapted to their environment over time.

The basic concepts of evolutionary algorithms mimic the above principals of adaptive processes observed in our nature. These concepts are summarised here with examples from the application area planning.

- Gene: The smallest information unit in evolutionary computation. One gene represents one property. For instance, a specific type of plan step.
- Species: An individual or chromosome with a sequence of genes. Where, each gene represents one property. A solution of the problem space is represented first in a phenotype, then mapped into a genotype. For instance, a specific plan consisting of several plan steps can be represented by a phenotype.
- Population: A number of individuals with similar properties. For instance, alternative plans.
- Cross–over: A major variation of a species by exchanging some of its genes with those of another randomly selected species. Tying out an alternative plan by exchanging major plan steps of two randomly selected plans.
- Mutation: A slightly variation of a species by randomly modifying a few of its genes. Tying out an alternative plan by slightly modifying a few of its plan steps.
- Reproduction: Forming the next generation of a population of parent species by selecting those species as offspring, which fit the best to the desired target. Forming a new set of plans by selecting those plans which are the closest to the desired goal.

In dead, the clearer these computational concepts became, and the more efficient hardware was available, the more successful evolutionary algorithms could be implemented over time. Finally, with this success, and the algorithms known as robust and powerful adaptive search mechanisms, evolutionary computation has established itself as an own research discipline and became more popular in the 1980s.

The "evolution" of evolutionary algorithms was driven on one hand by the fascination of the adaptation capability of species inside the nature, and on the other hand by their successful applications in various computational domains.

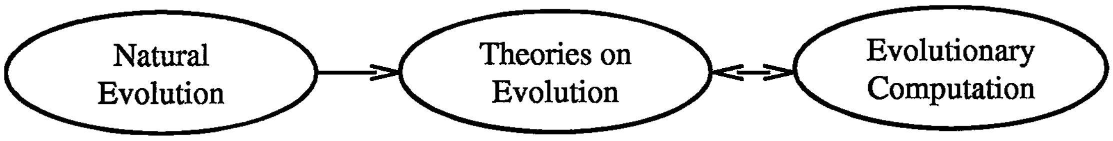


Figure 3.2: Sources For Evolutionary Computation

Accordingly, one could sketch this "evolution" by depicting the information flow between the problem domains natural evolution, evolutionary algorithms, and applications of evolutionary algorithms, as interrelationships among them (*Figure 3.2*).

The principle focus of this research is on the latter: the application of evolutionary algorithms in the problem domain of multi–agent planning. For this purpose, first the properties of evolutionary algorithms are examined, in order to apply them as efficient search mechanisms for multi–agent planning.

2.2 Evolutionary Algorithms

Evolutionary algorithms is a umbrella term, used to describe computer–based problem solving systems which use the above basic concepts as key elements in their design and implementation. Some of these concepts are emphasised in one evolutionary algorithm more and less in another, depending on the algorithm's specific paradigmas, such as the simulation of evolution of an object or the genetic recombination of information. Usually, they are classified with the following terms Evolutionary Programming, Evolution Strategies, Genetic Algorithms, and Genetic Programming. The following are short specifications of the different classes: [Bäck et al.; 1992], [Heitkötter et al.; 2001], [Mitchell; 1997]

Evolution Strategy

Evolution Strategies imitate the effects of genetic procedures on the phenotype. The presumption for coding the variables in the Evolution Strategy is the realisation of a sufficient strong causality, meaning small changes of the cause must create small changes of the effect. The underlying theory is the evolution window, which states that evolutionary progress takes place only within a very narrow band of the mutation step size. This fact leads to the necessity for a rule of self–adaptation of the mutation step size. Early work in this area is due to [Rechenberg; 1964], [Schwefel; 1977].

Below algorithm sketches the generic form of evolution strategy; where, μ is the number of parents and λ the number of offspring:

```
(define (evolution-strategy population)
(if (terminate? population)
    population
    (evolution-strategy
         (select
              (cond (plus-strategy?    // μ of (μ+λ)
    survive
                   (union (mutate
                        (recombine population))
                        population))
                   (comma-strategy?    // λ offspring
    survive
                        (mutate
                             (recombine
    population))))))))
```

Evolutionary Programming

Evolutionary programming is a stochastic optimisation strategy with emphasis on the behavioral linkage between parents and their offspring, rather than seeking to emulate specific genetic operators as observed in nature. Evolutionary programming is a useful method of optimisation when other techniques such as gradient descent or direct, analytical discovery are not possible. Combinatorial and real–valued function optimisation in which the optimisation surface or fitness landscape possesses many locally optimal solutions, are suitable for evolutionary programming. However, Evolutionary Programming typically does not use any crossover as a genetic operator. An early early contribution to this area is [Fogel; 1966].

Below is a sample algorithm for evolutionary programming:

```
BEGIN EP
g := 0                         // initial generation
Initialize population P(g)     // start with random species
Evaluate population P(g)       // i.e. compute fitness
    values
WHILE NOT done DO              // while solution not found
    Mutate P(g)                // perturb genetic
    information
    Evaluate P(g)              // i.e. compute fitness
    values
    g := g + 1                 // count generations
    Select P(g) from P(g - 1)// stochastically select
    survivors
END WHILE                      // ... time, fitness, etc.
END EP
```

Genetic Algorithm

A genetic algorithm is an iterative procedure that consists of a constant–size population of individuals, each one represented by a finite string of symbols, known as the genome, encoding a possible solution in a given problem space. Genetic algorithms are applied to spaces which are too large to be exhaustively searched. The symbol alphabet used is often binary due to certain computational advantages, such as shifting, flipping, storage, time etc. However, this has been extended in recent years to include character–based encoding, real–valued encoding, variable–length encoding, and tree representations [Michalewicz; 1996], which distinguish genetic programming.

Genetic algorithms are used for a number of different application areas. An example of this are multidimensional optimisation problems in which the character string of the chromosome can be used to encode the values for the different parameters being optimized. The chromosomes are manipulated by crossover and mutation operations. As earliest references to this area one can find [Holland; 1975] and [Goldberg; 1989] in the literature.

A sample genetic algorithm is the following:

```
BEGIN GA
g := 0                          // initial generation
Initialize population P(g)      // start with random species
Evaluate population P(g)        // i.e. compute fitness
    values
WHILE NOT done DO               // while solution not found
    g := g + 1                  // count generations
    Select P(g) from P(g - 1)// stochastically select
    survivors
    Crossover P(g)              // recombine genetic
    information
    Mutate P(g)                 // perturb genetic
    information
    Evaluate P(g)               // i.e. compute fitness
    values
END WHILE                       // ... time, fitness, etc.
END GA
```

Genetic Programming

Genetic programming is the extension of the genetic model of learning into the space of programs. That is, the objects that constitute the population are not fixed–length character strings that encode possible solutions to the problem at hand, they are programmes that, when executed, are the candidate solutions to the problem. These programmes are expressed in genetic programming as parse trees, rather than as lines of code. Thus, for example, the simple program (a + b * c) would be represented as a tree. In pre–fix notation it looks like this: (+ a (* b c)). To be precise, suitable data structures are linked together to achieve this effect. The programmes in the population are composed of elements from the function set and the terminal set, which are typically fixed sets of symbols selected to be appropriate to the solution of problems in the domain of interest. The crossover operation is implemented by taking randomly selected sub–trees as individuals and exchanging them. Genetic programming usually does not use any mutation as a genetic operator. First publications on this area are [Cramer; 1985], [Koza 1989].

Differences Between The Algorithms

Though, no one could draw strict boundaries between the algorithms implemented in praxis, the four classes have some obvious differences:

- Genetic algorithms usually simulate evolution at the level of the genome, while the others directly evolve phenotypes.
- Genetic algorithms and programming usually uses both crossover and mutation, whereas evolution strategies and evolutionary programming usually do not use crossover at all.
- Genetic algorithms usually represent individuals with binary strings, evolution strategies real–valued vectors, genetic programming trees, and evolutionary programming graphs or any other form.
- Genetic algorithms and programming usually emphasise crossover, whereas evolution strategies and evolutionary programming emphasise mutation.
- Genetic algorithms usually implement the mutate operator though bit flipping, evolution strategies though adding Gaussian noise, genetic programming by exchanging sub–trees, and evolutionary programming by adding, deleting, or changing states to finite state machines.

Single Population Applications

The application area of evolutionary algorithms encompasses all problem classes in which exhaustive search cannot be avoided with other techniques. Since the resulting application area is relative broad, we restrict ourselves to applications in the agent domain. Furthermore, from our perspective of multi–agent planning, we separate this area into single and multiple population applications. Some application areas for the former are mentioned in the following.

Hybrid systems combine evolutionary algorithms with more conventional "lifetime learning" techniques [Paredis; 1995], [Paredis; 1996]. Here, populations of agents evolve, while each agent possesses behavioral plasticity. For example, evolution may operate on a population of artificial neural network topologies, while back–propagation or the Hebbian rule facilitates lifetime improvements of each network via weight updates. In general in machine learning, to speed up the search process and to find alternative solutions. For instance, evolutionary adaptation of neural robot sensors [Balakrishnan; 1998]; learning rules [Juillé et al.; 1998]; improving learning by evaluating the whole population instead of only the representative for rule–based systems and artificial neural networks [Yao et al.; 1996]; evolutionary learning to discover complex emergent properties in complex domains [Funes; 2001]; evolution of co–ordination strategies represented by schedules for homogeneous agents [Baray; 1999]; simulation of adaptation with respect to resource exchange [Dominiak; 2001]; resolving social dilemma in multi–agent systems [Arora et al.; 1996]; etc.

2.3 The Co–evolutionary Approach

In this section, we will discuss an important aspect of evolutionary computation that is again borrowed from nature and that is utilised for improving the artificial adaptation capabilities in evolutionary computation by evaluating emergent inter–relationships.

History Of Co–evolutionary Computation

We have introduced evolutionary algorithms as a modelling approach for the adaptation capability of processes found in nature. Various such adaptation processes more or less interact with each other, which depends on diverse natural influences. As result, the adaptation processes keep the nature in a harmonic balance. Hence, the actual reason for evolution are the interactions that are necessary for adaptation to the environment. This observation inspired researchers on evolutionary computation to model these interactions, too, in order to improve the adaptation capabilities.

The algorithms are based on a biological concept, according which competing populations may reciprocally drive one another to increasing levels of performance and complexity by producing an evolutionary "arms race" [Dawkinset al.; 1979]. Competing species are engaged in a feedback loop, in which an adaptive change by one creates a new challenge for the other, leading to an adaptive change on its part and so on. For instance, the co–evolution of parasites and hosts [Hillis 1991], [Kniskern et al.; 2001]. Some further definitions for co–evolution are:

- Co–evolution is evolution that involves successive adaptive changes in two or more independent species.
- In co–evolution individuals evolve with respect to other individuals, whereas in evolution individuals evolve with respect to a fixed environment [Potter; 1997].
- According to the NKC model [Kauffman; 1993], in an evolving organism of N genes, co–evolution is the contribution of a gene to the fitness of the individual depends on K other neighbour genes on the same chromosome and C other genes from each of the S_i species. The fitness of the whole system is optimised when $K \approx C \times S_i$. When the fitness of each individual is roughly equally affected both by its own fitness landscape and by the complex couplings to other individuals [Bak et al.; 1992].

An early work is the so called classifier system, which extended the basic evolutionary model to allow co–adapted sub–components to emerge inside a single population [Holland et al.; 1978]. The application domain was rule–based systems, in which rules evolved using a genetic algorithm. Most of the publications on co–evolution are based on this idea of representing different species of an environment in form of interacting sub–population. Whereby, different variations are discussed in the literature, such as crowding [De Jong; 1975], island model [Grosso; 1985], fitness sharing [Goldberg et al.; 1987], competitive fitness sharing [Goldberg et al.; 1987], emergent fitness sharing [Smith et al.; 1993], niching [Goldberg et al.; 1987], [Mahfoud; 1995], [Darwen et al.; 1996], [Rosin; 1997], [Rosin et al.; 1997], [Horn; 1997].

All approaches have the sub–population as common concept, which represents a sub–optimal solution that sometimes can have useful additional properties for the final solution or that can represent alternative solutions to be preserved until the end of the learn process. The major concern of related research is how to discover sub–

populations and its significant properties, and how to preserve them until the end of a solution.

Improving Evolutionary Computation With Co–evolution

Experimental studies show that the co–evolutionary model is significantly more efficient than the evolutionary model [Paredis; 1995], [Paredis; 1998]. The following is a summary of the advantages of the co–evolutionary approach over evolutionary once:

- Competitive environments evolve better solutions for complex tasks [Angeline et al.; 1994].
- Co–evolutionary approach to learning sequential decision rules [Potter et al.; 1995].
- The co–evolution method solves a larger class of optimization problems [Tahk et al.; 1999].
- Co–evolution can be used in both learning and optimisation [Nolfi et al.; 1999].
- Co–evolution can enhance the adaptive power of artificial evolution [Nolfi et al.; 1998].
- The co–evolutionary approach improves the search by utilising techniques, such as niches [Rosin; 1997]. Further, it enables the representation of the evolution of inter–related populations.
- For a given task, the co–evolutionary model finds 82% of high quality strategies, whereas the evolutionary model finds only 2% of those strategies [Pagie et al.; 2000]. The large difference in search efficiency of the processes is analysed by comparing the evolutionary dynamics of the two models.

For our purposes we conclude: A co–evolution model may be interpreted as co–operative or competitive adaptation inside the domain of a specific concept. Further, fitness sharing can be utilised to strengthen/weaken co–operation/competition between the populations.

However, the generalisation capability of co–evolutionary algorithms seams to have a handicap in specific behavioural situations of the search space, wich are called in the literature the *Red Queen Dynamics/Effect* [Ridley; 1993], [Pagie et al.; 2000]. In simulations of co–evolutionary armes race, interacting populations alter each other's fitness landscapes, which is caused by dynamics of this landscape [Cliff et al.; 1995]. Solution approaches try to detect and escape from such situations, rather than preventing them.

Planning With Co–evolutionary Algorithms

We have already shown that the co–evolutionary model is an improvement of the evolutionary model. Hence, the search efficiency of any evolutionary algorithm for planning could be improved by a co–evolutionary one. Therefore, we do not differentiate in the following sample references for planning between evolutionary and co–evolutionary approaches. For instance, co–evolutionary computation for path planning [Paredis; 1997]; the automatic programming of agents that learn mental models and create simple plans of action [Andre; 1995]; global planning from local eyeshot, which is a implementation of observation–base plan coordination in CoboCup simulation games [Yunpeng et al.; 2001]; planning schedules for parallel genetic algorithms of which each one solves a sub–problem [Husbands et al.; 1991].

2.4 Co–evolutionary Algorithms In Multi–agent Systems

The agent concept emerged at the end of the 1980s as a synthesis of various artificial intelligence concepts and ideas, some of which are distributed knowledge bases, distributed expert systems, blackboard systems, intelligent user interfaces, combination of different techniques, such as evolutionary computation, neural networks, fuzzy logic, etc. On the other hand, related technological progress enabled its realisation, such as hardware improvements and Internet. The intention was, and still is, to project these techniques on the product agent, in order to have a concrete experimental target that can be equipped with any further technology that promises an improvement for the intelligent behaviour.

Initial Work On Rule–based Systems

The utilisation of evolutionary computation for automated learning of meta rules through generalisation of primitive rules of a domain was a promising approach for doman–independent systems. For historical reasons, rule–based system are sometimes called classifier systems. The following is a brief discussion of initial work that combined rule–based systems with evolutionary/co–evolutionary computation.

According the encoding of rules in chromosomes, two strategies were common. In the Pitt Approach, an entire rule set is represented with a chromosome, whereas, in the Michigan Approach, a single rule is represented with a chromosome [De Jong; 1990]. An improvement of the learn efficiency of SAMUEL [Grefenstette; 1989], [Grefenstette; 1991] was achieved with genetic algorithms. A system that can learn and adapt reactive decision rules from simulations of multi–agent environments by using genetic algorithms [Grefenstette; 1992]. GABIL continuously learns and refines concept classification rules by using a genetic algorithm [De Jong et al.; 1991]. Lamarckian learning in the multi–agent environment of SAMUEL [Grefenstette; 1991]. Evolving reactive agents for the central food foraging problem by using genetic programming [Koza; 1992].

A typical classifier system is one which simulates a game. Since the domain of a game is, with respect to its rules, complete, it provides an ideal environment for searching for related game strategies and allows relative easy justification of the results. Furthermore, it can be used as a benchmark. For instance, co–evolution of agents that simulate the game of Tag [Reynolds; 1994], Tic Tac Toe [Rosin et al.; 1997], Backgammon [Pollack et al; 1998], Poker [Barone et al.; 1998].

Multi–agent–oriented Requirements For Co–evolutionary Algorithms

Most of the here analysed work is based on the assumption that species co–evolve in the same environment and therefore each species can be represented with one sub–population. Most of the work on evolution in multi–agent systems makes this assumption, such as scheduling homogeneous sub–problems [Husbands et al.; 1991]; evolutionary computing in co–operative multi–agent environments [Bull et al.;

1996]; evolution of communicating agents with genetic programming [Iba; 1998]; the co–ordination model proposed in [Baray; 1999] pre–assumes homogeneous agents and therefore uses a single population for representing homogeneous strategies; automatic acquisition of strategies by co–evolutionary learning for the N–iterated Prisinor's Dilemma game [Yao; 1997]; evolutionary adaptation of the autonomy in multi–agent systems [Gerber; 1999]; even in distribution and parallelisation of an evolutionary algorithm [Nangsue; 1998] homogeneous populations are assumed. In most games the rules apply equally to all species, whether they co–operate of compete.

In multi–agent systems such homogeneous structures usually do not exist. The knowledge bases of the agents are mostly heterogeneous, with respect to the stored rules. Besides the heterogeneity aspect, we have identified the following properties of multi–agent systems as properties to be simulated by co–evolutionary computation:

- Interaction: Co–operation and competition structures of agents and plans
- Variety: Homogeneous and heterogeneous structures of agents and plans
- Scaleability: Multiple agents and plans
- Focus: Priorities between agents and plans
- Diversity: Alternative plans for the agents

Our goal is to find modelling capabilities within co–evolutionary computation that can represent all the desired structures of multi–agent systems.

CHAPTER FOUR
THE CEVOP METHODOLOGY

The most prominent characteristic of co–evolution is that the fitness of an individual depends on fitness of other individuals, which can be from the same population or a different population. Here we use co–evolution to generate plans for multi–agent systems with heterogeneous rule bases, which are represented with different co–evolving populations. Whereby, the co–operation/competition structures among the agents are represented by further fitness functions. In this chapter, a detailed specification of the methodology and its concepts are introduced and compared with other approaches.

4.1 Major Concepts

Based on the ideas and the major claims of the thesis, the required concepts are formulated and combined in this methodology. The required concepts are the data structures goal and plan, and the functionality to co–evolve several individual plans to alternative global plans, to adjust the suggested plans with the initial plan, and to negotiate agreement on one of the suggested plans.

To summarise the idea of the Co–evolutionary Planning (CEVOP) methodology, several agents join a planning process, whereby each agent publishes its goals and a set of all its possible plan steps. This information is processed in the planning process in order to find common goals that could enable co–operation and on the other hand to find goals that are contradicting and therefore represent competing goals.

Representation Of Rules/Goals/Plans

The knowledge base of an agent usually consists of a set of rules R, which are used for various purposes. Some of the rule are used to construct plans P or to be part of a specific plan P_a. Thus, each plan step p_i of a plan P_a is a rule of the knowledge base. The set of all possible plans PP, with $P \subseteq PP \subseteq R$, provides a basis for searching for plans $P_a^{'}$, which are similar to $P_a \approx P_a^{'}$. If this approach is applied to all agents in an adaptive environment, then we may find some plans, which are similar to all agents. Thus, these agents may co–operate with each other, since their goals will be at least partially compatible. Otherwise, for some or all plans we may not find compatibility. In that case those plans will be considered as incompatible, which we interprete as potential for competition.

In general, a plan is the result of a planning process. Constructing a plan in the planning process is driven by the goals that were set before the planning process. Principally, each rule of the knowledge base represents a goal. Hence, each step of a plan is a sub–goal that needs to be reached by applying the plan step. Because of this interchangeability of rule, goal, and plan step, we will use these terms in this work to refer to the same matter.

Co–evolving Several Plans To Global Plans

The planning process of the CEVOP methodology has two aspects: First, the adaptation process within the co–evolutionary algorithm. Second, the adaptation process among the agents, in order to negotiate a final agreement on the generated plans. A specific CEVOP solution may implement the aspects in the following ways:

- Alternative plans are generated already with the co–evolutionary algorithm. Thereafter, the agents may agree on specific plans or cancel the planning process.

- The co–evolutionary algorithm generates only one plan in one run. Thereafter, if the agents cannot agree on specific plans, then the co–evolutionary algorithm is called again with slightly modified possible plan sets PP. This process is iterated, until an agreement on specific plans becomes possible or this planning process is canceled.
- The third approach may combine these two in one solution.

The adaptation process of a plan from P_a to P_a' within the co–evolutionary algorithm is oriented on the possible combinations of all plan steps p_i. The semantics of a plan step or of a sequence of plan steps is ignored. Thus not all generated plans may comply with the plan execution semantics of the agents and need to be confirmed by the agents before the plan implementation. Hence, the final agreement on specific plans is necessary in all solution cases.

Design Issues

From previous discussions we conclude some major requirements for our co–evolutionary planning approach. The requirements are summarised in form of design issues for the CEVOP methodology as follows: (*Table 4.1*)

- Agent interaction: Co–operation/competition interrelationships of agents should be modelled.
- Plan variety: For agents with homogeneity/heterogeneity knowledge bases the algorithm should find compatible/contradicting plans.
- Scaleability: All interactions of a multi–agent system should be modelled.
- Focus: Intensity of a specific interaction should be represented in the final plan.
- Plan diversity: The algorithm should be able to generate several alternative plans.

Table 4.1: Concepts For Co–evolving Plans Of Multiple Agents

Concept	Range	Purpose
Interaction	Co–operation	Evolved Plan Sets Should Allow For Resource Sharing
	Competition	Evolved Plan Sets Should Disallow Resource Sharing
Variety	Homogeneous	Plan Sets Evolve According Similar Rules
	Heterogeneou s	Plan Sets Evolve According Different Rules
Scale–ability	2	Two Plan Sets Co–evolve At The Same Time
	N	Multiple Plan Sets Co–evolve At The Same Time
Focus	Great	Superiority Of A Specific Interaction Is Evolved Emphasized
	Small	Inferiority Of A Specific Interaction Is Evolved Emphasized
Diversity	1	Single Final Plan Generated
	N	Alternative Final Plans Generated

Some co–evolution–related requirements are further needed to improve the search process:

- Problem decomposition: The planning problem is decomposed into plan steps and solved step–wise, which is represented by the increased fitness within the algorithm.
- Parallelism: The algorithm can simultaneous evaluate plans within distinct populations.
- Credit assignment: In general, by evaluating each rule's contribution to a success of the whole plan, an optimal plan can be generated. This issue should be considered in the design of the chromosome evaluation criteria.

One may consider these issues also in a specific CEVOP implementation for both above discussed aspects of the planning process, which were the adaptation process within the co–evolutionary algorithm and among the agents.

4.2 Mapping Between The Problem Domains

In order to solve the planning process of a multi–agent system within the co–evolutionary algorithm domain, we need to define a mapping of the necessary elements of the planning process. At a lower level of abstraction the plan primitives are mapped and at a higher level the planning concepts.

Mapping Plan Primitives

We need to map at least the primitives plan and plan step. From the set of all possible plan steps PP, each plan step p_i is mapped to a gene value. The validity range of p_i is mapped to gene value ranges, in which mutation is allowed. A plan, consisting of a sequence of several plan steps, is represented in form of a chromosome (*Figure 4.1*).

Possible variations of different p_i sequences in form of alternative plans are explored by crossing over the genes of two chromosomes. Mapping the primitives into the helper domain we call encoding, and the backward transformation we call decoding.

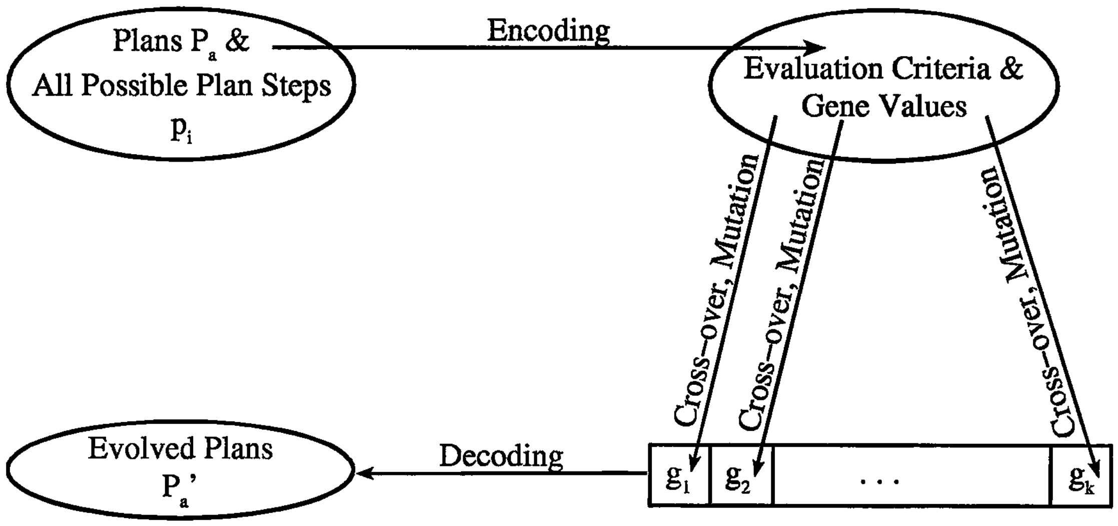


Figure 4.1: Mapping Plan Primitives Onto Chromosomes

Mapping Planning Concepts

At the conceptual level of the planning process, particularly the interaction types co–operation and competition, and the plan variety properties homogeneity and heterogeneity are mapped onto related structures of the co–evolutionary algorithm. Since the gene value ranges are fixed for a chromosome, all chromosomes of a population represent only those alternative plans, which are based on the related plan steps (*Table 4.2*). Hence, these populations can represent only homogeneous plans. In other words, plan steps that are not found in the knowledge base of an agent are not put into the set of all possible plan steps PP. On the other hand, this is the reason why we represent heterogeneous plans with different populations.

Table 4.2: Homogeneous Plans Within A Chromosome

P_1	P_2	. . .	P_n

Finding a solution for a chromosome is the result of exploring the search space opened by all possible plan steps PP. Since the search is guided by chromosome evaluation criteria, actually their design determines the required co–

operation/competition structures. Furthermore, it is simpler to design a criterion that evaluates all genes or a group of genes of a chromosome with equal criterion. This is the reason why we represent co–operation with the evolution of a chromosome and competition with co–evolution of chromosomes that belong to different populations (*Table 4.3*).

Table 4.3: Mapping Planning Concepts Onto Populations

Multi–agent Planning Domain		Interaction Type	
		Co–operation	Competition
Plan Variety	Homogeneous	Same Population	Different Populations
	Heterogeneous	Different Populations	Different Populations
		Co–evolutionary Algorithm Domain	

In the special case that all agents are homogeneous and only co–operation structures are sought, we can get the cheapest CEVOP implementation, in terms of storage and processing time.

4.3 The Generic Co–evolutionary Algorithm

In this section we will be discussing the inner loop of the CEVOP algorithm, which is the co–evolutionary part.

The Algorithm For N Populations

The algorithm is introduced for n populations and m_i interactions for each population A_i.

```
1) For all n populations
1.1) Randomly pickup plan steps from $P_i$ for population $A_i$
2) For all n populations
2.1) Cross-over inside $A_i$
3) For all n populations
3.1) Mutate distance values inside $A_i$
4) For all n populations
4.1) Calculate Fitness $F_i$ of Population $A_i$
4.2) For all $m_i$ interaction relationships of population $A_i$
4.2.1) Calculate fitness $F_{ij}$ for interaction of $A_i$ with $A_j$ $(i \neq j)$
4.2.2) $F_c = F_c + F_{ij}$
4.3) $F_i' = (F_i + F_c / m_i) / 2$ and $F_c = 0$
5) IF fitness $F_1$ AND ... AND $F_m$ satisfied OR other termination THEN GO TO 7
6) For all n populations
6.1) Fitness selection on $A_i$
6.2) GO TO 2
7) For all n populations
8) PRINT $A_i$
```

A_i is the plan population to be optimised. P_i is the set of all possible individual plan steps of agent i. Fitness F_i' of population A_i is determined by accumulating its own fitness F_i and all m fitness values for interrelationships F_{ij}. F_c represents the co–operation and/or competition strengths of agent i with the others.

Interpretation Of The Algorithm For The Case Of Homogeneous Co–operation

In case of k co–operating homogeneous agents, the set of all possible plans P_i is identical for all agents A_{i1}, ..., A_{ik}. Thus, the plans can co–operatively evolve inside a single population.

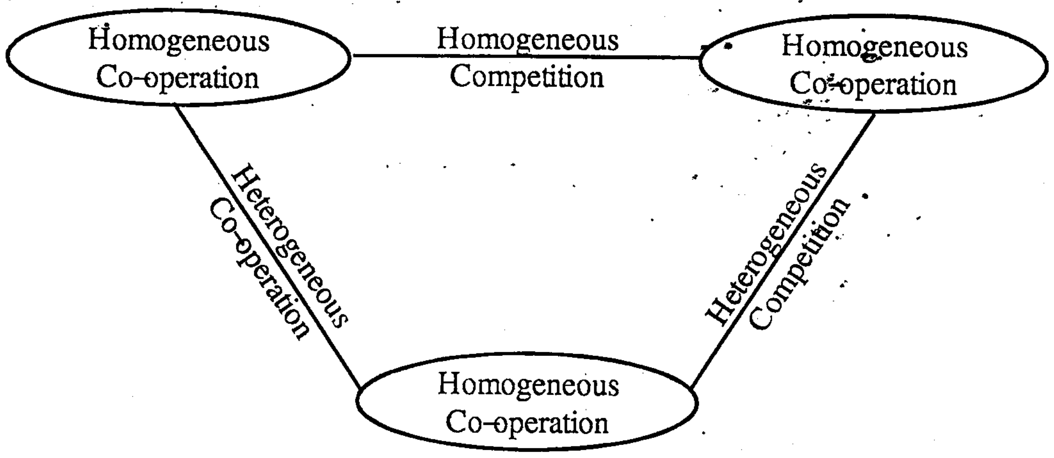


Figure 4.2: Sample Interaction Types For Three Homogeneous/ Heterogeneous Plan Populations

For this purpose the plans of all these players are represented inside a single chromosome. Another representation and sample mapping of these structures is depicted in (*Figure 4.2*).

Theoretically, it is possible to refine the interrelationships in (*Figure 4.2*) by directed ones. However, only one reasonable configuration arises in this case, which is co–operation in one direction and competition in the other direction between two populations. Even this case may not be reasonable for some domains.

Fitness Evaluation

According the algorithm, the fitness evaluation is a two–stage process, which we call local and global fitness evaluation:

- Local: First, the fitness F_i for each population is calculated and accumulated independently from the others.
- Global: Second, by adjusting each local fitness F_i by its co–operation/competition strengths with the other populations through $F_i' = (F_i + F_c / m_i) / 2$, the global fitness for each population is determined.

In the calculation of F_i and F_i' we have weighted all co–operation and/or competition strengths equal. In a specific domain, the relationships between the chromosome evaluation criteria may not be linear. In that case the accumulation function may be nonlinear.

Properties Of The Algorithm

The following summarises some important features of the algorithm:

- Plans with variable number of plan steps can be represented by variable chromosome lengths within a population. This property can make the algorithm more dynamic against related requirements of a specific domain.
- For the co–evolutionary algorithm we have provided the cross–over and mutation operations. However, to adapt the algorithm to a specific problem domain, one may emphasise these operations over their reproduction rates.
- The fitness and selection operations can be adopted to a specific domains, too. A general interpretation of the fitness function is: It evaluates the contribution of each plan step to a success of the whole plan.
- The computational effort of the co–evolutionary algorithm is $O(n^3)$ in the worst case and $O(n^4)$ for the case that multiple plan solutions are sought (*Appendix H*).
- Since each criterion evaluates a chromosome and seeking for an optimum value for all criteria requires an equilibrium, the algorithm solves an optimisation problem.

Convergence Behaviour

The generalisation capability of the algorithm depends on the design of the chromosome evaluation criteria, since they drive the chromosomes towards the desired fitness. Therefore, appropriate design of the criteria will enable the algorithm to converge for most cases. However, an inappropriately designed criterion may disable the algorithm to converge, such as contradicting multi–objective criteria. In general, if the criteria over–constraint the problem.

4.4 Interpretation Of The Generated Plans

Above we have defined that the source data for the co–evolutionary algorithm was the ste of all possible plans PP of an agent. Our justification for the claim that the methodology provides for learning while planning was based on the assumption that the set of known plans P is a subset of all possible plans PP: $P \subset PP$. Hence, principally any plan that was not in the set of known plans P_i is a new learned one. For the case of two agents, the intersections of both sets of possible plans $PP_1 \cap PP_2$, excluding both sets of plans $P_1 \cup P_2$, is the set of new learned plans for both agents $(PP_1 \cap PP_2) \setminus (P_1 \cup P_2)$. Each of the other two symmetric intersecting sets of plans $((PP_1 \cap P_2) \cup (PP_2 \cap P_1)) \setminus (PP_1 \cap PP_2) \setminus (P_1 \cup P_2)$ are new only for one of the agents (*Figure 4.3*).

Another interpretation of generated plans is related to the capability of the co–evolutionary algorithm to model co–operation/competition appropriately. Co–operative/competitive co–evolution evaluates candidate plans and and their variations against one another while searching for both, so that each drives the improvement of the other. Since this process is driven by the chromosome evaluation criteria, the appropriateness of the modelled co–operation/competition actually depends on the appropriateness of the criteria. Furthermore, a desired grade for co–operation/competition of plans may also be controlled by designing the criteria appropriately.

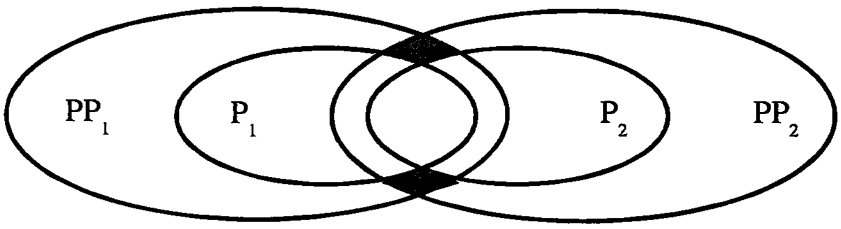


Figure 4.3: Interpretation Of Generated Plans For Two Agents

4.5 The Generic CEVOP Methodology

Above, we have discussed various properties of the CEVOP methodology. Now, we summarise the algorithmic steps, which are basically the co–evolutionary algorithm for plan generation and adaptation, and the negotiation schema for plan agreement.

```
1) Apply the co-evolutionary algorithm for all
   random/desired plans $A_i$ of agent i
2) IF all agents agree on the generated plans $A_i'$ THEN
   GO TO 4
3) Allow each agent to modify $A_i$ such that $A_i = A_i''$ GO
   TO 1
4) Each agent i applies the generated plan $A_i'$
```

A multi–agent system may utilise the co–evolutionary planning process in iterative calles of its planning process. After each such call the agents will need to enter a negotiation and adaptation process, in order to find a final agreement. First, the agents negotiate on the generated plans. If they cannot find an agreement, then each agent enters its own and independent adaptation process. In the adaptation, each agent has the decision alternatives to adapt either its initial plan A_i to the generated plan A_i', or to adapt A_i' to A_i, or to adapt both. The adapted plans are all collected and passed to a new call of the co–evolutionary algorithm.

The negotiation, adaptation, and agreement schema for the agents is a suggested scenario for embedding the co–evolutionary algorithm as a planning approach into a multi–agent systems.

4.6 Recapitulating The CEVOP Methodology

In the following the basic properties of the methodology are summarised:

- CEVOP addresses the problem of homogeneous and/or heterogeneous co–operative and/or competitive planning of multiple agents by mapping the structures onto a single co–evolutionary algorithm.
- The planning methodology is inherently domain–independent, since no assumptions on any specific domain have been made.
- The main advantage is that the multi–agent system can find a global plan in a more democratic way, since each agent's initial plan influences the co–evolution process, and the global plans are negotiated.
- The multi–population approach does not have a greater computational complexity, since it can be accumulated in a single population, resulting in the same complexity.
- Alternative plans could be produced by introducing the algorithm co–evolution within each population. Where, desired properties of sub–optimal solutions can be preserved until the end of the search by using the techniques discussed in the chapter on co–evolutionary computation.

CHAPTER FIVE
THE AGENTTEAM FRAMEWORK

AgentTeam is a generic framework for multi–agent systems (MAS). The idea is to collect and combine necessary generic concepts, which are sufficiently abstract to provide a domain–independent framework for MASs. Another intention is to initiate discussions on intelligent agent –oriented concepts. Thus, AgentTeam serves as a basic model for the design of specific MASs. Whereby, further domain–dependent concepts may be introduced in the design of a specific MAS, if needed. The concepts, initially defined earlier and prototype implemented in the distributed database management system domain [Kumova; 1998c], [Kumova; 2000a], [Kumova; 2001a], have been now generalised and abstracted further in this version. Especially, stronger concepts for co–operation and competition were introduced though the CEVOP methodology, whic was introduced in the previous chapter. A restricted prototype version of CEVOP for the Soccer domain is introduced in the next chapter. In this chapter, the co–operation, competition, and co–ordination models are emphasized as they are affected by the CEVOP approach.

5.1 Major Concepts

The AgentTeam combines various concepts of other existing multi–agent systems (MAS) in a framework The framework is discussed here by emphasizing two issues:

- The architectural model.
- The object–oriented design of the architectural model.

Before we start to discuss the concepts, we give a summarised brief overview of related work, since most of our concepts were influenced by those earlier solutions.

Definition Of An Intelligent Agent

The difficulty for finding a definition for the agent concept, on which the community could widely agree, is as difficult as finding a definition for intelligence. Nevertheless, many attempts are undertaken in the literature to describe an agent with few words.

- A programme with some architectural and behavioural properties that may appear a human as a smart application.
- A rule–based application with an inference mechanism that evaluates the rules. Where the rules are in first order predicate logic.
- Given its goals, limited resources, and dynamic real–time environment, an intelligent agent must decide which of many possible actions to execute at each point in time [Hayes–Roth; 1992].
- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. A human agent has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors. A robot agent substitutes cameras and infrared range fingers for the sensors and various motors for the effectors. A software agent has encoded bit strings as its percepts and actions [Russell; 1995].
- According a weak notion for agency, an agent enjoys the properties autonomy, social ability, reactivity, pro–activeness [Wooldridge et al.; 1995].

Various other definitions for agents are collected and classified according different aspects in [Franklin et al.; 1996], [Knapik et al.; 1998], [Çağlayan et al.; 1997].

Other Models For Multi–agent Systems

A standardisation efforts for MASs does not exist. But various MASs development environments and tools are available that propose standardisation of different aspects of MASs. We summarise here only a view popular MASs by mentioning only their most outstanding features briefly.

Microsoft Word's letter wizard and help assistants have rule–based knowledge of a restricted domain. They can be classified as intelligent modules, since they can exist only inside the environment of this application.

RETSINA [Sycara et al.; 1997] is a MAS framework for information retrieval. A hierarchical communication structure is proposed that involves several agent types.

InfoSleuth [Nodine et al.; 1998] is more a software–engineering tool for developing intelligent agents. Developed agents can be deployed into a system environment with uniform communication capabilities, i.e. layered communication semantics are defined for InfoSleuth agents, in order to facilitate communication standards. It is, like RETSINA, a framework for information retrieval.

Infomaster [Duschka et al.; 1997], [Genesereth et al.; 1997] is an information integration tool for information retrieval purposes. Its rule–based schema integration facilitator is an approach to homogenise heterogeneous data sources as well as structured and unstructured data.

ZEUS is a graphical development tool–kit for building distributed multi–agent systems [Nwana et al.; 1999] that provides both a FIPA [FIPA; 1997] or KQML [Wooldridge; 1998] based communications infrastructure and a planning engine for handling rule based conversations by means of automata models.

An evaluation and comparison of the most cited MASs can be found in [Eiter et al.; 2001]. It is also expected that these efforts will converge over time [Iglesias et al.; 1998] first to standards on particular concepts, such as the ACL, thereafter to behavioural and architectural standards.

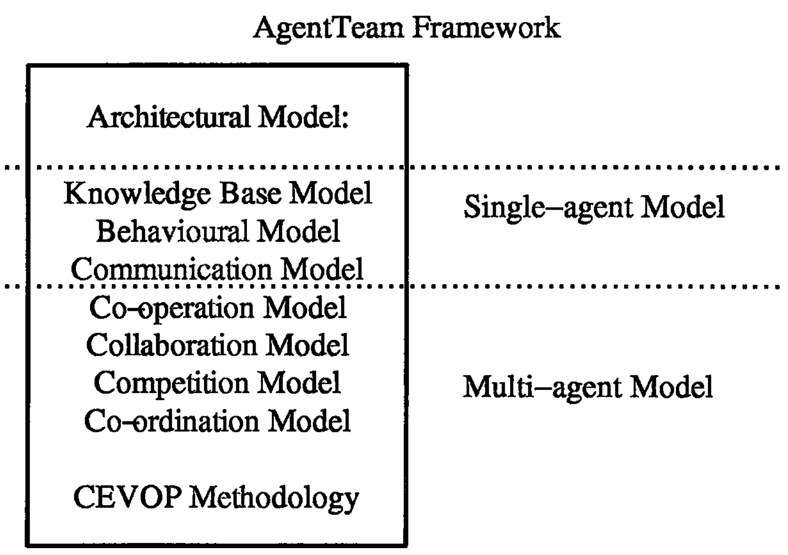


Figure 5.1: Concepts Of The AgentTeam Framework

Architectural Model Of The Framework

The architectural model of AgentTeam includes the elementary modals depicted in (*Figure 5.1*). Each model defines related concepts that are thought to be domain–independent, which will be shown in the discussions below. Although, the CEVOP methodology is integral part of the framework, we will not repeat it here, since it was extensively discussed in the previous chapter. Instead, the co–operation, competition, and co–ordination models are introduced in terms of the CEVOP methodology. In all models n–to–m relationships were assumed for the cardinality between the mentioned objects. The models are further grouped by the single–agent model and MAS model.

5.2 The Architectural Model

Before we proceed the major models of the architecture, some further generic properties are defined that are common to most agents.

Intelligent Components

Modern systems usually consist of several interacting software components, such as sources, resources, modules, and applications. On the other hand, from the point of view of the environment, all of them are again components. Even the environment itself may be a system made up of components, such as operating system, hardware, and so forth, until the lowest system level, consisting of drivers and devices. Therefore, a component can principally be software, hardware, or a system combining both of them. Compared with non–intelligent components, an intelligent one possesses some further architectural and behavioural features, which are described below.

According their environment, intelligent components can be grouped in three classes (*Figure 5.2*):

- *Intelligent module*: Is an intelligent component of an intelligent or non–intelligent application. It is imbedded in the application, where it exists within a restricted activity area. Since, its knowledge base structures are relative primitive, its reasoning capabilities are restricted. For instance, the smart help–modules of an application that assist the user in interactive dialogs.
- *Intelligent Agent*: Is an intelligent application. Is autonomous, since it can act independently from other applications. Its knowledge base contains advanced structures and it can reason in the context of global consistency concepts. For instance an intelligent broker agent that maintains keyword–service–server relationships to support related information queries.
- *Intelligent MASs*: Is a distributed system consisting of multiple co–operative intelligent agents acting successfully within a specific domain. The agents are only partially autonomous, since they act towards a common global goal. However, a MAS as a hole is autonomous, like a single intelligent agent. For instance, agents representing consumers and producers in an e–market place, and those agents, which broker between them.

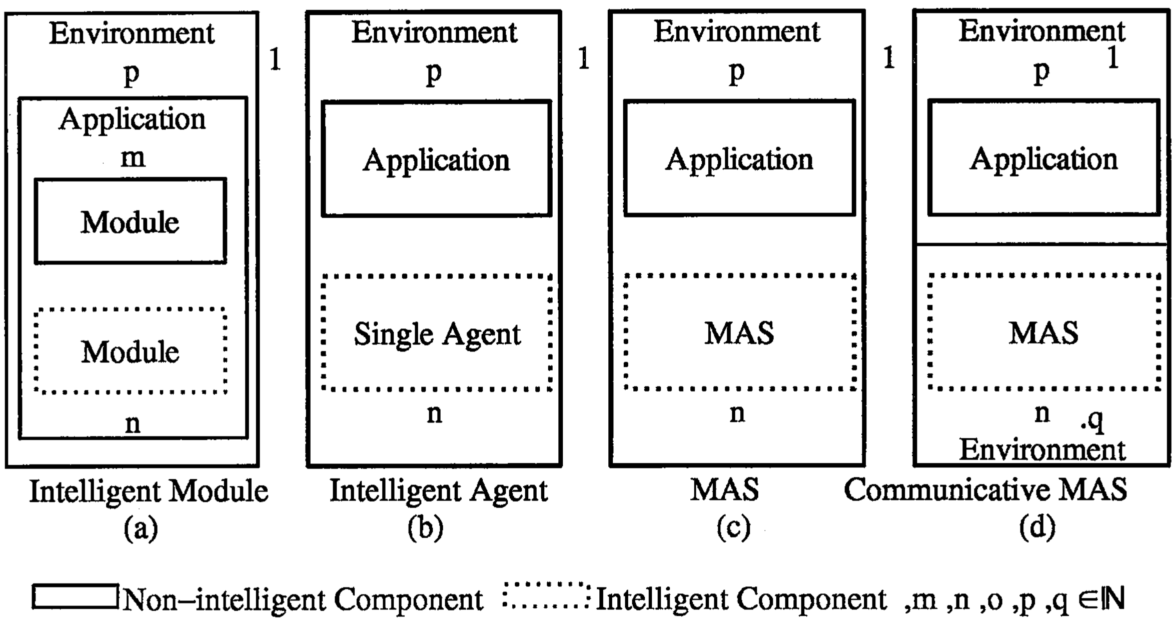


Figure 5.2: Possible Configurations For Intelligent Components

- *Communicative MASs*: Are domain–independent MAS that can act autonomously in any application domain and communicate among each other successfully. For instance, e–market places with interrelationships among each other and direct connections to related production systems.

World Model

By our definition, an intelligent agent is more advanced than an intelligent module, since it can have more capabilities enabled by its higher degree of autonomy. The agent possesses a generic architecture that is valid for all types of intelligent components. It interacts with users, resources, and sources of the environment (*Figure 5.3*). A resource is a data access interface that acts as a service provider for a data storage. Where, a resource may be again intelligent component. A source is pure data without any functionality. Usually, different languages are used to communicate with each object of the environment. For instance, user interfaces, agent communication languages, various communication protocols for resources and sources. Advanced internal structures of an agent are goals, control mechanisms for learning concepts, planning optimal solution strategies, and co–operation and co–ordination structures for the execution of tasks.

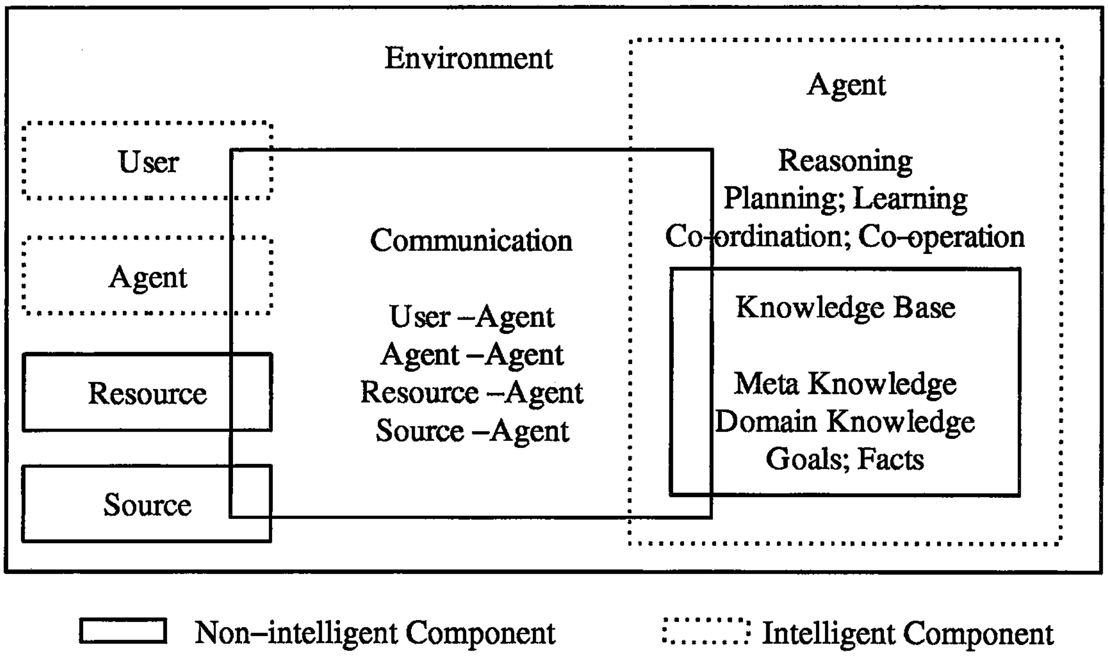


Figure 5.3: Generic Architecture And Environment Of An Intelligent Agent

Reasoning is a higher–level control mechanism, which is used to evaluate all the other concepts. Since it is usually based on formal logic, it enables the component to behave intelligent. The knowledge base contains all these concepts in form of domain–independent functionality that represents meta–knowledge, and domain–dependent functionality that represents domain knowledge.

Object Orientation

We propose object–oriented designe patterns for agents and MASs. This approach has influenced, besides the software engineering aspects of the models, also partially the other concepts, for example the AgentCom language. Since software engineering concepts are usually secondary in the discussion of artificial intelligence concepts, we refer here to the related publications [Kumova; 2000b], [Kumova; 2000f], [Kumova; 2000h].

Life–cycle Model

Distinguishing different states of an agent can facilitate its design, provided that it is used as a standard design concept.

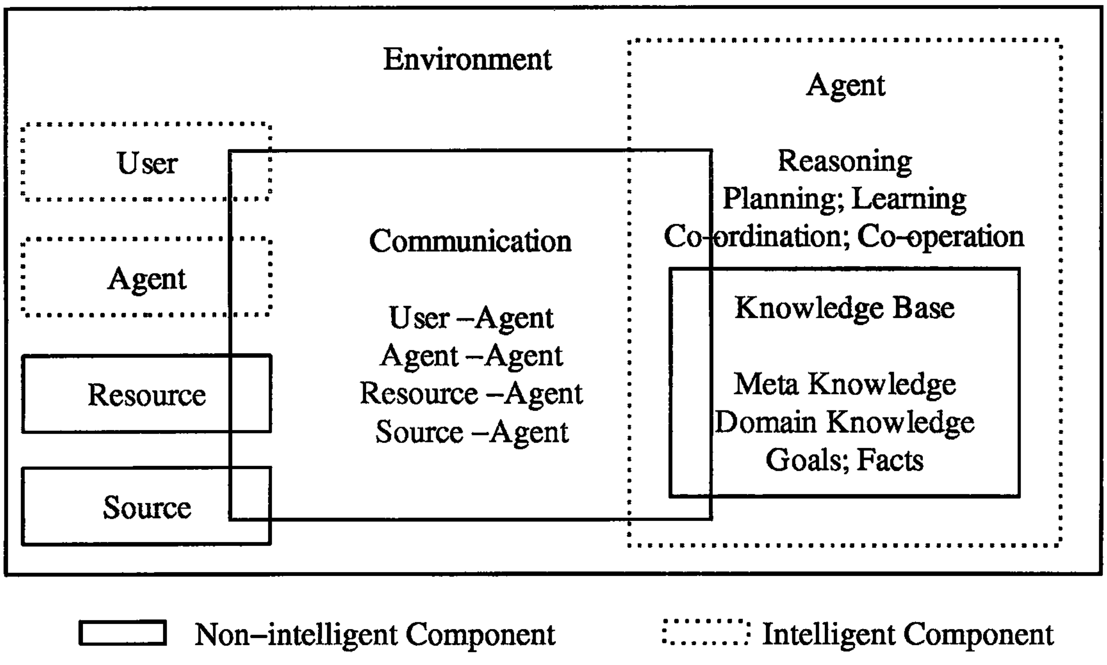


Figure 5.3: Generic Architecture And Environment Of An Intelligent Agent

Reasoning is a higher–level control mechanism, which is used to evaluate all the other concepts. Since it is usually based on formal logic, it enables the component to behave intelligent. The knowledge base contains all these concepts in form of domain–independent functionality that represents meta–knowledge, and domain–dependent functionality that represents domain knowledge.

Object Orientation

We propose object–oriented designe patterns for agents and MASs. This approach has influenced, besides the software engineering aspects of the models, also partially the other concepts, for example the AgentCom language. Since software engineering concepts are usually secondary in the discussion of artificial intelligence concepts, we refer here to the related publications [Kumova; 2000b], [Kumova; 2000f], [Kumova; 2000h].

Life–cycle Model

Distinguishing different states of an agent can facilitate its design, provided that it is used as a standard design concept.

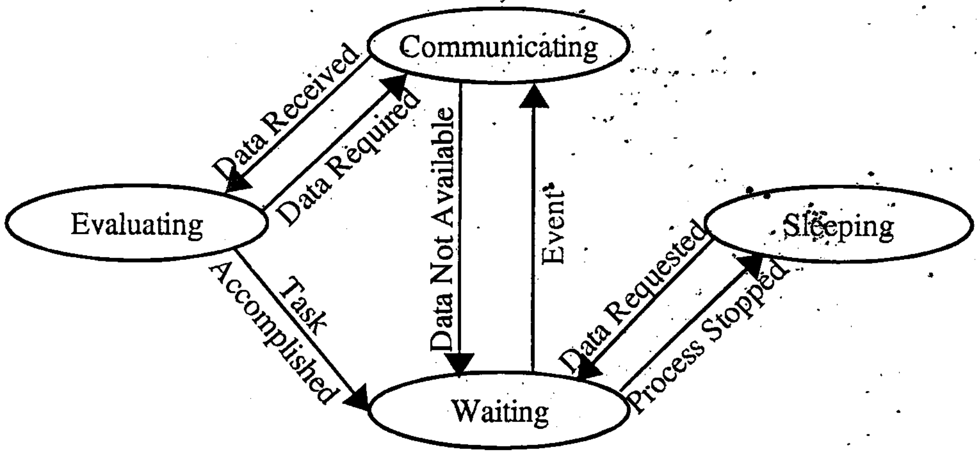


Figure 5.4: Life–cycle Model Of An AgentTeam Agent

An agent may enter one of the four generic states sleep, wait, communicate, or evaluate (*Figure 5.4*). The states and the possible state transitions are explained briefly below.

Sleeping: An agent is sleeping, when it is currently persistent or is moving. For example when it is currently inactive, because its state and data is stored in a database or is moving from one machine to another machine. An agent may transit from sleeping only to the waiting state, for example caused by a data request.

Waiting: An agent is waiting, if currently no event has activated any of its functionality. From waiting it may transit to sleeping, if no events have occurred for a while. It may transit to communicating, if a communication event is received, such as a message from another agent or an interaction from a user.

Communicating: An agent is communicating in all types of interaction with other agents, which can be co–operate, collaborate, or compete. It may change to waiting, when for example network congestion occurs. It may change to evaluating, when information was received that needs to be evaluated.

Evaluating: An agent is evaluating, when calculating, transforming a protocol, learning, reasoning, filtering, or retrieving data. It may transit to waiting, when it is awaiting further information from other agents, in order to complete its tasks, or if all tasks have been completed.

5.3 The Knowledge Base Model

Usually, knowledge interchange format (KIF) is used as common data structure for representing logical knowledge inside ontologies. Since, knowledge in KIF is in first order predicate logic it is used by the inference mechanism, without further transformations. This knowledge is the bases for logical behaviour and enables the agent to communicate logically with users and other agents. On the other hand, communication with non–logical resources and sources is usually non–logical and therefore requires other knowledge base structures than those for logical knowledge [Kumova; 2000d].

Semantic net is an alternative representation form for knowledge. Since, there are no restrictions on the data structures to be stored in semantic nets, the knowledge does not necessarily need to be available in first order predicate logic. A semantic net is a suitable structure for storing heterogeneous data. It is a connected, directed, and labelled graph, where a node represents an object or an attribute value and an arc between two nodes represents a relation, a property, or an operation. A semantic net can always be drawn as a connected graph.

Independent from an application domain, the knowledge base of an agent should contain several generic concepts (*Figure 5.5*):

- *Domain Knowledge*: Is the knowledge of a specific application domain. For instance, Soccer rules.
- *Meta Knowledge*: Is domain–independent knowledge for controlling the evaluation of domain–dependent knowledge. For instance, the CEVOP methodology contains domain–independent planning knowledge.
- *Goals*: Are desired solution alternatives without solution ways, i.e. without plans.

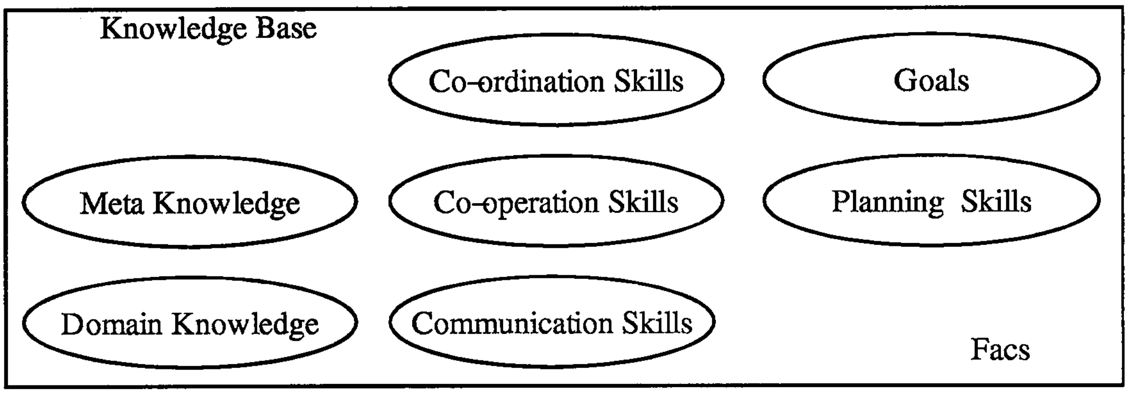


Figure 5.5: Knowledge Base Concepts Of An Intelligent Component

- *Plans*: Are solution ways for a given goals and/or several sub–goals.

The communication, co–operation, and co–ordination skills are discussed in the sections below.

Syntax and semantic of knowledge base structures need to be shared, too, in order to enable a successful communication. Syntactical structures are necessary to exchange knowledge. Whereas, semantical structures enable the meaningful evaluation of the exchanged syntactical structures.

Further, we propose in our framework to design a knowledge base in an object–oriented fashion. A knowledge base consisting of a semantic net, where the nodes are objects, can be a very flexible structure [Kumova; 2000a], [Kumova; 2001a].

5.4 The Behavioural Model

Our contribution to the behavioural model focuses on the planning capability of an agent. All other behavioural properties of an intelligent agent are summarised briefly.

Behavioural Properties

The discussion on the behaviour of agents in terms of human intellectual properties enables to compare the intellectual capabilities of the agents among each other and with those of human. Various such properties and aspects are discussed in the literature, such as capable, perceptive, successful, reactive, reflexive, predictive, imperative, rational, adaptive [Maes; 1995], delegation, autonomy, communication, perception, actuation, or intelligence as a whole. For formal definitions and informal descriptions of these properties we refer to the related publications and will not discuss them further [Goodwin; 1994], [Çaðlayan et al.; 1997].

The Planning Behaviour

Our basic planning approach is the Co–evolutionary Planning (CEVOP) methodology. According to our previous discussion we assume that theoretically all rules of a knowledge base R are a super–set of all possible plans PP and that all possible plans are a super–set of all known plans P: $R \supseteq PP \supseteq P$. For real implementations we assume that agents will converge over time to $PP \supset P$, due to the frame problem, and the difference is the potential area for alternative plans. If $\Sigma PP_i \cap \Sigma PP_j \neq \varnothing$, then the CEVOP algorithm is able to find compatible plans or compatible partial plans that satisfy common goals or sub–goals of the agents, respectively. Where PP_i and PP_j are all possible plans of agents i and j.

On the other hand, If $\Sigma PP_i \cap \Sigma PP_j = \varnothing$, then the CEVOP algorithm will find that the plans are incompatible or contradicting, which is a potential for competition between the related agents.

Since the search space is first explored with the co–evolutionary algorithm, thereafter the plans negotiated between the agents, compatible and/or contradicting plans are sought in a co–evolutionary planning process.

5.5 The Communication Model

Communication skills are communication inter–dependencies between users, agents, resources, and sources. For example, the communication structures inside a MAS or between MASs. They represent all possible passing of inter–agent messages of a MAS. Usually, ACLs are used as communication protocol, Such as Knowledge Query and Manipulation Language (KQML) [Finin et al.; 1993], [Finin et al.; 1994], [Mayfield et al.; 1995], ARtimis COmmunication Language (ARCOL) [Sadek et al.; 1997], AgentCom [Kumova; 2000d]. These languages are modeled in analogy to speech acts found in natural languages [Smith et al.; 1996]. Speech acts have been identified as basic logical concepts in natural language communication for requesting and asserting intentions.

Design Factors

An important complexity that influences the design of intelligent behaviour emerges from synergy effects in the design among the planned behaviour, application environment, agent communication language (ACL), and the knowledge base (KB). For instance, the design of the knowledge base structures is partially determined by the chosen ACL, partially by the planned behaviour of the component, and partially by the knowledge structures of the environment to be communicated over the ACL (*Figure 5.6*).

From the point of view of software engineering, to improve the accuracy and maintenance of these concepts also formalisms are used. For instance, for the specification, description, and implementation of behaviour different languages are proposed [d'Inverno et al.; 1997].

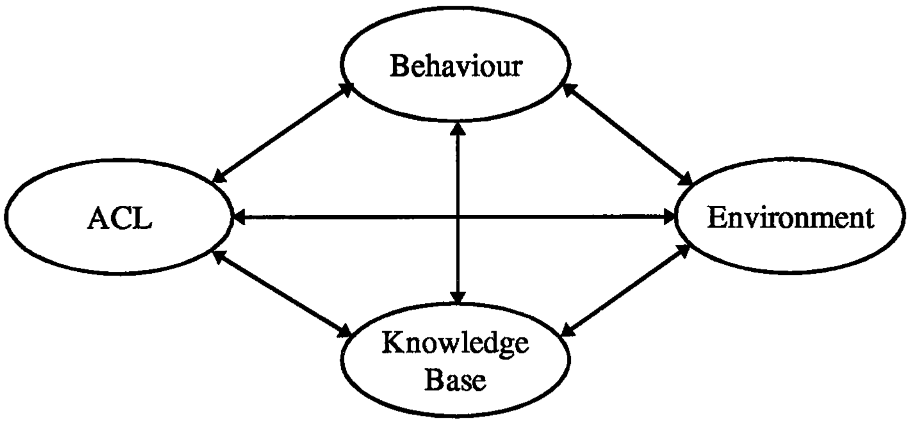


Figure 5.6: Factors That Influence Agent Communication

Client–broker–server Model

From the perspective of the communication structures, the generic architecture of a MAS consists of multiple clients and multiple servers, and multiple brokers between the clients and servers. A client can reach a potential server that could give the desired information in two steps (*Figure 5.7*):

```
1 A client passes the desired information to known
  brokers and gets the addresses of some potential
  servers.
2 The client contacts the related servers directly.
```

Related to the client–server model are some practical network–oriented implementation alternatives, such as synchronous versus asynchronous and stateless versus state–full communication. Usually, these techniques enable parallel processing at server as well as at client side.

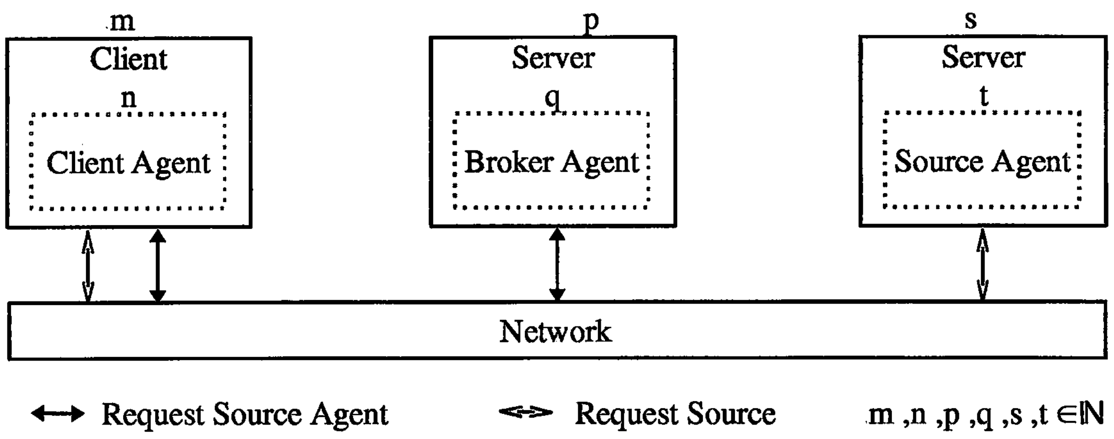


Figure 5.7: Communication Relationships Of A Client–broker–server–based MAS

Agent Types

We propose four agent types with a pre–defined communication structure as follows (*Figure 5.8*):

- *User Agent*: A user agent is mainly responsible for managing user transactions, which may be given interactive on a user interface or stored as time–conditional events. For each transaction one task agent will be invoked. Usually, one transaction may cause several tasks.
- *Task Agent*: A task agent performs a given task on its own behalf autonomously, but may need to co–operate dependent tasks with other task agents. Where, the user agent may take over a co–ordinator role, if necessary for a specific transaction.
- *Resource Agent*: A resource agent is mainly responsible for brokering messages between task agents and source agents. Only one should exists on a server, where it provides the server site connectivity. Resource agents should be able to co–operate with each other, in order to provide the task agents a unified service. Several source agent may be necessary on one site, depending on, for example, the diversity of the sources.
- *Source Agent*: A source agent knows how to access the local data sources, which can be heterogeneous. It knows various transformation alternative, in order to return the data in the required form to the task agents.

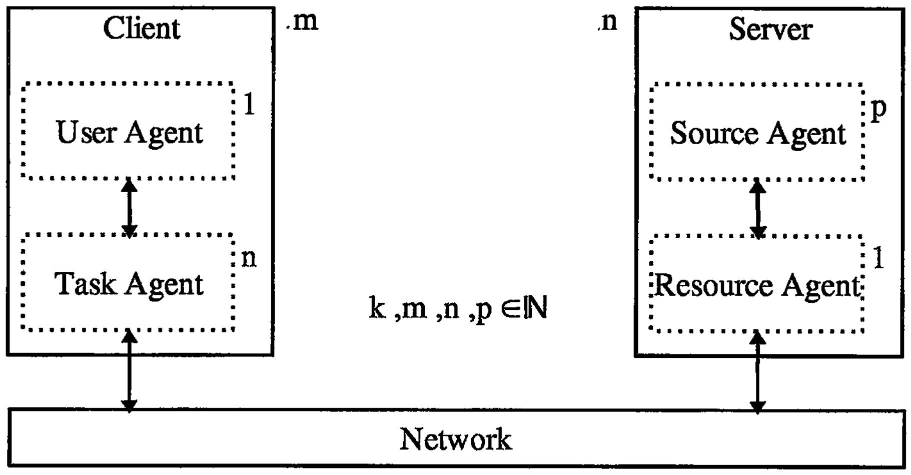


Figure 5.8: Communication Relationships Between Agent Types In Information Systems

However, in a specific domain not all agent types may be necessary.

Language Semantics

Intelligent behaviour can be simulated in many ways. One effective technique is, deciding non–deterministic functionality by using heuristics and employing mathematical logic for reasoning and explaining decisions. In order to communicate such logical concepts, a language should provide suitable syntax and semantics. For this purpose agent communication languages were developed. However, the difficulty to define domain–independent language semantics still persists [Sing et al.; 1998], [Hobbes; 1999]. One crucial handicap is the lack of concepts for abstracting from heterogeneous semantics of different domains. Though, some effort was made in the past to define language semantics for KQML, in form of conversation rules that pre–define performative sequences, [Smith et al.; 1996], [Labrou et al.; 1997], [Chaib–draa et al.; 1998], [Wooldridge; 1998] and the modal logic–based semantics with the Foundation for Intelligent Physical Agents (FIPA) standardisation effort [FIPA; 1997], no successful standards exist, yet [Sandholm; 1999].

Other approaches define language semantics over social agency [Sing et al.; 1998], [Ossowski; 1999]. Social agency is a suitable combination of differently emphasised agent properties, intended to achieve more communicative, more co–operative, and easier co–ordinated agents.

The Concept Of A Bilingual Agent

Behaviour is an abstract concept consisting of a combination of the above mentioned behavioural properties. It is the result of mutual influences of cognitive processes, such as planning and learning. Intelligent agents can influence each other's behaviour by communicating, co–operating, collaborating, and competing. In order to understand the concepts of another agent, standard semantics are necessary. Some common generic standards are first order predicate logic, modal logic, and object–orientation. The more a concept is domain–specific, the less it is generic with respect to other domains and the less it may comply with logical concept. On the other hand, higher abstractions can be found with logical concepts than with non–logical concepts. Furthermore, it is possible to reason over logical concepts, whereas reasoning over non–logical concepts is difficult and can result in logical inconsistencies. Finally, in real implementations knowledge at various abstraction levels is necessary for successful agents. It is obvious that the communication of a concept at different abstraction levels may have different semantics, due to different interpretations. Multi–agent systems that implement the above discussed ACLs approach this problem by hiding heterogeneous data or introducing domain–specific language semantics for their interpretation [Sycara et al.; 1997], [Nodine et al.; 1998], [Genesereth et al.; 1997], [Martin et al.; 1999]. This approach results in various domain–specific language dialects [Chaib–draa et al.; 1998], [Dam; 1997], which is a handicap for intra–MASs communication.

We address the semantics problem of an ACL, with respect to heterogeneous data exchange, with the bilingual agent concept [Kumova; 2001b]. Where, logical concepts are communicated though a logical language, such as the above discusses once, and non–logical domain–dependent concepts trough a non–logical language (*Figure 5.9*).

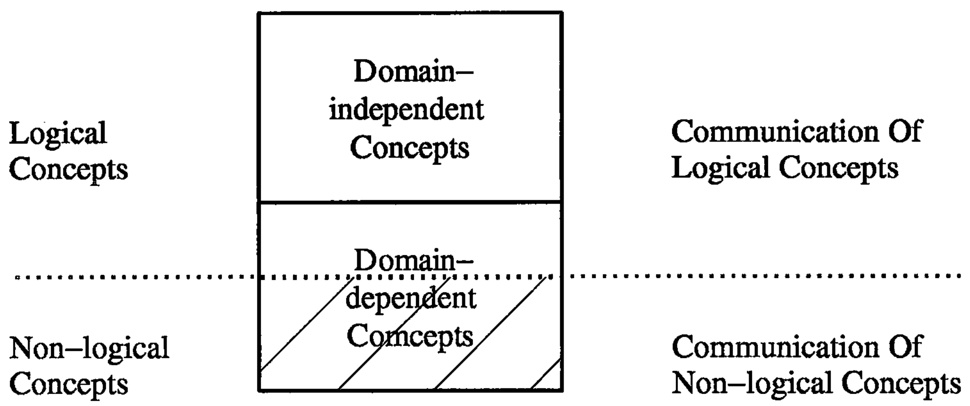


Figure 5.9: Concept Classes And Their Communication

With logical we mean predicate logic or its extensions, such as modal logic, deontic logic, temporal logic, or more–valid logic. Accordingly, with non–logical we mean data–oriented, functional, or object–oriented, but not logical (*Appendix A*).

This model clearly distinguishes between indirect access to the communication partner's knowledge base over logical ACLs and direct access over non–logical ACLs. Furthermore, it can help to avoid the parallel use of multiple direct access languages if it is standardised, such as XML SOAP [SOAP; 1999].

Thus, a bilingual agent can communicate with other agents in three modes: (*Figure 5.10*)

- Only in a logical language with another intelligent agent, to exchange only logical concepts.
- Only in a non–logical agent, to exchange only heterogeneous objects.
- In both languages in parallel, to exchange both types of information simultaneously.

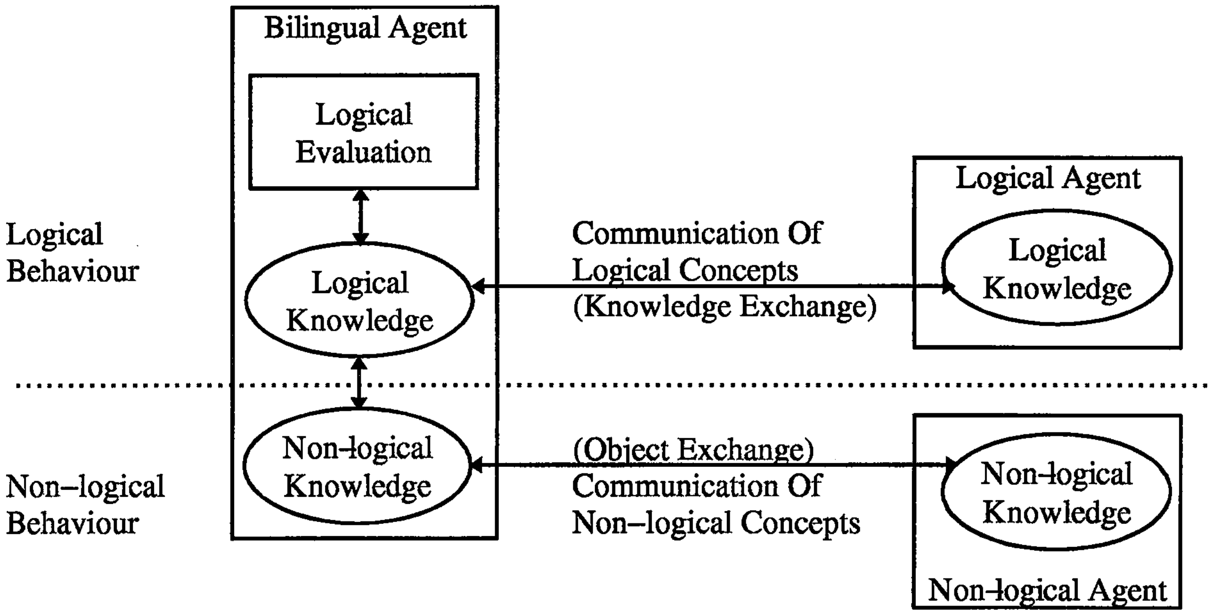


Figure 5.10: The Bilingual Communication Concept

We propose further that objects should be exchanged in a non–logical communication, since object–orientation can be used to hide the heterogeneity of the carried object. Most agents already implement various protocols to access different resources and sources. Exchanging them among the agents over a standardised non–logical ACL can simplify the communication structures. Another advantage is that domain–independent logical concepts can now be freed from domain–dependent non–logical concepts.

For this purpose we had designed AgentCom, as no comparable language existed at that time, which satisfied our requirements [Kumova; 1998c]. Its capability to carry objects is meanwhile realised, for instance with XML SOAP. However, its list–oriented syntax is still more advantages for agents that use list structures to store knowledge than the tag syntax of XML.

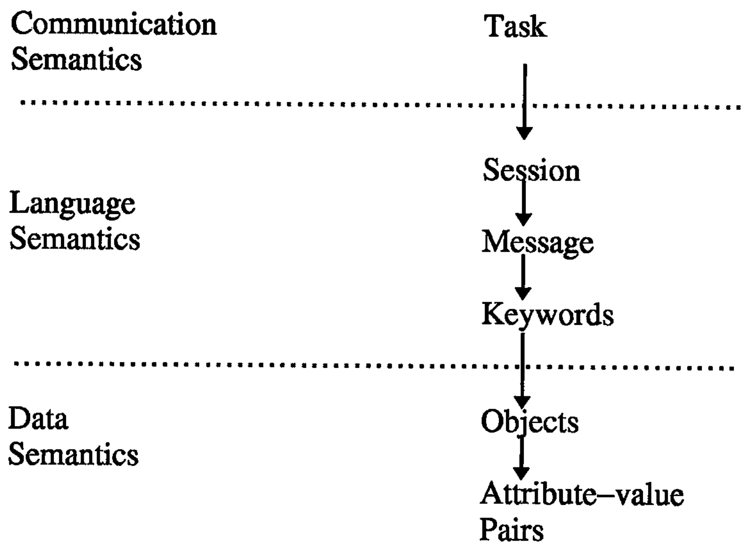


Figure 5.11: Hierarchical Communication Semantics Of AgentCom Agents

The Agent Communication Language AgentCom

AgentCom was designed to transfer knowledge in form of objects between the knowledge bases of agents and to initiate their evaluation [Kumova; 2000d]. The language consists of the message types SESSION_BEGIN, GIVE, TAKE, CREATE, DELETE, RESPOND, SESSION_END, and nine keywords for specifying various parameters of an object. Since, an object is represented in the language in form of text, any programming language that supports string manipulation can implement AgentCom. According the hierarchical syntax structure of the language, the semantics are embedded into communication levels (*Figure 5.11*).

The sample code below requests the receiver *DBAgent1* to return the specified attribute values of the instance *enrol* of the class *Table* to the sender.

```
(GIVE (ADDRESS 193.140.x.x 193.140.x.y)
    (SESSION_ID 13) (MESSAGE_ID 65) (NAME TaskAgent1
    DBAgent1)
    (OBJECT Table enrol (HAS
         (OBJECT == courseId CSE509)
         (OBJECT > studId 99001)
         (OBJECT Attribute assignments NIL)
         (OBJECT Attribute midterms NIL)
```

```
                    (OBJECT Attribute final NIL))))
```

This example illustrates the direct access of the sender to the receiver's knowledge base, since the receiver's data structures are known by the sender. The OBJECT attribute enables to represent an object in form of a list. The complete syntax of AgentCom is specified in (*Appendix B*). A detailed discussion of the language syntax and semantics is provided in [Kumova; 2000d]. An evaluation of AgentCom against the other ACLs and some close Internet protocols can be found in [Kumova; 2001b].

Communicative Interaction Types

We interprete the relationships between communication, co–operation, collaboration, and competition as it is sketched in the figure (*Figure 5.12*).

The relationships state that communication is necessary for the other structures. The other structures are discussed in the following sections, in terms of alternative modi an agent may enter.

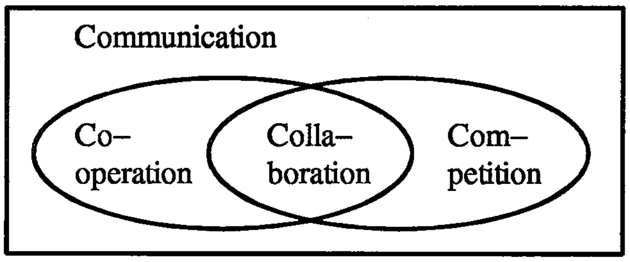


Figure 5.12: Relationships Between Co–operation, Collaboration, And Competition

5.6 The Co–operation Model

In co–operation mode, an agent knows only team goals It has no antagonistic goals. It may have individual goals, but these will be sub–goals required to reach the team goals independently. The communication partners can share their knowledge, since their goals are not contradicting.

Co–operation skills are semantics defined over communication skills. They define how some communication structures need to be applied to accomplish a specific task that depends on other agents. Various strategies are discussed in the literature [Kandzia et al.; 1997], [Kumova et al.; 1999], in order to enable homogeneous and/or heterogeneous agents to co–operate with each other. For example, co–operation between the RETSINA information gathering agents is supported by query delegation to other agents [Paolucci et al. 2000]. The need for co–operation arises for an agent, when at least one of its sub–goals can be satisfied alternatively or only by another agent.

Planning already as a centralised algorithm is known to be NP–complete. Planning as a distributed algorithm in a MAS introduces further complexity caused by additional iterations for negotiation. Even knowledge base heterogeneity can increase the complexity, if the homogenisation process requires further iterations. We address all these problems with the co–evolutionary approach of the CEVOP methodology, where the complexity is reduced, through the constraints of some evaluation criteria, to the polynomial time $O(n^8)$ (Appendix H).

The provided algorithmic solution for abstracting over heterogeneous knowledge bases is generic, since there are no limitations for the populations of the co–evolutionary algorithm. In general, the plans of homogeneous agents evolve in the same search spaces, which is represented by one population, whereas the plans of heterogeneous agents evolve in different search spaces, which is represented by different populations.

5.7 The Collaboration Model

In this discussion, we identify collaboration as a special cases and as a sub–set of co–operation and competition. This case is defined here over the co–operation process of different systems.

Rationale For Differentiating Collaboration From Co–operation

In most cases and in almost all domains, except the military, the term collaboration is used interchangeably with the term co–operation. Whereas, a strict distinction is made between both in military terminology. Here, collaboration is associated with supporting the enemy, in reaching its goals, against the goals of the own side. Independently of how similar or opposite the final goals of the parties are, any inter–party communication aims at satisfying a common sub–goal. Thus, collaboration is usually a timely restricted kind of co–operation, compared to the total time required for the final goal. On the other hand, a collaborating agent may compete against its initial side, for this duration.

Since, even collaboration needs to be controlled by the related parties, we need a clear differentiation between intra–MAS and inter–MAS communication. For this purpose, collaboration between MASs is thought as a kind of co–operation with resource restrictions, such as time, knowledge, and participants.

A Definition For Collaboration

An AgentTeam MAS collaborates with an autonomous agent or another MAS, if all conditions below are satisfied:

- A sub–goal of a MAS does not conflict with a goal of an autonomous agent or another MAS.
- Some agents of the MAS co–operate directly with an autonomous agent or another MAS by sharing their knowledge, until the agreed and non–conflicting sub–goal is reached.

In order to provide for a harmonic and conflict–free co–operation inside a MAS, no collaboration is allowed within a MAS. This can be guaranteed by building the plans of the MAS by involving all agents into all planning processes.

5.8 The Competition Model

Provided that individuals' objectives are initially not towards aggressivity, damage, disruption, sabotage, or any other non–ideal, then competition is no goal in that sense. However, contradicting goals can rise competition on shared resources, if the resources are insufficient for all agents at a given time. In that case, shared resources will be accessed concurrently. Generally, agent groups or single agents, with conflicting goals will compete against each other, in order for each one to reach its antagonistic goals.

Planning in a competitive environment makes sense only if it aims at reducing competition in favour to co–operation. The CEVOP methodology provides a relative simple algorithmic solution to achieve this by seeking always for co–operative plans or sub–plans, if the evaluation criteria are defined appropriately. Where, the remaining part represents potential for competition. On the other hand, it is also possible to explicitly define competition, in order to explicitly learn plans with improved competitive structures. This is again controlled by appropriate definition of the evaluation criteria.

Principally, the plans of competing agents evolve in different search spaces, which is represented by different populations. Competitive solutions will evolve by mutually sharing the fitness of the other populations.

5.9 The Co–ordination Model

A control structure is necessary for each specific task. It synchronises conflicting messages inside a task or between different tasks. Co–ordination is thought to be orthogonal to computation [Papadopoulos et al.; 1998]. Therefore, co–ordination languages [Ciancarini et al.; 1998] are used to implement related structures independent from the computational structures. A practical approach to distinguish between communication, co–operation, and co–ordination structures is to implement each one in form of a different protocol. On the other hand, all communication–oriented behavioural properties require these skills. For instance, negotiation is a concrete concept that requires a combination of some communication, co–operation, and co–ordination skills. Various co–ordination models and languages are proposed in the literature [Papadopoulos et al.; 1998].

Control structures for the time of the application of a plan are different from control structures for the planning process. The control structures of CEVOP are inherent in the methodology. Control structures for the implementation of a plan are not explicitly stated with CEVOP. After its generation though CEVOP a plan may be extended by existing control languages to control its application.

CHAPTER SIX THE SOCCERTEAM PROTOTYPE

The AgentTeam framework is domain–independent, since its concepts are designed domain–independent. Here, we introduce a prototype of AgentTeam, which is implemented in the Soccer game domain. For this purpose, the generic concepts of AgentTeam have been adopted for the SoccerTeam domain and additional Soccer–specific concepts introduced. The CEVOP methodology influences mainly the planning, co–operation, competition, co–and ordination structures of the prototype. Therefore, these concepts are emphasised in this description.*

6.1 Existing RoboCup Client Simulations

One international forum for testing Soccer playing agents is the RoboCup simulation league [Chen et al.; 2001], in which eleven team members co–operate whilst competing against another team of eleven members. The success of the teams is constantly increasing in each year's competition. The goal set by the RoboCup initiative is:

"By mid–21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA (Fédération

* In the coding it became clear that the implemented of all here introduced features of SoccerTeam and those of the officiel RoboCup Soccer simulation server, to be considered at client side, would require a relative strong programming effort, which forced us to exclude it from this dissertation. Hence, only the CEVOP methodology was implemented. Therefore, the remaining parts of this chapter are design issue for SoccerTeam and are currently implemented only partially.

Internationale de Football Association), against the winner of the most recent World Cup."

The current state of the art however does not allow any comparison with this goal, yet. The capabilities of current simulation teams are still far behind of those of any human junior Soccer team. From the analysis of recent RoboCup competitions [Asada et al.; 1999], [Tanaka et al.;2000] one can conclude that still much progress is required in all areas of technology used in multi–agent Soccer teams.

About 30 teams are registered to each annual RoboCup competition since the first in 1997. Each year some new teams appear, new ideas and design approaches are introduced. Since our approach combines evolutionary computation and planning, we will discuss here only related teams. First, each team's major design issue is given, then the team is discussed on this issue briefly. All of them are clients that run with the official RoboCup Soccer simulation server [Chen et al.; 2001].

magmaFreiburg2000 [Dorrer; 1999]: Concurrent behaviour networks [Maes; 1989]

- 2. place RoboCup'99
- To increase the reactivity of an agent; e.g. the concurrent evaluation and execution of say, kick, turn_neck commands
- Planning of team strategies is not supported
- Analytical approach

TsinghuAolus [Yunpeng et al.; 2001]: Global planning from local perspective

- 1. place RoboCup'01
- Distributed planning by evaluating and synthesizing individual agent behaviours
- Analytical approach

FC Portugal 2001 [Reis et al.; 2001]: Strategy levels; strategy tactics and formations; agent roles

- 3. place RoboCup'01
- Three strategy levels: strategic behaviour; ball possession behaviour; ball recovery behaviour
- Analytical approach

Brainstormers 2002 [Riedmiller et al.; 2002]: Neural network reinforcement learning of sample team strategies

- 2. place RoboCup'01
- Neural network and analytical approach

Harmony [Hashimoto et al.; 2002]: Co–evolutionary learning within a neural network [Pollack et al.; 1998]

- Registered for RoboCup'02
- Decision making with behaviour networks
- Evolution of team positioning from individual Soccer strategies in sample games

Sean Luke [Luke; 1998]: Learning from sample random competitions by using genetic programming

- No success at RoboCup'97, but RoboCup Scientific Challenge Award [Hedber g; 1997]
- Evolution of a team from genetic programming (Soccer) functions through competitive fitness

Darwin United [Andre et al.; 1999]: Genetic programming of agents that learn team strategies

- No success at RoboCup'98
- Evolution of a team from genetic programming (Soccer) functions through credit assignment

The latter two teams are the closest to our approach, but with the following significant differences:

- Since at the team strategy level no domain information is given to the algorithm, an exhaustive search is performed at this level of abstraction; we give team strategies to the algorithm and seek for strategic game constellations.
- Team strategies are evolved from low–level Soccer strategies and random games; we use low–level Soccer strategies, too, but simulate random games through the evolutionary search.
- The team is evolved by using genetic programming; whereas we use co–evolutionary algorithms.

In general, this approach is useful to show that evolutionary computation is capable to evolve from primitive strategies of a specific domain higher–order concepts through generalisation. However, the reason why they were not successful in RoboCup competitions is that good Soccer team strategies were already implemented by the teams with analytical approaches. On the other hand, in a CEVOP run of SoccerTeam several team strategies are already given and the algorithm evolves for sample constellations the best plan solution, which can be interpreted as seeking for optimal applicable team strategies.

The major property that decided winners in all previous RoboCup competitions was clearly the successful application of individual strategies, such as interception, dribble, pass, shoot, and goal keeper skill. Successful application of team strategies however is still a challenge for all teams.

To summarise the RoboCup competition from the perspective of human Soccer games, an average spectator may observer some general properties that are common to all successful teams and conclude for the simulation league with: Relative simple individual and team strategies, which are comparable with those of human primary school players. And for the robot leagues: Dummy actions like those of two year old humans, however more or less goal–oriented.

SoccerTeam is not fully implemented, yet, so we cannot compare it with existing teams. We believe that the CEVOP methodology can make an existing team implementation more effective at the team strategy level, since, with the specific co–evolutionary approach, it can be more successful than any analytical approach. Some of the above discussed teams utilise soft–computing approaches, too, however only to learn some selected specific game constellations, which introduces an analytical component into their approaches. With the CEVOP algorithm we explore and learn *all* possible game constellations, without introducing too much analytical components, which makes the solutions more generic.

6.2 Major Concepts

Before we start with the software design of SoccerTeam, it is important first to realise the major domain–dependent and domain–independent concepts that will influence the prototype the most. First of all, the sources for the sought concepts are the Soccer game domain and the software engineering domain. From the former we will synthesise concepts that will enable its successful projection onto the software system SoccerTeam. The latter will provide us with concepts from the AgentTeam framework and facilitate SoccerTeam's engineering.

AgentTeam Framework Concepts

The concepts that will be borrowed are all AgentTeam models except the knowledge base model, communication model, and collaboration model (*Figure 6.1*).

In the knowledge base model of AgentTeam semantic nets are used to store heterogeneous data and manage it object–oriented. However, because of the real–time constraints of the Soccer simulation server and the relative homogeneous knowledge structure of a Soccer player, a simple knowledge base model will suffice for SoccerTeam.

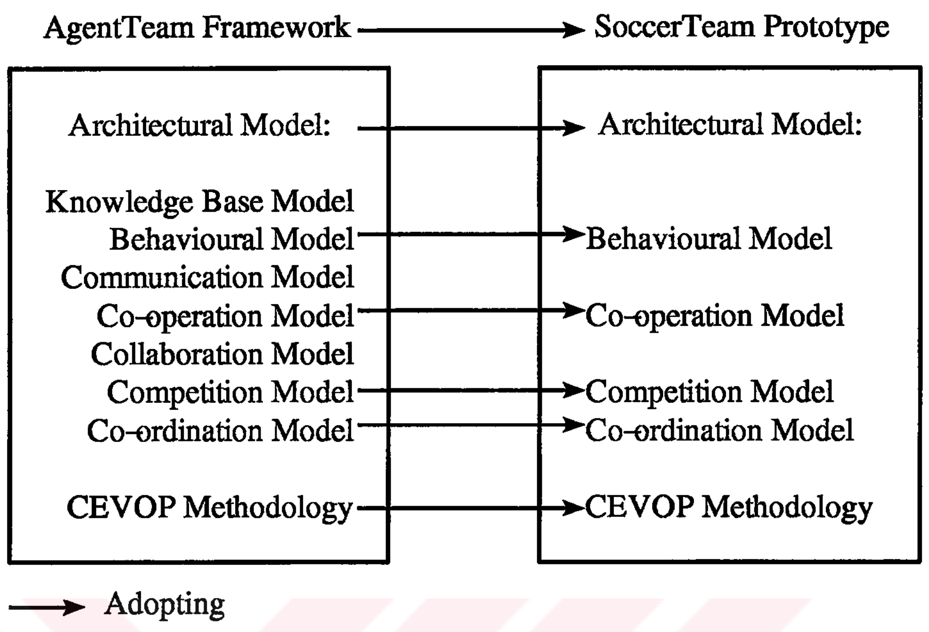


Figure 6.1: AgentTeam Concepts Adopted By SoccerTeam

The communication model of a bilingual agent proposed in AgentTeam also addresses heterogeneity. Since, communication with the Soccer simulation server and among a team has homogeneous structures, the simple protocol of the server satisfies the requirements.

In case of Soccer and in terms of collaboration as it is defined in the AgentTeam framework, a whole team or a member of it would collaborate with the opponent, if they would give the opponent a chance to win a match. Such as to shoot an own goal, to pass the ball to an opponent, never try to get the ball, etc. Since this is an uninteresting case, by definition, we do not allow SoccerTeam or its members to collaborate with the opponent.

Soccer Game Concepts

The Soccer regulations (*Appendix C*) define all necessary rules and the condition for performing a valid game. Accordingly, the basic concepts of Soccer are the following objects, which are only briefly described here:

- Game field: Is a rectangular play ground divided into two half, one for each team. Each half has a goal, a goal area, and in front of it a penalty area.
- Ball: Is an elastic leather filled with air.
- Players: A player can shoot the ball with the feet several times, in order to move it into the opponent's goal.
- Teams: Each team consists of 11 players, from which one is the goal keeper.
- Coaches: Each team may have a coach who communicates his team frequently some meta knowledge.
- Game rules: Are set to be applied by all participants while a match.
- Match: Two teams meet on a game field and play according the rules for a restricted time.
- Referees: Usually three referees monitor a match, in order to penalise rule violations.
- Game constellation: The current location of the ball and the 22 players on the field.

Architectural Abstractions

Based on the mentioned Soccer game abstractions and some additional information on the objects of the game and their movements (*Appendix D*), we construct following natural rule hierarchy (*Table 6.1*):

- Physics: At this level the game objects, ball and players, move according the physical rules, location, gravity, momentum, and acceleration. These rules are implemented in the RoboCup Soccer simulator through related functions. Corresponding functions are implemented in SoccerTeam and are considered for each movement. Correct implementation of the physical rules enables the players to play more precise.

Table 6.1: Soccer Game Abstraction Levels

Abstraction	Objects Move According	Purpose
Teams	Combined Game Strategies	Co–operative/Competitive Offensive/Defensive Game
Individuals	Primitive Game Strategies	Individual Offensive/Defensive Game
Game Code	Game Rules	Enabling Soccer Game
Physics	Physical Rules	Enabling Physical Objects World

- Game code: The officiel Soccer game regulations define the rules according to which a specific match is carried out. The RoboCup server implements most of the game rules. However, for some foul situations a human referee may decide for a free–kick, such as surrounding the ball, intentionally blocking the movement of other players, or flooding the server with messages, etc. [Chen et al.; 2001]. Corresponding rules are implemented in SoccerTeam and are considered for each movement. Always complying to the game rules enables the players to avoid fouls.
- Individuals: These are the autonomous players of the game that implement individual game strategies, which are called primitive game strategies in SoccerTeam. A player decides to apply a specific primitive game strategy depending on the current game constellation. Optimal application of primitive game strategies can provide a game advantage for the individual.
- Teams: The highest level for object movements are those of the teams. A team movement is represented by several primitive game strategies. For a specific game constellations, a combined game strategy is an individual plan that proposes several successive primitive game strategies. Optimal application of combined game strategies can provide a game advantage for the whole team.

We have combined these Soccer game abstractions in a hierarchy of object movements. This hierarchy will serve as a skeleton for the system architecture of SoccerTeam.

Soccer Game Strategies

Soccer game rules are necessary and sufficient to play correct Soccer and to win a match randomly. A soccer game consists of a combination of various type of re–occurring simple object movements, which must comply with the game rules. In order for a team to win a game, these allowed movements are combined to different types of strategies. When and how to apply a strategy depends on the current field constellation. All here discussed game strategies have been considered in the plan construction process of the CEVOP methodology for SoccerTeam.

In order to construct generic plans, the planning process within CEVOP does not consider lower level Soccer strategies, such as interception, dribble, and kick. The strategies will be considered in the plan implementation by each agent, depending on the current constellation. In this sens, an individual strategy represents an individual goal that can be reached by applying several low level strategies.

Learning While Planning

For SoccerTeam to be successful against competitors, it is essential that all players should know all rules of all abstraction levels, before making decisions about a specific game constellation. Meaning that a decision process for applying a combined game strategy will require to evaluate recursively all related lower level rules. However, recursively exploring all rules is too exhaustive. Instead, the CEVOP methodology explores the search space randomly, towards a a desired fitness function. The result after applying the CEVOP methodology is a pair of offensive/defensive team plans for an initially random game constellations. This pair represents a learned plan.

6.3 The Architectural Model

Our goal is to construct a team of Soccer players in form of a multi–agent system. For this purpose, all the other models that were mentioned above, including the

CEVOP methodology, are combined in the architectural model to construct the SoccerTeam prototype system.

Object Orientation

To fasten the development process of the software and to facilitate further maintenance of the system, we have designed the prototype in an object–oriented fashion. The basic idea is to abstract as much as possible Soccer game concepts and specify them inside a hierarchy of super–class. In SoccerTeam the super–class *Agent* implements nearly all functionality of a generic Soccer player and inherits its properties to the classes Player and GoalKeeper (*Appendix K*).

6.4 The Knowledge Base Model

The knowledge base of a SoccerTeam agent has a relative simple structure that consists of an array for each different types of knowledge. Following knowledge structures are stored:

- All combined game strategies that resulted from a planning process with CEVOP.
- The most recently communicated messages from the other team members and from the simulation server, which are necessary to decide the next primitive game strategy.

6.5 The Behavioural Model

Since no other than the planning behaviour is implemented, the intelligence of the players is restricted to the capabilities of the combined game strategies and the decision mechanism.

Planning

Planning in SoccerTeam is a two–stage process:

```
1) Plan generation through CEVOP runs
2) Plan execution in SoccerTeam simulation runs
```

One CEVOP run usually output one plan for a specific game constellation. The number of constellations, even for non–continues positions, is huge. Therefore, the planning process is iterated until the overall grade of success of SoccerTeam becomes satisfactory in simulation runs.

Objects Of The Planning Process

Following objects of the Soccer planning domain and their relationships are essential for the planning process:

- A plan usually consists of several plan steps.
- A plan step is interpreted as a primitive game strategy that promises individual success in a specific game constellation.
- A plan is interpreted as a combined game strategy, consisting of three successive primitive game strategies that promise individual success for three successive game constellations.
- A team strategy consists of one combined game strategy for each player, that can be applied to the same specific game constellations. However, each player autonomously decides which combined game strategy to apply to a specific game constellations.

Planning Strategies

We have designed different types of game strategies as potential parts of a plan. They are grouped into static game strategies (*Appendix E*) and dynamic game strategies. The latter is further subdivided into basic game strategies, goalkeeper strategies, primitive offensive game strategies, primitive defensive game strategies, primitive game starting or continuing strategies, combined game strategies, and coach's game strategies (*Appendix G*). Where, the latter are meta strategies over the others.

The applicability of a specific primitive strategy depends on the following aspects:

- Allowed: According to the Soccer rules, some are allowed for all team members, some only for the goal keepers. For example, catching the ball is allowed only for the goal keepers within the penalty area.
- Possible: Applying a specific strategy may be allowed, but may not be possible to a specific agent. For instance, only those sufficiently close to the ball may dash it.
- Appropriate: Depending on the game situation, some are appropriate in the specific situation of an agent, some are not. For instance, shooting a goal is usually inappropriate for goal keepers; or defensive strategies are usually inappropriate for the team members, if they are currently in the offensive situation.

A primitive game strategy consists of the strategy type S and the relative distance Δx, Δy, to be applied on a specific game constellation (*Table 6.2*).

Table 6.2: Structure Of A Primitive Game Strategy

S	Δx	Δy

In the plan generation process for constructing a combined game strategy, the change of the game constellation, after all players' first steps have been applied, is successively considered in each of the remaining two plan steps. However, in the plan execution process, after all players' first steps have been applied, the resulting game constellation may not be the same as planned. Two factors cause this effect:

- Each player has restricted sensor capabilities, which can cause to interprete the current constellation differently and therefore plan differently.
- Each player has restricted actor capabilities, caused by noise, which may lead to unexpected results for a plan.
- The opponent's planning and execution approaches may be different, in case of heterogeneous teams.

With respect to the objective of a plan, we differentiate two approaches:

- Relative goal shooting: The number of combined strategies is set to a small number, so that a team plan must not necessarily end up with goal shooting. The objective here is to find, for a given constellation, the best plan.
- Absolute goal shooting: The number of combined strategies is set to a large number, so that a team plan must end up with goal shooting. The objective here is, from a given constellation, to reach the opponent's goal quickly.

While the planning process trough CEVOP, each defensive and offensive team learns one team strategy. For the learning process, one team is fixed to be offensive for all steps of a combined game strategy, and the other team is fixed to be always defensive. Thus, planning three steps ahead aims at finding the most successful first plan step for the current constellation. The remaining two steps represent actually a success factor for the first step and strengthens its strategic value. After the first step of a plan has been applied and the constellation has changed, another more suitable combined game strategy may be chosen.

Strategies For Co–operation And Competition

Based on Soccer rules and individual strategies (*Appendix E*), we define team–oriented strategies for co–operation and competition (*Appendix G*). These strategies are expressed in form of the following chromosome evaluation criteria:

- Offensive goalkeeper: moveInFiled, focusBall
- Offensive players: moveInField, distanceTeamGoal, densityAroundBall
- Defensive goalkeeper: moveInFiled, focusBall

- Defensive player: moveInField, coverClosest

Optimising The Planning Process

In unsupervised learning, in iterative applications the CEVOP algorithm is initialised always with random values. In this case, the set of combined game strategies that covers all possible game constellations found is a statistical mean value. This value is usually better than that of an exhaustive search. However, the statistical mean value can further be reduced, if some heuristics are utilised, which means that the learning process becomes supervised, with respect to initial game constellations.

In supervised learning, the CEVOP algorithm is initialised with specific game constellations, in order to get a solution for which no successful plan exists, yet. Sample heuristics are specific game starting positions for throw–in, kick–off, penalty, or free–kick. For this purpose, related game constellations schemes will be identified and the players' initial co–ordinates randomly concentrated around the ball's co–ordinates.

Learning

Theoretically, any plan that was not known by the agent and was found by CEVOP from the set of all possible plans, represents a learning process. If the set of primitive game strategies remain the same for all CEVOP runs and the most significant game constellations have been learned, then the learning factor turns to decrease with any further CEVOP run. Meaning, that any new plan will increasingly approximate the already learned plans.

6.6 The Communication Model

The communication protocol of the Soccer simulation server [Chen et al.; 2001] will be utilised additionally for inter–team communication of some primitive concepts. A team member may send a message to the others, in order to transmit its intention expressed by the information below:

- The coach wants the players to apply a specific meta strategy.
- The sender wants the receiver to pass the ball to the sender.
- The sender wants the receiver to know that the sender is about to pass the ball to the receiver.

6.7 The Co–operation Model

Although, all 11 players of a team co–operate, because of the difficulty to apply the same fitness function to heterogeneous populations, we must distinguish types for co–operativ evolution depending on population variety:

- Co–operating homogeneous players: Since, the knowledge bases are homogeneous, the plans of the 10 players can evolve in the same search space, which is represented by one population. The same applies to goalkeeper plans.
- Co–operating heterogeneous players: Since, the knowledge bases are heterogeneous, the plans of the 10 players evolve in a different search space than those of the goal keeper, which is represented by two populations. Besides the fitness function of each population, a fitness sharing function is defined, which represents the co–operation relationship between these populations.

6.8 The Competition Model

We represent competition by different populations, since the sought solutions are orthogonal and therefore require different fitness functions. Population variety no more effects this situation, since we already have different populations. An explicit fitness sharing function is not defined, since the competition relationships are implied by designing the chromosome evaluation criteria accordingly orthogonal.

6.9 The Co–ordination Model

The co–operation structures, discussed above, imply some means for co–ordination within the players of a team. Further control is implemented with the meta strategies, which the coach may apply depending on specific game constellations and overall match situations. In other words, we do not need to define explicit control structures here.

6.10 Planning With CEVOP In SoccerTeam

The generic CEVOP methodology is adopted to the Soccer game domain by simplifying some of its concepts and including further Soccer–specific attributes. Particularly, the negotiation and adoption process can be simplified here, since Soccer rules already provide predefined basic co–operation and competition structures. The objective here is to generate for already given co–operation and competition structures, represented in form of strategies, concrete plans.

In the co–operation, competition, and co–ordination models the principle interaction structures found in Soccer game have implicitly been mapped onto the principle interaction structures of the co–evolutionary algorithm. Now we present the mapping of these structures in a compound form, inside the co–evolutionary algorithm and discuss the further details of the algorithm.

Domain Mapping

Mapping the Soccer game strategies onto the co–evolutionary algorithm is one crucial part of the methodology. It is give as follows:

- Individual strategies of the team members (*Appendix F*) are encoded within the genes, which is explained more detailed in the phenotype description below.

- Strategies for co–operation (*Appendix G*) between the goal keeper and the other team members are represented in form of chromosome evaluation criteria.
- The static strategies, strategic areas and learn granularity (*Appendix E*), are considered directly within the chromosome evaluation criteria.
- Strategies for competition are enforced indirectly by designing the offensive/defensive chromosome evaluation criteria appropriately (*Appendix G*).

The result of a CEVOP run are field locations for all 22 team members, which represents a pair consisting of an offensive and a defensive plan with three successive plan steps.

Populations

The homogeneous/heterogeneous and co–operation/competition structures of SoccerTeam are depicted in (*Figure 6.2*). The co–evolutionary algorithm will operate with this schema of populations and relationships.

In Soccer game the goal keepers are competing, too. However, it is quite unlikely that both meet on the field. Therefore this case is unrealistic and can be avoided in the learning phase.

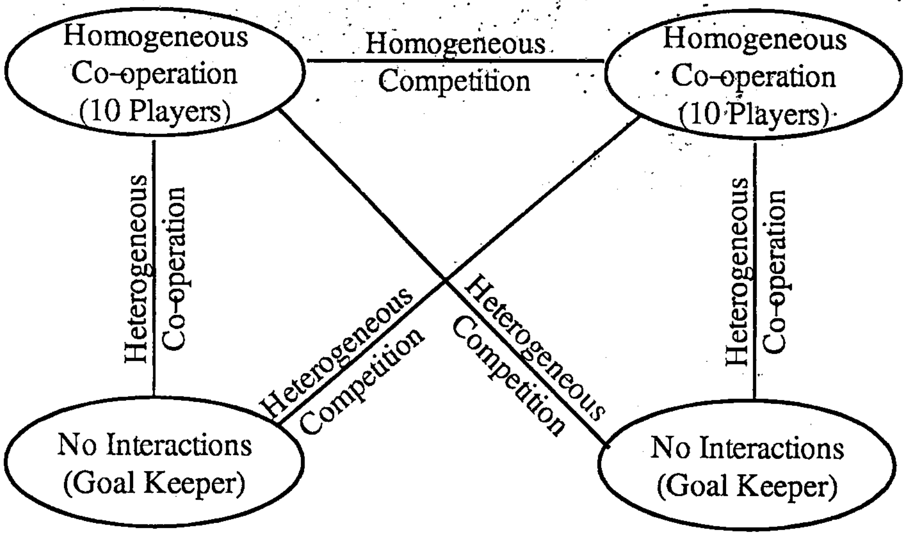

Homogeneous Co-operation (10 Players)
Homogeneous Competition
Homogeneous Co-operation (10 Players)
Heterogeneous Co-operation
Heterogeneous Competition
Heterogeneous Competition
Heterogeneous Co-operation
No Interactions (Goal Keeper)
No Interactions (Goal Keeper)

Figure 6.2: Co–operative/Competitive Evolution Of The Four Homogeneous/Heterogeneous Plan Populations

Phenotype Structure

A phenotype represents 11 combined game strategies $S_{p,i}$, one for each player. Where p = 1, ..., 11. Each of them consists of i = 1, 2, 3 primitive game strategies $S_{p,1}$, ..., $S_{p,3}$ with their relative locations $\Delta x_{1,1}$, $\Delta y_{1,1}$, ..., $\Delta x_{11,3}$, $\Delta y_{1,3}$, one for each player (*Table 6.3*). Each row of the table represents one combined game strategy. Three succeeding columns represent all first primitive game strategies of all 11 players. Thus, one phenotype represents, for a specific game constellation, a team plan solution consisting of 11 plans, one for each player. Where each plan consists further of three plan steps.

Table 6.3: Phenotype Structure Of A Combined Game Strategy

$S_{1,1}$	$\Delta x_{1,1}$	$\Delta y_{1,1}$	$S_{1,2}$	$\Delta x_{1,2}$	$\Delta y_{1,2}$	$S_{1,3}$	$\Delta x_{1,3}$	$\Delta y_{1,3}$
	...			...			...	
$S_{11,1}$	$\Delta x_{11,1}$	$\Delta y_{11,1}$	$S_{11,2}$	$\Delta x_{11,2}$	$\Delta y_{11,2}$	$S_{11,3}$	$\Delta x_{11,3}$	$\Delta y_{11,3}$

Genotype–Phenotype Mapping

The structure of a genotype is defined by a function that maps sequentially each symbol of a phenotype to its value range inside the genotype. Therefore, a transformation in any direction is a sequentially ordered one–to–one mapping between phenotype symbols and genotype genes. However, the combined game strategy that is intended for the goal keeper is transform to genes of a separate chromosome.

The Generic Co–evolutionary Algorithm

Following algorithm co–evolves the n = 4 populations according their m_i interaction types:

```
9) For all n=4 populations
9.1) Randomly pickup plan steps from $P_i$ for population $A_i$
10) For all n=4 populations
10.1) Cross-over inside $A_i$
11) For all n=4 populations
11.1) Mutate distance values inside $A_i$
12) For all n=4 populations
12.1) Calculate Fitness $F_i$ of Population $A_i$
12.2) For all $m_i$ interaction relationships of
  population $A_i$
12.2.1) Calculate fitness $F_{ij}$ for interaction of $A_i$ with
  $A_j$ $(i \neq j)$
12.2.2) $F_c = F_c + F_{ij}$
12.3) $F_i' = (F_i + F_c / m_i) / 2$ and $F_c = 0$
13) IF fitness $F_1$ AND ... AND $F_m$ satisfied OR other
  termination THEN GO TO 7
14) For all n populations
14.1) Fitness selection on $A_i$
14.2) GO TO 2
15) For all n populations
```

```
16) PRINT $A_i$
```

A_i is the plan population to be optimised. P_i is the set of all possible individual plan steps of agent i. Fitness F_i' of population A_i is determined by accumulating its own fitness F_i and all m fitness values for interrelationships F_{ij}. F_c represents the co–operation and/or competition strengths of agent i with the others. The software design of this algorithm is attached as (*Appendix J*).

Interpretation Of The Algorithm For The Case Of Homogeneous Co–operation

In case of the 10 co–operating homogeneous players of a team, the set of all possible plans P_i is identical for all agents A_{i1}, ..., A_{i10}. Thus, the plans can co–operatively evolve inside a single population. For this purpose the plans of all these players are represented inside a single chromosome.

On the other hand, the two goal keepers' plans evolve in distinct populations, since there is no interaction defined between them and since their plans are heterogeneous to the other players.

Crossover

Uniform cross–over is applied on all genes, i.e. the primitive game strategies $S_{1,1}$, ..., $S_{13,3}$ and the distance values $\Delta x_{1,1}$, $\Delta y_{1,1}$, ..., $\Delta x_{11,3}$, $\Delta y_{1,3}$.

Mutation

Mutation is applied only on the distance values $\Delta x_{1,1}$, $\Delta y_{1,1}$, ..., $\Delta x_{11,3}$, $\Delta y_{1,3}$. Where, the range is dependent on the related primitive game strategy.

Local Fitness

The definition of the fitness functions is a further critical part of the CEVOP methodology, since they drive the populations towards the desired fitness. In local fitness evaluation each population is evaluated independently by some criteria. However, common to all is some functionality, which is further grouped depending on the strategy to be applied. A brief description of these groups is given below.

Before the chromosomes of a population are evaluated by a criterion and after cross–over and mutation has been applied, first some services routines perform some semantic checks:

- First, all combined game strategies are applied on the current game constellation.
- A physically impossible move is evaluated to zero.
- A move that violates the game rules is evaluated to zero.

Some fitness evaluations for offensive strategies:

- A move of the ball towards the opponent's goal is evaluated higher than a move off the opponent's goal.
- A move of the ball to a location closer to 90° angle to the goal is evaluated higher than a move to a location closer to 0° angle.
- The fitness value of a combined game strategy is calculated by a distance function that sorts all combined game strategies in decreasing distance, in order to determine the offensive player with the ball O_b, to which the ball will be passed next.

Some fitness evaluations for defensive strategies:

- If a player is the closest one from its team to the ball, then trying to get the ball is evaluated higher for that player.

Global Fitness

In case of Soccer, there was no need to explicitly define interaction types for the above designed co–operation/competition structures. They are implied by related design of the criteria for local fitness evaluation.

Generalisation

The convergence behaviour of the co–evolutionary algorithm depends on each criterion (*Appendix L*), since each fitness criterion shows different convergence behaviour. However, since the fitness functions are normalised to the range [0..1], the accumulated fitness value of each chromosome is expected to converge to 1. The relationships between the criteria, especially those applied on the same chromosome, is essential for the convergence (*Figure 6.3*).

The convergence behaviour of some of the criteria is currently inappropriately designed and/or implemented. For instance, for some initial random constellations the algorithm does not converge, such as if the ball is initially located in a distance to the opponent's goal less then the minimum required by the criterion teamGoalDistanceRelative, then the algorithm will not converge, since this criterion can never be satisfied. This may be a good example for an inappropriate criterion.

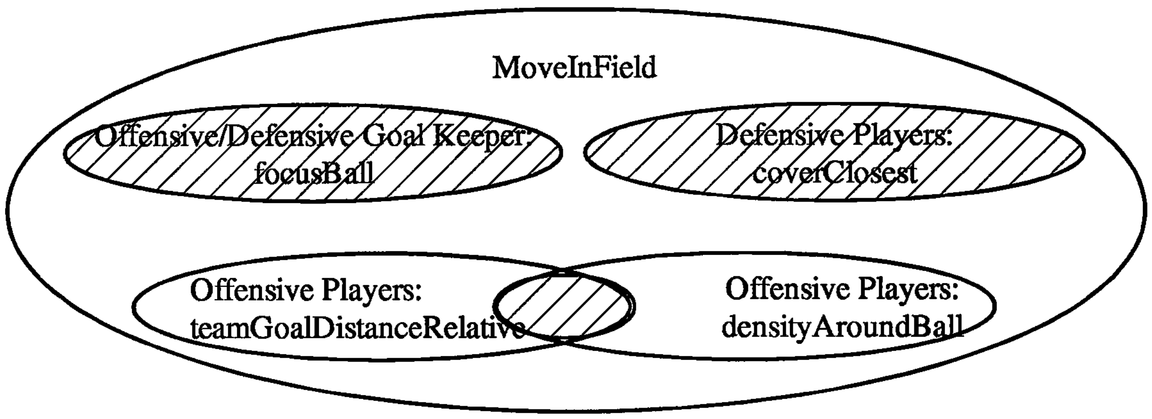

MoveInField
Offensive/Defensive Goal Keeper: focusBall
Defensive Players: coverClosest
Offensive Players: teamGoalDistanceRelative
Offensive Players: densityAroundBall
Solutions For One Population

Figure 6.3: Relationships Between The Chromosome Evaluation Criteria

Combined Evaluated Structures

Implied by the properties plan variety and interaction type, the structures evolve in some combinations. These combinations are recapitulated in the following:

- Each primitive game strategy and its distance values evolve combined to one player strategy.
- Three primitive game strategies with their distance values evolve to a combined game strategy for one player.
- 11 combined game strategies evolve combined to one team strategy; one for each team member.
- Two team strategies evolve combined to one pair of offensive/defensive team strategy.

Overall Algorithmic Steps

The algorithmic steps of the CEVOP methodology are given below.

```
5) Apply the co-evolutionary algorithm for a
  random/desired game constellation
```

```
6) Store the resulting combined game strategies for
  players/goalkeepers in the knowledge base of each
  player/goalkeepers, respectively
7) IF SoccerTeam is not successful in a simulation run
  THEN GO TO 1
```

The number of iterations of this algorithm depends on the grade of success of the team over another team in a simulation run. However, after a specific threshold, each found combined game strategy will look increasingly similar to the previously found strategies. Thus, after a number of iterations the found set of combined game strategies will saturate with respect to its successfulness in simulation runs. Therefore, the goal of the planning process is to find a set of combined game strategies for a team, such that they cover all possible game constellations. In other words, we are seeking for a set of combined game strategies that can provide for any constellation a successful team plan.

The current SoccerTeam implementation can apply just one team primitive game strategy for all team members, in order to inspect the generated plans on the monitor.

CONCLUSIONS

Our conclusions are many–fold. Therefore, we summarise them according some aspects. Several new ideas are introduced here, which reflect potential improvements and future research directions.

The CEVOP Methodology

- Improved plans: Co–operative and competitive planning with the CEVOP methodology has the advantage that plans improve each other through co–evolution.
- Domain independence: The methodology is though for all MAS domains. It is principally independent, but specific solutions may restrict the algorithm to a domain depending on the generality of the used evaluation criteria.
- Parallelism: The algorithm is inherently parallel. For example the computation of each generation of a population can be parallelised to those of the other populations, for the duration of each generation.
- Fuzzyfication: In order to introduce further domain information into the search, the evaluation criteria could be designed so that Fuzzy operations evaluate the chromosomes. For instance, to represent multi–objective criteria in a more flexible way.
- Multi–objective criteria: Could be utilised to represent more flexible co–operation and competition strengths.

- Application areas: The domain–independent algorithmic CEVOP methodology could principally be used as basic approach in any planning component and in any MAS domain. Sample application domains, e–market MASs, information processing MASs, production control MASs, etc.

AgentTeam

- Domain independence: All models of the framework are designed independent from any domain.
- Multi–agent planning: The planning process includes co–operative and competitive relationships of homogeneous and heterogeneous agents.

SoccerTeam

- Off–line planning: Because of the real–time constraints we have placed the planning phase before the playing phase.
- Absolute goal–oriented plans: According the current policy relative goal–oriented plans are generated by allowing only three primitive strategies for one player with a maximum movement range of 60 meters and requiring a relative move towards the goal. An absolute goal–oriented plan could initially start close to the home goal, allow 20–30 primitive game strategies, and require to end always with a goal.
- Variable–length chromosomes: Since the movement within one primitive strategy can be variable, we do not need variable length chromosomes here.
- Plan generation: With the CEVOP methodology individual plans, co–operative plans, and competitive plans are generated within a single planning process.
- Further criteria: We have used only some important meta strategies as evaluation criteria. Further meta strategies could improve the resulting plans, such as covering the goal, allowing relative long passes, or fuzzyfying the criteria.
- Covering all game constellations: By iterating the CEVOP algorithm multiple times, one can find the smallest representative set of team plans that covers all possible game constellations.

ACRONYMS

Following acronyms have been used throughout this work.

ANSI	American National Standards Institute
ARCOL	ARtimis COmmunication Language
B	Ball
BNF	Backus–Naur Form
CEVOP	Co–evolutionary Planning
d_a	Defensive player a attacks O_b, to get B
d_i, d_j	Defensive playing opponent i, j; where $i \neq j$
D	Population of defensive combined game strategies
FIPA	Foundation for Intelligent Physical Agents
G	Goal
HTML	HyperText Mark–up Language
HTTP	HyperText Transport Protocol
IP	Internet Protocol
KB	Knowledge Base
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
MAS	Multi–agent System
O_b	Offensive player b with the ball
O_i, O_j	Offensive player i, j from the same team of player b; where $i \neq t$ and $j \neq t$ and $i \neq j$
o_t	Offensive target player t from the same team of player b, to which b will give a pass
O	Population of offensive combined game strategies

p_i, p_j	Plan step i, j
P_i, P_j	Plan i, j
PP	Possible plans
R	Rule
S_{pi}	Primitive game strategy i
S_{ci}	Combined game strategy i
S_{oi}	Offensive game strategy i
S_{di}	Defensive game strategy i
SG	Goal shoot area
T_i, T_j	Population i, j
URL	Universal Resource Locator
WWW	World Wide Web

REFERENCES

Referenced URLs where valid by May, 2002 !

Andre, David; 1995; "The automatic programming of agents that learn mental models and create simple plans of action"; IJCAI–95; Morgan Kaufmann

Andre, David; Teller, A.; 1999; "Evolving Team Darwin United"; Asada, M.; RoboCup–98; LNCS; Springer; Heidelberg

Angeline, Peter J.; Pollack, Jordan B.; 1994; "Competitive environments evolve better solutions for complex tasks"; Australian Electronics Engineering; Thomson Business

Arora, Neeraj; Sen, Sandip; 1996; "Resolving Social Dilemmas Using Genetic Algorithms"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Asada, Minoru; Kitano, Hiroaki; Noda, Itsuki; Veloso, Manuela; 1999; "RoboCup: today and tomorrow – what we have learned"; Artificial Intelligence; Elsevier

Çaðlayan, Alper; Harrison, Colin G., 1997, "Agent Sourcebook: A Complete Guide to Desktop, Internet, and Intranet Agents", John Wiley Sons Inc., New York

Bäck, Thomas; Schwefel, Hans–Paul; 1992; "An Overview of Evolutionary Algorithms for Parameter Optimization"; Evolutionary Computation 1(1), pp.1–23

Bak, Per; Flyvbjerg, Henrik; Lautrup, Benny; 1992; "Evolution and Co–evolution in a Rugged Fitness Landscape"; *ALIFE III*;

Balakrishnan, Karthik; 1998; "Biologically inspired computational structures and processes for autonomous agents and robots"; Iowa State University

Baral, Chitta; Trejo, Raul; Kreinovich, Vladik; 1999; "Computational complexity of planning and approximate planning in the presence of incompleteness"; *IJCAI'99*

Baray, Cristobal; 1999; "Evolution of Coordination in Reactive Multi–agent Systems"; PhD thesis; Indiana University

Barone, Luigi; While, Lyndon; 1998; "Evolving Computer Opponents to Play a Game of Simplified Poker"; ICEC'98;

Bicchieri, Cristiana; Pollack, Martha E.; Rovelli, Carlo; 1996; "The Potential for Cooperation among Web Agents"; *Adaptation, Coevolution and Learning in Multiagent Systems*; AAAI Press; Menlo Park

Bigus, Joseph P.; Bigus, Jennifer, 1998, "Constructing Intelligent Agents with Java – A Programmer's Guide to Smart Applications", John Wiley Sons Inc., New York

Bonneville, Jacques; Hassas, Salima; 1996; "Towards a self–organizational approach for a parallel computation in a distributed production rule based system"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Box, G.E.P.; 1957; "Evolutionary operation: a method of increasing industrial productivity", Applied Statistics, 6, 81–101

Brassard, Gilles; Bratley, Paul; 1988, "Algorithmics – Theory & Practice"; Prentice–Hall; Englewood Cliffs

Bui, H.H.; Kieronska, D.; Venkatesh, S.; 1996; "Negotiating agents that learn about others' preferences"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Bull, Lawrence; Fogarty, Terence C.; 1996; "Evolutionarey Computing in Cooperative Multi–agent Environments"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Çaðlayan, Alper; Harrison, Colin G.; 1997; "Agent Sourcebook: A Complete Guide to Desktop, Internet, and Intranet Agents"; *John Wiley*; New York

Cavazza, Marc; Mead, Steven J.; Strachan, Alexander I.; Whittacker, Alex; 2001; "A Blackboard System for Interpreting Agent Messages"; *AAAI*;

Chaib–draa, Brahim; Vanderveken, Daniel; 1998; "Agent Communication Language: Towards a Semantics Based on Success, Satisfaction, and Recursion"; *Intelligent Agents V – Agent Theories, Architectures, and Languages*; Springer; Berlin, 1999

Chavez, Anthony; Moukas, Alexandros; Maes, Pattie, 1997, "Challenger: A Multi–agent System for Distributed Resource Allocation", Autonomous Agents'97, Marina Del Ray

Cheikhrouhou, Morsy M.; Conti, Pierre; Labetoulle, Jacques; 1998; "Intelligent Agents in Network Management – A State of the Art"; *Networking and Information Systems*;

Chen, Liren; Sycara, Katia; 1997, "WebMate: A Personal Agent for Browsing and Searching", TR, Carnegie Mellon University, Pittsburgh

Chen, Mao; Foroughi, Ehsan; Heintz, Fredrik; Zahn, Xiangh Huang; Kapetanakis, Spiros; Kostiadis, Kostas; Kummeneje, Johan; Noda, Itsuki; Obst, Oliver; Riley, Pat; Steffens, Timo; Wang, Yi; Yin; Xiang; 2001; "RoboCup Soccer Server"; RoboCup Federation; http:// www.robocup.org/

Ciancarini, P.; Rossi, D.; 1998; "Coordinating Distributed Applets with Shada/Java"; *SAC'98*

Cliff, D.; Miller, G.F.; 1995; "Tracking the Red Queen: Measurements of adaptive progress in coevolutionary simulations"; Moran, F.; Moreno, A.; Merelo, J. J.; Cachon, P.; Advances in Artificial Life: Proceedings of the Third European Conference on Artificial Life; ECAL95; LNCS; Springer; Heidelberg

Cohen, Philip R.; Cheyer, Michelle; Wang, Soon; Cheol, Baeg; 1994, "An Open Agent Architecture", AAAI Spring Symposium, pp. 1–8, March

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; 1990; "Introduction to Algorithms"; MIT Press; Cambridge

Cramer, Michael Lynn; 1985; "A representation for the adaptive generation of Simple Sequentisl programs";Grefenstelle J.J.; Proceeding of the International Comference on Genetic Algorithms and Their Applications; Morgan Kaufmann

Dam, Mads; 1997; "Analysis and Verification of Multiple–Agent Languages", *LOMAPS'96*; Lecture Notes in Computer Science, Springer; Berlin

Dawkins, R.; Krebs, J.R.; 1979; "Arms races between and within species"; Royal Society of London;

Darwen, P.; Yao, X.; 1996; "Every niching method has its niche: fitness sharing and implicit sharing compared"; Proc. of Parallel Problem Solving from Nature; Lecture Notes in Computer Science; Springer

Darwin, Charles; 1859, "The Origin of Species", John Murray; London

Desai, Bipin C.; 1990, "An Introduction to Database Systems", West Publishing, St. Paul

De Jong, Kenneth A.; 1975; "An Analysis of the Behavior of a Class of Genetic Adaptive Systems"; PhD thesis; University of Michigan

De Jong, Kenneth A.; 1990; "Genetic–algorithm–based learning"; Kodratoff, Y.; Michalski, R. S; Machine Learning; Morgan Kaufmann

De Jong, Kenneth A.; Spears, W. M.; 1991; "Learning Concept Classification Rules Using Genetic Algorithms"; 12th International Joint Conference on Artificial Intelligence; Sydney

d'Inverno, M.; Fisher, M.; Lomuscio, Luck, A.; M.; de Rijke, M.; Ryan, M.; Wooldridge, M.; 1997; "Formalisms for multi–agent systems"; *Knowledge Engineering Review*;

Dix; Jürgen; Muñoz–Avila, Héctor; Nau, Dana S.; 2000; "IMPACTing shop: Planning in a Multi–Agent Environment"; *CL–2000 Workshop on Computational Logic in Multi–Agent Systems*

Dominiak, Dana M.; 2001; "Genetic algorithms for agent evolution and resource exchange in complex adaptive systems"; PhD thesis; Illinois Institute of Technology

Dorrer, Klaus; 1999; "Motivation, Handlungskontrolle und Zielmanagement in autonomen Agenten";PhD thesis; Albert–Ludwig Universität Freiburg

Durfee, E. H.; Lesser, V. R.; 1991; "Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation"; *IEEE Transactions on Systems, Man, and Cybernetics*

Duschka, Oliver M.; Genesereth, Michael R.; 1997; "Infomaster–An Information Integration Tool", *International Workshop on Intelligent Information Integration*; Freiburg

Eiter, Thomas; Mascardi, Viviana; 2001; "Comparing Environments for Developing Software Agents"; INFSYS research report; Technische Universität; Wien

Ellis, Margaret A.; Stroustrup, Bjarne; 1990; "The Annotated C++ Reference Manual"; Addison–Wesley; New York

Engelmore, Robert; Morgan, Tony; 1988, "Blackboard Systems", Edison–Wesley; Wokingham

Evans, Eric; Rogers, Daniel; 1997, "Using Java Applets and CORBA for Multi–User Distributed Applications", IEEE Internet Computing, Vol.1 No.3 pp.43–55

Ferber, Jacques; 1999; "Multi–agent Systems – An Introduction to distributed artificial intelligence"; Addison–Wesley; New York

FIFA; 2001; "Laws of the Game"; Fédération Internationale de Football Association

Finin, Tim; Fritzson, RichMcKay, Donald, 1992, "A Language and Protocol to Support Intelligent Agent Interoperability",

Finin, Tim; Wiederhold, Gio; Weber, Jay; Genesereth, Michael; Fritzson, Richard; McKay, Donald; McGuire, James; Pelavin, Richard; Shapiro, Stuart; Beck, Chris; 1993, "Specification of the KQML Agent–Communication Language", draft, http:// www.cs.cmu.edu/ ~softagents

Finin, Tim; Labrou, Yannis; Mayfield, James; 1994, "KQML as an Agent Communication Language", Draft, http:// www.cs.umbc.edu/ kqml, University of Maryland Baltimore County, Baltimore

FIPA; 1997; "Foundation for Intelligent Physical Agents"; FIPA 97 Specification, part 2, Agent Communication Language, http:// drogo.cselt.stet.it/fipa/ spec/fipa97/fipa97.htm

Fogel, J. Lawrence; Owens, A.J.; Walsh, M.J.; 1966; "Artificial Intelligence Through Simulated Evolution"; John Wiley & Sons; New York

Franklin, S.; Graesser, A.; 1996; "Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents"; *Intelligent Agents III: Agent Theories, Architectures, and Languages*; Proceedings of ECAI'96 Workshop

Fraser, A.S.; 1957; "Simulation of genetic systems by automatic digital computers"; Australian Journal of Biological Sciences, 10, 484–491.

Friedman, G.J.; 1959; "Digital simulation of an evolutionary process", General Systems Year Book; 4:171–184

Funes, Pablo Jose; 2001; "Evolution of complexity in real–world domains"; PhD thesis; Brandeis University

Furguson, Innes A.; Karakoulas, Grigoris J.; 1996; "Multiagent Learning and Adaptation in an Information Filtering Market"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Garland, Andrew; Alterman, Richard; 1996; "Multiagent Learning through Collective Memory"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Genesereth, Michael R.; Keller, Arthur M.; Duschka, Oliver M.; 1997; "Infomaster: An Information Integration System"; *ACM SIGMOD*

Gerber, Christian; 1999; "Evolution–Based Self–Adaptation as an Expression for the Autonomy Degree in Multi–Agent Societies"; DFKI;

Goldberg, David Edward; Richardson, J.; 1987; "Genetic algorithms with sharing for multimodal function optimization"; Proceedings of the Second International Conference on Genetic Algorithms

Goldberg, David Edward; 1989; "Genetic Algorithms in Search, Optimization and Machine Learning"; Eddison–Wesley

Goldberg, David Edward; Borgerson, J.; Vaughn, A.; Hawley, K.; Cunningham, C.; Milner, J.; Zacarias, K.; Wagus, B.; Gadient, R.; Sutton, B.; Pelikan, M., Roth; 1997; "Genetic algorithms: a bibliography"; University of Illinois at Urbana–Champaign

Goodwin, Richerd; 1994; "Formalizing Properties of Agents"; *Journal of Logic and Computation*; Vol. 5, Issue 6

Grecu, Dan L.; Brown, David C.; 1996; "Learning To Design Together"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Green, Shaw; Hurst, Leon; Nangle, Brenda; Cunningham, Pádraig; Somers, Fergal; Evans, Richard; 1997, "Software Agents: A review"; http:// www.cs.tcd.ie/ research_groups/aig/iag/pubreview.ps.gz

Grefenstette, John J.; 1989; "A system for learning control rules with genetic algorithms"; Proc. Third Intl. Conf. Genetic Algorithms; Morgan Kaufmann

Grefenstette, John J.; 1991; "Lamarckian learning in multi–agent environments"; Proc. Fourth Intl. Conf. of Genetic Algorithms; Morgan Kaufmann

Grefenstette, John J.; 1992; "The evolution of strategies for multi–agent environments"; Adaptive Behavior

Grefenstette, John; Daley, Robert; 1996; "Methods for Competitive and Cooperative Co–evolution"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Grosso, P.; 1985; "Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model"; PhD thesis; University of Michigan

Gruber, Thomas R.; 1994, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing", Guarino, Nicola; Poli, Roberto; Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic

Hashimoto, Keisuke; Kawamura, Hidenori; Yamamoto; Masahito; Ohuchi, Azuma; 2002; "Team Description for Harmony Co–evolutionary Positioning"; RoboCup'02; LNAI; Springer; Heidelberg

Hayes–Roth, Barbara; Hayes–Roth, F.; Rosenschein, S.; Cammarata, S.; 1988, "Modeling Planning as an Incremental Opportunistic Process"; Engelmore, Robert; Morgan, Tony; *Blackboard Systems*; Edison–Wesley; Wokingham

Hayes–Roth, Barbara; 1992; "Opportunistic Control of Action in Intelligent Agents"; TR, Stanford University; Palo Alto

Haynes, Thomas; Lau, Kit; Sen, Sandip; 1996; "Learning Cases to Complement Rules for Conflict Resolution in Multiagent Systems"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Hedberg, Sara; 1997; "Robots playing Soccer ? RoboCup poses a new set of AI challenges"; IEEE Expert;

Heitkötter, Jörg; Beasley, David; 2001; "The Hich–Hiker's Guide to Evolutionary Computation"; Uunet Deutschland GmbH; http:// surf.de.uu.net/ encore/www/

Hillis, W.D.; 1991; "Co–evolving parasits improve simulated evolution as an optimization procedure"; Langton, C.; et al. Artificial Life II; Addision–Wesley

Hobbes, Thomas; 1999; "Agent Communication Languages: The Current Landscape"; *IEEE Intelligent Systems*; March

Holland, Jhon H.; 1975; "Adaptation in natural and artificial systems"; The University of Michigan Press

Holland, Jhon H.; Reitman, J. S.; 1978; "Cognitive systems based on adaptive algorithms"; Waterman, D. A.; Heyes–Roth, F.; Pattern–Directed Inference Systems; Academic Press

Holland, O. E.; 1996; "Multiagent systems: Lessons from social insects and collective robotics"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Horn, J.; 1997; "The nature of niching: genetic algorithms and the evolution of optimal, cooperative populations"; PhD thesis; University of Illinois at Urbana–Champaign

Huhns, Michael N.; 1997, "Ontologies for Agents", IEEE Internet Computing, Vol.1 No.6 pp.81–83

Huhns, Michael N.; Singh, Munindar P.; 1997, "Conversational Agents", IEEE Internet Computing, Vol.1 No.2 pp.73–75

Husbands, Philip; Mill, Frank; 1991; "Simulated coevolution as the mechanism for emergent planning and scheduling"; Belew, Richard K.; Booker, Lashon B.; *Proceedings of the Fourth International Conference on Genetic Algorithms*; Morgan Kaufmann

Iba, Hitoshi; 1998; "Evolutionary Learning of Communicating Agents"; Journal of Information Sciences; Elsevier;

Iglesias, C. A.; Garijo, M.; González, J. C.; 1998; "A Survey of Agent–Oriented Methodologies"; Singh, M. P.; Müller, J. P.; Rao, A. S.; *Intelligent Agents V. Agent Theories, Architectures, and Languages*; Lecture Notes in Articial Intelligence; Springer

Joseph, W. Sullivan; Sharman, W. Tyler; 1991, "Intelligent User Interfaces", ACM Press, New York

Juillé, Hugues; Pollack, Jordan B.; 1998; "Coevolutionary Learning: A Case Study"; International Conference on Machine Learning

Kandzia, Peter; Klusch, Matthias; 1997; "Cooperative Information Agents"; *CIA'97*; *Lecture Notes in Artificial Intelligence*; Springer; Berlin

Kaufman, Lar ; Welsh, Matt; 1995; "Running Linux"; O'reilly;

Kauffman, S. A.; 1993; "The origin of order"; Oxfort University Press

Kernighan, Brian W.; Ritchie, Dennis M.; 1983; "Programmieren in C mit dem C–Reference Manual in deutscher Sprache"; CarlHanser Verlag; Wien

Knapik, Michael; Johnson, Jan; 1998; "Developing Intelligent Agents for Distributed Systems: Exploring Architecture, Technologies, and Applications"; McGraw–Hill; New York

Kniskern, Joel; Rauscher Mark D.; 2001; "Two models of host-enemy model"; Population Ecology; Springer; Toyo

Koza, John R.; 1989; "Hierarchical Genetic Algorithms Operating on Populations of Compiler Programs"; Sridharan, N.S.; Eleventh Joint Conference on Artificial Intelligence; Morgan Kaufmann; San Mateo

Koza, John R.; 1992; "Genetic Programming"; *The MIT Press*; Cambridge

Krone, Oliver; Chantemargue, Fabrice; Dagaeff, Thierry; Schumacher, Michael; Hirsbrunner, Béat; 1998, "Coordinating Autonomous Entities", SAC'98

Kummeneje, Johan; 2001; "RoboCup as a Means fo Research, Education, and Dissemination"; Ph. Lic. Thesis; Stockholm University

Kumova, Bora İ.; 1998c; "System Specification for an Example Distributed Database"; Msc thesis; Dokuz Eylül Üniversitesi; İzmir; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; Kut, Alp; 1999; "Agent-based System Co-operation"; PDP'99, Madeira, http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 1999; "The Agent Model of AgentTeam"; TAINN'99; İstanbul; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 1999c; "Ağ bağlantılı Uygulamaların Akıllı Bileşenlerle Yeteneklerini Artırmak"; inet-tr'99; Ankara; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 2000a; "Flexible Distributed Database Management with AgentTeam"; IASTED-Applied Informatics'00; Innsbruck; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 2000b; "Object-oriented Design Concepts for Client-Broker-Server-based Multi-agent Systems"; TAINN'00; İzmir; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 2000d; "The Agent Communication Language AgentCom and its Semantics"; IASTED-AI'00; SAIS'01; ACTA Press

Kumova, Bora İ.; 2000e; "Software Design Patterns for Intelligent Agents"; SSGRR'00; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 2000f; "Object-oriented Design and Implementation of the Multi-agent System AgentTeam"; ECOOP'00; OOPSLA'00; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 2000g; "Yapay Us"; TRT2; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 2000h; "Evaluation of the Object-orientation of a Multi-agent System"; ADVIS'00; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 2000i; "On the Improvement of Networked Applications with Intelligent Components"; IASTED-SAE'00; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 2000j; "D.E.Ü. Internet Bağlantılı Dağıtık Veritaban Taslağı"; inet-tr'00; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 2001a; "Dynamic Re-configurable Transaction Management in AgentTeam"; Euromicro PDP'01; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 2001b; "Evaluating Protocols for Intelligent Agent Communication"; BAS'01; http:// cs.deu.edu.tr/ ~kumova

Kumova, Bora İ.; 2002a; "The Game Strategies of SoccerTeam"; *TAINN'02*; İstanbul Üniversitesi; İstanbul

Kumova, Bora İ.; Imada, Akira; 2002b; "The Concepts of the SoccerTeam Prototype"; *ADVIS'02*; Dokuz Eylül Üniversitesi; İzmir

Kumova, Bora İ.; Kut, Alp; 2002c; "Akıllı Yazılım Bileşenlerin Yapısal Özellikleri"; *Fen ve Mühendislik Dergisi*; Dokuz Eylül Üniversitesi; İzmir

Labrou, Yannis; Finin, Tim; 1997; "Semantics for an Agent Communication Language", Sing, Munindar P.; Rao, Anand; Wooldridge, Michael J.; 1998; *Intelligent Agents IV - Agent Theories, Architectures, and Languages*; Springer, Berlin

Lander, Susan E.; 1997, "Issues in Multiagent Design Systems", IEEE Expert, Vol.12 No. 2 pp.18-26

Lesser, Victor R.; Corkill, Daniel D.; 1988; "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks"; Engelmore, Robert; Morgan, Tony; *Blackboard Systems*; Edison-Wesley; Wokingham

Luke, Sean; 1998; "Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97"; GP98; J. Koza et al.; Morgan Kaufmann; San Fransisco

Maes, Pattie; 1989; "The dynamics of action selection"; IJCAI'87; Morgan Kaufmann

Maes. Pattie; 1995; "Modelling adaptive autonomous agents"; Langton, C.; Artificial Life: An Overview; MIT Press; Cambridge

Mahfoud, S. W.; 1995; "Niching methods for Genetic Algorithms"; PhD thesis; University of Illinois at Urbana–Champaign

Martin, David; Cheyer, Adasm J.; Moran, Douglas B.; 1999; "The Open Agent Architecture: A Framework for Building Distributed Software Systems"; *Applied Artificial Intelligence*;

Matsubara, Hitoshi; Noda, Itsuki; Hiraki, Kazuo; 1996; "Learning of Cooperative actions in multi–agent systems: a case study of pass play in Soccer"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Maturana, F.; Balasubramanian, S.; Norrie, D.H.; 1996; "A Multi–Agent Approach to Integrated Planning and Scheduling for Concurrent Engineering"; *Proceedings of the International Conference on Concurrent Engineering: Research and Applications*; Toronto

Mayfield, James; Labrou, Yannis; Finin, Tim; 1995, "Evaluation of KQML as an Agent Communication Language", Intelligent Agents Volume II – Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages; Wooldridge, M.; Muller, J.P.; Tambe, M.; 1996; Lecture Notes in Artificial Intelligence; Springer; Berlin

McCarth, J.; Hayes, P.J.; 1969; "Some Philosophical Problems from the Standpoint of Artificial Intelligence";*Machine Intelligence*; Meltzer; Michie; Edinburgh University Press

Michalewicz, Z.; 1996; "Genetic Algorithms + Data Structures = Evolution Programs"; Springer–Verlag; Berlin

Miller, Brad L.; Shaw, Michael J.; 1995; "Genetic Algorithms with Dynamic Niche Sharing for Multimodal Function Optimization"; ILLIGAL Report No. 95010

Nangsue, Phaderm; 1999; "An agent–oriented, massively distributed parallelization model of evolutionary algorithms"; Clarkson University;

Nodine, Marian; Perry, Brad; Unruh, Amy; 1998, "Experience with InfoSleuth Agent Architecture", AAAI–98 Workshop on Software Tools for Developing Agents

Nodine, Marian; Chandrasekara, Damith; 1998; "Agent Communities"; technical report, MCC

Nolfi, Stefano; Floreano, Dario; 1998; "How co–evolution can enhance the adaptive power of artificial evolution: Implications for evolutionary robotics"; LNCS; Springer

Nolfi, Stefano; Floreano, Dario; 1999; "Learning and evolution"; Autonomous Robots; Kluwer

Nwana, H. S.; Ndumu, D. T.; Lee, L. C.; Collis, J. C.; 1999; "ZEUS: A Toolkit for Building Distributed Multi–Agent Systems"; *Applied Artificial Intelligence Journal*;

Odubiyi, Judé B.; Kocur, David J.; Weinstein, Stuart M.; Wakin, Nagi; Srivastava, Sadanand; Gokey, Chris; Graham, JoAnna; 1997, "SAIRE – A Scalable Agent–based Information Retrieval Engine", Autonomous Agents'97, Marina Del Ray

Oei, C. K.; Goldberg, D. E.; Chang, S. J.; 1991; "Tournament selection, niching, and the preservation of diversity";

Orfali, Robert; Harkey, Dan; Edwards, Jari; 1996, "The Essential Distributed Objects Survival Guide", John Wiley & Sons; New York

Ossowski, Sascha; 1999; "Co–ordination in Artificial Agent Societies – Social Structures and its Implications for Autonomous Problem–Solving Agents"; Springer; Berlin

Pagie, Ludo; Mitchel, Melanie; 2000; "A comparison of evolutionary and coevolutionary search"; Santa Fe Institute

Paolucci, M.; Kalp, D.; Pannu, A.; Shehory, O.; and Sycara, K. 2000; "A planning component for retsina agents"; Wooldridge, M.; Lesperance, Y.; *Lecture Notes in Artitcial Intelligence*; Intelligent Agents VI; Springer

Papadopoulos, Georg A.; Arbab, Farhad; 1998; "Modeling Activities in Information Systems using the Coordination Language MANIFOLD", *SAC'98*

Papadopoulos, Georg A.; Arbab, Farhad; 1998; "Coordination Models and Languages", Advances in Computers, No 46, Academic Press

Paredis, Jan; 1995; "Coevolutionary Computation"; Artificial Life;

Paredis, Jan; 1996; "Coevolutionary Life–time Learning"; Voigt, Hans–Michael; Ebeling, Werner; Rechenberg, Ingo; Schwefel, Hans–Paul; Parallel Problem Solving from Nature; LNCS; Springer; Heidelberg

Paredis, Jan, Westra, R.; 1997; "Coevolutionary Computation for Path Planning"; Proceedings 5th European Congress on Intelligent Techniques and Soft Computing; H.–J. Zimmermann; Verlag Mainz

Paredis, Jan; 1998; "Coevolutionary Algorithms – The Handbook of Evolutionary Computation"; Fogel, T.; Michalewicz, D. Z.; Oxford University Press

Pearson, Glen; 1985; "Mission Planning within the Framework of the Blackboard Model"; Karna, Kamal N.; *Expert Systems in Governmont Symposium*; IEEE

Petridis, Miltos; Knight, Brian; 2001; "A blackboard architecture for a hybrid CBR system for scientific software"; University of Greenwich

Pollack, Jordan B.; Blair, Alan D.; 1998; "Co–evolution in the successful learning of Backgammon strategy"; Machine Learning;

Potter; Mitchell A.; De Jong, Kenneth A.; Grefenstette, John J.; 1995; "A Coevolutionary Approach to Learning Sequential Decision Rules"; Proceedings of the Sixth International Conference on Genetic Algorithms; Morgan Kaufmann

Potter, Mitchell A.; 1997; "The Design and Analysis of a Computational Model of Cooperative Coevolution"; dissertation; George Mason University; Virginia

Potter; Mitchell A.; De Jong, Kenneth A.; 2000; "Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents"; Evolutionary Computation; MIT Press

Potter, Mitchell A.; Meeden, Lisa A.; Schultz, Alan C.; 2001; "Heterogeneity in the Coevolved Behaviors of Mobile Robots: The Emergence of Specialists"; Proceedings of The Seventeenth International Conference on Artificial Intelligence; Morgan Kaufmann

Prasad, M. V. Nagendra; Lesser, Victor R.; Lander, Susan E.; 1996; "Learning Organizational Roles in a Heterogeneous Multi–agent System"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Puppala, Narendra; Sen, Sandip; Gordin, Maria; 1998; "Shared memory based Cooperative Coevolution";IEEE Society;

Rechenberg, Ingo; 1964; "Kybernetischer Lösungsweg eines Experimentellen Problems", lecture at WGLR; Berlin

Rechenberg, Ingo; 1973; "Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution", Fromman–Holzboog; Stuttgart

Reis, Luis Paulo; Lau, Nuno; Oliveira, Eugénio Costa; 2001; "Situation Based Strategic Positioning for Coordinating a Team of Homogeneous Agents"; Balancing Reactivity and Social Deliberation in Multi–Agent Systems; Hannebauer, Markus; Wendler, Jan; Pagello, Enrico; LNCS; Springer

Reiher, P.; Guy, R.; Yavis, M.; Rudenko, A.; 2000; "Automated Planning for Open Architectures"; *OpenArch '2000*;

Reynolds, Craig W.; 1994; "Competition, coevolution and the game of tag"; Brooks, Rodney A.; Maes, Pattie; Fourth International Workshop on the Synthesis and Simulation of Living Systems; MIT Press; Cambridge

Rich, Elaine; Knight, Kevin; 1991; "Artificial Intelligence", McGraw–Hill; New York

Ridley, M.; 1993; "The Red Queen: Sex and the evolution of human nature"; Viking; London

Riedmiller, M.; Merke, A.; Hoffmann, A.; Nickschas, M.; Withopf, D; Zacharias, Z.; 2002; "Brainstormers 2002 – Team Description"; RoboCup'02; LNAI; Springer; Heidelberg

Rosin, Christopher Darrell; Belew, Richard K.; 1997; "New methods for competitive coevolution"; *Evolutionary Computation*

Rosin, Christopher Darrell; 1997; "Coevolutionary search among adversaries"; University of California, San Diego

Roychowdhury, Shounak; Araro, Neeraj; Sen, Sandip; 1996; "Effects of local information on group behavior"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Russell, Stuart J.; Norvig, Peter; 1995; "Artificial Intelligence – A Modern Approach"; Prentice Hall; Englewood Cliffs

Sadek, D.; Bretier, P.; Panaget, F.; 1997, "ARCOL agent communication language and MCP, CAP and SAP agent's cooperativeness protocols", proposal, France Télécom

Sandholm, Tuomas W.; 1999; "Distributed rational decision making"; Weiss, Gerhard; *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*; MIT Press, Cambridge

Schalkoff, J. Robert; 1990, "Artificial Intelligence: An Engineering Approach", McGraw–Hill, New York

Schmidhuber, Jürgen; 1996; "A General Method for Multi–agent Reinforcement Learning in Unrestricted Environments"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Schwefel, Hans–Paul; 1977; "Numerische Optimierung von Computermodellen mittels der Evolutionsstrategie", Birkhäuser; Basel

Sing, Munindar P.; Rao, Anand; Wooldridge, Michael J.; 1998; "Intelligent Agents IV – Agent Theories, Architectures, and Languages"; *ATAL'97*; Springer; Berlin

Singh, Narinder; 1998, "Unifying Heterogeneous Information Models", Communications of the ACM, Vol.41 No.5

Smith, I.; Cohen, P.; 1996; "Toward a Semantics for an Agent Communication Language Based on Speech Acts"; *AAAI'96*;

Smith, R. E.; Forrest, S.; Perelson, A. S.; 1993; "Search for diverse cooperative populations with genetic algorithms"; Evolutionary Computation;

SOAP; 1999; "Simple Object Access Protocol (SOAP) 1.1"; *Apache*; http:// www.w3.org/ TR/SOAP

Stone, Peter; Veloso, Manuela; 1996; "Towards Collaborative and Adversarial Learning: A Case Study in Robotic Soccer"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

Stone, Peter; 1998; "Layered Learning in Multi–agent Systems"; PhD thesis; Carnegie Mellon University

Sycara, Katia; Decker, Keith; Pannu, Anandeep; Williamson, Mike; Zeng, Dajun; 1997; "Distributed Intelligent Agents"; *IEEE Expert*; http:// www.cs.cmu.edu/ ~softagents

Tahk, Min–Jea; Kim, Youdan; Nam, Changho; 1999; "Coevolutionary approaches to structural optimization"; AIAA Journal; AIAA

Tanaka–Ishii, Kumiko; Frank, Ian; Arai, Katsuto; 2000; "Trying to Understand RoboCup"; AI Magazine; AAAI

Vitek, Jan; Tschudin, Christian; 1997; "Mobile Object Systems – Towards the Programmable Internet"; Springer; Berlin

Wooldridge, Michael; Jennings, Nicholas R.; 1995; "Intelligent Agents: Theory and Practice"; *The Knowledge Engineering Review*

Wooldridge, Michael; 1998; "Verifiable Semantics for Agent Communication Languages"; *ICMAS'98*; IEEE Computer

Wooldridge, Michael; Jennings, Nicholas R.; 1999; "The Co–operative Problem Solving Process"; *Journal of Logic & Computation*;

Yao, X.; Liu, Y.; Darwen, P.; 1996; "How to make best use of evolutionary learning"; Complex Systems – From Local Interactions to Global Phenomena; IOS

Yao, Xin; 1997; "Automatic acquisition of strategies by co–evolutionary learning"; Proc. of the International Conference on Computational Intelligence and Multimedia Applications

Yunpeng, Cai; Jiang, Chen; Jinyi, Yao; Shi, Li; 2001; "Global Planning from Local Eyeshot: An Implementation of Observation–base Plan Coordination in CoboCup Simulation Games";

Zeng, Dajun; Sycara, Katia; 1996; "Bayesian Learning in Negotiation"; Adaptation, Coevolution and Learning in Multiagent Systems; AAAI Press; Menlo Park

APPENDIX

The following appendices provide additional information to the main work in two alternative ways:

- Support a matter through a more detailed discussion of this work. For instance the CEVOP methodology, the AgentTeam framework, and implementation–oriented details of the SoccerTeam prototype.
- Support a matter through a more detailed discussion of another work. For instance the official Soccer game regulations or the RoboCup Soccer simulation server.

It was an important design issue to keep the appendices modular, so that they can be studied independent from each other.

APPENDIX A

SYSTEM REPRESENTATION FORMS

Any system consists primarily of some functionality and some data. Early in the history of software systems, in order to improve their maintenance, a separation of mainly functional concepts from data–oriented concepts were proposed for the design phase of the development process. For instance, representing a system in data flow diagrams and flow charts in the design, or collecting data in a database and functionality in libraries in the implementation. Theoretically, any pure data can be represented by some pure functionality, and vice versa. For instance, for any data sequence a representation in form of a function can be found, and vice versa. Thus, various representation forms can be found for a single system.

To improve the maintainability even more, further representation forms have been introduced over time, which can be classified in control–oriented, logic–oriented, and object–oriented concepts. Since, any system represented in these abstractions can be reduced to some functionality and data, function–oriented and data–oriented representations appear again as a super–set (*Figure A.1*). This claim is supported by the fact that the reverse is not always true. That is, not all functionality or data is object–oriented or logical. Where, with logic we mean mathematical logic, such as predicate logic and its extensions.

Since, functionality can be sub–divided into conditional and non–conditional sequences, control–oriented representation emerges as another true subset of function–orientation. For instance, it is possible to represent a system pure control–oriented and data–oriented, without object–orientation or logic–orientation.

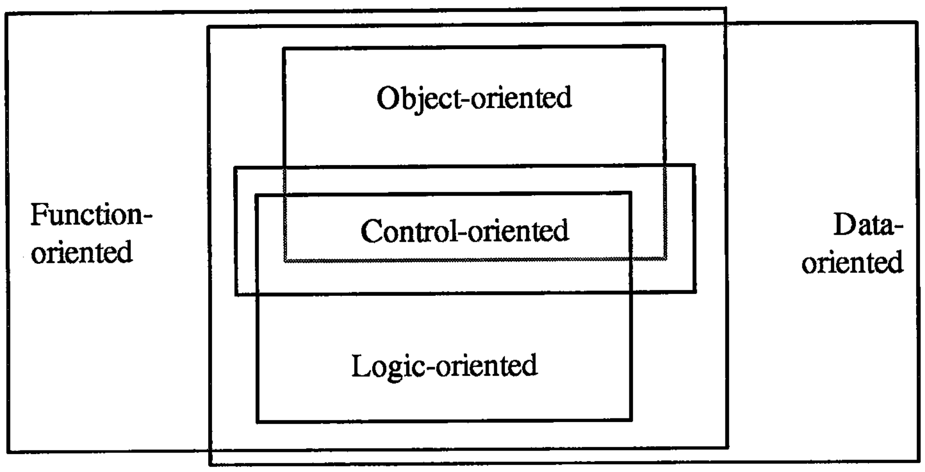


Figure A.1: Relationship Between System Representation Forms

The above information can be useful to design a system that is more flexible against various types of heterogeneity. Solution approaches that cupe with heterogeneity can increase the consistency of data and functionality of distributed systems. Object–orientation as already proved itself as a successful approach for hiding heterogeneity. We believe however that a system which provides transformation capabilities for all discussed representation forms could be more powerful, since in that case even a higher system abstraction would be reached.

APPENDIX B
BNF OF AGENTCOM

In order to shorten the way iterative rules are expressed and thereby increase readability, exponential non–terminals extend this BNF. Thus, an exponential non–terminal represents an iterative rule.

```
<session> ::= <message>
<message> ::= <sessionBegin>
   ({<give> | <take> | <create> | <delete>} <respond>)*
   <sessionEnd>
<sessionBegin> ::= (SESSION_BEGIN <keyword>+)
<sessionEnd> ::= (SESSION_END <keyword>+)
<give> ::= (GIVE <keyword>+)
<take> ::= (TAKE <keyword>+)
<create> ::= (CREATE <keyword>+)
<delete> ::= (DELETE <keyword>+)
<respond> ::= (RESPOND <keyword>+)
<keyword> ::= (ERROR eValue) | (ADDRESS aSource+
    aDestination+) |
   (NAME nSource+ nDestination+) | (DOMAIN dValue+) |
   (SESSION_ID i) | (MESSAGE_ID j) | <object>+
<object> ::= (OBJECT oName instanceName {instanceValue+ |
   (<property> <object>+)+})
<property> ::= ATTRIBUTE |VALUE | HAS | METHOD |
    <operation>
<operation> ::= && | || | → | ↔ | < | > | != | == | = |
    += | -= | *= | /= | + | - | * | /
```

In case of synchronous communication, a response message is returned synchronously. In case of asynchronous communication, a response message is sent back from server to the client explicitly.

APPENDIX C
SOCCER GAME REGULATIONS

Soccer game regulation include the official game rules and the official rules for controlling a match. Game strategies are applied by the players and are based on game rules. They can improve the play strength of a team by utilising advantages player constellations on the field and thus increase its chances to shoot a goal.

Official Rules

Although the official rules of FIFA [FIFA 2001] define the general standards for Soccer, different types of leagues usually have their own specialised regulations and measures for the field. Depending or player type, league, and size of the field, such as men, women, robots, software agents, each league type possesses its different standards. Common to all leagues however is that the symmetries of the field lines are similar, which are referred here as the official lines (*Figure C.1*). Common are also most of the basic rules for the game and the referees. For our purposes, we have restricted our self to the official rules of the RoboCup Federation for the Simulation league [Chen et al.; 2001].

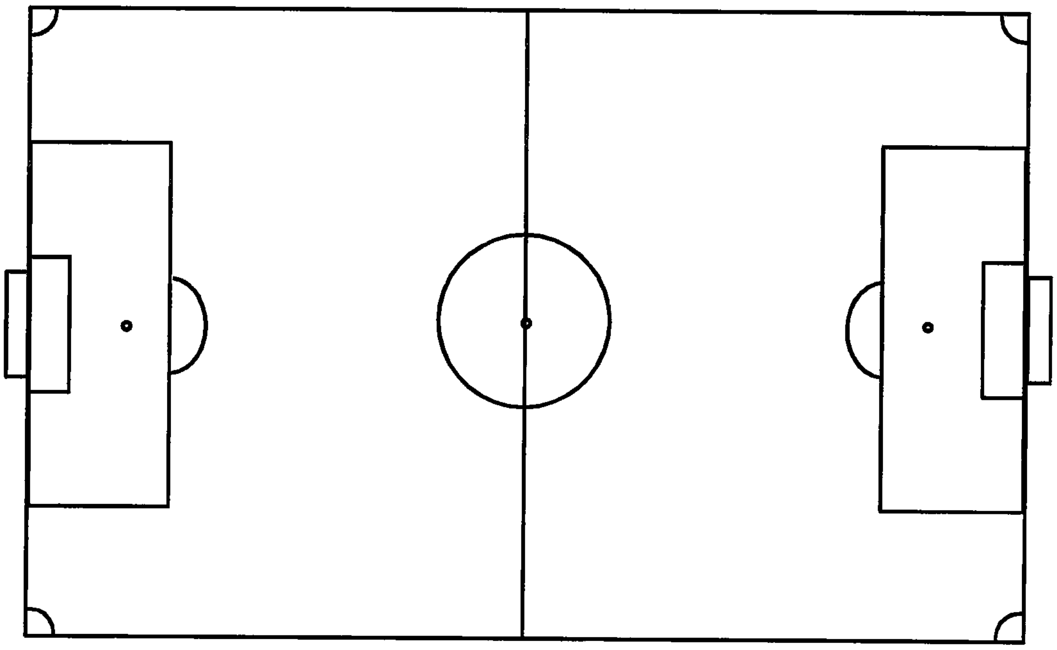


Figure C.1: Official Soccer Field Definition

Official Lines

The main field is surrounded by the four outer border lines, in which the players may hit the ball continuously (*Figure C.1*). The small four corner areas are located at the corners of this field. The goal area is located in front of each goal and marked by small boxes inside the field. The penalty areas are marked with the greater boxes that incapsulate the goal areas. The penalty arcs are situated immediately in front of the penalty areas. The cycle in the middle of the field is the kick–off area. The small boxes behind the two goal areas and outside the field are depicted here only to show the breadth of a goal.

APPENDIX D

ABSTRACTION LEVELS OF THE SOCCER OBJECTS

The two major concepts of Soccer are the game object and the rules. Principally, the rules are applied on the objects, in order to move them inside the field. One can observe that the objects actually move at different levels (*Figure D.1*). Where each higher level movement recursively consists of multiple lower level movements.

- Physical rules: Physical objects move according to physical rules. Human soccer game is affected by the physical rules weight, volume, location, gravity, momentum, friction, and acceleration. Current Soccer simulation server versions [Chen et al.; 2001] have implemented the location, gravity, momentum, and acceleration concepts for most types of possible game movements.
- Game rules: Official Soccer game objects move according some game rules. All game rules that require a movement must comply to physical rules. Such as, turn of a player with a momentum factor, accelerate with a momentum factor, sensor restrictions for hearing and seeing, or resource restrictions in form of stamina.
- Individual rules: Individuals move according the game rules. All individual rules of a single player, known in SoccerTeam as primitive game strategies, should comply with the game rules, in order to be successful. For instance, a short pass, a through in, or a corner kick.

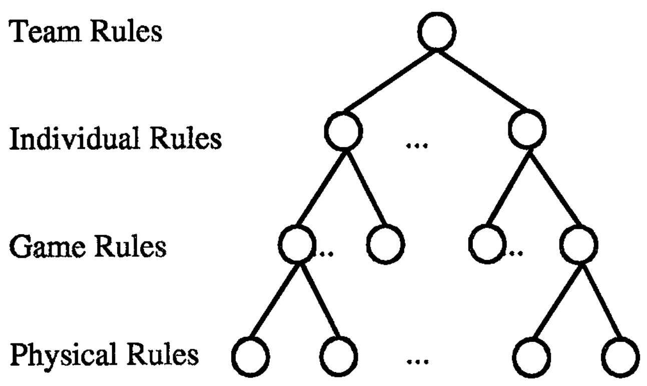


Figure D.1: Object Movements Performed At Different Abstraction Levels

- Team rules: A team moves according individual rules, since a team move consists of the movements of its individuals. All team rules, known in SoccerTeam as combined game strategies, usually involve several players and consist of several steps. Where each player performs a step simultaneously with the others, by applying some individual rules.

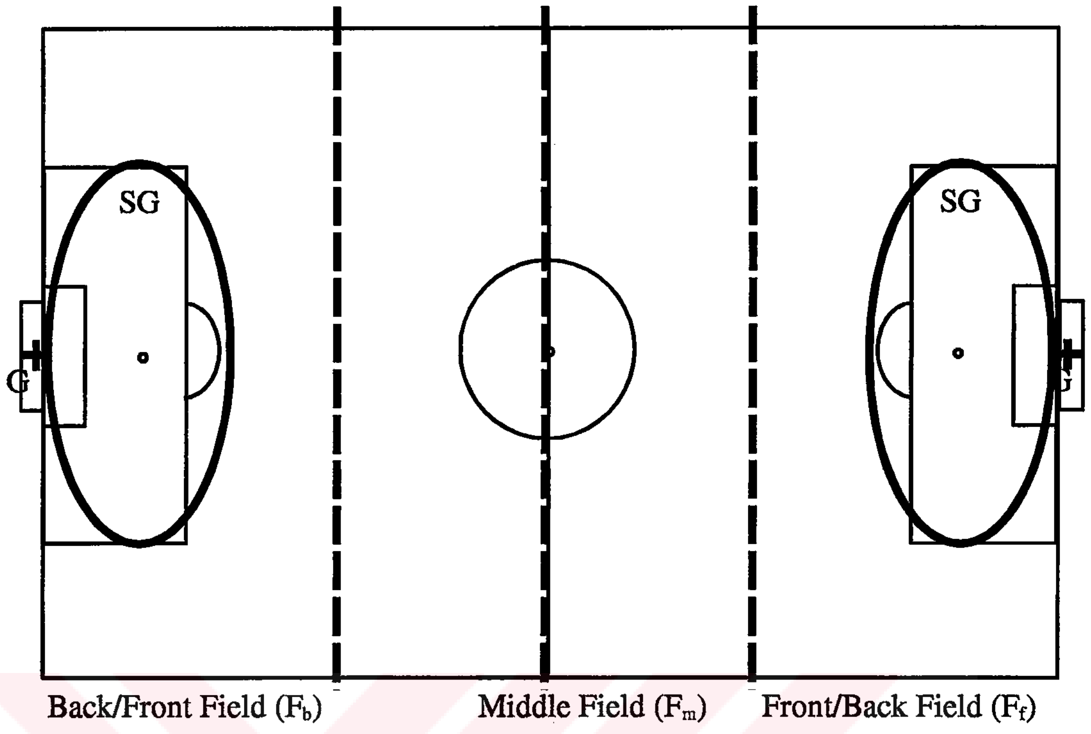
APPENDIX E
SOCCERTEAM GAME STRATEGIES: STATIC

In this section, we describe the strategies that do not change during a match, and thus remain constant in the planning process.

Strategic Areas

In human soccer game one can observe that attacking teams usually follow some common strategies that depend on specific field locations, such as major acting areas, major attack direction, and shoot areas (*Figure E.1*).

- Major acting areas: Back field, middle field, and front field are the major acting areas on the Soccer field. Before each match the coach decides which players and how many players to locate in these areas, which we call here team distribution.
- Focus area: For a team to implement its strategies successfully, its members should be located strategically near to the ball. To represent this information, we define a circle around the location of the ball, in which most of the members will be positioned most of the time of a match. The goalkeeper however, may be outside this area, but will still focus the ball.
- Major attack direction: The major attack direction is clearly towards the opponent's goal. However, depending on the current player constellation, it may be sometimes suitable to pass the ball backwards, in order to continue the attack from a more advantages position.



▪▪▪▪▪ $F_b/F_m/F_f$ ⬭Shoot Area (SG) ✚Goal (G)

Figure E.1: Strategic Field Lines And Areas

- Shoot/defence areas (SG): These are the areas in which statistically most of the goals are shoot. They incapsulate the official goal areas. Each team's shoot area is located immediately in front of the opponents goal. Each team's defence area is at the home side.

Learn Granularity

The selection of the grid line granularity of the field requires a compromise to be made between faster learning of location insensitive rules and slower learning of location sensitive rules. As result of our experiments, we have defined 1 meter grid lines, which divide the field into 60x100 squares. Thus, a generated plan will have player constellations with this distance granularity.

APPENDIX F

SOCCERTEAM GAME STRATEGIES: INDIVIDUAL

In a specific game constellation each team member may apply one out of a set of allowed possible, and/or appropriate individual strategies that we call primitive strategies. The intention of the following dynamic game strategies is to move the ball and/or the team member from its current location to a more advantages location. We have considered here only two–dimensional strategies, but no three–dimensional once, such as header, shoulder, jumping, kicking the ball high or over an opponent, etc.

Basic Game Strategies

Depending on specific conditions, some basic strategies may be applied by all 22 team members. Some of them may be interpreted different in different situation, which are discussed in the next sections.

(1)Focus ball: This strategy is not only always allowed, possible, and appropriate, it is a necessity for each team member for all the time of a match. It gives the team members the advantage to recognise immediately any change of the current game situation.

(2)Move: This strategy is simply to move the team member to a new location, which is allowed, possible, and appropriate for all team members in any situation. However, following semantics need to be considered:

- Self pass: If the team member has the ball, then also the ball is moved to the new location.

- Wait: The move distance may be zero. To fill–up some time by not moving can be a tricky strategy. It has important advantages, such as saving the team member's stamina, waiting for a better match constellation, or initiating the covering to disassociate.

(3)Pass: If a team member is sufficiently close to a ball, then it may pass the ball to a suitable team member or shoot it to an advantages location.

Goalkeeper Strategies

Additionally to the capabilities of the other players, the goalkeeper can catch the ball with the hands. Since this capability is restricted to the penalty area, a goalkeeper can play more effectively inside this area. Usually, a goalkeeper does not shoot a goal or is directly involved in attack or defence strategies at the front. The goalkeeper becomes a major figure in those strategies, which are applied near the home goal. Accordingly, following strategies are possible:

(4)Carry ball: The goalkeeper may move with the ball up to the penalty line, by kicking or carrying it. This situation gives that team usually the advantage to start a new attack. However, carrying the ball is allowed usually only for several seconds.

(5)Catch ball: Catching the ball with the hands is allowed only within the penalty area. It enables the goalkeeper to hold the ball from moving into the goal.

Primitive Offensive Game Strategies

The topmost goal of a team is to win the game. For this purpose, offensive strategies enable it to shoot goals. Some important strategies are:

(6)Shoot goal: If a team member has the ball and is within the shoot area, then it may shoot the ball towards the opponent goal. For the offensive goalkeeper this strategy is usually inappropriate and for the defensive goalkeeper not possible.

(7)Move free: Usually, each offensive team member will be covered by a defensive one. Each offensive player will try to free itself from the covering by moving away, in order to enable a pass to itself.

Primitive Defensive Game Strategies

On the other hand, the primarily objective of the defensive team is to make it difficult for the offensive team to shoot a goal and to get back the ball. This intention is expressed with the following strategies:

(8)Get ball: Usually some closest defensive players attack the offensive team member, in order to get the ball. This strategy is always allowed, possible, and appropriate, even in the case when the goalkeeper has the ball.

(9)Cover closest: The most effective strategy to prevent the offensive team to establish a strategy is to cover all its members. However, it is not always appropriate, for instance to cover those which are too far from the attacked goal.

Game Starting Or Continuing Primitive Strategies

A Soccer game may be interrupted by various types of faults. All different starting or continuing rules, such as kick–off, free–kick, corner kick, touch, penalty are principally similar in the sense that they can be thought as special game constellations, if we considered them as special snapshots that could occur any time in normal play randomly. Therefore, for our purpose, which is to construct generic plans for random constellations, it suffices to neglect them for the planning process.

Combined Game Strategies

Principally, any combination of the above discussed individual strategies may be combined to an individual plan. Currently, for each random match constellation, an individual plan consists of up to three primitive game strategies. The interpretation of each primitive game strategy is as follows:

- 0. step: Initial random game constellation is generated, therefore all 22 team members and the ball may not be located at strategic positions.

- 1. step: All 22 team members will be located now at strategic positions relative to the ball. However, the applicability of this pair of offensive/defensive team strategy within a specific SoccerTeam match is relative low, since the assumed random constellation, which had led to this strategy, will probably never occur.
- 2. step: All 22 team members will be located again at strategic positions relative to the ball. Since in the previous constellation all team members will definitely be at strategic positions and therefore this situation is more realistic, this strategy is significant more significant.
- 3. step: The same like for the previous step applies here. However, the interpretation of this strategy is twofold. Like in chess, each step looked ahead can improve the decision of the previous step applies here, too, but since, in contrast to chess, all 22 figures may move simultaneously, each further step looked ahead becomes much less probable. Nonetheless, exploring the third strategy brings the advantage that more complicated solutions can be found. For instance, depending on the current field constellation, it may be sometimes suitable to pass the ball backward, in order to continue the attack from a more advantages position.

APPENDIX G

SOCCERTEAM GAME STRATEGIES: CO–OPERATIVE

Actually only the individual strategy *pass* directly implies co–operation. However, to apply even this strategy requires a suitable team constellation, hence it alone is not sufficient to establish co–operation. In order to generate suitable constellations for a team and to enable an advantages application of the *pass* strategy, we now propose some strategies for co–operation. These strategies can also be considered as meta strategies, since they assume the existence of individual strategies, such as the *pass* strategy, but do not explicitly specify them.

Team Strategy

Some offensive team strategies are:

- Pass distance: The *pass* strategy can be applied successfully only if there is at least one sufficiently close team member. Preferably several team members should be in pass distance, since alternative attack situations may irritate the defensive team and thus give the offensive team an advantage.
- Density around ball: If most of the team members are located around the ball, preferably some of them in pass distance, but most of them not too close and not too wide, then the ball could be passed among the players quickly, in order to gain more advantages constellations. The goalkeeper however, will usually not directly be involved in this strategy.

- Ball–goal located: The most advantages passes are made to those players, which are closer to the opponent's goal than the current ball location. Therefore, several team members should be positioned between the ball and the opponent's goal.

The definition of these strategies may appear similar, since they are usually applied conjunctively. But this is not the case, because each of them could also be applied independently from others relative successfully.

Following are important defensive team strategies:

- Cover goal: If the ball is relative close to the home goal, then most of the defensive players should cover it by covering the closest offensive players and/or by building a wall. This strategy is multi–objective. Depending on a specific constellation, the defensive players may be required to comply to both objectives in all three possible combinations.
- Cover back field: This strategy is though to become effective when the offensive team is about to establish an attack strategy. The objective here is to move most players into the home back field before the ball passes the middle field line. This strategy is multi–objective with respect to the individual strategy clover closest, since in a given constellation not all team members should cover the back field, but some should cover those offensive players behind the ball in the middle field.

Coach's Game Strategies

In real soccer games, prior to a match and if necessary sometimes in brakes, each coach sketches major strategies to be considered in all other strategies. These strategies remain relative stable and can be disregarded only, when temporarily another strategy appears to be more promising. For instance, they may be determined by the coach dynamically throughout a match, in order to adapt his team to the opponent's capabilities and the current match situation. We represent most of the coach strategies with a single generic strategy called teamDistribution. Its special cases will explain its application varieties.

The distribution of the team over the major acting areas back field (F_b), middle field (F_m), and front field (F_f) is represented in form of the triple (F_b, F_m, F_f).

- Defensive team distribution: For example (F_b, F_m, F_f) = (8, 2, 0).
- Offensive team distribution: For example (F_b, F_m, F_f) = (1, 2, 7).
- Initial team constellation for each game starting or continuing strategy.
- Strategies for the duration of each half and/or full time of a match, depending on expected skills for both teams and their current situation; if related information is available apriori.

A sample scenarios may demonstrate the significance of teamDistribution:

- Back field attack scenario: If the ball is in the back field and one of the own team members gets the ball, then most of the team members should move towards the opponent goal.

  ➩ Goal: Most of the team members should be distributed over the middle and front fields, in order to support the attacking team member.

  ➩ Rule: IF ball is in F_b AND O_b THEN (F_b, F_m, F_f) = (F_b—, F_m++, F_f++)

Where, O_b is the offensive player with the ball, — indicates a decrease of the value and ++ an increase.

All cases of teamDistribution may further be relativised, so that the players do not actually need to be located in the respective acting areas, but within the crowd around the ball relative in the back, in the middle, and in the front.

APPENDIX H

ALGORITHMIC COMPLEXITY OF CEVOP

A standard approach to determine the increase of the running time of an algorithm with the size of the input data is the asymptotic notation [Brassard et al.; 1988], [Cormen et al.; 1990]. We need to discuss this aspect of the CEVOP algorithm, since it generates in one run only one pair offensive/defensive plan. However, usually multiple such plans will be required and therefore the algorithm must be iterated multiple times. Here we calculate the complexity of the inner loop of the CEVOP algorithm, which consists solely of the co–evolutionary algorithm, according to the asymptotic notation in general and for the SoccerTeam case.

Complexity Of The Generic CEVOP Algorithm

The major nested loops in CEVOP run for the number of populations, chromosomes, and genes. Accordingly, the asymptotic lower, upper, and exact bounds are, respectively:

- $\Omega(n^2)$, if the fitness is reached in the first generation
- $\Theta(n^3)$, even if the algorithm converges, the exact number of generations is usually unknown apriori
- $O(n^3)$, for the worst case of the algorithm

This is the general complexity of the algorithm without domain–specific information However, without any chromosome evaluation criteria, the algorithm is impractical. On the other hand, the complexity can increase with any further nested loop of a criterion.

Complexity Of The CEVOP Algorithm Of SoccerTeam

In an extension of the CEVOP algorithm, we have inserted several criteria, which introduce the algorithm some further nested loops. wherefore the complexity of the whole algorithm increases. The criteria and their complexity is as follows:

- moveInField: tm * cS $\rightarrow \Omega(11 * 3) = \Omega(1)$; $\rightarrow \Theta(n^2)$; $\rightarrow O(n^2)$
- densityAroundBall: tm * cS$\rightarrow \Omega(11 * 3) = \Omega(1)$;$\rightarrow \Theta(n^2)$; $\rightarrow O(n^2)$
- teamGoalDistanceRelative: tm * cS$\rightarrow \Omega(11 * 3) = \Omega(1)$;$\rightarrow \Theta(n^2)$;$\rightarrow O(n^2)$
- coverClosest: cS * (tm + tm * tm + tm * tm * tm * tm) = cS * $(tm + tm^2 + tm^4)$

  $\rightarrow \Omega(3 * (11 + 11^2 + 11^4)) = \Omega(1)$; $\rightarrow O(n * (n + n^2 + n^4)) = O(n^5)$;

  $\rightarrow \Theta(n^3)$, if the distances are always already sorted
- focusBall: 2 * cS $\rightarrow \Omega(2 * 3) = \Omega(1)$; $\rightarrow \Theta(n)$; $O(n)$

Where the acronyms have following meaning:

- cS: number combined strategies, which is currently set to 3; but can be increased.
- tm: number team members, which can be either 1, 10, or 11, depending on the specific criterion; which cannot be increased.

With the further assumptions that the number of populations and chromosomes may be increased, we get the following complexity of the CEVOP algorithm for SoccerTeam:

$\Omega(n^2) * \Omega(1) = \Omega(n^2)$; $\Theta(n^3) * \Theta(n^3) = \Theta(n^6)$; $O(n^3) * O(n^5) = O(n^8)$

Introducing further nested loops however, would increase the asymptotic upper bound accordingly. Therefore, potential nested loops of a criterion should be designed carefully. On the other hand, embedding the co–evolutionary algorithm into a further loop for the purpose to generate another plan in each iteration, will increase each of the above calculated complexity values by one dimension.

APPENDIX I

CONVERGENCE BEHAVIOUR OF A CEVOP RUN

In various experimental runs we have tested different parameter value combinations, ranging from 40% to 0% cross–over, 10% to 1% mutation, different random value generator seeds, and different combinations of the chromosome evaluation criteria. Relative good results were found for the following parameters:

- Chromosomes in each population: 100
- Cross–over rate: 30%
- Mutation rate: 5%

For these parameter settings, we have selected one representative case for the random value generator seed 810520 and the below chromosome evaluation criteria. These criteria represent most of the discussed individual and co–operative Soccer game strategies.

- Offensive goalkeeper: moveInFiled, focusBall
- Offensive players: moveInField, distanceTeamGoal, densityAroundBall
- Defensive goalkeeper: moveInFiled, focusBall
- Defensive player: moveInField, coverClosest

Although the evaluation criteria do not require each other, most of them depend on the value of the other. This can increase the number of generations. For instance, focusBall aims at positioning the goal keeper in relative distance to the ball. Therefore, the offensive players, including the ball, usually converge first, before the others converge.

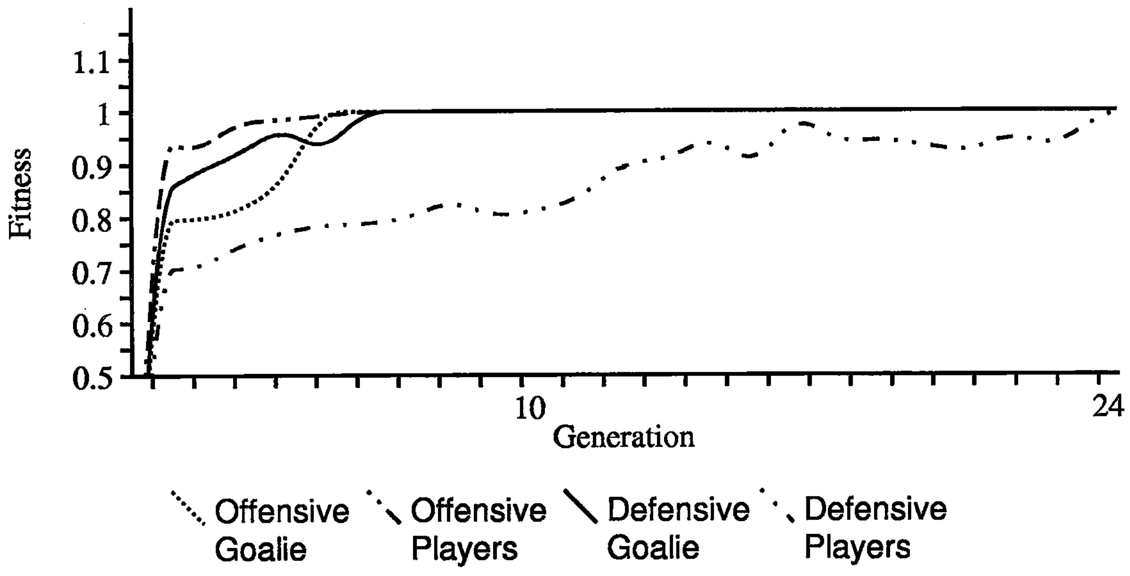


Figure I.1: Fitness Versus Generation Of Each Population

On the other hand, slightly changes if the ball location can cause the dependent criteria to evaluate lower values for the other players, for instance densityAroundBall. In this sense, at the end of this dependency relationships are the criteria for the defensive players. After all offensive players have been positioned, the defensive players are located relative to them. Therefore, the defensive players converge the latest.

coverClosest is implemented as an optimisation problem by itself and therefore is more complex (*Appendix H*) than the others. Hence the defensive players population requires relative more generations to converge to the desired fitness value, although the offensive players had already converged (*Figure I.1*).

The average fitness of the four populations reaches 1 at generation 24 (*Figure I.2*). In most of the test cases that we have made, the algorithm converged within 20–30 generations in the average.

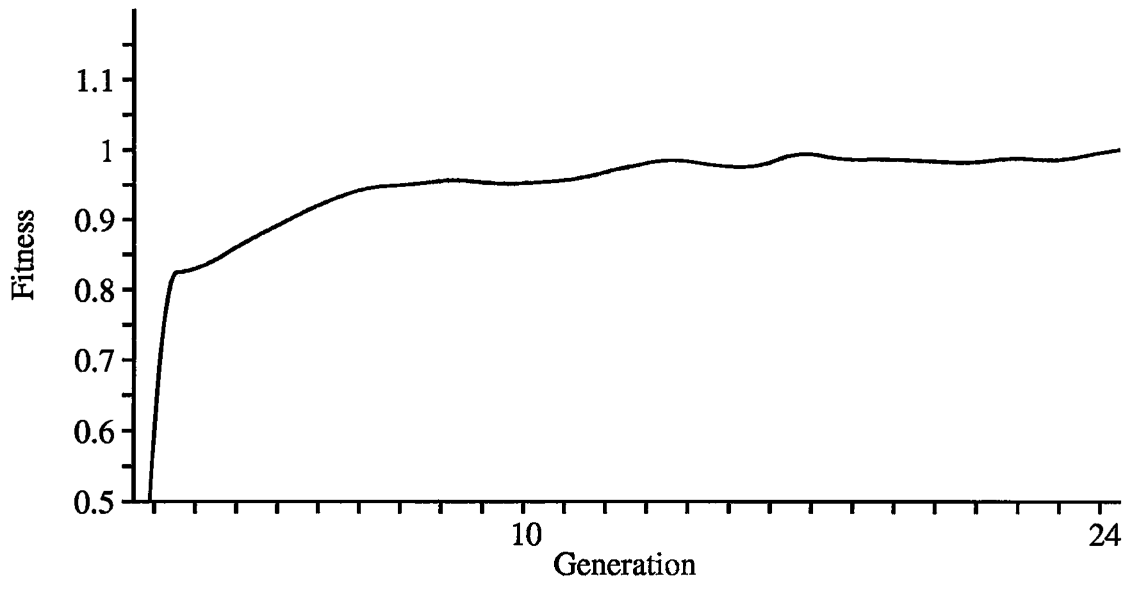


Figure I.2: Average Fitness Versus Generation Over All Populations

APPENDIX J

CLASS DIAGRAM OF THE CEVOP SIMULATOR

The learning component of the SoccerTeam prototype is an application independent form the SoccerTeam simulator client. Usually it is run several times to generate several team plans for both, the offensive and defensive team. The result of one run is principally one offensive/defensive pair of team plans.

Below is the object–oriented specification of the software design of the CEVOP component. It is written entirely in C++ [Ellis et al.; 1990], [Kernighan et al.; 1983] under Linux [Kaufman et al.; 1995]. First, each class, of the class diagramme (*Figure J.1*), is described separately, thereafter the major control flow of a programme run is traced.

SoccerTeamRule

This is the entry point of the programme where the main function is located. Therefore, this part is actually not a class. It checks the inline–passes parameters, creates an instance of CEVOP, and forwards it the parameters. A loop is provided here, in which CEVOP may be called several times, in order to generate several pairs of offensive/defensive team rules.

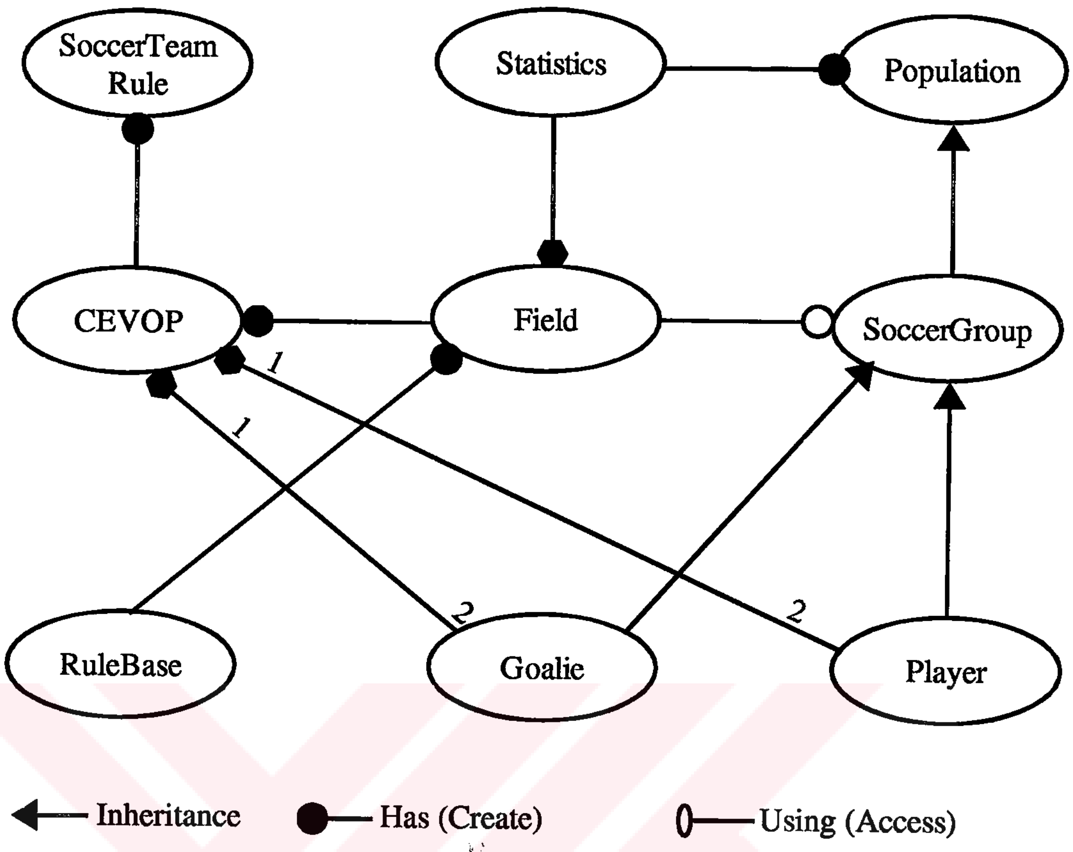


Figure J.1: Class Diagram And Object Cardinality Of The CEVOP Simulator

CEVOP

This class first creates the required instances of the classes Field and Goalie. It implements the major loop and the termination condition of the co–evolutionary algorithm. In each generation the genetic operations of the populations are called. After the loop has terminated, the related calles to output the final result are placed here. The global view of the co–evolutionary algorithm that is necessary to call related methods for evaluating required co–operation and/or competition structures is implemented here.

Statistics

This class utilises the standard random value generator to produce random values in the desired ranges.

Population

A generic genetic algorithm is implemented with this class. It provides the basic population and chromosome operations, such as cross–over, mutation, and fitness selection. Some abstract methods enable to specialise the operations.

SoccerGroup

This class represents abstract Soccer players and their individual strategies as a genetic algorithm population. Genetic algorithm related information is inherited from the population class and specialised for the Soccer domain. This is done by initialising the chromosome value ranges for cross–over and mutation. Further, in each generation of the genetic algorithm, related methods are provided to call to apply and evaluate each chromosome's genes. Some abstract methods are used to enable the implementation of special Soccer player types.

Goalie

The goalkeeper is a special Soccer player type, which is implemented in this class. It inherits the generic Soccer player type behaviour and extends it with goalkeeper related information, such as individual Soccer strategies that can be applied only by a goalkeeper.

Player

The player is a special Soccer player type, which inherits the generic Soccer player type and extends it with player specific behaviour. Player strategies that are usually not applied by a goalkeeper, such as attacking and shooting a goal, are implemented in this class.

Field

The class Filed represents a Soccer match consisting of a field and the playing teams. First, an initial random game constellation is generated, which represents a snapshot of a match. Thereafter, in each generation, the constellation can be modified according the related chromosome quadruples. The chromosome evaluation criteria, which represent co–operation and competition structures of the teams, are implemented in this class. Since the criteria are independent from each other, they can be called in any order. Some methods are provided to restore the initial field constellation at the begin of each genetic algorithm generation. Some further functionality is implemented to save the current field constellation in a rule base.

RuleBase

This class provides some functionality to store the given data in a permanent file. Some methods allow to redirect the output to different files, whenever this is desired.

Programme Control Flow

The purpose of the CEVOP component is to generate pairs of team plans for an initial random field constellation. The input data to this process is the number of plans to be generated and a seed value for the random value generator. The flow of control starts with these arguments in SoccerTeamRule and iterates within the CEVOP loop. Inside one loop the control is given to SoccerGroup and Field, depending on the desired execution sequence. In sub–sequent calles, control flows from SoccerGroup and Field to Statistics and RuleBase.

The initial control flow starts with the following Linux shell command:

```
SoccerTeamRule <Rules#> <randomGeneratorSeed>
```

APPENDIX K

SOFTWARE DESIGN OF THE SOCCERTEAM PROTOTYPE

According the objective of the thesis, our goal was to show the feasibility of the proposed planning approach. The implementation of SoccerTeam as a client to the RoboCup Soccer simulation server was secondary. The following specification represents the software design of the SoccerTeam prototype from an object–oriented point of view (*Figure K.1*) to be used in a future implementation. In this first development stage, we have restricted our selfs to the implementation solely of the functionality for displaying the planning results within the official RoboCup Soccer server monitor. This functionality is implemented in the SoccerTeam class.

SoccerTeam

The SoccerTeam class provides a few methods to read a given team plan from a permanent file and to send the related field location co–ordinates to the RoboCup Soccer simulation server. Each instance of this class represents one team member.

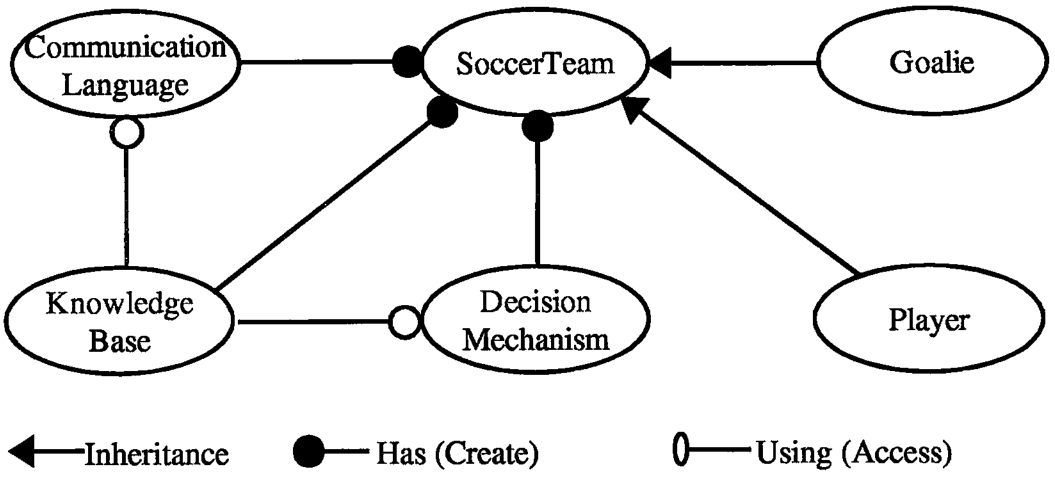


Figure K.1: Class Diagram Of The SoccerTeam Software

Programme Control Flow

The control flows from the main function of the socket–connected test client of the RoboCup Soccer simulation server to the related methods of a SoccerTeam instance.

The initial control flow starts with the following Linux shell command:

```
SoccerTeam <TeamMember#> <offensiveDefensiveIndicator>
    <planStep#>
```

APPENDIX L

SAMPLE OUTPUT OF A CEVOP RUN

The purpose of this discussion is to show the significance of the chromosome evaluation criteria of the co–evolutionary algorithm. Where each criterion represents a Soccer game strategy. The output of one run of the CEVOP programme is one pair offensive/defensive plan. The RoboCup Soccer server monitor is used by SoccerTeam only to display one steps of a team plan. The three steps of a plan are discussed separately for both, the offensive and defensive team, with respect to the criteria. The offensive team's home is always on the left side, the defensive team's home on the right. The player's numbers have no meaning in this discussion, since the start sequence of the client processes does not always mach with the server registration sequence .

Parameters Of The Sample CEVOP Run

The sample run was started with the following parameters, whereby some of them are coded within the programme and some are passed as command line arguments:

- Random value generator seed: 810520
- Chromosomes in each population: 100
- Cross–over rate: 30%
- Mutation rate: 5%

Interpretation Of The CEVOP Output

In the initial random constellation the players are positioned anywhere inside the field and the goalkeepers anywhere inside their goal area. (*Figure L.1*).

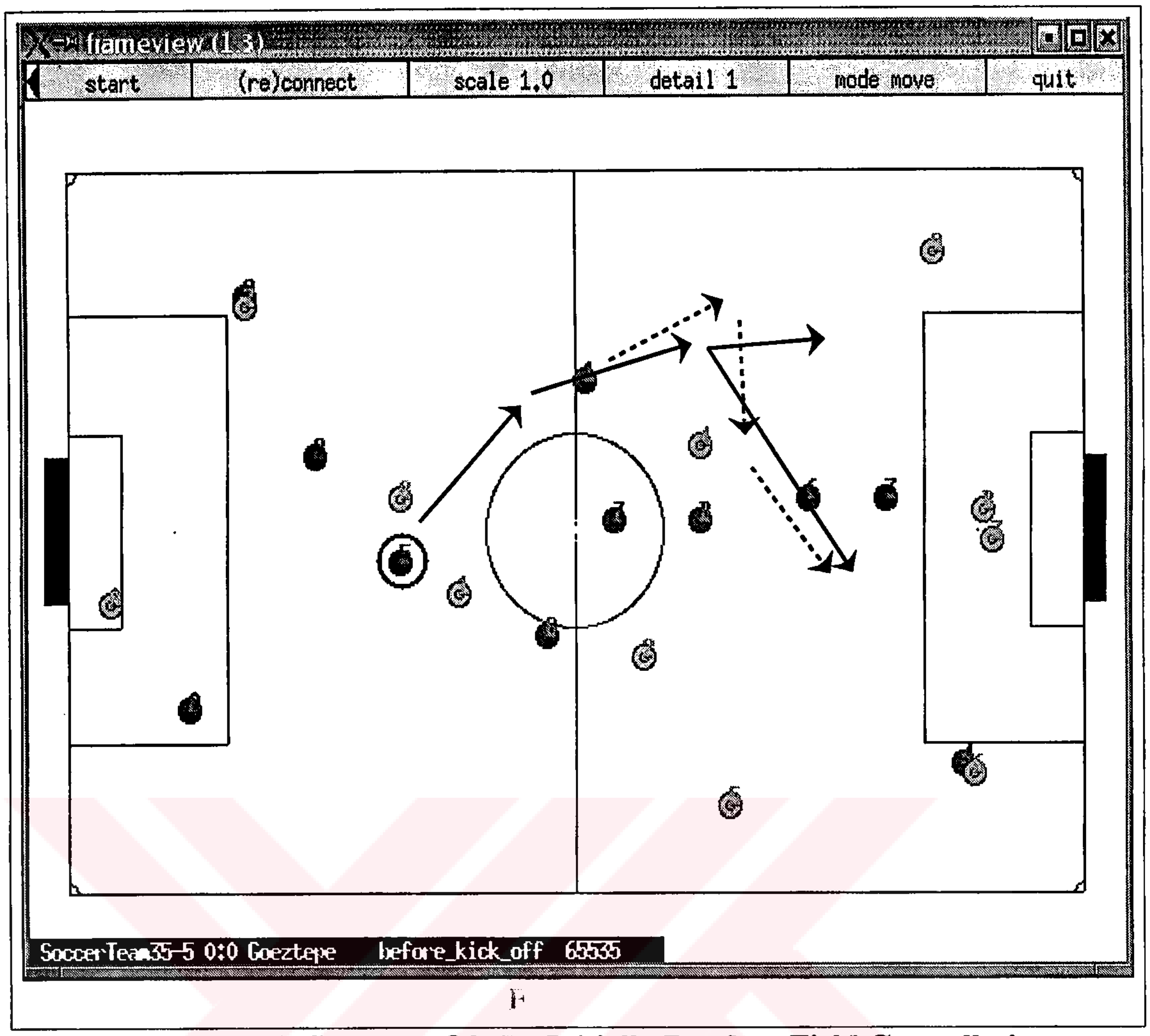


Figure L.1: Snapshot Of The Initially Random Field Constellation

In the following figures a cycle indicates the player with the ball, solid arrows show the movement of the ball and the related player, and dashed arrows show the movements of the player to which the ball is passed. The sample CEVOP process has generated following ball positioning and passing:

- 1. strategy: Player 2 makes a self pass
- 2. strategy: Player 2 makes a self pass
- 3. strategy: Player 2 passes to player 1

As already mentioned above, the player numbers of the CEVOP output do not match with those of the RobuCup Soccer simulation server !

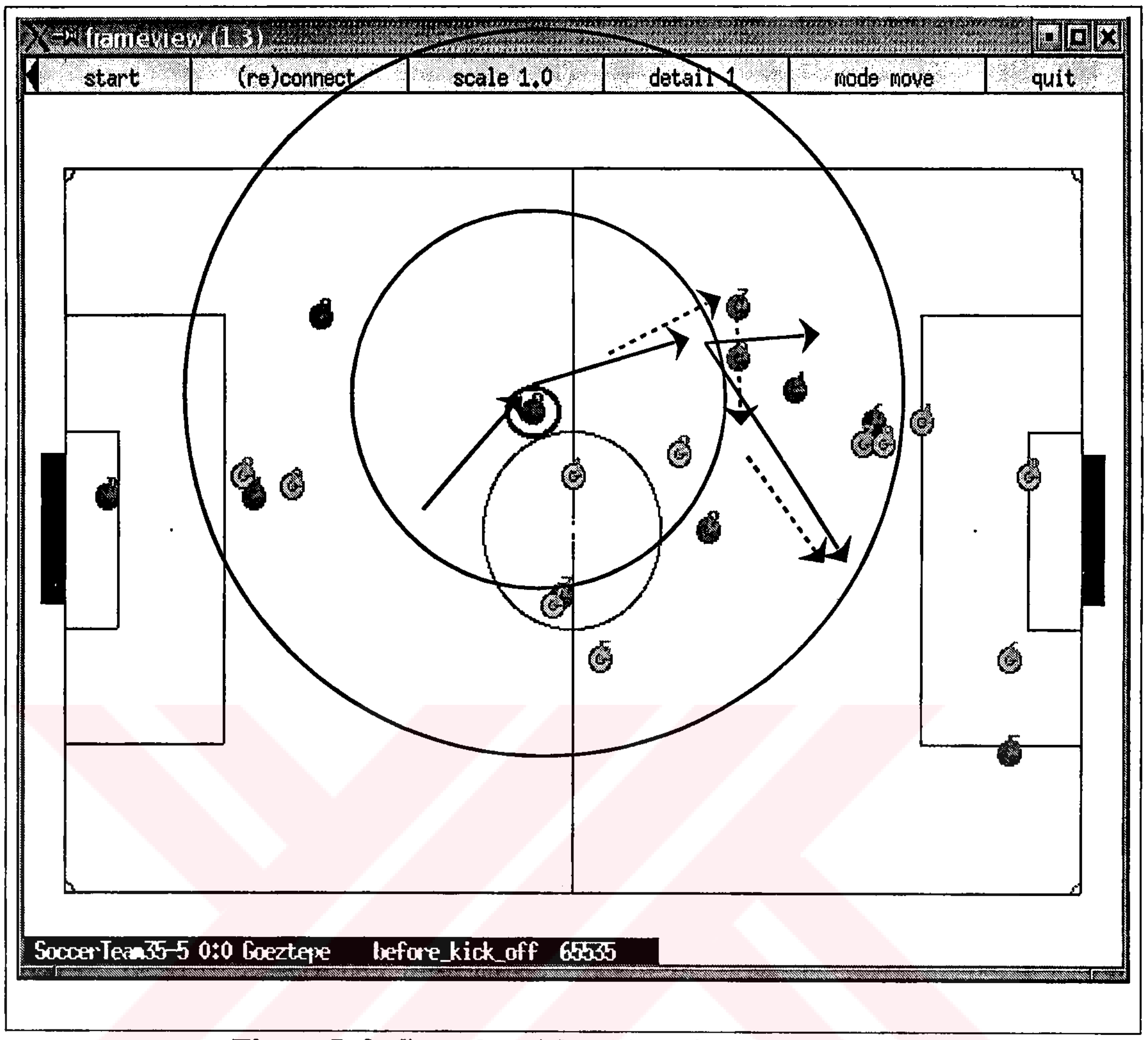


Figure L.2: Snapshot After The First Team Strategy

After the first team strategy has been applied to the initial constellation (*Figure L.2*), the interpretation is as follows:

- moveInField: Since all players are inside the field borders, this criterion is fulfilled.
- distanceTeamGoalRelative: The ball moves a total of around 45 meters towards the goal and therefore the required 30% of maximum 60 meter, accumulated for all three steps, is satisfied.
- densityAroundBall: Four players are required to be in a distance up to 20 meter and five players up to 40 meter. Since the accumulated distance of all players is calculated, this criterion is satisfied. The ranges are indicated in the figure with cycles around the ball.

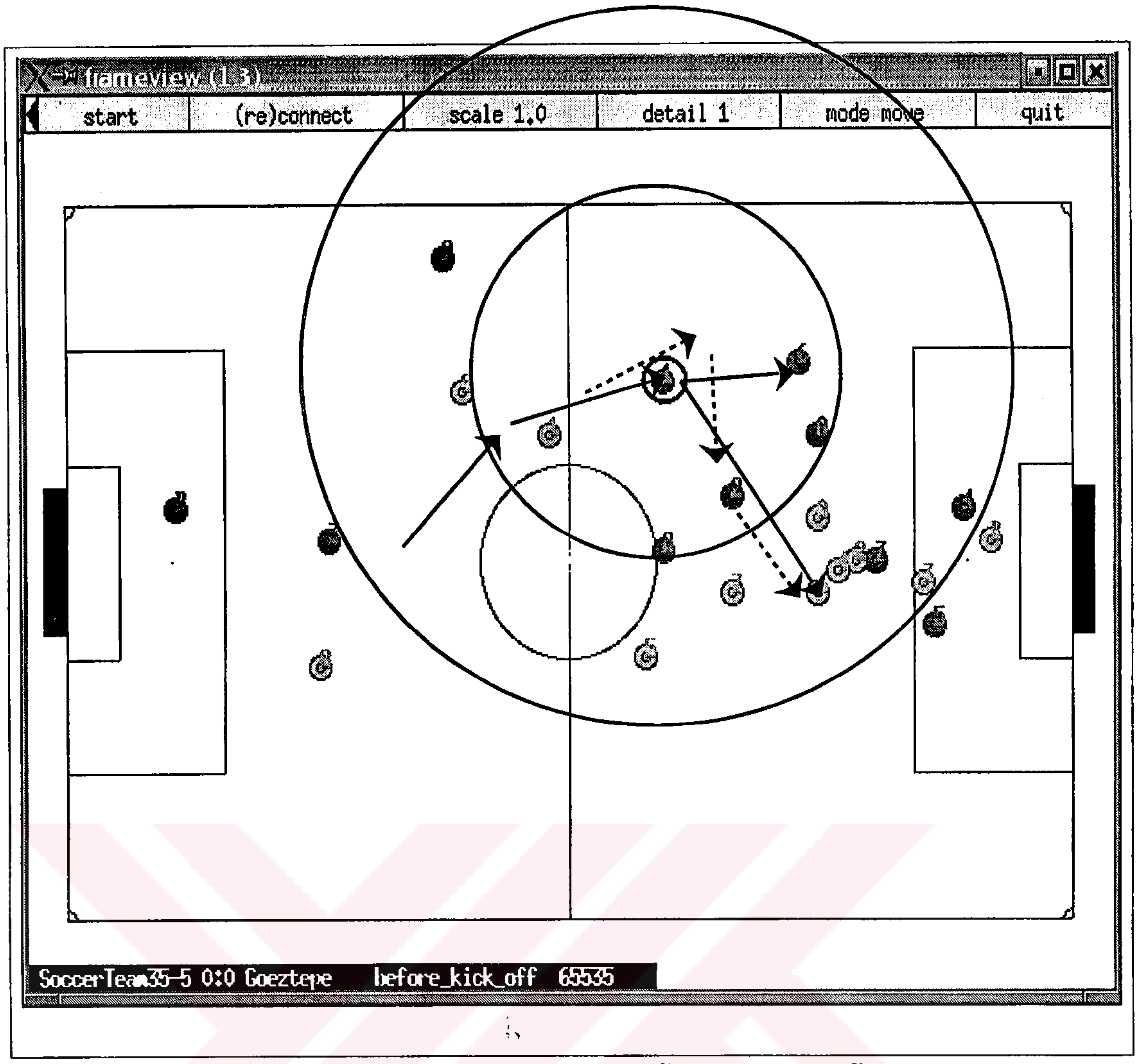
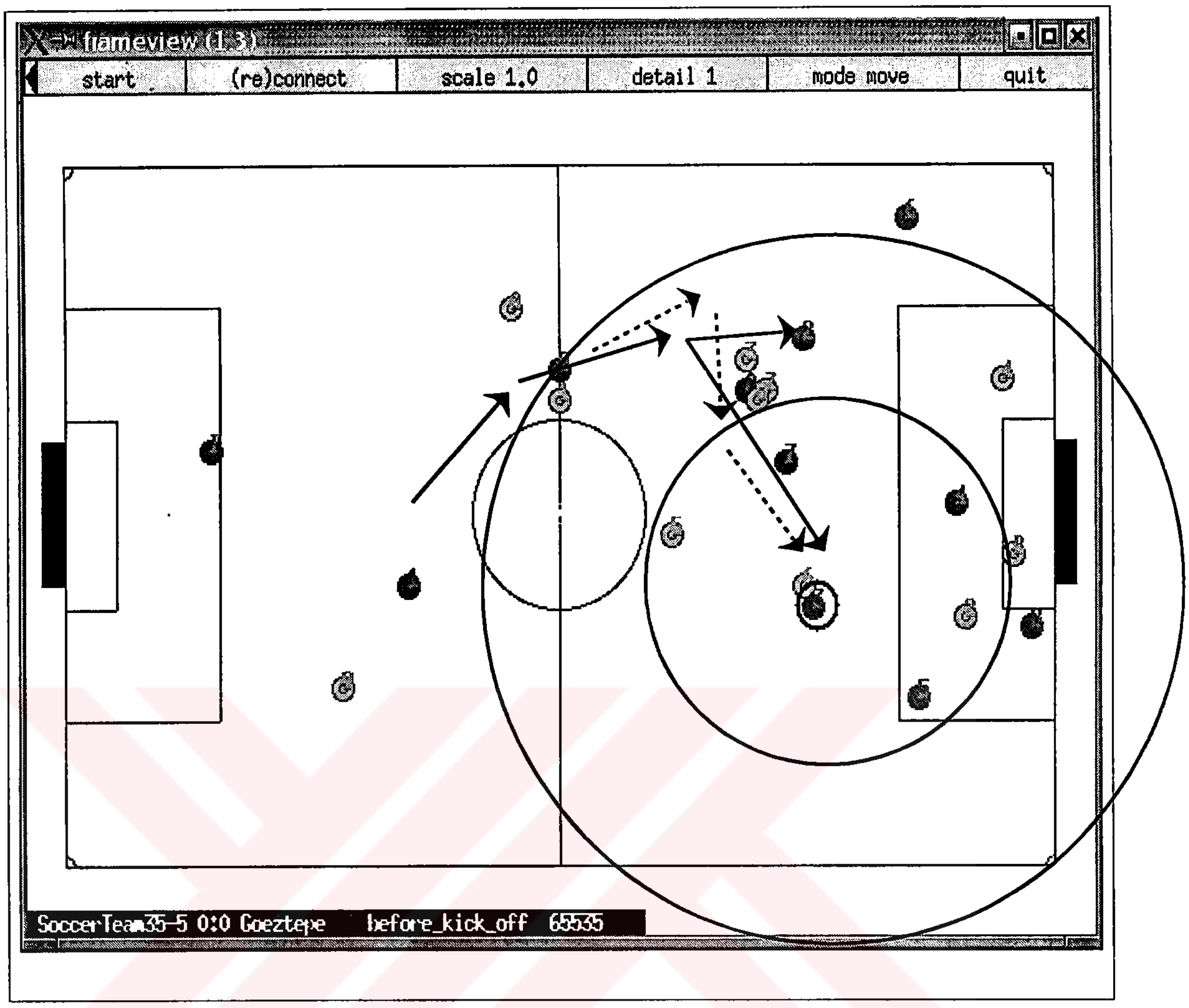


Figure L.3: Snapshot After The Second Team Strategy

- coverClosest: This criterion states that all defensive players should be in the average at most up to 5 meter from an offensive player, which is here fulfilled.
- focusBall: The goalkeepers are required to be located less than 16% on the abscissa and 10% on the ordinate of the distance of the ball from the goal. This criterion is also satisfied for both goalkeepers.

The above criteria are all fulfilled also for the remaining constellation, as can be seen in the figures (*Figure L.3*), (*Figure L.4*).



SoccerTeam35-5 0:0 Goeztepe before_kick_off 65535

Figure L.4: Snapshot After The Third Team Strategy