1918 TALLINNA TEHNIKAÜLIKOOL

INFOTEHNOLOOGIA TEADUSKOND

Automaatikainstituut

Reaalajasüsteemide õppetool

Johannes Ehala

ISP70LT

VAHENDATUD INTERAKTSIOONI KAUDU JUHITUD ILMNEV KÄITUMINE MULTI- AGENT SÜSTEEMIDES: KATSESÜSTEEM

Magistritöö

Juhendaja: Leo Mõtus
reaalajasüsteemid
professor

Tallinn 2012

Autorideklaratsioon

Olen koostanud antud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem esitatud kaitsmisele kusagil mujal.

Kuupäev:

Autor: Allkiri:

Eessõna

Magistritöö teema valikul andsid palju inspiratsiooni juhendaja Leo Mõtuse huvitavad loengud reaalaja tarkvaratehnikast ja agentsüsteemidest. Lõplik teema sai paika pandud arvestades Proaktiivtehnoloogiate teaduslabori uurimussuundasid ja käimasolevaid projekte ning autori soovi luua midagi praktilist.
Autor soovib tänada juhendaja Leo Mõtust hariva kriitika ja näpunäidete eest ning Proaktiivtehnoloogiate teaduslaborit igakülgse toe ja soosiva hoiaku eest töö kirjutamisel. Veel sooviks tänada oma pere, eelkõige isa Martin Ehalat, tagant utsitamast ja Kristine Roosi vaimu erksana hoidmise eest.

Vahendatud interaktsiooni kaudu juhitud ilmnev käitumine multi- agent süsteemides: katsesüsteem

Annotatsioon

Hajusad sardsüsteemid moodustasid juba eelmise sajandi lõpul valdava osa kasutusel olevatest arvutisüsteemidest ja tehnoloogia pideva arengu tulemusena on tänaseks jõutud autonoomsete kognitiivsete tehissüsteemide loomiseni. Selliste süsteemide käitumise ennustamine ja hoidmine lubatud kitsenduste piires on omaette probleem. Raskusi põhjustab komponentide osaline autonoomia, vahetu kokkupuude keskkonnaga ja selle tulemusena dünaamiliselt tekkivad interaktsioonid. Nende interaktsioonide loomine ja haldamine ei ole enam kirjeldatav klassikaliste algoritmiliste mudelitega. Selleks et tulla toime lisandunud keerukusega on kasutusele võetud agendi mõiste. Agentidel põhinevad mudelid keskenduvad interaktsioonide kirjeldamisele jättes komponentide sisu detailselt lahti kirjutamata.
Magistritöö analüüsib üldisemalt agenttehnoloogiatega seotud teooriat ja toob välja tähtsamad agentide omadused ning levinumad definitsioonid. Eraldi tähelepanu all on agentide vaheline suhtlus, vahendatud interaktsioon ning ilmnev käitumine ja selle kasulikkus multi-agentsüsteemides. Kogutud teoreetilistele teadmistele tuginedes on töös loodud targa tolmu kübemetel põhinev katsesüsteem. Katsesüsteemi agente ja vahendatud interaktsiooni realiseeriv tarkvara tugineb targa tolmu tehnoloogias levinud TinyOS operatsioonisüsteemil ja on jagatud mitme eraldiseisva mooduli vahel. Katse eesmärk on demonstreerida agentsüsteemis esinevat ilmnevat käitumist ja iseorganiseerumist, mida saab osaliselt juhtida vahendatud interaktsiooni kaudu. Töös tehakse algust formaalse lõimearvutusel põhineva mudeli koostamisega katsesüsteemis esineva agentidevahelise suhtluse kirjeldamiseks.

Lõputöö on kirjutatud inglise keeles ja sisaldab teksti 84 leheküljel, 6 peatükki, 5 joonist ja 1 tabelit.

Mediated Interaction Directed Emergent Behaviour In Multi-Agent Systems: a Pilot System

Abstract

Since the end of the last century majority of computing applications have been embedded ubiquitous computing systems. With the advancement of technology the systems have grown more complex giving rise to pervasive computing and cognitive artificial systems. The behaviour of these systems is hard to predict and keep within permissible bounds. Difficulty rises from dynamic interactions formed between the autonomous components of the system during operation. Managing these interactions has proven to be quite a challenge. In order to study interaction a useful abstraction has been created in the form of agents. Agent models focus on describing interactions and omit a detailed description of the components themselves.

This thesis presents a generalised theoretical analysis of agent technologies focusing on agent characteristics, origin and definition. Separate attention is directed to interaction among agents especially mediated interaction, the resulting emergent behaviour and its additional value to multi-agent systems. The theoretical experience gathered is put to practice in the form of a simple multi-agent experiment using smart dust motes. TinyOS based software components are developed for the smart dust motes to represent agents and mediated interaction. It is shown that the system exhibits emergent behaviour and self-organisation and that this behaviour can be partially controlled by mediating agent interactions. Finally an attempt is made to formally describe the interactions of the system using a multi-stream computation based approach.

This thesis is written in English and contains 84 pages of text, 6 chapters, 5 figures and 1 table.

Contents

List of tables:

List of figures:

Glossary of terms and abbreviations

adaptivity	ability to adapt
ADC	analog to digital converter
AI	artificial intelligence
DAI	distributed artificial intelligence
first principle	fundamental axiom or postulate in a theory
flash	flash memory, non volatile memory
ID	identification
JADE	Java agent development environment
Laboratory	in this thesis the Laboratory for Proactive Technologies
MAS	multi-agent system
node	a single element in a network
PC	personal computer
RAM	random access memory
SA	situation awareness
stigmergy	a form of indirect communication among insects [47]
XML	Extensible Markup Language

1. Introduction

Since Alan Turing's paper introduced universal computing machines in 1936 and the first programmable electronic computers built during the Second World War, computer science and computer technology have seen remarkable progress. The 1950s and 1960s showed quick promise of intelligent computers presently assisting humans in all areas of life, but this was not to be. Creating computers that could think like humans ran into difficulties and computer technology took a different direction towards integrated computers and smart devices. This gave rise to interest in distributed computing systems, where many embedded computers work together on a common task. One of the earliest examples of such systems was the parking assistant for cars and other motor vehicles [35]. Work on the parking assistant started already in the 1970s but commercially ready applications appeared only during the last decade. These systems are to date still not autonomous (not to be confused with automatic) and operate under heavy safety restrictions usually with the occasional assistance of humans. This has turned out to be common practice with autonomous systems and is an example of the autonomy paradox [3]. The autonomy paradox holds that autonomous systems instead of easing the workload of humans tend to increase it in stead. The reason for this is that the more autonomous a system gets the more supervision and maintenance it needs.

A part of the unpredictability and uncontrollability of distributed autonomous systems stems from the dynamic structure of interactions between its components. The central role of interactions was noted already in the end of the 1970s by Milner [29] and demonstrated by Brooks [5] a few years later. Brooks' subsumption architecture showed how intelligent behaviour can emerge from the interaction of components with much simpler functionality. It was not until the 1990s that the importance of interactions was widely acknowledged and interactive computing [58] became the standard in modelling distributed systems. Some examples of interaction based models of computing include the Q-model [36], π-calculus [30] and persistent Turing machines [16]. The essence of interactive computing lies in emergent behaviour which provides distributed systems with additional value that components individually do not possess. This value emerges from the dynamic interactions of autonomous entities inconspicuously and is very hard to predict. Controlling or simply detecting emergent behaviour is therefore one of the main challenges of developing distributed computing systems that comprise autonomous components.

Another difficulty besides handling interactions is modelling complex autonomous behaviour. Interactive computing centres on interactions and devotes little energy to

describe the elements of the system. Often these components have to be extremely sophisticated and software intense (sometimes beyond what is technologically possible) to mange the tasks expected of them. To deal with this shortcoming the notion of a computing agent was introduced. An agent is a single computing entity, usually the least element in interactive computing models. Agents have several unique characteristics such as autonomy, context awareness, personal goals, self-sustainability, etc. that make them useful abstractions when precise descriptions are not needed or possible. They are divided between intelligent, context aware cognitive agents and simpler reactive agents [12]. Cognitive agents help model complicated cooperation scenarios where interactions involve complex negotiation and reasoning semantics. Such models are widely used in social sciences as well as artificial intelligence studies. Reactive agents are intended for models of distributed computing systems such as the parking assistant application described earlier. These agents do not necessarily exhibit complex behaviour and usually implement simpler communication routines.

Distributed artificial systems, multi-agent systems, embedded devices, autonomy, interactive computing, agents, emergent behaviour and many other related subjects are all research objectives at the Laboratory for Proactive Technologies (Laboratory). The Laboratory is a fairly young organisation but comprises a profound theoretical background and fair amount of practical experience in these areas of study. A part of the work done in the Laboratory focuses on practical distributed artificial systems of embedded computers. The areas of interest in these systems include interactions and emergent behaviour. A novel concept of middleware is promoted that can help manage dynamic interactions, create situation awareness and possibly discover emergent behaviour [33]. Practical models are being developed but no working systems exist yet. The technological asset of the Laboratory is among other things made up of smart dust motes. These small devices equipped with radio transceivers can be easily networked to create distributed computing systems and provide the means for practical testing.

1.2 Objectives of thesis

There are two main goals set for this thesis. The first goal is to provide a general overview of agent technology and interactive computing. The novelty and importance of these computing models is discussed. Also a brief introduction to emergent behaviour, multi-stream computing and the Q-model will be given. The second objective is to create a distributed computing system using smart dust motes found at Laboratory. This system

must demonstrate the characteristics of agents and interactive computing. It should also be compatible with novel research done in the Laboratory and be suitable for further development and usage.

The importance and usefulness of agents and multi-agent systems in computer science is evident. Agent based modelling and technology has evolved rapidly during the last two decades up to the point of becoming a research field in its own right. Separate courses are taught at universities and several textbooks devoted to multi-agent systems have been issued [12, 63, 44]. It is therefore prudent to discuss agent technology more thoroughly and analyse what makes agents so special in computer science. Naturally the whole field of research is too vast to be summarised so a selection about what to view has to be made. This thesis focuses on agents as building blocks for models of complex computing systems, their properties and usefulness. Multi-agent systems will only be discussed from the point of view of reactive agents and distributed artificial systems. MASs that compose cognitive agents and model complex social behaviour will not be reviewed. Special attention must be put on interactions, the centrepiece of interactive computing, and emergent behaviour. The analytic overview carried out in the first half of the thesis must serve as a complete theoretical basis for constructing the practical experiment. For this reason a short introduction to Q-models and multi-stream based computing is also necessary.

The second goal of the thesis is to build a distributed computing system using smart dust motes available in the Laboratory. Constructing such a system serves several purposes. First of all it is a practical exhibition of the theoretical knowledge learned about agents, autonomy, interactions and emergent behaviour. Secondly it will provide experience about designing distributed systems and may reveal still undiscovered obstacles of handling emergent behaviour. This is an especially important issue as there are currently no working systems in the Laboratory that possess the needed functionality to exhibit and monitor emergent behaviour. Finally the pilot system serves as a basis for future experiments and development of more software intense complex systems. In order to cater these objectives the system must meet certain requirements. It is desired that the components of the system possess some autonomy so that it would not be possible to predict their exact behaviour. At the same time these components must be open to communication and cooperation with other parts of the system. This hopefully will lead to emergent behaviour that should then be made visible in some way. The Laboratory is interested that software for the pilot system abides the standards of, and is compatible with, other systems developed at the

Laboratory. The final task is to formally describe the system in Q-model and stream-computing terms.

The outcome of this thesis is an incomplete but thorough overview of modern computing paradigms and challenges. Focus is centred on practical models and applications. For this purpose a distributed computing system is developed. An attempt is made to formally describe the system according to the Q-model and multi-stream computing concepts.

1.3 Outline of thesis

Chapter 2 starts off by introducing the notion of a computing agent. Section 2.1 discusses the origin of the term along with some of the reasons why a new concept to describe computing entities was needed. The numerous definitions of an agent are then summarised in the next section. Chapter 2 concludes with the analysis of various agent characteristics that are common to the different agent definitions. Among these features autonomy, situation awareness, self-x properties and communicativeness are reviewed in more detail. Subsection 2.3.4 explains the notion of proactivity and proposes a simple classification of agents that will be used throughout the thesis.

Chapter 3 is devoted to interactions and interactive computing. The importance of interactions in distributed computing systems is explained and examples of new interactive computing models are introduced. Section 3.1 analyses the different types of interactions with special attention put on mediated interactions. The next section sheds light on the notion of emergent behaviour and its central role in interactive computing. Section 3.3 takes a detour from interactions to discuss environments in multi-agent systems. Environments have a significant effect on distributed computing systems, on interactions in these systems and on emergent behaviour. Chapter 3 ends the theoretical part of the thesis with a brief insight to Q-models and multi-stream computing.

Chapter 4 outlines the reasons and requirements of an experimental distributed computing system. The necessity and use of this system is discussed and general requirements are listed. Section 4.2 describes the different smart dust motes available at the Laboratory for Proactive Technologies and briefly reviews the TinyOS operating system and nesC programming language used on the motes. The last section of chapter 4 proposes a structure of the software components that would be easily reconfigurable and compatible with software of other projects in the Laboratory.

An interesting distributed computing experiment is described in chapter 5 that involves system wide synchronisation. Similar smart dust mote based synchronisation experiments

are first reviewed and objectives and requirements for the test are discussed. Section 5.2 follows with a description of the different components and software solutions used in the experiment. A short overview of the operation of the system and its success is given. Section 5.3 attempts to describe the system formally using Q-model and stream computing notation. The chapter continues with a discussion of possible future developments and experimentation and concludes with suggestions of possible practical applications that could benefit from the features introduced in the pilot system.

2. Agents

An agent is a term that has become very popular in computer science during the last two decades. Agents are referred to as autonomous units capable of carrying out actions and often forming networks and working together. These networks are called multi-agent systems and they are used to model complex computing situations. However the term agent is not tied to computer science.

It is possible to find descriptions of agents in other sciences as well such as chemistry, medicine, social sciences and others. Often it is used in conjunction with multi-agent systems to model whatever phenomenon is under study and is suitable to be modelled with MAS. There are also examples of agents being used in other meanings. In medicine for instance a pharmacological agent refers to a chemical substance that can have an effect on a biological organism. In social sciences an agent is an individual that acts independently and autonomously as opposed to an individual that is controlled and influenced by social structures and is not free in its choices. Though these agents have their own meaning and definition they all still share a very important common characteristic, that is, they act or are in some other sense active.

In this thesis the focus is on computing agents and agents used in MAS. These agents were abstractions created in AI research in the end of the last century. It started from the idea of individual autonomous elements forming a system to work together to achieve higher level goals which would be unattainable to a single element alone. This idea gradually formed into the theory of multi-agent systems that is today successfully utilized in many other sciences [12].

The following chapter starts with a short explanation of the origin of the term agent. Section 2.1 gives an overview of the origin of the term agent. Section 2.2 presents some of the more known definitions the different definitions of an agent in computer science since the 1990s. There is no intention to single out one correct meaning of an agent but rather discuss the differences between each approach. Section 2.3 describes the main characteristics of an agent.

2.1 Computing agent origin

The computing agent model was not something that suddenly appeared. As it often is with novel ideas they are proposed as answers to problems that have proven themselves difficult to solve with present available methods. To understand the need that created MAS it is

necessary to start with the research of artificial intelligence in the 1970s. At that time researches of AI were mainly concerned with different aspects of what was considered intelligence (e.g. learning, adapting, problem solving etc.). Focus was put to the symbolic representation of these problems and their algorithmic solutions. It was assumed that once these areas were sufficiently studied then putting together an artefact from these components would be a straightforward task. However it was soon discovered that real world problems tend to be too complex for any reasonable algorithmic solutions. In a time-constrained real world situation it would take too long for any first principle based problem solver to reach a solution that would still be valid and applicable. In many cases such an algorithm won't be able to reach a reasonable conclusion at all. [22]

This realisation slowly settled in and by the middle of the 1980s many new approaches to the implementation of artificial intelligence were proposed. The most important of these ideas, which contributed to the future realisation of agents and multi-agent systems, was the idea of distributed artificial intelligence (DAI). The most popular example of a intelligent computing system of the time was the mobile robot constructed by Rodney Brooks [5]. Brooks showed that intelligent behaviour can emerge from the interaction of a number of simpler behaviours. He presented a design and partial implementation of a novel robot architecture which was called subsumption architecture. According to this design a robot (or any AI entity) would have different layers of control, with the lowest layer dealing with simple behaviours (e.g. sensor and actuator control) and the higher layers with more difficult ones (e.g. perception, reasoning, pro-active behaviour). Information in the system moves bottom to up while commands move up to down. Each layer would be able to influence the work of the layers below it while being monitored and, when necessary, controlled by layers above it. At the same time each layer is a whole in its self, having its own objectives and being able to function independently from the other layers. This design resembles multi-agent systems in many aspects but the layers of Brooks' subsumption architecture were not the single autonomous elements that future agents ascended from.

The first concurrent computing entities that could be considered the predecessors of agents were actors as defined by Agha and Hewitt [1, 19]. Work on the Actor model started already in the 1970s, but the focus at that time was on issues of parallel versus serial computing. Since the communication mechanism between the different actors was the central part of the model, it later suited well as a model to describe distributed artificial intelligence. [12].

The new concept of distributed artificial intelligence gradually became acknowledged and at the beginning of the 1990s many new models were proposed [36, 30, 16] all defining

some sort of agent as the primitive element of the model. There is no prevailing definition to the term agent to date, but from the many existing definitions, certain common characteristics can be educed and a generalisation can be made.

2.2 Agent definition

The idea of distributed artificial intelligence is almost thirty years old now and multi- agent systems as a practical model of the idea has been around since the 1990s, but surprisingly there is still no universally accepted formal agent definition. Instead there are different formal definitions that are often formulated together with a formal description of a multi-agent system. Each new proposed MAS defines an agent in a way most suitable to its needs. Since there are numerous MAS definitions and no general acceptance of 'the right one', there are also numerous formal agent definitions. It would appear that the meaning of an agent is always something dependant on the context it is used in, but this is not the case. Most agent definitions share a lot of common characteristics and since the formalisation of DAI there have been many attempts to merge the numerous descriptions into one powerful concept.

One relatively successful attempt was made by Stan Franklin [13], who summarised agent definitions of several AI researchers and went on by starting taxonomy of agents.

Before reviewing the conclusions Franklin made, it would be informative to go over some of the works and definitions of AI researchers that Franklin included in his paper.

Pattie Maes, in her paper discussing the potential of using artificial intelligence in entertainment solutions [27], defines an agent as a computational system that is placed in some environment which it can sense and act upon according to its perception of the environment. An agent has goals that direct its actions and it is capable of adapting or learning form previous experience to better deal with new unfamiliar situations. Maes suggest that these new types of agents can be useful in creating novel entertainment solutions, for instance videogame characters that are able to determine the users' skills and adjust accordingly to provide a richer gaming experience.

Barbara Hayes-Roth gives a description of an adaptive intelligent system (AIS) [18] that shares common characteristics with agents described by Maes. AIS agents are capable of perceiving a dynamic environment, affecting that environment and reasoning to understand conditions and determine a course of action. In her paper, she suggests that agents are bound by their architecture to their intended environments and can thus function only in those types of environments. She proposes a more powerful agent architecture that allows

agents to dynamically construct their own control plans according to situational information to guide their actions.

Michael Wooldridge and Nicholas Jennings are much more philosophical in their discussion over the notion of agenthood [64]. They admit that an agent must be able to act and that one most likely is autonomous and rational, but find these characteristics too general, too insufficient to make a good agent definition. They reach the conclusion that an agent is an intentional system, an entity capable of beliefs and desires. In a later paper [21] Jennings is more practical and defines an agent as an encapsulated computer system placed in an environment that is capable of acting in that environment to meet its design objectives.

The definition of Stuart Russel and Peter Norvig [44] is perhaps the most general. They define an agent simply as anything capable of perceiving its environment and acting upon it. At first nothing is mentioned of autonomy or having personal goals to direct its actions. Russel and Norvig do go on by stating that a rational agent, one that is in some way useful to its creator 'does the right thing' and that an ideal rational agent acts in a way that maximises its performance measure. They also admit the usefulness of autonomy, but have not included it in their definition of an agent. Although the definition of Russel and Norvig seems too simplistic to give any practical understanding of what an agent is, there might still be some value in their intention of keeping the definition this general.

Guided by these discussions Franklin [13] comes to define an agent in the following way: an agent is an autonomous system that is situated in and part of an environment that it can sense and act upon in a timely manner and according to its agenda. It will inevitably affect what it will sense in the future. This definition, although being quite general, does capture the essence of what an agent is. The problem with this definition, which Franklin also admits, is that it does not really describe an agent in any useful way. By this definition a human is an agent just the same way that a thermostat is. They both sense the environment and are able to act and affect their surroundings in a timely manner. But the idea and value behind Franklins definition is to provide the minimal required characteristics of an agent, so that more meaningful and useful definitions can be made by adding additional restrictions to the definition.

Jacque Ferber has done just this [12]. He has added additional characteristics to the basic agent definition to describe a new type of agent useful in building multi-agent systems. The Ferber agent is a physical or virtual entity that is capable of acting in an environment. It is capable of perceiving its environment but to a limited extent and thus its representation of the environment is partial at best. An agent can communicate directly

with other agents and has skills and services that it can offer. It possesses resources of its own and is driven by individual objectives or goals. Its behaviour is directed towards achieving these objectives while taking into account the resources and skills available to it, the perception it has of the state of the environment and the communication it receives from other agents. Clearly this definition is much more descriptive and immediately rules out thermostats as agents. It also points out a limitation of an agent that is the inability of any agent to fully perceive and understand its environment. Even humans, who are considered to be the most sophisticated agents [18, 13], never have complete and accurate representation of the environment they are depending upon. Another novel idea of Ferber is that an agent has resources, skills and services. If an agent is a physical entity it most likely has some sort of power source which could be considered as a resource and in case of virtual agents their whole purpose usually is to provide some kind of services. Although it is possible to define an agent without these properties it right away becomes evident that such an agent is not very useful. The value of a Ferber agent then, is that they are effective single elements for building multi- agent systems, which he also suggested in [12].

Taking into account the agent definitions reviewed this far, it is possible to list some of the more common characteristics of agents. Thus an agent has among others the following characteristics:

- autonomy
- capability to perceive its environment
- capability to act in its environment
- reactive and pro-active behaviour
- goal oriented and situation dependent behaviour
- means of communication with other agents, social ability
- adaptivity, capability to learn
- ability to reproduce itself
- has skills and (access to) resources

Next some of the more important characteristics will be discussed in more detail.

2.3 Common characteristics

2.3.1 Autonomy

Autonomy is one of the basic properties of agents and a very useful one as well. Not only does autonomy provide an entity with freedom to decide for its self how to achieve its goals but it is also the basis for such properties as adaptation and learning. Moreover, in multi-agent systems individual autonomy of agents is the source of emergent behaviour of the system. Emergent behaviour (see section 3.2) is a principal characteristic of MAS and one of the aspects that makes MASs very interesting in terms of distributed computing research. It may be surprising then that agent autonomy is usually defined and understood in a fairly simplistic way.

The simplest definition of autonomy is that an entity is autonomous when it can operate without the intervention of a human or any other master. This description is not very useful as even a digital clock can operate independently for some time but it would be far fetched to call it autonomous. Researchers of AI and DAI usually give more detailed definitions which all overlap in most parts. The definition given by Nicholas Jennings in [21] for example states that an agent is autonomous when it has control over its internal state and behaviour. Jacque Ferber [12] considers an agent autonomous when it is not directed by commands from other entities but by its own goals or incentives. Having control over ones behaviour or following personal goals and incentives are important requirements as they are what distinguishes agents from ordinary software objects, modules, procedures and the like. Jennings and Ferber both acknowledge that an object (e.g. in object-oriented programming language) may have control over its inner state and structures that realise its behaviour, but it can not decide which behaviour to realise. Neither can an object normally refuse to carry out a method that has been called out. In contrast, agents can decide for themselves which behaviours to pursue and whether it is in their interest to perform tasks that someone else asks them to do. Autonomy thus opens the door for cooperation and negotiation in a system, rather than agents being obligated to carry out commands issued by some master [22].

In [12] Ferber introduces an additional aspect of autonomy besides having control over ones state and behaviour. He points out that possessing or having control over some resource is also part of being autonomous. An agent may be self-governing but will fail at its tasks if it can not obtain the resources it needs to realise its goals. Having resources though is not a requisite for autonomy.

In theory autonomy seems to be a straightforward and simple notion that is very useful in MAS design but building truly autonomous agents is a bit more difficult matter. These problems however exit the scope of this thesis and will not be discussed here. Knowing what constitutes as autonomous and how agents differ from programming objects is enough to understand the importants and necessity of autonomy in agents.

2.3.2 Situation awareness

It is hard to imagine useful agents without the ability to perceive their environment. This is the reason why most agent definitions mention agents having sensors or being able to sense their surroundings. Without any situational knowledge it would be hard for agents to demonstrate autonomy and goal oriented behaviour. It is possible and as it turns out very useful for agents to share their situational information. This helps agents cope with insufficient understanding of the environment to make decisions and carry out their tasks. It also makes it possible to create a system wide overview of the current situation. In order to achieve such capabilities it is necessary that all situational information is understood the same way over the whole system. The following paragraph will briefly go over the main aspects concerning situation awareness in agents and multi-agent systems.

First of all it should be explained exactly what is meant by situation awareness and how it differs from the similar concept context awareness. The most typical definition of SA used is the one provided by Mica Endsley in 1988 "the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future" [10]. In practice however it is never possible to monitor all the elements in the environment and a selection must be made. Context awareness was introduced by Bill Schilit in [45] and it was used to describe computing applications that could adapt their operation according to the changing local context. This local context was made up of elements of the immediate surroundings that are important to a human user. The idea behind Schilit's work was that these applications would provide a better user experience than ones not using context awareness. The difference between situation and context awareness stems form their usage and function. Context awareness is local situational information that the observer (i.e. computer application or agent) creates for its own usage and understanding from environment elements important to it. Situation awareness on the other hand is not observer specific and comprises general information that other entities can also use and interpret. Another difference is that context awareness was intended to be used by the application to provide

users with a better usage experience while the intention of situation awareness is to provide other agents (or in a sense users) with accurate and valid information of the current state of the environment [11]. Admittedly context and situation awareness deal with the same problems and share more similarities than they have differences but the terms mean different things and must not be confused.

In case of situation awareness it is important that information gathered from different parts of the system by different agents using different means is conditioned somehow to make it uniform in terms of the whole system. This is not a trivial task. Any sensor measurement taken by an agent has to be validated and supplied with temporal and spatial information to make it useful for other parts of the system. A temperature measurement without any information about where and when it was taken can not be used by anybody else but the agent that took the measurement and even it has limited use for this value. In [34] Mõtus et alii have proposed using a three-tuple $S \in \{S_p, S_t, S_a\}$ to describe simple sensor and other situational data. S_p is the actual value of the situational parameter, S_t holds temporal information and S_a spatial information. Unfortunately this is not very useful without rules that define how this temporal and spatial information is to be coded so that it could be explicitly understood over the whole system. Problems arise from the fact that in real multi-agent systems agents have their own different temporal domains and the accuracy of spatial measurements can vary over a system. Currently there are no good formal methods to validate situational data over MASs.

One of the benefits of using situational information is the possibility to create complex situational parameters from simpler ones such as sensor measurements. In [42] a hierarchical build up of situational parameters is proposed, where lower level parameters are fused together to form higher level parameters. Lower level parameters are sensor measurements for example, that when viewed together can have higher meaning. Air temperature and humidity measurements for instance can be lower level parameters that determine whether it is raining or not. The fact that it is raining (or not) is a higher level parameter that was created by understanding and reasoning over the sensor measurements. Using such higher level parameters is useful in ubiquitous computing systems because it helps to reduce the size of data flow. It must be noted though that creating higher level parameters relies heavily on previous data validation and that small aberrations in lower level parameters and their validity can result in completely different values for higher level parameters.

It is evident that realising situation awareness on an individual level is no trivial task. The difficulty begins with choosing the relevant parameters to measure and take into account followed by data validation problems and higher level parameter creation issues. Complexity increases when moving from individual SA to team SA which among other things means introducing standards on situational information. In addition team SA comprises not only of sharing the SA of single agents but also the knowledge of who knows what. Research on SA is far from being definitive despite its importants in multi-agent computing.

2.3.3 Self-X properties

There is much reason to discuss the self-X properties of both single agents and multi-agent systems. IBM's vision of autonomic computing [20] states eight elements of a computing system that need to be solved in the near future to take the next step forward in computer science. These abilities are knowledge of self, self configuration, work optimization, self healing, self protection, situation awareness, operation in heterogeneous environment and hidden complexity. As Roy Sterritt points out in [54] most of these elements fall under some self-X property or attribute. He summarises IBM's vision of the next step in autonomic computing as achieving self-management of computer systems. Most of the self-X properties that are common to single agents are also properties of multi-agent systems or have very similar counterparts. This subsection will discuss some of these self-X properties that are required to constitute a self-managed computing system in relation to both agents and MAS.

The study of self-X properties such as self-learning and self-organisation began already in the 1960s with (even earlier but not in relation to AI) the study and research of AI [35]. Both properties are still areas of active research and are not limited to AI any more. Self-organisation can be considered more of a property of MAS than an individual agent. It is defined as the evolution of an organisation without any external coordination or centralised planning from outside the system. The components organise themselves through local interactions and gradually a global system-wide pattern emerges [47]. Self-organisation is usually observed together with emergent behaviour but the two are actually not causally related. Emergence and self-organisation in a system can appear independently from one another [48]. The study of self-learning is another field of AI that has seen remarkable research over the years. On a single agent level self-learning is very close to machine learning. Whether it is supervised, reinforcement or unsupervised learning the output of the

learning process is used to improve the understanding of the current situation and predicting future situations to help the agent to plan its future actions. Learning is closely tied to adaptation being one of the possible mechanisms that helps achieve adaptation. On a system level self-learning is a bit more complex. Interactions between agents start to play an important role as the success of the learning process depends not only on the abilities of each agent but also on the way they are put to work together. An interesting new area of research is that of artificial immune systems. Susan Stepney introduces the subject in [53] and explains how a human immune system has great analogy to a complex multi-agent system. Ferber takes it one step further by stating that self-learning coupled with reproduction mechanisms leads to an evolutionary system and even artificial life [12].

Other self-X properties include self-healing, self-protection, self-configuration, self-moderation and self-optimization. IBMs vision of autonomic computing points out several of these properties and associates them with future computing systems. Self-protection and self-healing for instance are necessary parts of future agents and multi-agent systems as autonomic agents will have to deal with defending themselves against malicious attacks or be able to heal themselves in case their defence fails. Both are equally important and vital to agent existence. A malicious attack can destroy an agent if it fails to protect itself just as well as the multi-agent system can demand the destruction of the agent who has survived an attack, but is incapable of repairing itself to continue normal work. It can be undesirable to leave a broken agent free to roam in MAS. As explained in [55] the broken agent can cause undesired emergent behaviour in the system and has to be killed. The method proposed in [55] for killing such agents was self-destruction of the agent very similar to the way cells self-destruct in biological organisms. In cell biology it is known as the apoptosis of a cell in a multi-cellular organism.

Self-configuration and self-optimization form another pair of self-X properties that are closely tied. Self-configuration means the configuration of ones parameters according to the current situation to be able to operate normally. In a dynamic environment the situation changes constantly and an agent must be capable to reconfigure itself to meet these changes [20]. Self-optimization is actually finding the configuration that is optimal to the current situation. In a multi-agent system self-configuration can sometimes mean that an individual agent must changes its own goals (or goal-functions). This may elicit from the current situational information or when the systems precedent goals become in conflict with an agents own goals [35].

It is also important to point out that in order to achieve autonomic computing systems that exhibit self-X properties these systems must be self-aware [33]. It is not enough that the

system has situational information about the environment and other agents it must also have knowledge of its own resources, components and characteristics. The system must know the status of these self-properties and the rules to adjust and use these properties and what effect they might have on future performance.

As IBM called out ten years ago the future of computing systems should lie in autonomic computing. It is the components of systems that must be self-managed and the system that must be capable of self-governance in order to take computing systems to the next level. Biological systems provide for a lot of inspiration and analogies that could help take this next step. But what comes so easily to the natural world is not a trivial task in computer science. It is reasonable to believe that achieving self-X behaviour could be the key to autonomous computing.

2.3.4 Reactive, cognitive and proactive

This section aims to briefly explain the notions of reactivity, cognition and proactivity in terms of agents and computing systems. These three terms are often used to describe autonomous computing components or systems so it is necessary to elaborate on their meaning. The following is by no means taxonomy of agents nor complete comparison but rather it tries to explain how these notions are generally used and understood.

Figure 2.1, taken from Leo Mõtus lecture slides [31], illustrates the division (and evolution) of computing systems from transformational to proactive.

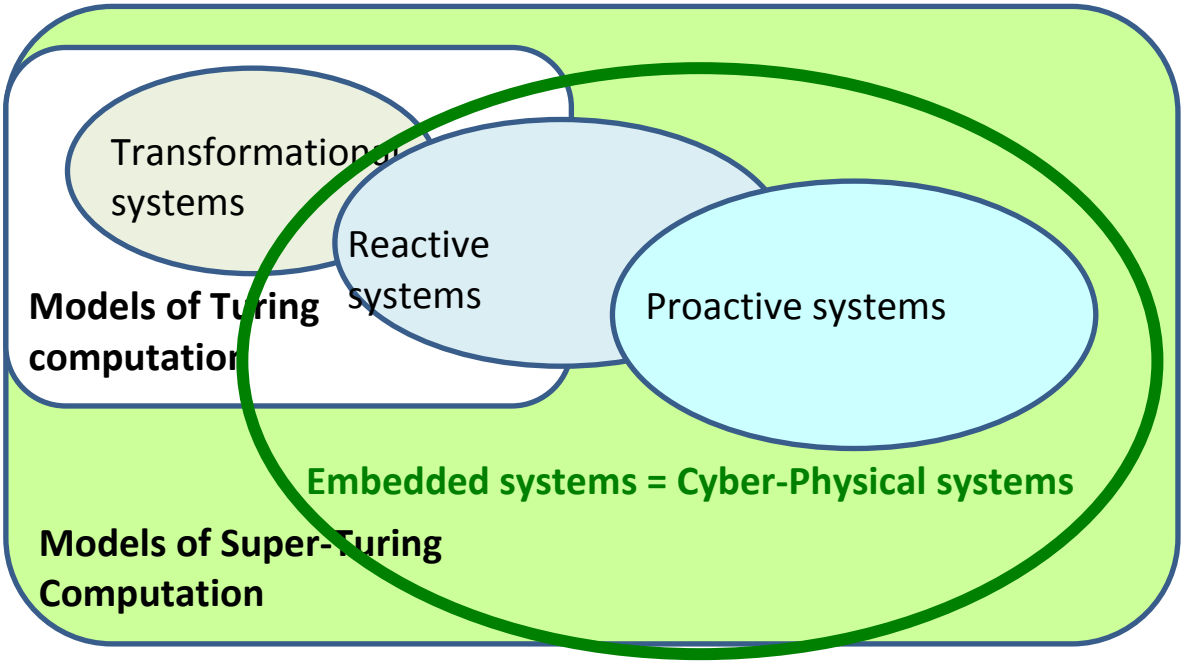

Figure 2.1 Classes and models of computing systems [31]

As computer technology has advanced and computers have become smaller and more ubiquitous computation has shifted from algorithms carried out on single computers to complex computations processed on networks of computers. These new classes of reactive and proactive computing can not be modelled by Turing machine based models any more. Instead alternative approaches that take into account the non-deterministic characteristics of these systems are needed.

Reactive computing is most often used to describe computing systems that react to external stimulation. The system finds an output response to the input it receives. The input signals to the system are more or less known at design time or are simplified and the output is then defined for each input situation. Reactive systems do not exhibit autonomous goal oriented behaviour. Their actions are fixed by the constraints that the designer has imposed on the system. However this does not mean that a reactive computing system should not be considered an agent. Ferber [12] classifies agents as reactive and cognitive. Reactive agents are agents that on their own are not particularly impressive. They do not reason about the state of the environment or their own actions and are not intelligent enough to solve complex problems on their own. Reactive agents become useful when they are put to work together. Their ability to solve complex problems stems from cooperation and the interactions they form. It must be noted that Ferber does not imply that reactive agents are completely without personal goals or autonomy. Reactive agents usually have some internal motivational mechanisms that direct their behaviour in addition to external stimulations that they react to. These internal goals are also defined by the designer.

Cognitive computing systems are systems that exhibit goal directed behaviour and are more or less autonomous in their actions. Similar to reactive systems cognitive systems also react to external stimuli but in a proactive way. The difference between proactive and reactive behaviour is that a proactive system is anticipatory while reactive systems only react in response to already happened situations. A proactive system evaluates input stimuli and tries to understand the current situation and predict the possible future situations. It may take into account its previous situational information or compare the current situation to other known similar situations and draw conclusions to determine its next action. A proactive system tries to foresee the future situations and sometimes it even attempts to direct the future towards situational states that serve its own goals. It takes initiative unlike a reactive system that has to deal with already happened situations and does not try to predict the future. Cognitive systems are thus situation aware, goal oriented proactive, autonomous systems that adapt and learn as they try to achieve their goals. Cognitive agents according to Jacques Ferber are no different. They possess more

knowledge than reactive agents and have a deeper understanding of the processes occurring in the environment and of the current situation. Ferber compares cognitive agents to classical expert systems. Each agent is a kind of expert with its deep knowledge of the environment and its capabilities. These agents come together and form complex interactions and solve their differences to achieve some overall goal that they have all agreed upon before.

2.3.5 Communicativeness

Interaction and communication are somewhat equivalent notions. Still multi-agent textbooks such as [12, 63] devote separate chapters to both concepts meaning that there must be some underlying differences between the two. It appears that interaction is a more general term that comprises all sorts of ways how two agents might influence each other. The collision of two mobile robots, for example, might be considered an interaction as it has effect on both participants. Exchanging meaningful messages is another, more intentional form of interaction. Communication usually means the latter but it can sometimes include indirect and somewhat unintentional messages as well. In this section the communicativeness of agents will be discussed while interactions will be left for the next chapter.

Communicativeness is a characteristic of an agent that illustrates its desire and capability to form interactions with others in order to cooperate and coordinate work. Both Ferber [12] and Wooldridge [63] view communication as an act of an agent. Acts are what agents are capable of in order to influence their environment and help them achieve their goals. In this sense being able to communicate with others in the system is a first order necessity for an agent. Sometimes exchanging simple command messages and data is enough to collaborate but not always. Agents have different goals and desires and at one point they will come into conflict with each other and have to resolve these differences to be able to cooperate. A number of different communication techniques such as auctions, negotiation, argumentation and others exist to help agents reach consensus and coordinate work. Some of these methods will be listed below with short general overviews of the ideas behind these approaches. The language ontology of these methods will not be reviewed in this thesis.

- **Auctions** are a way for agents to determine the use of shared or limited resources. The method assumes that there is one agent, known as an auctioneer, who is in charge of the resource or has been appointed to arbitrate the use of this resource.

Other agents in need of this resource bid for it and the highest bidder usually wins. Auction protocols are fairly simple and effective for allocating goods to agents. Different types exist: English auctions, Dutch auctions, Vickrey auctions, first-price sealed-bid auctions, etc. [63]

- **Negotiation** techniques are used when agreement has to be reached on more complicated matters than simple resource usage. This might mean reaching consensus on different matters of interest. Negotiation is a good method in case of individualistic agents with conflicting goals. Generally negotiation requires four components: a set of all possible proposals, a protocol to determine currently legal proposals, strategies for making proposals (this is usually agent specific) and a rule to determine an agreement. Wooldridge [63] references two types of negotiation: task-oriented and value-oriented. In the first agents cooperate to accomplish a set of tasks with minimal cost to every participant. In the latter agents try to achieve the best environmental state in terms of some personal value function. Since every agent affects the environment they must agree on an optimal state for all.
- **Argumentation** is a development of the negotiation technique where agents can explain the reasons for their proposals. The additional value of this method is that agents can justify their proposals, criticize other proposals, use persuasion (threaten, reward, appeal, etc.) and change their initial views. Argumentation is extensively studied in many sciences, (social sciences, psychology) a formal framework for multi-agent systems has been proposed by Sierra and Jennings [49].
- **A contract net** is a simple technique used to accomplish tasks in MASs. According to this method an agent will propose a contract for any work it needs done. Other agents that are capable and willing to carry out this task will bid to the issuing agent (manager) to receive the contract. The manager then chooses a suitable bidder and rewards it with the contract. The winner is now responsible of completing the work. It can issue sub-contracts if needed. A similar technique has been developed in the Laboratory that uses subscriptions to coordinate the work of agents [33].
- An interesting form of communication is the **blackboard** model. In its MAS variant each agent represents a 'knowledge base' that has certain know-how and can perform certain actions. Additionally there is a blackboard (which can be represented by an agent also) that comprises a common problem that needs to be solved. Individual agents can solve parts of the problem and post their results on the blackboard for others to see. A control mechanism coordinates the work of the

individual agents to reach a desired solution to the problem posted on the blackboard [12]. Different variants of the blackboard model exist.

Each of these techniques represents a different communication model with its own rules and methods. If agents of a system are to cooperate this functionality has to be provided to them before hand. This is what is meant by communicativeness, that agents possess the necessary tools and knowledge to engage in meaningful communication. The approaches listed above are only some of the possible coordination techniques. They are mainly directed at cognitive agents because they require more intensive computations than reactive agents can offer. In [12] Ferber points out that reactive agents can easily communicate via simple broadcasted messages that propagate at different speeds and intensities in different directions. Coordination in such systems stems form self-organisation and is not achieved by complicated argumentation. Even though being indirect this type of communication also expresses an agent's communicativeness since it is completely intentional.

3. Interactions and emergent behaviour

Agents have proved powerful computing entities and a useful abstraction to model component autonomy, goal oriented operation (in contrast to submissive servicing of commands) and complex behaviour (context awareness, reasoning, proactive behaviour). Whether these agents are reactive or cognitive they find additional merit in interaction and cooperation. Working together enables to achieve goals otherwise unreachable for individual agents. Consider for example the breaking system of a modern car as a system of interacting agents. Suppose separate agents control the breaking of each wheel and the steering system of the car. When breaking the common goal of the system is to slow the car and keep it on the road. The agents have to cooperate to slow each wheel so it does not lock and to steer the car safely when some wheels do. It is a joint effort of the system that relies on interaction and situational information (inertia, temperature and humidity outside, road conditions) from various sensor(agent)s (gyroscope, thermometer, etc.).

Systems of agents comprise features and functionality that is not specific to individual agents making up the system. Often this functionality can not be derived from the separate behaviours of the components of the system. Rather it seems to emerge from the complex non-transparent interactions between cooperating agents. This emergent behaviour is currently a major interest in distributed computing systems and the task of handling emergence one of the principle challenges. Interaction of agents is not the only source of emergent behaviour as the environment contributes its own share of unpredictability. The following chapter is devoted to these fundamental features of multi-agent systems.

Interaction and its importance in distributed embedded systems are discussed in section 3.1 followed by an overview of classification methods for different types of interaction. The different approaches to implementing mediated interaction and the common characteristics of these mediation artefacts are reviewed in subsection 3.1.2. Section 3.2 explains emergent behaviour followed by an overview of the importance of environments in multi-agent systems in 3.3. Chapter 3 is concluded with a short synopsis of the multi-stream interactive computing in section 3.4.

3.1 Interactions

A characteristic of embedded pervasive computer systems is their continuous exposure to changing environmental conditions. Because of complex unknown causal relationships inside the environment there will always be conditions and states of the environment that

the designer of the system can not predict and prepare for. This means that determining the whole behaviour of such systems is not feasible and often not possible. Another source of uncertainty in embedded systems is the existence of autonomous components. These may include artificial entities with partial autonomy or completely autonomous (in terms of the embedded system) humans or both. Their autonomy introduces additional indeterminacy for the system's designer. The earliest examples of such systems are parking assistants and air traffic control systems.

The first implementations of the park assistant system were built by 'trial and error' and involved the use of heavily restrictive safety envelopes (and still do). With certain admission the commercial parking assistant applications of today can be considered (partially) autonomous. The same can not be said about air traffic control systems that are still under human supervision. Although parts of these systems operate autonomously (e.g. aircraft trajectory calculations and pilot health assessment) the overall work has to be monitored because consequences of arbitrary errors are too great [35]. The first designers of these systems realised that the difficulty of controlling embedded systems stemmed from the autonomy of its components and the unknown factors of the environment. However the solution to this problem did not lie in understanding autonomy and the processes of environment better. Instead it was discovered that the interactions of system's components were the source of unpredictable behaviour and that better handling of these interactions would allow monitoring and control of the system.

Before the appearance of embedded autonomous computing systems not much attention was put to studying interactions. Communication channels between processes were strictly fixed and known at design time. The same approach was used with embedded systems but it failed immediately. The reason is that embedded autonomous systems compose a much richer topology of interactions. A component may form interactions with the environment other autonomous components and humans. These interactions are dynamic by nature, meaning that they can be altered or replaced during operation which is necessary functionality when parts of the system are autonomous or exhibit obscure behaviour. Interactions may change for a variety of reasons: changes in the environment; changes in context; component goals change; changes in the system when some components appear or disappear or regulation by a third party (system manager or other mediator in case of mediated interaction). A model of embedded systems must support these changes if it is to be successful.

Earlier formalisms that described distributed computing and dealt with interactions between processes were Petri nets for example [see 40]. Unfortunately first Petri net based

computing models did not include time and assumed that transition functions that link places execute instantaneously. This shortcoming was tackled with timed Petri nets, but the new model was still unsuitable for describing embedded systems because the topology of links between places in Petri nets does not support dynamic altering. Milner started work on a new computing model in the end of the 1970s and presented 'a calculus of communicating systems' in [29]. This work later evolved into the π-calculus [30] which successfully separated interaction from computation. However neglecting time [32] and clinging to algorithmic processes [17] still limits its use as a general model for embedded systems. The importance of interaction in computing systems was realised and strongly advocated by Peter Wegner et alii in the mid 1990s. Together with Dina Goldin they introduce the notion of sequential interactive computing [17]. Their sequential interactive thesis suggests that any sequential interactive computation can be performed by a persistent Turing machine – a three tape modification of the Turing machine as proposed by Goldin et alii in [16]. The persistent Turing machine has one read only tape for input symbols, one read-write tape for performing calculations and one write only output tape. The machine reads a symbol from its input tape, performs calculations on its work tape and after completing computations writes a symbol to its output tape. The input tape can not be erased and in this sense the machine is history dependant as it may read previous input values. If the work tape is not erased after each computation then the machine is persistent as previous calculations can affect future ones. About the same time as Wegner started advocating interactive computing Mõtus and Rodd introduced the Q-model, which also dealt with interactions between processes [36]. An important feature of the Q-model is that it includes complete time sets for all processes and channels (interactions) between processes. The Q-model was further developed in [32] to represent multi-stream computing and in [33] to include location specific information in addition to time sets.

The scope of this thesis does not leave room to discuss these models in more detail. Some insight to multi-stream computing as it is used in the Q-model modification is given in section 3.4 as it will be later made use of in chapter 5 to describe the pilot system more formally. This section will continue with descriptions of different types of interaction in subsection 3.1.1 and a separate review of mediated interaction in 3.1.2.

3.1.1 Types of interaction

The multi-agent system approach to typifying interactions evolves around understanding the reasons for interaction and the nature of relationships between agents [12, 63]. MAS

interactions are different whether agents are equal partners, competitors or in master-servant relations and whether they intend to cooperate, compete for resources or resolve conflicts. In this thesis a more distributed autonomous systems approach is taken where interactions are classified according to procedural characteristics. This enables to divide interactions between to classes: direct and indirect. At first sight it may seem plausible to distinguish interactions by interacting partners between agent-agent, agent-environment and agent-human interactions but this classification is discouraged. The reason is that interaction based computing models may model humans and the environment as regular (or special) agents making it impractical to have special types of interaction for them. Additionally to direct and indirect interactions an interesting approach to sort interactions according to causal relations evoking these interactions is suggested by Pena et alii [39]. Mediated interactions will be view separately in subsection 3.1.2.

Direct interaction is the exchange of information that may take place between two familiarised agents in the form of passing messages or influencing partner's state. Direct interaction does not apply that agents must communicate directly to each other, rather interaction can be sequential and messages may pass through several links (other agents) before they reach their destination. The requirement is that messages must be directed to specific recipients. The ontology for communication in direct interaction may vary from simple boolean notifications to complex communication languages. In [12] Ferber notes that direct interaction takes place between cognitive agents capable of abstract communication sometimes to the extent of creating their own private ontology to understand each other. Influencing partner's state usually takes place between agent and environment but can also occur between regular agents. Consider for example raising a signal line from low to high on an electrical circuit that connects one agent to another. If this line directly connects only two agents and both are aware of each other then the requirement of direct interaction is met.

Indirect interaction happens between two agents that share a common environment. One agent causes a change in the state of the environment that is observable by another (or many other) agent(s). The two agents need not ever meet or know of each others existence. The prerequisite for indirect interaction is that agents share the same environment and that changes in the environment are persistent for a bound period of time [15]. Indirect interaction has been observed in biology and natural systems. Societies of ants and termites use indirect interaction as one means of communication. Food foraging process of ants, for instance, involves indirect interaction between members of the colony. If an ant finds food, it brings it back to the nest leaving a trail of pheromones along the way. When other ants

come across this trail they follow it to the food and intensify the rout with their own pheromone. Termites have been observed to cooperate through indirect interaction to build their nest. The construction of the nest itself becomes the common environment that termites use to coordinate the building process. This self-organising mechanism is known as stigmergy and was introduced by Grasse in 1959 [47]. Some of the useful properties of indirect interaction according to [15] are:

- anonymity – interacting partners need not know each others identity
- time decoupling – there (may) exists a time delay between one agent actuating the change and the other observing it
- space decoupling – a change in the environment may propagate to other agents at other locations; or mobile agents without ever meeting may pass through places and observe environment state changes others have left behind
- non-intentionality – interaction may be (but does not have to be) completely unintentional

An additional property that is not mentioned in [15] is the bonded broadcast nature of indirect interactions. This may seem too obvious to be mentioned separately, but it is perhaps the key characteristic of indirect interaction that makes self-organisation and emergence possible. A state change made in the environment by one agent often reaches a number of other agents who are time and space wise in an observable range.

In [39] interactions are classified based on the complexity of causal relations of interactions and the level of abstraction taken to model them. A similar method to partition interactions is discussed in [32]. In [39] distributed multi-agent systems are divided according to the complexity of its interactions between two domains: ordered and unordered. The ordered domain is further divided into known domain and knowable domain. Systems where the causal relationships invoking interaction are completely known and described make up the former and systems where these relations are not fully understand or partially known make up the latter. All interactions in the ordered domain have definitive causal relationships whether they are known or not. The unordered domain is made up of systems where the causal relations of interactions are unstable or unpredictable. This domain is divided between complex domain and chaos domain. The complex domain includes systems with causal relations where the number of relationships is too great for analytic processing. Certain relations can be deduced by analysing system's history but predicting future interactions is implausible. Interactions in the chaos domain

are without any perceivable causal relations. In [32] interactions are also classified according to their complexity but complexity of interactions is based on the time domain used in the system (topological, metric, relative, etc.).

3.1.2 Mediated interaction

Contemporary distributed computing systems compose self-aware reactive and proactive components operating to achieve self-set goals. The autonomy of these components implies the possibility to form different interactions with partners dynamically at run time. The second epoch of interactive computing beginning in the 1990s with Milner [30], Wegner [58] and Goldin et alii [16] has acknowledged this fact and modelled computing around interaction. The models of persistent Turing machine and π-calculus capture the essence of contemporary computing systems better than algorithm based models but they lack practical methods for engineering these systems. Perhaps the most immediate problem in distributed computing systems is the lack of practical methods to manage interactions so that a system's operator has an overview of the progress of the system and might be able to direct its course. No general techniques exist jet, but the growing understanding is that some sort of mediated interaction mechanism could help tackle this problem [see for instance 50, 38, 33].

Mediated interaction is usually seen as a collection of intermediating components that manage and monitor interactions between agents. Its complete functionality is still a matter of open debate but a few commonly acknowledged characteristics can be identified. First of all mediated interaction should be able to replace any direct interaction and most of indirect interactions in a system where required. Secondly the components of mediated interaction should accumulate situation awareness and share this awareness with each other to create a system-wide understanding of context. Thirdly mediated interaction should be capable of on-line alteration of interaction topology and filtering of messages. Finally mediated interaction should exhibit proactive and goal oriented behaviour that seeks to fulfil its own goals or goals set by a third party (operator of the system or an autonomous component of the system). These characteristics in various forms can be found in [33, 50, 38]. The aim of mediated interaction is to manage interactions of a system by first creating an understanding of situational context and then using this knowledge to monitor the progress of the system and intervene by altering interactions when necessary. Situation awareness is achieved by analysing changes in the environment, parsing the contents of messages passed between agents and reasoning about this content. As soon as signs of

undesired behaviour are noticed mediated interaction can start filtering messages and modifying interaction mapping to direct the system towards desired performance.

Three practical examples of mediated interaction between agents each a little different will be discussed next. Focus is concentrated on models that explicitly deal with mediating interactions in artificial systems and communication mediation as it is studied in social sciences is not reviewed. Also multi-agent development and simulation environments such as JADE for example will not be discussed. Although one of JADE features is managing interactions and hiding the complexity from agents it does not demonstrate any of the four mediated interaction features mentioned above.

In [50] the notion of electronic institutions is introduced as a mediating environment for agent interactions. Electronic institutions are supposed to mimic similar human institutions for autonomous agents. The term human institution stands for the conventions, norms and socially acceptable behaviour that humans follow in their societies. The authors of [50] argue that agents should have a similar environment that establishes the rules and normative that agents who want to act in the environment should follow. Agents that participate in an electronic institution must accept its rules and demonstrate that they are capable to keep the commitments they make while interacting in the institution. In return the institution provides agents its common ontology and communication protocols and functions as a mediator for interaction partners. Although it is not explicitly mentioned in [50] it is clear that such an institution has its own goals (in the form of rules and norms) and is capable of accumulating situation awareness. Electronic institutions could be modelled as proactive entities that dynamically modify the communication environment, provide an overview of system progress (to an operator for example) and filter and mediate interactions. The drawback of electronic institutions is that it only supports direct interaction between agents and neglects indirect communication completely. In this sense it is intended more for cognitive than reactive agents.

Another framework for interaction mediation is proposed in [38]. The authors describe a collection of coordination artefacts that help agents manage their interactions. A coordination artefact can be understood as a resource of the environment. It is not active itself it only encapsulates the complexity of interactions and provides this functionality to agents requiring it. A coordination artefact is characterised by a usage interface, operating instructions and specification of behaviour that it provides for the agents. There can be many types of different coordination artefacts in a system. The authors claim that the behaviour of coordination artefacts can be dynamically changed at run-time (by operator or agents of the system), which would in fact make them active entities of the environment

and allow them to be designed to express self-interest and proactive behaviour. Other useful properties of coordination artefacts are that they can log communication history (thus accumulate situational information) and that their operation is completely transparent and controllable. Combining these to properties allows predicting the future behaviour of the artefact although it is unclear how this is achieved especially if the functionality of the artefact can be dynamically altered by other agents or an operator of the system. Still a lot of useful observations about mediating interactions have been made in [38].

Finally the work done in [33] is discussed. The authors present a formal model of mediated interaction based on the Q-model and the notion of proactive middleware. This middleware serves as a distributed mediator component that manages interactions for agents connected to the middleware. The distribution of middleware means that each agent is appended with a middleware interface that provides a communication channel for the agent. This means that all messages and data essentially pass through middleware making it a powerful tool in creating situation awareness. The practical application discussed in the article uses a producer-consumer based communication model according to which consumer agents make requests for data from producer agents. Middleware handles these requests via subscriptions meaning that consumers make subscriptions for data that interest them and middleware finds a producer that provides the data. This inherently renders all interactions indirect (direct interactions are still possible) making it extremely easy to modify the topology of the system at run-time. The middleware presented in the article is an active component that possesses means to control the system according to its own interests or to pass this control to another master (agent or operator). The prototype of this middleware was implemented on an ad hoc sensor network communicating over wireless media. This illustrates its possible use in multi-agent systems with reactive agents but leaves unclear how it would suit a system of cognitive agents with more complex communication languages.

3.2 Emergent behaviour

One of the main reasons that agents have been adopted as a suitable model of computing entities is that their nature comprises autonomy. Similarly one of the main reasons why interactive computing and MASs are currently preferred computing models is their ability to capture emergent behaviour better than traditional algorithmic computing. Autonomy and emergent behaviour have a strong causal relationship but the former is not the only source for the latter. Other sources such as interaction between system components and

their abidance to the changing environment are the cause for emergent behaviour in most multi-agent systems. Whether these systems compose reactive agents engaged in simple cooperation or cognitive agents committed to a complex reasoning procedure the overall outcome of both processes is likely the result of emergent behaviour [35].

Emergent behaviour in various systems had been noticed long before the appearance of computing systems. It is present in all natural systems, can be observed in chemistry, physics, biology, cybernetics, etc. and is currently most studied in complex systems theory [62]. As a result emergence as a phenomenon is understood quite unanimously in computer systems studies although informal definitions of different works vary to a small extent and emphasise different aspects of it. This thesis also avoids giving a formal definition of emergent behaviour and consents with a general discussion on the subject.

In [21] Jennings states that emergent behaviour is the behaviour of a system that can not be deconstructed in terms of the behaviour of the components of the system. According to [46] emergence can be observed when global behaviour of a system is the nonlinear combination of local behaviours. A similar stance is taken in [48] where emergence is a macro level phenomenon of a system that is the result of nonlinear activities on the micro (component) level. The same authors impose a few more requirements on the phenomenon to truly qualify it as emergence. The macro level behaviour must be novel in the sense that it did not exist before [48] and that it is not observable in the behaviours of parts of the system [62]. Both works also stress that the overall emergent behaviour must express coherence i.e. must have its own identity or pattern that is consistent over time and relies on the correlation of system's components. Another property pointed out in [62, 46] is the robustness of emergent behaviour. Removing single components from the system does not lead to immediate system failure or disappearance of the behaviour. Rather the quality and performance of emergent behaviour will depreciate slowly as components from the system are removed. Irrespective of these additional characteristics it seems that majority of researchers still restrain to the simpler definition of emergent behaviour. Generally put, emergent behaviour is the behaviour of a system that can not be explicitly derived from the behaviour and actions of the components.

The reasons for the occurrence of emergent behaviour have already been mentioned. Complex dynamic interaction (or plain action [48]) of components is the most common explanation for emergence [21, 46, 35, 62]. Usually systems that exhibit emergent behaviour compose numerous different components forming implicit interactions that constantly change and are thus too complex to follow. The complexity of interactions stems from the (partial) autonomy of components and the unpredictable influence of the

environment on the components, interactions and the system as a whole. Thus it is also autonomy of components and the influence of the environment that shape the overall emergent behaviour of the system. These ideas are summed up in [35, p 4]: "... *emergent behaviour is enabled by autonomous behaviour of components, dynamically changing topology of interactions, and incompletely known impact of the environment upon the system* ..."

A few examples of emergent behaviour have already been mentioned in relation to stigmergy in subsection 3.1.1. The foraging for food of ants and nest building of termites are processes where emergence can be observed. Among artificial systems the internet is a good source for examples of emergent behaviour. The way some web-pages or videos gain popularity for a short time and then loose it again follows the specification of emergence. Similar indeterminacy can be observed in the results listing of a query to an internet search engine. For a particular query to the same search engine by two different people at two different locations the results will (always) be different. This is because search engines keep history of users previous actions and monitor current interests. The answer to each query emerges because software agents have to interact to produce the best result for a particular user.

So far there are no sufficient and widely accepted methods to direct and use emergent behaviour. In some applications emergence is completely undesired and its appearance is restricted by defining strict bounds to interaction and by fixing interactions to a rigid structure. This inevitably decreases the efficiency and usefulness of complex artificial systems so designers of these systems look for methods that permit emergence but keep it bounded [21, 35, 33].

3.3 Environment

First of all, it must be made clear what is meant by the word environment. In MAS literature it can mean many things depending on context. For example when talking about modelling of MAS, environment can mean any simulation environment where the model is tested or in some cases even the hardware it is tested on. Sometimes environment is a part of the MAS model as a separate element in itself that is designed along with the other elements of the model. Other times the environment must be included in the MAS model as is and can not be manipulated or modified. This is usually the case with situated MASs. Here environment refers to the overall surrounding that the agents are embedded in and is included in the MAS model as a separate active element with its own states and processes

[60]. These environments represent either the real world or parts of it or some virtual setting (e.g. the internet). In this thesis environment should be understood the same way – it is the surrounding settings that agents are embedded in.

Environments play an important role in the design of agents and multi-agent systems. Agents situated in some environment need some kind of input from the environment in order to successfully carry out their tasks. The types of sensors and actuators an agent needs and the amount of state needed to interpret and understand the environment depends on the task it is designed to do as well as the environment it will be placed into. All of the higher level logic of reasoning, learning, adaptation, pro-active behaviour etc. depend heavily on the characteristics of the environment and as it turns out environments range from simple and straight-forward to complex. It is thus necessary to discuss some of the different types of environments and the possibilities and restrictions they impose on agent design.

A general description of the natures of different environments is given by Stuart Russel and Peter Norvig [44]. They bring out six characteristics that should be considered when describing environments. It is not the intention of this thesis to give a complete overview of all possible characteristics of environments because this would need a whole chapter at least. Rather only the six characteristics of Russel and Norvig will be briefly referenced here as a general overview of what must be taken into account concerning environments in MAS and the reader can turn to [44] for examples and a more detailed description or any of [12, 60, 63] for other work on the subject.

The first characteristic that Russel and Norvig point out is that an environment can be either fully or partly observable from the point of view of the agent. Fully observable here means that everything relevant in relation to the tasks of the agent can be measured and taken into account. Partial observability can stem from a noisy sensor, poor agent design that has overlooked some task related relevant parameters or because the environment has changed and the agent can not acquire the information it needs with the sensors it has any more.

The second important characteristic to consider is whether an environment is deterministic or stochastic. In a deterministic environment the next state is completely determined by the current state and the actions taken by the agents. The reverse is true for a stochastic environment. Most real world situations are stochastic and even some deterministic environments must be considered stochastic from the point of view of an agent. For example if an environment is partially observable to the agent it must consider the environment stochastic, because it can not predict the future state even if the environment

really is deterministic. This is often the case when the future state of a deterministic environment depends on the current action of more than one agent. In this case an agent might not know how the other agents are going to act and thus the future state can not be predicted.

Some environments are episodic while some are sequential. In an episodic environment the agent's actions and decisions in one episode do not affect the decision making and outcome of other episodes. An example of an episodic environment is that of an assembly line robot that has to detect defective parts. Each inspection of a particular part is a separate episode that does not influence other episodes. In contrast a chess game would be a sequential environment where each turn is a separate episode. The actions taken in one episode have an effect on the actions of future episodes. The sequential aspect of an environment actually introduces feedback to the system. Actions that change the environment now will most likely be sensed in the future and could provide useful information about the nature of the environment and the processes running it.

From the point of view of an agent the environment can be either dynamic or static. If the environment does not change while the agent is deciding on its next action then the environment is static. The agent need not worry about the conditions changing during the time it is making its decisions. In a dynamic environment the conditions change constantly even during the time an agent is deciding what to do next.

Another characteristic of the environment is whether its states are discrete or continuous. The distinction that Russel and Norvig have made here is that some environments have a finite number of different states while others have an infinite number of possible states. According to them a chess game environment has a finite number of possible states. The number of possible different actions an agent can take in a chess game is also numerable as well as the set of rules to the game. In contrast in a continuous environment there is a continuous flow of states with no end. In this case distinction between neighbouring states is somewhat artificial and depends on the design of the system.

The final characteristic of an environment according to [44] is whether it is a single agent or multi-agent environment. This distinction is important because each agent acts on the environment in a way that best suits its own goals and each action has an effect on the work of the other agents in the environment. Depending on the individual goals of agents they can either compete with each other or cooperate. Multi-agent environments are more complex in this sense and present the need for communication between agents.

There are of course many more aspects of an environment that are important when modelling agents and multi-agent systems, but as mentioned earlier it would be a

demanding task to go over all of them here. There is however one property that was not discussed by Russel and Norvig that needs to be mentioned. This is the environments' role in communication. James Odell points this out in [37] when he differentiates between the physical and the communicational environment. He states that the physical environment provides the principles and processes that govern and support agents while the communicational environment provides the principles and processes that allow agents to convey information. The communicational environment defines different communication protocols, languages, social policies and coordination strategies that are necessary to design and implement MASs while the physical environment sets the rules and means how messages are passed through the environment. As an example, radio communication between smart dust motes is made up of a communication language and a message passing protocol and the laws of physics that define the way electro-magnetic waves propagate through the environment. According to Odell the first two would be a part of the communication environment and the latter a part of the physical environment.

The purpose of this brief and incomplete overview of the characteristics of an environment is to stress the important role that it plays in agent and MAS design. It is necessary to understand and recognise these properties to be able to take full advantage of them. In real world situations the environment is always complex - partially observable, stochastic, sequential, dynamic, continuous and multi-agent, but it may be possible to simplify these aspects from the point of view of the agent and thus simplify its design significantly. The environment can seldom be designed or controlled but rather it has to be adapted with and that is why it is necessary to know exactly what possibilities it offers and what restrictions it demands.

3.4 Q-model and stream computing

This section will discuss a formal model for interactive computing, namely the Q-model, and give a brief introduction to stream computing. Several models for artificial systems of interacting components that focus on interaction have been presented since interactive computing succeeded traditional algorithmic models. Some of the more known formalisms are the π-calculus by Milner [30], persistent Turing machines introduced by Goldin et alii [16] and the Q-model as presented in [36]. In this thesis the Q-model is favoured for two reasons. Firstly because it is supported by a strong formal time model particularly useful in practical engineering projects and secondly because recent work in the Laboratory for

Proactive Technologies uses the Q-model extensively (see for instance [33]). The stream computing notation adopted here relies on the work of Dosch et alii [9].

The basis for the Q-model was first introduced in 1977 by Quirk and Gilbert to describe the properties of complex real-time systems. The model was further developed to suit the timing analysis of distributed computing systems in [36]. At the heart of the Q-model are processes and channels. Processes represent components of the system while channels model interactions between the components. A Q-model process *p* is a mapping from the process' domain of definition to its value range [36]

$$p : T(p) \times dom\ p \rightarrow val\ p \qquad (3.1)$$

Here *dom p* and *val p* represent the domain of definition and value range of process *p* respectively and *T(p)* stands for the process time set. *T(p)* contains all the start times for process *p* and the mapping is repeated for each start time. In terms of stream computing the definition and value range of a process are the input and output streams of that process. These ideas are developed in [32] and will be reviewed later on.

A channel is a one-to-one, one-way connection between two processes. It is used by one process to acquire information from another. The process initiating the interaction is called a consumer process and its partner a producer process, as the latter produces data that the former needs. A channel is a mapping from the producer process value range onto the consumer process domain of definition, i.e. it links the producer's output to the consumer's input. A channel σ_{ij} between a producer p_i and a consumer p_j is [36]

$$\sigma_{ij} : val\ p_i \times T(p_i) \times T(p_j) \rightarrow proj_{val\ p_i} dom\ p_j \qquad (3.2)$$

where *val* p_i and $T(p_i)$ stand for the value range and time set of p_i, *dom* p_j and $T(p_j)$ stand for the domain of definition and time set of p_j and $proj_{valpi}$ stands for a projection of the value range of p_i to the domain of definition of p_j. A projection of values is used because the value range of the producer process might not explicitly define the whole domain of definition for the consumer process, i.e. the consumer may use input from other processes as well. The mapping depends on the time sets of both processes. For a specific consumer start instance *t* from the consumer time set $T(p_j)$ the mapping can only use values of the producer process that have been produced for that moment in time. This is defined by the channel function $K(\sigma_{ij}, t)$ as follows [36]

$$K(\sigma_{ij}, t) \subset T(p_i),\ t \in T(p_j) \qquad (3.3)$$

The channel function $K(\sigma_{ij}, t)$ defines for a time instance *t* of $T(p_j)$ a subset of $T(p_i)$.

Streams are a common method for interactive computing design [32], where each stream usually represents the communication history of an interaction between two components. A set of finite streams over a set of symbols $\mathcal{A}$ is denoted A^* and defined as [9]

$$A^* = \{\langle\ \rangle\} \cup \mathcal{A} \times A^* \qquad (3.4)$$

Here $\langle\ \rangle$ denotes an empty stream and $\mathcal{A} \times A^*$ a prefix operation that attaches an element from $\mathcal{A}$ in front of stream A^*. This illustrates the recursive nature of streams and helps model the progress (i.e. history) of interactions, a current stream is made up of the previous stream and a new element. The prefix operation $\lhd$ is defined as [9]

$$\lhd : \langle a \rangle \times A^* \to A^* \qquad (3.5)$$

where a is an element of $\mathcal{A}$ and $\langle a \rangle$ denotes a one element stream. Single streams will be denoted by capital letters while their elements are lower case. The concatenation $X\&Y$ of two streams $X = x_1 \lhd, x_2 \lhd, \ldots x_k \lhd \langle\ \rangle$, $k \neq 0$ and $Y = y_1 \lhd, y_2 \lhd, \ldots y_l \lhd \langle\ \rangle$, $l \neq 0$ of the same set $\mathcal{A}$ results in a new stream $\langle x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_l \rangle$ [9]. The concatenation operation is denoted by $\rhd$ and defined as

$$\rhd : X\ \&\ Y \to W \qquad (3.6)$$

where $X = \langle x_1, x_2, \ldots, x_k \rangle$, $Y = \langle y_1, y_2, \ldots, y_l \rangle$ and $W = \langle x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_l \rangle$. There are many more operations that can be performed on streams that will not be discussed in this thesis. The brief introduction given so far is enough to analyse the pilot system presented in chapter 4 and 5. For a more thorough overview of streams the reader can turn to [56].

Before turning to chapter 4 and the pilot system proposed in this thesis it is necessary to elaborate how streams and the Q-model are related. A starting point for this discussion is given in [32]. The domain of definition *dom p* and the value range *val p* of a process *p* can be viewed as a collection of streams for every time instance *t* of *T(p)*. If $\mathcal{I}$ was the set of all possible input symbols for *p* then the domain of definition for this process *p* would compose a set of finite streams defined by (3.4) such that for each t_i of *T(p)* there exists an input stream $I_i \in I^*$. Similarly if $\mathcal{V}$ was the set of all possible output symbols of process *p* then the value range for *p* would compose a set of finite streams defined by (3.4) such that for each time instance $t_i \in T(p)$ and input stream $I_i \in I^*$ there exists an output stream $V \in V^*$ defined by the mapping (3.1).

The domain of definition for process p_x may include output from several processes p_1, ..., p_m. The set of input streams I_x* for p_x is then made up output streams of V_1*, ..., V_m* according to the channel mappings σ_{1x}, ..., σ_{mx} (3.2) and the appropriate channel functions (3.3). If the output of processes p_1, ..., p_m is the only input for process p_x for all t_i of $T(p_x)$ then $I_x^* \subseteq V_1^* \cup ... \cup V_m^*$.

An example of combining streams and Q-model processes base on the pilot system experiment can be viewed in section 5.3.

4. A practical demo application

The Research Laboratory for Proactive Technologies (Laboratory) was established in 2007. It resides in the premises of Department of Computer Control at Tallinn University of Technology. The Laboratory is based on a group of researchers that had been dealing with real time systems at the Department of Computer Control and collaborating with researchers from Institute of Technology at the University of Tartu and hence, have inherited their theoretical results and experience. The Laboratory consists of more than ten PhD level researchers, several PhD students and a few MSc students. The Laboratory has several partners from other departments at Tallinn University of Technology and University of Tartu and cooperates closely with different private companies and businesses in the manufacturing, electronics and software industry [43].

The main research domain of the Laboratory includes the theoretical study and practical implementation of networked pervasive computing systems. Some of the topics include multi-agent systems, real-time systems, ad-hoc networks, emergent behaviour, situation awareness, smart sensor networks and self-organisation. Theoretical and practical work in the Laboratory is carried out on each topic daily and involves formal modelling of said systems and networks and practical experimentation on smart dust motes or virtual simulation. A large part of the practical work is done in cooperation with the Laboratory's partners from industry and some of the research results have already found use in production lines or end products.

This thesis adds another experiment to the practical results of the Laboratory in the form of a pilot system for future MAS studies. The pilot system will provide a well structured platform for a multi-agent system and comprise all the necessary components to build such a system. The software written for this pilot system will be divided into explicit components that can be easily scaled to be used elsewhere in other experiments or applications. An additional objective of this work is to create a demo application that displays various characteristics of a multi-agent system including self-organisation, autonomy and emergent behaviour. It also introduces and explains the concept of mediated interaction and shows how mediated interactions can be used to partly control the system's emergent behaviour. Such an application can be used as an effective visual demonstration for visiting researchers and guests. It should introduce the problems and properties of autonomous computing systems and the work done in the Laboratory in this domain.

This chapter introduces the setup for the demo. Its necessity, objectives and possible use is described in section 4.1. Section 4.2 gives an overview of the hardware and software used

in the experiments, the operating system of the smart dust motes and the programming language. Section 4.3 describes the software parts of the system and explains why such architecture is reasonable for this demo.

4.1 Reasons, necessity, use

During the first five years the Laboratory has focused on networked pervasive computing systems. This includes theoretical work as well as various experiments and practical applications for the Laboratory's partners in industry. All together these experiments and practical developments have produced a significant amount of software components for smart dust motes and the know-how for utilizing these motes in ad-hoc networks. So far these solutions have mostly been practical applications and there have not been many experiments that deal with aspects of autonomous computing systems, for example such as emergent behaviour. Therefore there is a need for a software architecture that can be used on smart dust motes and would provide a structured basis on top of which a multi-agent system could be built. Building only practical applications for the industry gives little chance to try out the more exclusive theoretical ideas. One of these ideas is using mediated interaction as a method for partially controlling the emergent behaviour of a system. Creating a new primitive platform for experimenting with these ideas would be the first step towards building more complex and powerful systems in the future.

Another application that the Laboratory could benefit from is a visually effective demo that would illustrate some of the properties of autonomic computing systems. Computer software is hard and uninteresting to promote unless it has a graphical user interface. Oft times there is a need to demonstrate the practical work done at the Laboratory to visiting researchers and officials. So far most of the applications have been developed to work seamlessly in the background as a requirement of their design. There is no monitoring system that could give an overview of what such a system is doing. A demo application that can provide some tangible or visual conformation of its work would be much more convincing and interesting to promote. A special application for this purpose could be built on top of the experimental MAS platform and it would verify the usefulness of the platform.

If this pilot system is going to be used as a test platform for simulating autonomic computing systems, researching mediated interaction and emergent behaviour it should meet certain requirements concerning structure and performance. First of all, it should be structured in a way that supports the changing of different parts or components of the

system without having to make appropriate changes everywhere in the system. This means that the system should be easily decomposable into components that have their own purpose. Components with similar purpose must have a similar structure so that they can replace one another without having to make too many changes to other components. This way the system can be updated and used in future experiments by substituting only the required components with new ones.

Since the system will be used to study complex behaviour and the interactions that are formed between components during runtime it is required that different processes and interactions occurring in the system should be easily monitored. Mechanisms should exist that can log the process information from any specified component and save this information so that it can be analysed later.

From the demo application point of view this system should comprise autonomic components that interact and exhibit emergent behaviour. The objective of the demo should be first to demonstrate the existence of these characteristics and then to exercise control over the system using mediated interaction. The autonomy and the emergent behaviour of the system should be tangible or visible in some way so that it would be possible to see a change in the work of the system once control over these properties is enforced. Control is achieved through mediated interaction which means that there must be an operator component that can mediate the interactions and take orders through some user interface.

The following list clarifies the requirements that have been mentioned so far. The pilot system must:

- be decomposable to standardised components
- be easily assembled from those components
- allow monitoring of processes and interactions
- keep a log of its processes
- display autonomy
- display emergent behaviour
- display control over emergent behaviour
- be visually effective

The main purpose of the pilot system is to provide a structural and component wise basis needed for MAS studies on smart dust motes. The first implementation of the system will be a simple but convincing demo application that has all the basic characteristics of a multi-agent system and utilises mediated interaction to control the system's work. Although the demo is intended for MAS studies it will prove useful in other fields as well.

For starters there has been long debate in the Laboratory about the possible lengths of radio message time delays. Knowing the range of delays would help improve sensor data time stamping and validating the timestamps at different parts of the system. The demo will provide sufficient data to evaluate transport delays of messages in a smart dust based multi-agent system.

4.2 The setup

The following subsection will describe the different software and hardware that will be used in the construction of the pilot system and MAS experiment proposed in this thesis. Since the Laboratory has been working on sensor networks for the past five years most of the components needed for the demo application already exist. This includes different smart dust motes, sensors and various software components.

4.2.1 Smart dust mote hardware

Different smart dust platforms used in the Laboratory form the component basis of the demo system. These include *iris*, *cricket*, *micaz* and *telosB* motes formerly produced by Crossbow and now by Memsic and two motes produced by Defendec *denode* and *iris2*. Table 4.1 gives a brief overview of some characteristics of these smart dust mote platforms. In addition to these sensor motes the Laboratory also uses more powerful *gumstix* motes that stand comparison with smaller computers. For this reason *gumstix* motes are not listed in Table 4.1.

Table 4.1 Overview of smart dust mote platforms

Platform	Producer	Microcontroller	RAM / Flash memory	Sensor board / Sensors
iris	Memsic	ATmega1281	8Kb / 128Kb	sensorboard
cricket	Memsic	ATmega128L	4Kb / 128Kb	onboard/sensorboard
micaz	Memsic	ATmega128	4Kb / 128Kb	sensorboard
telosB	Memsic	TI MSP430	10Kb / 48Kb	onboard/sensorboard
iris2	Defendec	ATmega1281	8Kb / 128 Kb	onboard/sensorboard
denode	Defendec	ATmegaRFA1	16Kb / 128Kb	onboard/sensorboard

For the demo application the Defendec *denode* platform will be used, because it is currently the most widely used platform in the Laboratory and has the best software support. With certain limitations it would be possible to run the demo application software on other motes as well if there were need for it. The *denode* mote has an Atmel Atmega128rfa1 microprocessor with a built in radio chip. It runs an 8 bit architecture at 16MHz, sufficient enough for basic sensing applications. The microprocessor has 16 Kbytes of SRAM memory for running the main program and an additional 128 KBytes of flash memory for storing sensitive data. There are 38 general purpose I/O ports to connect other digital media or sensors and a 10 bit ADC with 8 channels. The ADC can sample at 330 KHz or at 570 KHz with an 8 bit resolution. The built in radio operates at the 2.4 GHz ISM band and is compatible with ZigBee and the IEEE 802.15.4 standard. The transceiver will be discussed in more detail in the following paragraph. The *denode* platform has temperature, humidity, pir (pyroelectric infrared sensor) and photodetector sensors built into the circuit and an extension board connector for other sensor boards and media. It is necessary to mention that the platform has three LEDs which will play an important role in the demo application. The smart dust motes used in the experiment will have an additional sensor board with a single microphone SPM0204HE5 produced by Knowles Acoustics.

Since communication is the central part of any multi-agent system it is justifiable to discuss the mechanisms that provide the communication channel to the agents in more detail. As mention the smart dust motes use the built in radio chip that is compatible with the IEEE 802.15.4 standard. It has 16 channels in the range of 2400 - 2483.5 MHz according to the IEEE 802.15.4 standard. The transceiver allows data rates up to 2 Mbit/s and supports 128 byte frame buffers for message packets. The lower layers of the radio software are supplied by the TinyOS operating system but the message protocol has been developed specially in the Laboratory to be used in sensor networks. It is based on three-tupled expressions and has a hierarchy similar to XML [41]. Each expression consists of an object, subject and value field. The object field describes the data type and the value field holds its value if it has any. The subject field refers to other tupled expressions that are tagged to the current object (expression). For example a temperature reading with situational data coded into such a message would mark temperature in the object field and its value in the value field. The subject field is left empty because temperature is the top level expression. The temperatures situational data such as a spatial interval would have a subject value of one which refers to the first expression in the message which is temperature. That way it is clear that this spatial information describes the temperature

value and no other in case there is more sensory data in the message. Such a protocol is a compact and expressive way of coding data for message exchange in sensor networks.
For more information on *denode* hardware refer to [2, 23].

4.2.2 TinyOS and NesC

TinyOS is an operating system developed for low-power wireless devices. These include (smart dust) sensor networks, smart buildings and other ubiquitous computing systems. TinyOS software is open source and was first developed by Berkeley, University of California in cooperation with Crossbow Technologies and Intel Research. It was intended to suit small remote wireless devices with limited memory and computing power. TinyOS has one control stack and is completely non-blocking, a useful feature in ubiquitous computing systems. TinyOS code is written in NesC programming language and compiled into a small binary using a custom GNU toolchain. NesC language promotes component based programs and the TinyOS package comes with basic components for communication, sensing, actuators and data storage. Today TinyOS is a joint venture of a community of developers from all over the world known under the name the TinyOS Alliance. The latest release is TinyOS 2.1.2 [61, 57].
NesC is an extension to the C programming language. It was specifically designed for the TinyOS operating system. Programs in NesC comprise components that are linked together by interfaces. An interface is a bidirectional interaction channel between two components. It allows threads of control to pass from component to component and back. Interfaces can be split-phase meaning that a component can issue a start command to another component that returns immediately and then wait for the other component to answer back with a finished event once the command is actually completed. A single-phase command in contrast would have the caller wait and halt until the other component finishes and returns. A component can use and provide different interfaces. Provided interfaces represent functions that the component provides for the other components in the form of executable code. They determine part of the components' functionality. Used interfaces are those that the component needs to use to carry out its work. This is what is meant by interfaces establishing a two way connection between components. At compile time the components are statically linked together via their interfaces. The component based structure of NesC satisfies the first two requirements of the pilot system developed in this thesis (see section 4.1). NesC programs run a series of tasks that cannot pre-empt each other and hardware

interrupts that may interrupt tasks and other hardware interrupts. For a more detailed overview of NesC the reader may refer to the NesC programming manual [14].

4.3 Structure of the system

Considering the reusability of components and easy to assemble requirement the software of the demo application will be divided into three parts. These parts are agents, middleware and mediator. Figure 4.1 illustrates how these parts share the common memory of a smart dust mote and shows the way they are linked together. The lilac boxes on Figure 4.1 represent components that come with the TinyOS operating system. These components are ready-to-use meaning that the software developer needs only to link them properly to the necessary places. The real complexity of these interfaces and the lower level components that actually drive the hardware are hidden from the user and are not depicted on the schema. It should be mentioned that the *Sensor* component uses, in addition to TinyOS components, specific software that has been developed in the Laboratory. This software is commonly used in other projects and is not a part of the work done in thesis.

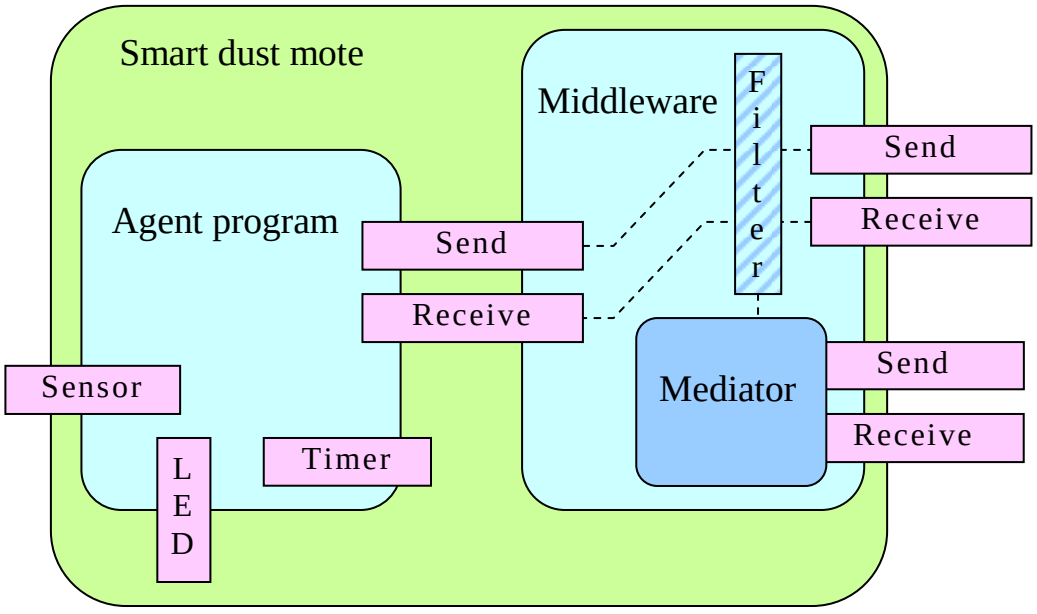

Figure 4.1 Software structure of a smart dust mote

As Figure 4.1 illustrates all the three parts of the multi-agent system have to reside on each smart dust mote. The only exception is the mediator component that has an additional user interface (the operator) situated on some PC. An alternative to this structure is to place the middleware and mediator component on a separate mote. Then an agent program would occupy a whole mote for itself just as might be expected from an autonomous agent. This

would mean that agents have to communicate through middleware motes if there is to be any interaction mediation at all. The problem with such an approach is that middleware motes would have to agree on who is mediating which agent, rising the complexity of the middleware component. If there is to be only one middleware mote then it is most likely that it will become congested with all the messages passing through it. As the size of the multi-agent system grows the middleware mote will become a communication bottleneck paralyzing the system. The simplest and most effective way then is to place the middleware component on each mote where it is directly tied to its own agent.

The agent component comprises all the functionality of the agent. It has access to all the hardware interfaces and services that the mote can offer. These are for example the *Timer*, *LED* and *Sensor* components as depicted on figure 4.1. As far as the agent program knows it is independent and communicates directly with other agents that it comes to contact with. It uses the *send* and *receive* primitive operations that TinyOS offers assuming that these components directly control the radio hardware. The agent is not aware of the middleware component that is situated between it and the real radio driver, nor is it aware of being mediated. Using the *send* and *receive* software between the agent and middleware enables the designer of the system to design agents as if there is no middleware at all. If the middleware component was removed then the agent's *send* and *receive* would be connected directly to the radio hardware without any need for modification. This also means that any agent designed for smart dust motes using TinyOS can be placed in stead of the agent program component and used in this system.

As mentioned the middleware component has control of the motes radio. It links the agents *send* and *receive* operation interfaces to the real radio through a *filter* (see Figure 4.1). The *filter* is used to gather information about the agent as well as to filter incoming and outgoing messages. It is used by both middleware and mediator, but it is more a part of the former than the latter. If there was no mediator component there would still be a *filter* in the middleware. Middleware makes use of the *filter* by monitoring and analysing messages that the agent sends and receives. It can use and store any information that it finds relevant in the messages that pass through it. Middleware might for instance keep track of the agent's state or the number and type of the motes that the agent is communicating with. This information might be used by the mediator to adjust the *filter* or by the developer to monitor and debug the system.

The mediator component is placed inside middleware. This is not a requirement but rather a matter of convenience. The mediator component could reside outside middleware and manipulate the filter from there, but since it usually needs to use the information gathered

by middleware it is sensible to bond the two together. Part of the mediator is in the form of a user interface that may be located on any human-machine interaction utility for example a PC. The operator can issue commands via this user interface to the mediator components in the system and possibly get feedback of the current state. What separates the mediator from middleware is that the mediator is in charge of filtering out unwanted messages and creating messages of its own that must be passed to the agent. Middleware only transfers, monitors, registers and extracts information from the messages while the mediator configures the *filter* to only forward desired messages. The goal of the mediator is to control (at least partially) the behaviour of the system.

Together the three parts form a multi-agent system that can be monitored and controlled to some extent. Chapter 5 describes an experiment that uses a multi-agent system to solve a synchronisation task. It is demonstrated that synchronisation can be achieved by self-organisation and that using the agent-middleware-mediator setup will allow the operator to direct the course of self-organisation in a suitable direction.

5. A distributed computing system experiment

On the theoretical side this thesis has, so far, tried to give a brief overview of agent technologies in computer science. The different aspects of agenthood have been described and the characteristics and perspective of MAS has been discussed. It is evident from this review that the key element and most intriguing aspect of distributed computing is the interaction of agents. Interaction among embedded ubiquitous computers provides additional value to these systems that separate operating components cannot produce. Ironically interaction is also one of the most common sources of failures in these systems. Although the importance of interaction is well acknowledged there is still too little attention devoted to it during the design of computer systems. It is possible that some of this neglect is due to the fact that there are numerous different formalisms for modelling interaction but no comprehensive widely accepted techniques. One of these methods for modelling interactions has been favoured over others in this thesis as well.

On the practical side this thesis wishes to demonstrate a simple multi-agent system. For this task the smart dust motes from the Laboratory will be used. They will be programmed as suggested in chapter 4 composing software of agents, middleware and mediator. Different agents with different features will be used in this system, some of them more interested in cooperation than others. It is suggested that those components qualify as simple agents and that their operation as a system exhibits self-organisation and emergent behaviour. A middleware and mediator component will also be implemented and it will be shown that using mediated interaction can help to stabilise and control emergent behaviour.

This chapter describes a multi-agent system experiment. Section 5.1 proposes a distributed computing problem that can be solved with MAS. The problem is solved using smart dust motes and relevant software in section 5.2. Section 5.3 attempts to model this system formally. In section 5.4 the limitations and prospects of this particular experiment are analysed with suggestions for future experiments and section 5.5 will discuss the possible real life applications for such systems.

5.1 Description of the experiment and similar problems

The objective of the experiment is to demonstrate work of multi-agent systems and the benefits of mediated interaction. Additionally it is desired that the experiment meets the

design principles described in chapter 4 and is practicable using the technology available in the Laboratory. From these requirements the most limiting is the 'must be visually effective' demand. Since the only readily available actuators on smart dust motes are LED lights they will have to play a central role in the experiment. The following experiment is proposed. The demo application will consist of different agents each implemented on a smart dust mote and each blinking their LED light. These agents must synchronise their blinking by exchanging messages about their blink frequency and phase. Each agent is essentially in control of a discrete periodic signal that it can change according to its own desire, and the info it gets from other agents about their signal parameters. The LED's on and off state are linked to this signal, making it possible to observe the progress of the signal without any other monitoring software. As a result of the systems work agents should come to some consensus and settle all at a common frequency and phase.

5.1.1 Clock synchronisation in computer networks

The discussed synchronisation task is very similar to clock synchronisation problems in distributed computing systems. These problems are well known and studied and different methods to achieve synchronisation exist. First of all it must be noted that no two clocks can ever be absolutely synchronised. Jennifer Lundelius and Nancy Lynch prove in [26] that there is a lower bound to how close two clocks of different processes can be synchronised. Their bound depends on the number of processes in the system and stems from the uncertainties of message exchange between processes. Message exchange is always accompanied with uncertainties that can not be known and this limits the precision of synchronisation. Another factor that complicates synchronisation is the indeterminable jitter of hardware clocks on each distributed device. This jitter is caused by the difference of oscillator characteristics of hardware clocks in different environmental and other conditions. It means that the oscillator cycle period will vary due to changes in temperature or electromagnetic interference. Jitter is one of the main reasons for clock drift which means that once synchronised, clocks will slowly drift apart again making it necessary to resynchronise periodically. A lot of scientific work has tackled these problems since the 1970s already. Theoretical work has dealt with different aspects of synchronisation sometimes focusing on initial synchronisation and resynchronisation separately. Some works have assumed that clocks are ideal while others have taken into account two-faced clocks that announce different time to different partners of the system. The proposed methods to synchronise clocks could broadly be divided between two categories: internal

and external clock synchronisation. In external synchronisation the clocks of a system will be synchronised to some external clock while in internal synchronisation the clocks of a system will be synchronised to each other. These works and problems will not be discussed in more detail in this thesis as they clearly exceed the scope. The reader can refer to [24, 52, 8, 7 or 6] for more details on clock synchronisation.

The beginning of the TinyOS and *mica* (named after sheet silicate minerals) type smart dust mote epoch a dozen years ago made ad-hoc sensor networks accessible to everybody. As a result numerous theoretical and practical works have appeared on clock synchronisation in sensor networks using *mica* type smart dust motes. Distributed sensor networks have imposed additional requirements and limitations to time synchronisation and have therefore raised quite some research interest lately. The previous work done on time synchronisation is not that easily applicable anymore because it is computationally too intensive for small embedded computers, or causes too much communication overhead. The dynamically changing structure of sensor networks also raises additional problems. Some of the sensor network specific solutions will now be discussed in more detail as they have many similarities with the experiment in question in this thesis.

In [28] Miklos Maroti et al. introduce a flooding time synchronisation protocol (FTSP) for TinyOS based smart dust motes. This method provides a multi-hop clock synchronisation protocol with low communication overhead and good precision. The protocol works by electing one node from the network to be the root node to whose clock all the other nodes have to synchronise to. This root node broadcasts its local clock and the neighbouring nodes that catch this message compensate their local clocks to follow the root node. Once this is done the neighbouring nodes are in synchronisation and broadcast their local time forward until the furthest nodes in the network are synchronised. This process must then be repeated once in a while to compensate for hardware clock drifts. Maroti et al. show that their protocol can achieve synchronisation precision in the range of ~2 μs.

In [25] the authors propose alterations to the FTSP. They claim that the FTSP is not scalable because in FTSP synchronisation errors grow exponentially with diameter growth. As an alternative they suggest forwarding synchronisation pulses as fast as possible without intermediate computations or waiting as in FTSP.

The flooding time synchronisation protocol uses a root node to whose clock all other nodes synchronise to.

In [59] a reach-back firefly algorithm (RFA) is proposed that lacks a central root node. The authors draw inspiration from fireflies that are known to synchronise their flashing when grouped in swarms. The reach-back firefly algorithm achieves synchronisation in a

completely decentralised manner without any need for network formation or coordination. As a result the algorithm is impervious to changes in network structure and adapts well when new nodes are added or removed from the network. An important distinction of the algorithm is that it only synchronises the phase of each nodes pulsating signal while the frequency of the signal stays the same over the whole network. This means it actually is not suitable for synchronising clocks, but that it rather synchronises the period and phase for firing pulses simultaneously. The problem with RFA is that communication overhead is significantly greater than in FTSP and global precision is worse.

Another decentralised synchronisation algorithm is gradient time synchronisation protocol (GTSP) described in [51]. The goal of this protocol is to provide the best synchronisation between neighbouring nodes while global synchronisation accuracy of the whole network is of second importants. With GTSP the nodes broadcast their local clock rate and current value. Each node then adjusts its own local clock values to the average of all the neighbouring clocks. This method has about the same communication overhead as FTSP.

The experiment carried out in this thesis shares many similarities with the distributed synchronisation algorithms GTSP and RFA. In the experiment the nodes will broadcast their blinking period and phase and listen to others. There will be no root node or any attempts to organise network structure. This makes the system robust to changes enabling new nodes to join and old ones to leave without the work of the system being influenced much. Such a property is important in multi-agent systems and in most pervasive distributed computing systems. Similarly to RFA and GTSP systems the experiment will focus on local interactions and let global synchronisation emerge from the work done on each node locally.

In terms of communication overhead the nodes in the experiment will exchange more messages than FTSP but less than RFA. There will be no danger of increased message collisions that trouble the RFA algorithm when nodes become more synchronised. This is because broadcast intervals in the experiment will be selected randomly for each node at start time and be fixed for the duration of the experiment.

The blinking in the experiment does not synchronise clocks and is in this sense very similar to RFA where synchronicity is achieved instead. The precision of synchronicity in the experiment will be greatly poorer than in any of the other protocols and this is understandable as the goal will not be to maximise performance but rather to settle with what is sufficient enough. The estimated precision is in the range of 10 milliseconds which is enough to make it seem that the LEDs on different smart dust motes are blinking in unison.

5.1.2 Experiment requirements

The purpose of the experiment is to construct a distributed computing system that would conform to the definition of a multi-agent system as much as possible. Behaviour of any MAS contains essential amount of uncertainty. The operation of such a system depends on the dynamic interactions created at run-time. The outcome depends on whether there are enough agents of the right type (for a particular task) in the system, and whether these agents have been able to find each other and communicate. To mimic this uncertainty the experiment must introduce some random values and choices. Five different types of agents are used in the experiment. Some types are keener to accomplish the task of synchronisation while the others will follow more reluctantly. One type of agents used in the experiment will be totally indifferent to the attempts of reaching synchronicity and will unintentionally prevent others from reaching consensus. The type of behaviour each agent will have is determined randomly at start time. This means that each test is unique and it is impossible to say whether the agents will reach synchronicity during the trial run or not.

The blinking interval in the experiment is to be in the interval between half a second to ten seconds. The agents are allowed to change their signal periods only within this interval. The initial period is taken randomly from this range. This is another method to make each test unique. Since the initial conditions are revealed only after start there is no way to predict the resulting synchronic blinking interval. Any true multi-agent system deployed in an environment will be just as unpredictable.

It has already been mentioned that the precision of synchronicity in the experiment is of little importance. The requirement is that the LED lights of each smart dust mote must all seem to a human observer to blink at the same time. This means that synchronicity with an error of less than 100 ms is sufficient enough. The actual interest is whether the system achieves any synchronicity at all as the start conditions and agent types are always unknown. From the point of view of the system's operator there is a system of agents all blinking chaotically with unknown interval change functions. The operator's interest is to get the agents to blink in unison and if possible determine their blinking interval. In the experiment the mediator component is used to achieve such control over an unknown system. The mediator's bounds are that it can only know and use the information that passes through the communication layer of agents because the hypothetical systems operator can only monitor the messages passed between agents. Nothing is known about the agents and thus no agent specific knowledge can be used in the design of the mediator

component. It will be interesting to see whether a mediator component with such bounds can be used to control the operation of the described multi-agent system.

5.2 Setup and components

This section will describe the three software components introduced in section 4.3 in more detail. The specific algorithms the agents use to change their blinking parameters will be reviewed, as well as mediation principles and their implementation methods. Finally different test will be discussed. There will be no formal trials with performance logging and statistical analysis which would produce comparable results. The reason for this is that there are too many different possible configurations for setting up these tests and too many interesting parameters and processes to follow. For instance it would be interesting to observe a test with say 20 agents all of the same type and see how they progress. Or to study a mixture of agents with different types or to separate the 20 agents into two larger groups that are out of communication reach of each other and then bridge this reach with one or two agents. There are too many possible combinations and a formal specification and later analysis for all these tests is out of the scope of this thesis. Some ideas about possible future tests have been given in section 5.4. It is the desire of the author not to make a selection from these interesting trials, as that would only provide incomplete results without a conclusive overview of the real potential of the system. Instead informal tests will be carried out and the results and observations will be discussed generally. The reason for informal testing, beside the fact that a complete trial is too extensive, is that it is currently unknown whether this experimental setup is feasible at all.

5.2.1 Agents in the experiment

There are five different types of agents that will be used in the experiment. They differ from each other by the way they change their LED light's blinking period. The code for each agent type is all comprised in one software component. After the smart dust motes boot one of these types will be selected randomly for use and the rest will be ignored. The agent type can only be changed by restarting the mote which will invoke the random selection function again. The agent types will be marked by different letters from A to E to distinguish them later on in the thesis. A short characterisation of each type follows.

- **type A**

  This agent is the most eager to cooperate with others. As soon as it receives a message from another agent it immediately changes its' blinking period to match the other agent's period. However the timer that controls the blinking is not reset immediately. The new interval is stored and scheduled to become the new period after the timer fires. This means that if the agent receives another message with a new interval before the timer has fired then the period from the first message is overwritten.

- **type B**

  Agents with type B will store all the blinking periods they receive. After the blink timer fires they will calculate the average of all stored periods and assume the result as the new interval and set the timer with this value. The stored period for a particular agent is updated each time a new message from this agent is received. The periods are stored for three timer fired cycles. If the value is not updated in three cycles it is removed and the agent is thought lost. This means that in some cases old values will be used when calculating the new interval. This is considered acceptable behaviour as a good autonomous system must cope with out of date or erroneous data.

- **type C**

  Type C agents are very similar to type B with the only difference that instead of calculating the average of all stored neighbour agent periods they will find the median of the set. At early stages of operation when the difference between agents' periods is greater the average and median will differ a lot. As the agents' periods start to even out so will the median and average values until eventually becoming the same.

- **type D**

  Type D agents are also similar to type B but they are less eager to change their blinking periods. These agents will calculate the average of all the stored neighbour node periods but unlike type B agents they will not apply the result as the new blink interval. Instead they will only change their current interval by a small amount in the direction of the found average value. The size of this small step is randomly chosen but can not exceed 300 milliseconds. If the difference of the current and average value is smaller than the randomly chosen step then the agent will accept the average value as the blink interval.

- **type E**

  This type of agent is not interested in cooperating with other agents at all. It will change its blinking period randomly by steps that are not larger than three seconds. It will broadcast its period just as the other agents do and it will receive messages from others, but ignore their contents completely. Smart dust motes that pick this type after boot will light the yellow LED so these agents could be distinguished from others.

The message exchange of each type of agent is the same. All the agents will broadcast their current period and next LED on event in one single message at regular intervals. The intervals are picked randomly after boot from the range of two to ten seconds. The range has a lower bound because sending messages too frequently will overcrowd the communication band and lead to message collisions and an upper bound because infrequent messaging will make the synchronisation process too slow. Message exchange can not be linked to the LED blink period either, because once the agents become synchronised they will all want to send their messages at the same time which will lead to packet collisions.

Agents with type B, C and D need to store the period and phase values of neighbouring agents. Each agent has a buffer that can hold the necessary info for 50 agents. This is not the limit for these smart dust motes, but for the experiment no more is needed. The stored values are tagged with the senders ID and time-stamped upon arrival so it would be possible to clean the buffer from time to time. Every time a new message from a known neighbour arrives the old value is replaced and the time-stamp renewed. Each value is valid for three blink periods of the host agent. This means that when the agents calculate the average or median period some of the values will be fresher while others will be older. This is not a problem but rather a strength of the system that it can operate with partially out-dated input. The effect of using old data is temporary and the long term process of reaching synchronicity is not affected.

The phase of blinking is altered in a common way by all types of agents. With the blink period info each agent also broadcasts the time remaining in millisecond until it turns its LED light on again. This info is also buffered and used when it is time to set the timer. Every agent finds the average of the next LED on event of all the neighbour agents and compares this to its own LED on value. If they differ more than one milliseconds then the agent alters the timer argument before it sets the timer. The timer argument is usually the blinking period of the system found by each agent type differently. If the phase needs to be

shifted then this argument is either shortened or lengthened by a small amount. This amount is set to 300 milliseconds in the experiment. Phase synchronisation is actually more complicated than synchronising the period of the signal and would be interesting to study and implement with different methods for each agent type. Unfortunately it complicates matters a lot and is too extensive to be tested in this thesis.

All of the agents use random figures to determine certain actions they take. This is because randomness is a simple way to mimic autonomy in a system. Normally there might be some continuous input stream of data that the agent processes to determine its next action. If this stream of data is complex and dynamic enough and takes into account past events then it is hard or impossible to determine and direct the agents actions and the agent could be considered as acting autonomously. Smart dust motes are computationally not powerful enough to process streams or run complex algorithms therefore randomness is used to make them unpredictable and uncontrollable. However generating random numbers is not such a straight forward task as it may seem.

In TinyOS there is a special component for generating pseudo-random numbers. It uses the Park- Miller minimal standard pseudo-random number generator and can produce 32 bit random numbers. This algorithm however needs an initiating value. The results of the algorithm depend on this value and will always be the same for the same value. Therefore a random number is needed to start generating more random values. To solve this problem the agents are equipped with microphones and will read a single value from the microphone input. Since there is always ambient noise around the agents the result of the microphone reading will produce a random value in the range of the ADC. These readings can not produce truly random values since the range of the ADC is limited but they will provide sufficient randomness for the experiment. The pseudo-random number generator of TinyOS is used instead of just sampling the microphone each time a random figure is required because it produces results faster than a reading of the ADC.

So far the characteristics and capabilities of the agents used in the experiment have been discussed but no attempt has been made to attest their true agenthood. In order to determine whether the smart dust motes used in the experiment qualify as agents the different properties of agents listed in section 2.2 should be discussed again. Autonomy has already been reviewed in this paragraph and it is concluded that using randomness as a control signal instead of a complex input stream is sufficient to exhibit autonomy. This is substantiated by the fact that there is no reason (except current hardware limitations) why randomness could not be replaced in the current agent design by a complex control signal. Smart dust mote hardware is what limits the range of agent acting and perceiving as well.

The motes used in the Laboratory are capable of sensing the environment and acting upon it (blinking can be considered an act that influences environment) although one might expect more ingenuity from an agent. But as Ferber suggests (see subsection 2.3.4) not all agents have to be impressive [12]. The agents in the experiment are reactive agents with simple goals and their effectiveness comes from working together not acting on their own. None of the agents in the experiment can be directly controlled or commanded but most of them have a desire to cooperate with others and each has its own goals (the way blinking parameters are changed) in mind all the time. There is no question about the communicative ability of agents as smart dust motes were originally designed to work as a system and the whole essence of the experiment relies on interaction. The two properties that the agents in the experiment do not possess are the ability to reproduce oneself and adaptivity. The ability to reproduce self can be overlooked as it is more a property of software agents and does not apply as strongly to agents with their own hardware. As far as adaptivity and learning goes it is possible to have the agents in the experiment change their type during operation as well not only once after boot. If this change was tied to a learning algorithm or other external stimuli then the agents could exhibit adaptive behaviour.

Considering the above discussion and arguments it is claimed that the smart dust motes used in the experiment have enough of the right properties to be considered agents. They are certainly not very powerful agents capable of doing anything useful on their own but still have the necessary characteristics to work as a system and achieve behaviour that is unattainable to any of them working alone.

5.2.2 Middleware and mediator

The middleware and mediator component filter the interaction of agents. As figure 4.1 indicates the component has control of the physical radio of the smart dust mote and provides communication services to the agent. The agent is unaware that there is another component between it and the radio.

After boot the middleware component immediately starts filtering messages. It scans all the agent's messages and saves the current blink period information for its own use before broadcasting the messages over the network. It also starts blocking all the incoming messages. There are several reasons for this. First of all it is useful to forward all the outgoing messages to the network because it enables the operator to capture the initial blink period and phase information. Currently there is no tool for this but it is likely that future experiments will want to log the initial state of the system before real operation

begins. Secondly it is important to block agents from receiving blink information from others to halt the system's work until everything is set and ready. At an early stage of development it became evident that if messages are not blocked at first then the motes that are switched on will instantly start cooperating and sometimes will reach certain synchronicity before the last ones are turned on. This may be acceptable and normal behaviour in real life systems but it is inconvenient in an experiment since the starting point is left undefined and the before and after state will be hard to compare.

Operation of the system starts after the operator broadcasts an initiating message. This opens the filter and allows agents to receive each others messages. The mediator component will fallow all the packets and monitor the blink period and phase info. If the operator does not assign a blink period for the system then the mediator component will do nothing and let the system settle on its own period and phase. If at one point the operator decides on a fixed period for the system then it sends this value to the mediator component and activates mediation. The mediator component compares the operator sent interval to the agent's current period and starts blocking undesired messages. Since the mediator component does not know how the agents change their periods it must experiment a little. If the agent's current period is shorter than the operator appointed interval then the mediator might only let messages with longer periods pass and block others. It monitors the agent's behaviour and if its period starts to grow longer then it continues with this method. If this has no effect then it will try forwarding only messages with shorter intervals. If there are no suitable messages for a length of time then the mediator can create messages of its own. It will use a new neighbour ID so as not to conflict with other agents and start seeding the agent with suitable blink period information to direct its behaviour. It must be noted that this is not direct control from the operator as the operator has no knowledge of the agent's inner functionality. It can only manipulate the interactions by blocking unwanted messages, creating false communication and parsing the exchanged messages.

5.2.3 Experiment results and observations

During development the system was tested on smart dust mote networks of sizes 6, 13 and 24 nodes. The six node setup was most useful for debugging purposes because it was still possible to keep track of all the interactions while not loosing the interesting properties (self-organisation, emergence) of the system. The 13 node setup was used as an early test of communication resilience and the 24 node setup as the final test in the late stages of

development. Despite complications during development the result and final setup of the system work as required on the 6 and 13 node test, exhibiting emergent behaviour and self-organisation in a visible way. The 24 node test does not reach system wide synchronicity most likely because of frequent message collisions and a drop in communication efficiency. The largest number of nodes to reach synchronised blinking was 18. It is interesting to note that from the 24 node setup 16-18 nodes would usually reach synchronicity while others are left to stray. This section will discuss some of the properties of the developed system while leaving improvements and further research problems to section 5.4 and application areas to section 5.5.

The most interesting properties are with out a doubt self-organisation and emergent behaviour. In the experiment these properties result from the interaction of agents and essentially describe the same phenomenon. Individually the agents do not appear to be doing anything useful else than blinking a LED light. They do not think about cooperating with any other agents, they merely observe their environment (other agents) and adjust their blinking accordingly. The agent has no knowledge of what the other agents are doing or that there is an operator of the system who wishes to see all the agents work together. The experiment shows that these agents will synchronise their blinking without any central coordination from any of the agents or the operator. This is even true for the case when no mediation is applied and the system is left to linger. The only case when the system can not organise on its own is when there is at least one agent with type E in the system. This agent is not affected from the interaction with other agents while it has effect on other agents. Since agent E changes its blink interval constantly and in no relation to the other agents, the system never reaches synchronicity. If agent with type E were to keep its blink period constant and there was only one agent of this type then the system should eventually synchronise to this agent. If there were more agents of type E then no synchronicity can be reached. All these processes were observed in the tests as well with the exception of using a special type E agent with a constant blinking interval.

Although achieving self-organisation is an accomplishment in its self there is still little use of such a system. This is because the course and final outcome of an emergent process is always undetermined. In the experiment it would be impossible to say whether the system reaches synchronicity if agents of type E were not specially marked. An observer of the system might wait indefinitely long and see no emerging organisation if there were concealed type E agents in the system. The observer would conclude that the system does not exhibit self-organisation but as soon as the disobedient agents were to be removed the system would organise. It is also not possible to determine the final state of the

organisation. In the experiment the final blinking period and phase are never predictable because of the unknown interactions that the agents form. Suppose the starting state and parameters of all the agents are the same in two different tests then the two different runs of the system would still produce different final states. This is because if even one message is lost during the test or arrives a little later in the second test then the final result will be influenced. This naturally makes using such systems in fault sensitive applications very problematic.

Mediating interactions between agents is one way of making the system more predictable and perhaps even partially controllable. The experiment shows that it is possible to create a middleware-mediator application that can direct the course of self-organisation and emergent behaviour as well. The work of this component is completely based on the information contained in the messages that agents exchange. No knowledge of agent types or their operation directives is required. The mediator first tries to establish how its host agent reacts to different input. It blocks the agents communication and feeds it with sample messages and monitors the agents output. This establishes whether the agent reacts to other agents and helps filter out the ones who refuse to cooperate. Once this is done communication is opened again and the system is left to self organise while the mediator component starts to filter out undesired messages. In all the tests where mediation was used the system always reached synchronicity even in the presence of agents of type E. These agents were isolated by middleware and left unsynchronised while the rest organised themselves. Mediation also proved the possibility of directing the course of organisation towards a desired final state. In all the experiments where a specific blink period was set by the operator as the desired goal, the agents would synchronise to that period each time. It was also observed that if the operator changed this period after initial synchronicity had been achieved the system would loose its synchronicity temporarily and regain it again with the new interval. The system looses synchronicity temporarily because the agents change their blink periods differently in different amounts at a time.

On the whole the experiment can be considered a success. At first a system of agents was created and it was shown that this system is capable to self-organise. Then a middleware-mediator component was designed that could direct the course of self-organisation and allow the operator of the system to determine the accumulation point (period and phase).

5.3 A model of the system

The pilot system constitutes a distributed real-time system that can be described in terms of the Q-model and stream computing as discussed in section 3.4. An agent can be viewed as a Q-model process and its interactions, input and output can be modelled by data streams. Figure 5.1 illustrates two agents that exchange information streams with each other and the environment. The environment in case of the pilot system comprises other agents and a systems operator. Figure 5.1 also shows how mediator components mediate streams for their host agents. The streams between two physical smart dust motes are a different colour than the streams inside the motes. This is to show that mediator components alter interactions and filter or change the data streams passing through them.

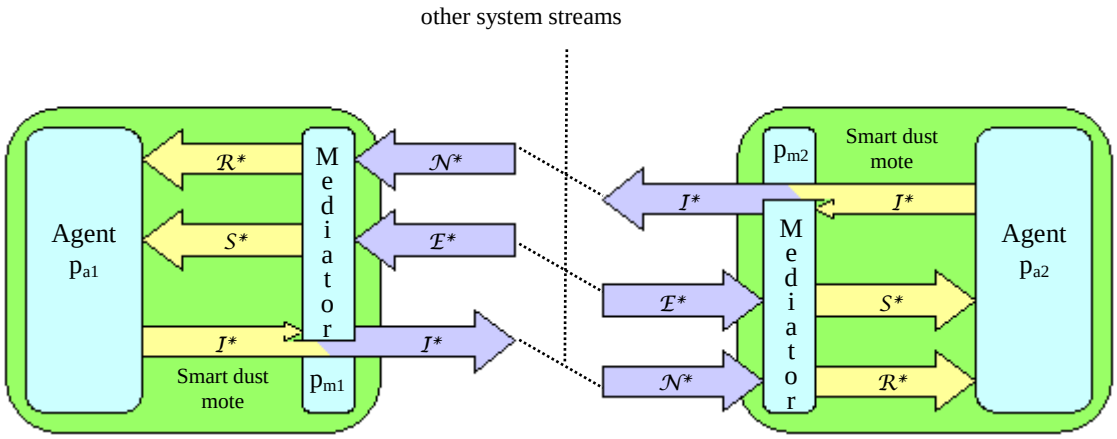

Figure 5.1 Overview of different streams in the pilot system

In the pilot system each agent has many interaction partners that will be called neighbours. With each neighbour is associated a set of streams N^*. The set cannot be empty and must have at least one stream element N. The set of streams and one stream of this set is defined as follows

$$N^* = \{N_1,..., N_k \mid \ k \geq 1\} \qquad (5.1)$$

$$N = \{n_1 \lhd ... \ n_s \lhd \langle \ \rangle \mid \ s \geq 1\} \qquad (5.2)$$

Empty streams $\langle \ \rangle$ will not be allowed for the moment, although their existence in a distributed artificial system is possible. Similarly a set of streams will be defined for the interaction channel between the mediator and operator component. The set will be denoted E^* and its element as E

$$E^* = \{E_1,..., E_l \mid \ l \geq 1\} \qquad (5.3)$$

$$E = \{e_1 \lhd ... \ e_t \lhd \langle \ \rangle \mid \ t \geq 1\} \qquad (5.4)$$

These two sets N^* and E^* are mapped to the mediator component of each agent. The mediator itself will be viewed as process p_m (3.1) in Q-model terms. The stream sets form a part of the domain of definition for process p_m. The value range of p_m is made up of two different sets of streams R^* and S^*. R^* will represent a neighbours set of streams N^* that the mediator has filtered and altered. S^* denotes the set of streams that the mediator additionally wishes to communicate to the agent. The set S^* will be completely produced by the mediator. These sets and their corresponding stream elements are

$$R^* = \{\mathrm{R}_1,...,\mathrm{R}_k \mid \ k \geq 1\} \tag{5.5}$$

$$\mathrm{R} = \{r_1 \lhd ... \ r_s \lhd \langle \ \rangle \mid \ s \geq 1\} \tag{5.6}$$

$$S^* = \{\mathrm{S}_1,...,\mathrm{S}_m \mid \ m \geq 1\} \tag{5.7}$$

$$\mathrm{S} = \{s_1 \lhd ... \ s_u \lhd \langle \ \rangle \mid \ u \geq 1\} \tag{5.8}$$

Sets R^* and S^* make up part of the agents input. An agent is viewed as a process p_a. The output of this process is a set of streams I^*. The set is channelled to the agent's mediator that uses this set as an input and to the rest of the system unaltered. It must be stressed that the mediator does not filter or modify in any way the output stream of the agent. I^* and I are defined as

$$I^* = \{\mathrm{I}_1,...,\mathrm{I}_n \mid \ n \geq 1\} \tag{5.9}$$

$$\mathrm{I} = \{i_1 \lhd ... \ i_v \lhd \langle \ \rangle \mid \ v \geq 1\} \tag{5.10}$$

The whole domain of definition for the mediator process $\mathrm{p_m}$ is made up of stream sets from many neighbours, possibly many operators and the host agent's own output. To simplify modelling processes and their inputs and outputs calligraphic letters will be used to note a collection of similar stream sets. $\mathcal{N}^*$ will denote a collection of neighbour node stream sets, $\mathcal{E}^*$ will denote a collection of operator stream sets and $\mathcal{I}^*$ a collection of agent output stream sets as follows

$$\mathcal{N}^* = N_1 * \cup ... \cup N_x *, \ \ x \geq 0 \tag{5.11}$$

$$\mathcal{E}^* = E_1 * \cup ... \cup E_y *, \ \ y \geq 0 \tag{5.12}$$

$$\mathcal{I}^* = I_1 * \cup ... \cup I_z *, \ \ z \geq 0 \tag{5.13}$$

Similarly the streams modified and forwarded by mediator to its host agent will comprise a number of different sets and the collection of these sets will be denoted $\mathcal{R}^*$ and S^* as follows

$$\mathcal{R}^* = R_1 * \cup \ldots \cup R_x *, \quad x \geq 0 \qquad (5.14)$$

$$S^* = S_1 * \cup \ldots \cup S_w *, \quad w \geq 0 \qquad (5.15)$$

In Q-model terms the collection of streams $\mathcal{N}^*$, $\mathcal{E}^*$ and $\mathcal{I}^*$ will make up the domain of definition *dom* p_m for the mediator process p_m and the collections $\mathcal{R}^*$ and S^* will make up its value range *val* p_m. A channel between the mediator and its agent is σ_{ma} (3.2) and defines a part of the domain of definition *dom* p_a for agent process p_a (3.1). *dom* p_a is complete if its own output $\mathcal{I}^*$ is added to its input along with other data streams from sensors. The value range *val* p_a of process p_a is made up of the collection of streams $\mathcal{I}^*$. Figure 5.2 depicts agent and mediator (same as in figure 5.1) as Q-model processes and channels. The graphical notation abides that of [36].

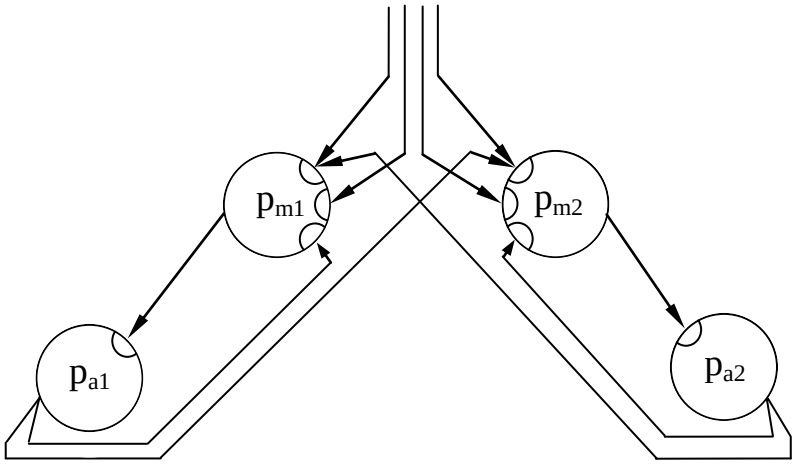

Figure 5.2 Overview of pilot system in Q-model notation

Processes p_m denotes a mapping from input to output. This mapping is defined as a multi-stream function $\mathcal{T}$ for the mediator process p_m as follows

$$\mathcal{T} : (\mathcal{I}^*, \mathcal{N}^*, \mathcal{E}^*) \rightarrow (\mathcal{R}^* \triangleright r, S^* \triangleright s) \qquad (5.16)$$

where $\triangleright$ is the concatenation of two streams (3.6). $\mathcal{R}^* \triangleright r$ means that for each set of streams R^* in collection $\mathcal{R}^*$ the operation $R^* \triangleright \langle r \rangle \rightarrow R^*$ has been carried out to create a new stream R in R^*. The same follows for $S^* \triangleright s$.

Similarly for agent process p_a a multi-stream function $\mathfrak{K}$ is defined as

$$\mathfrak{K} : (\mathcal{R}^*, \mathcal{I}^*, S^*) \rightarrow \mathcal{I}^* \triangleright i \qquad (5.17)$$

So far nothing has been said about the set of symbols that can be used to create streams. This set is a subset of real numbers for the pilot system. All streams N, E, R, S and I are constructed of symbols of set $O \in \mathbb{R}$.

The experiment is a special case of stream computing where all the streams compose only one element. In this case s = t = u = v = 1 and the streams N = n_1, E = e_1, I = i_1, R = r_1, S = s_1. The set of symbols O is a subset of natural numbers and defined O = {512, ..., 10240}. Also y = z = w = 1 which means there is only one operator stream set E^*, one control stream set S^* and one agent output stream set I^*. The number of neighbours a node may have is $0 \leq x \leq 23$. As a result the mediator function (5.16) for one time instance of the mediator process p_m takes on the following form

$$\mathfrak{T} : (\langle i \rangle, \mathcal{N}^*, \langle e \rangle) \rightarrow (\mathcal{R}^* \triangleright r, S^* \triangleright \langle s \rangle) \qquad (5.18)$$

and similarly the agent function (5.17) is altered

$$\mathfrak{K} : (\mathcal{R}^*, \langle i \rangle, \langle s \rangle) \rightarrow I^* \triangleright \langle i \rangle \qquad (5.19)$$

As a result the mediator function can be described for one time instance t of $T(p_m)$ in the following way

$$\mathfrak{T} : r = 0, s = e \begin{cases} i < e \;\&\; n_x < i \\ i > e \;\&\; n_x > i \end{cases}; r = n_x, s = 0 \begin{cases} i < e \;\&\; n_x > i \\ i > e \;\&\; n_x < i \end{cases} \qquad (5.20)$$

where e denotes $\langle e \rangle \in E^*$, s denotes $\langle s \rangle \in S^*$, i denotes $\langle i \rangle \in I^*$, r denotes $\langle r \rangle \in R^*$ and n_x denotes $\langle n \rangle \in N_x^*$ of x^{th} neighbour.

There are five different agent processes used in the pilot system which means there are five different multi-stream agent functions. Some of these functions are quite complicated and will not be presented here. For the sake of completeness of this section one agent multi-stream function should be reviewed. For agent type B the multi-stream function (5.19) is calculated by

$$\Re : \mathrm{i} = \frac{i + s + \sum_{i=0}^{x} r_i}{x + w + z} \qquad (5.21)$$

where i denotes $\langle \mathrm{i} \rangle \in I^*$, s denotes $\langle \mathrm{s} \rangle \in S^*$ and $\mathrm{r_i}$ denotes $\langle \mathrm{r} \rangle \in R_i^*$ of i^{th} altered neighbour stream. The number of neighbours is x, the number of output streams is w and the number of control streams is z.

5.4 Discussion and further problems

The system designed in this thesis presents very promising characteristics but is still as the title suggests a pilot system. The current version has a few noteworthy limitations and lots of room for improvements and future development. One considerable drawback for example is that in the experiment all the agents are within communication range of each other. That means that each agent can influence every other agent in the system. In real applications this is seldom the case. It would be interesting to observe for example what happens in systems that form long chains where each agent can only communicate with a few nearest neighbours. There are other intriguing questions as well and three follow-up experiments will be briefly discussed now.

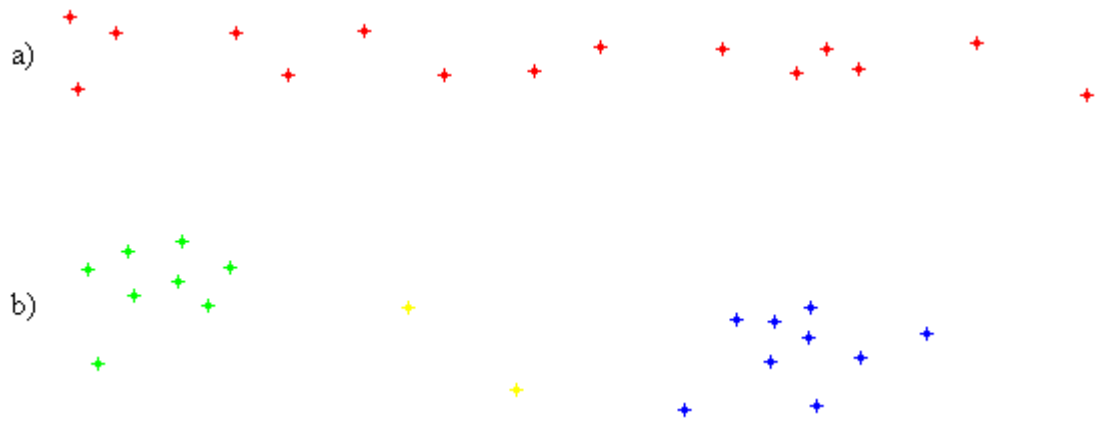

Figure 5.3 Agent distribution into a) a chain and b) two sets with a narrow bridge

Figure 5.3 depicts agents (marked by the stars) distributed over some area. In figure 5.3 a) they are stretched into a line and in b) two groups of agents (blue and green) are separated from each other except for a communicational bridge created by the two yellow agents. It is assumed that in a) the agents communicational range spans to the two-three nearest neighbours and no further. In b) the blue and green agents are out of reach of each other, but both groups can communicate with their nearest yellow agent who in turn can

communicate with each other. As a follow-up experiment it would be interesting to observe in a) how long it takes the chain of agents to reach synchronicity or whether it is achieved at all. It stands to reason that self-organisation in such a displacement takes considerably longer than in the case when every agent is in communicational reach of the others. In a chain setup an agent at one edge can influence the other edge only through the intermediate agents. Influence is much slower and weaker in this case, but exists none the less and should eventually synchronise the system. In b) it remains unknown whether the agents are capable of organising at all. The communicational bridge between the two yellow agents might be too narrow to bring the two larger groups to a common phase and interval.

In the current experiment the blinking period and phase were synchronised but it was only possible to mediate the period. Phase was left out of mediation and control of the operator in order to keep things simple. However there would be great value for the operator to be able to set the blinking phase as well. In this case the system could be used to synchronise clocks with the possibility of setting time from outside of the system. This is a feature that the latest sensor node network clock synchronising algorithms do not support. The new system could maintain synchronicity among itself and would only need periodic updates of the real operator time to compensate hardware clock drift. Therefore it would be useful to implement a version of middleware that could mediate blink phase as well.

Time is an interesting feature to study in any distributed system. As a follow-up experiment it would be useful to determine some of the temporal characteristics of the current setup. First of all the total time it takes to synchronise a network with *n* number of nodes. Of course this will always be different even for the same *n* value but a probabilistic range could still be found along with a relation to the size of the system. Another area to develop is the precision of synchronicity. Currently the system settles in the range of a 100 milliseconds, but there is definitely room for improvement. Using some of the methods from [28] it might be possible to reach accuracy in the range of 100 microseconds which would make the system suitable for clock synchronisation.

5.5 Application areas

At first when contemplating about a multi-agent system experiment for this thesis inspiration was drawn from the idea of a multi-agent based control system for configuring the traffic lights system of a city. As it turns out this is not a novel idea and previous research has already been done on the subject. Currently there are numerous ways used to

manage traffic networks. One method is to run pre-programmed controllers at each junction and keep them at fixed time intervals in order to create "green waves" in certain directions. The drawback of this approach is that the system is too rigid and cannot react to changes in traffic flow. The other approach uses traffic flow sensitive controllers making the system more dynamic but also very complex and unmanageable at larger numbers of junctions. In [4] the authors propose using traffic flow responsive controllers and smart controllers that are capable of emergent behaviour and self-organisation. The authors describe a model that could be used for such a system, but present no experimental results of its efficiency. The belief in this thesis is also that a self-organising smart network of traffic light controllers could smooth traffic flow and minimise wait-times at junctions. The developed pilot system could be a start in that direction. Understandably controlling traffic networks is a far more complicated task than synchronising blinking, but the basic ideas used in the experiment can be applied to traffic control as well. For instance the junctions would stay in "green wave" synchronisation as a result of self-organisation and mediated interaction could be used to adjust this synchronicity according to changes in traffic flow. Since these traffic-light systems ideas are currently actively studied, further investigation would certainly be beneficial.

Another area of use for the pilot system is anywhere where a distributed system needs to keep a system wide level or degree of some parameter. In a chemical plant for instance there might be a need to blend different substances together evenly over some process and at a fixed relation. Suppose this mixing is done in large quantities and several nozzles dose the different substances together. It is important to keep a certain ratio of these components which means if the flow of one of the solution increases then the others must adjust their flow as well. A self-organising system could keep this process under control without the need for constant surveillance. Or perhaps in another application there is need to keep a surface levelled at some offset. Suppose there are numerous actuators and sensors controlling different areas of the surface possibly lifting or lowering the area as need be. It would be easy for the operator to simply set the offset of the surface and have all the smart controllers self-organise at the specified level. Such a system would not need a clever self-organising system of course if the actuators were all homogenous. Then the operator could easily instruct each point directly. Unfortunately these systems are seldom homogeneous. Each area may have its own adjustment algorithm, specific local requirements and constraints and usually actuators and sensors from different manufacturers with different software versions. In this case it is difficult for the operator to keep track of all the details and a self-organising system is much more efficient.

6. Conclusion

The objective for the first half of the thesis was to introduce the relatively new computing paradigms of agents and interactive computing. For this purpose a thorough theoretical overview of the subject matter was given. Owing to the fact that a complete analysis of either of these research fields greatly exceeds the scope of the thesis a selection had to be made. Consequently the survey focused on general properties of agents and disregarded some topics of multi-agent systems. Special attention was put on reactive agents as used in distributed artificial systems. The different characteristics of agents such as autonomy, situation awareness, self-x properties and communicativeness were discussed extensively along with different definitions and understandings of the nature of agents. The subject of interactive computing had to be truncated as well. A substantial part of the topic was dedicated to interactions and its importants in modern computing applications and theoretical models. Different types of interactions, including mediated interactions, were reviewed thoroughly followed by an analysis of emergent behaviour. Emergence of a system wide behaviour that cannot be decoupled in terms of component behaviours is perhaps the most important feature of interactive computing. It has also proven to be a great challenge of computer science because current theory and practice lack good methods to predict, discover and control this phenomenon. A short introduction to environments in multi-agent systems has been given, seeing how environments have a significant role in the design and operation of embedded computer systems. The theoretical survey in this thesis was concluded with a short review of Q-models and multi-stream based interactive computing. A deeper discussion of persistent Turing machines, the π-calculus and other models of interactive computing would have exceeded the scope of this thesis.

The second objective of this work was to construct a distributed computing system that would demonstrate the characteristics of agents and interactive computing. The necessary software components for this system were developed and an experiment was carried out using smart dust motes available at the Laboratory for Proactive Technologies to prove the feasibility of the design. The distributed system had to compose partly autonomous components capable of interaction and exhibit emergent behaviour as a result of these interactions. A secondary goal was to direct emergent behaviour with the help of a smart mediator component that monitors and filters the exchange of messages between agents. The concept of an active middleware responsible for managing dynamic interactions in distributed systems had been proposed already some time ago at the Laboratory but until now no working prototypes have been implemented. The system developed in this thesis

proved this concept feasible and partial control over emergent behaviour was achieved and demonstrated in the experiment. The experiment involved synchronising the blinking of LED lights over a network of smart dust motes. Each mote was autonomously in control of its LED and system wide synchronicity was reached through cooperation and mediated interactions. The pilot system and the synchronisation experiment show great promise for further development. Improving the performance of synchronisation could be a good starting point. An analysis of other similar works suggests that the pilot system can reach equal performance levels with these systems by enhancing precision and estimating message uncertainties better. Examining the effect of the structure of the network on interactions is another interesting topic for development. As for practical applications using the features and concepts of the pilot system for control of traffic lights systems or simple set point control systems seems feasible.

Resümee

Magistritöö käsitleb kaasaegseid arvutussüsteeme, nende eripära ja probleeme ning uusi mudeleid ja teooriaid. Arvutitehnoloogia tormiline areng eelmise sajandi viimastel kümnenditel ja uuel sajandil on oluliselt muutnud arvutusprotsessi ja laiendanud arvutite poolt lahendatavate probleemide ringi. Arvutusi ei tehta enam ühel seadmel vaid hajutatult paljudest arvutitest koosnevates võrkudes. Selle tulemusena ei ole seni kehtinud Church-Turing teesile tuginev algoritmiline arvutusmudel piisav, et kirjeldada suurtes arvutivõrkudes aset leidvaid protsesse. Arvutiteaduse ringkondades ollakse üksmeelel, et uus teooria peaks paremini kujutama hajussüsteemides toimuvaid arvutite vahelisi interaktsioone, kuid üldist ühiselt aktsepteeritud mudelit ei ole veel leitud.

Antud tööl on kaks peamist eesmärki. Esiteks soovib magistritöö analüüsida ja anda ülevaate kaasaegsete arvutisüsteemide põhilistest eripäradest ja lahendamata probleemidest. Vaatluse alla tulevad hajusarvutus ja multi-agentsüsteemid, nende komponendid ja komponentide vahelised interaktsioonid. Teine eesmärk on luua targa tolmu kübemete põhine katsesüsteem, mis demonstreeriks hajussüsteemides esinevat ilmnevat käitumist. Antud katsesüsteemi lisaülesanne on tutvustada vahendatud interaktsiooni ja selle kasulikkust ilmneva käitumise avastamisel ja juhtimisel.

Teoreetiline osa algab ülevaatega agentidest ja agentteooriast. Agendi mõiste on algselt pärit tehisintellekti valdkonnast, kuid on viimastel kümnenditel laialt kasutusele võetud üldises arvutiteaduses ja mujal komplekssüsteemide modelleerimisel. Arvutusteoorias on agent abstraktsioon kujutamaks teatud omadustega aktiivset süsteemi komponenti. Agendi kontseptsiooni kasutatakse, kui ei osata või ei soovita süsteemi komponente detailselt kirjeldada. Agentidel on palju omadusi nagu näiteks autonoomsus, situatsiooniteadlikkus, iseorganiseeruvus, suhtlemisvõimelisus jm, mida soovitakse süsteemis ekspluateerida, ilma et peaks neid omadusi ja komponente täpselt formuleerima. Hajusarvutussüsteemide kirjeldamisel kasutatakse sageli just agentsüsteeme, sest need keskenduvad süsteemi komponentide kirjeldamise asemel muude süsteemi nähtuste analüüsile.

Magistritöö teoreetilise osa teine pool arutleb arvutisüsteemi interaktsioonide ja nende omaduste üle. Interaktsioone on üldistatult kahte liiki: otsesed ja kaudsed. Otsene suhtlus toimub kahe või enama osapoole vahel, kui kõik osalejad teavad üksteist ja sõnumid on suunatud kindlatele adressaatidele. Kaudse suhtluse puhul edastatakse sõnumid mingisse keskkonda (suhtluskanalisse) ilma kindla adressaadita ja suhtlus toimub anonüümselt nende osalejate vahel, kes on võimelised sõnumit vastu võtma ja lugema. Interaktsioonide jälgimine ja kirjeldamine on hetkel üks suuremaid väljakutseid hajussüsteemide

mõistmisel. Arvutisüsteemide komponentide osalise autonoomia ja süsteemi ümbritseva keskkonna ettearvamatuse tõttu pole võimalik ennustada kõiki tekkivaid interaktsioone. See muudab selliste süsteemide formaalse kirjeldamise keeruliseks ja põhjustab süsteemi ilmnevat käitumist. Ilmnev käitumine on süsteemi käitumine viisil, mis ei ole tuletatav süsteemi osade käitumistest ega nende otsesest kombineerimisest. Ilmneva käitumise käsitlemiseks on välja pakutud vahendatud interaktsiooni mõiste. Vahendatud interaktsioon võimaldab jälgida dünaamiliselt tekkivaid interaktsioone süsteemi töö käigus ja sobib teatud tingimustel ilmneva käitumise avastamiseks ja juhtimiseks.

Magistritöö teine eesmärk on koostada katsesüsteem Proaktiivtehnoloogiate teaduslabori (Labor) jaoks, mis kujutaks üht hajusarvutussüsteemi. Loodav süsteem peab käsitlema ilmnevat käitumist ja looma meetodid vahendatud interaktsiooni kasutamiseks. Eraldi nõue oli, et süsteemis esinev ilmne käitumine ja selle osaline kontroll ning juhtimine oleks visuaalselt jälgitav. Süsteemi koostamiseks olid Laboris kasutada targa tolmu kübemed. Targa tolmu kübemed on pisikesed sardarvutid, mis on varustatud mõningate anduritega ja on võimelised omavahel raadio teel suhtlema. Katse täpne sisu ei olnud paika pandud ja on autori välja mõeldud.

Katsesüsteemi nõuete täitmiseks sai loodud targa tolmu kübemete põhine arvutussüsteem, mille eesmärk oli sünkroniseerida LED tulukeste vilkumine. Süsteem koosnes kuni 24 kübemest, mis vilgutasid üht oma LED tulukest. Igal süsteemi komponendil oli autonoomne kontroll oma LED tulukese üle ja keegi teine seda otseselt juhtida ei saanud. Süsteemi eesmärk oli ühtlustada tulukeste vilkumise sagedus ja faas läbi omavahelise suhtluse. Seejuures mõned süsteemi komponendid olid rohkem koostööaldid kui teised ning esines ka komponendi tüüpe, mis ei huvitunud koostööst üldse. Katse alustamisel määrati igale komponendile tema tüüp juhuslikult ja samuti valiti juhuslikult tema vilkumise algsagedus ja -faas. Juhuslikkuse tõttu polnud ühegi katse puhul ennustatav, kas süsteem saavutab sünkroonse vilkumise või mitte. Katse jaoks loodi ka vahendatud interaktsiooni realiseerivad tarkvaramoodulid. Need moodulid vahendasid komponentide suhtlust ja filtreerisid sõnumeid. Vahendatud interaktsiooni tulemusena saavutas süsteem alati sünkroniseeritud vilkumise ja selle sagedust ja faasi oli võimalik eraldi määrata. Antud süsteemi ja vahendatud interaktsiooni kirjeldati ka formaalselt kasutades Q-mudeli ja lõimearvutuse meetodeid.

Magistritöö käigus vaadeldi hajusarvutussüsteemide erinevaid omadusi ja nende süsteemide modelleerimisel tekkivaid probleeme. Suurema tähelepanu all olid agendid ja agentpõhised mudelid ning interaktsioonid. Töö tutvustas ilmnevat käitumist ja vahendatud interaktsiooni mõistet, mis võiks olla lahenduse ilmneva käitumise avastamiseks ja

juhtimiseks. Teoreetilistele teadmistele tuginedes koostati hajusarvutuse katsesüsteem. Katsesüsteem demonstreeris olulisemaid hajusarvutussüsteemi nähtusi ja omadusi ning tõestas vahendatud interaktsiooni sobivust ilmneva käitumise juhtimiseks.

REFERENCES

[1] Agha, G. A. ACTORS: A Model of Concurrent Computation in Distributed Systems Massachusetts Institute of Technology, Cambridge, MA, 1985.

[2] Atmel, ATmega128RFA1 [WWW] www.atmel.com/Images/doc8266.pdf (20.11.2012)

[3] Blackhurst, J. L., Gresham, J. S., Stone M. O. The Autonomy Paradox: Why 'unmanned systems' do not shrink manpower needs [WWW] http://www.armedforcesjournal.com/2011/10/7604038/ (05.12.2012)

[4] Branke, J., Mnif, M., Müller-Schloer, C., Prothmann, H., Richter, U., Rochner, F., Schmeck, H. Organic Computing – Addressing Complexity by Controlled Self-Organization – *Proceedings ISoLA 2006 Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, Paphos, Cyprus, November 15-19 2006,* 2006, 185-191.

[5] Brooks, R. A Robust Layered Control System for a Mobile Robot – *IEEE Journal of Robotics and Automation* 1986, 2 (1), 14-23.

[6] Cristian, F. Probabilistic Clock Synchronization – *Distributed Computing*, 1989, 3 (3), 146-158.

[7] Dolev, D., Halpern, J. Y., Simons, B., Strong, R. Dynamic Fault-Tolerant Clock Synchronization – *Journal of the ACM*, 1995, 42 (1), 143-185.

[8] Dolev, D., Halpern, J. Y., Strong, H. R. On the Possibility and Impossibility of Achieving Clock Synchronization – *Journal of Computer and Systems Sciences*, 1986, 32 (2), 230-250.

[9] Dosch, W., Stümpel, A. Transforming Stream Processing Functions into State Transition – *Machines Software Engineering Research and Applications – Lecture Notes in Computer Science*, 2006, 3647, 1-18. [*Online*] SpringerLink (03.12.2012)

[10] Endsley, M. R. Designing for Situation Awareness in Complex Systems – [WWW] http://www.satechnologies.com/Papers/pdf/SA%20design.pdf (19.11.2012)

[11] Feng, Y. H., Teng, T. H., Tan, A. H. Modelling Situation Awareness for Context-Aware Decision Support – *Expert Systems With Applications*, 2009, 36 (1), 455-463.

[12] Ferber, J. Multi-Agent Systems : An Introduction to Distributed Artificial Intelligence. Harlow : Pearson Education Limited, 1999.

[13] Franklin, S., Graesser A. Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents – *ECAI'96 Workshop on Intelligent Agents III Agent Theories, Architectures and Languages Budapest, Hungary, August 12–13, 1996 Proceedings. – Lecture Notes in Computer Science*, 1997, 1193, 21-35. [*Online*] SpringerLink (19.11.2012)

[14] Gay, D., Levis, P., Culler, D., Brewer, E. nesC 1.1 Language Reference Manual [WWW] http://nescc.sourceforge.net/papers/nesc-ref.pdf (20.11.2012)

[15] Goldin, D., Keil, D. Towards Domain-Independent Formalisation of Indirect Interactions – *WET ICE 2004. 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Modena, Italy, 14-16 June 2004 , Proceedings*, 2004, 393-394. [*Online*] IEEE Xplore (27.11.2012)

[16] Goldin, D.Q., Smolka, S. A., Attie P. C., Sonderegger E. L. Turing Machines, Transition Systems and Interaction – *Information and Computation*, 2004, 194 (2), 101-128. [*Online*] ScienceDirect (25.11.2012)

[17] Golding, D., Wegner P. The Church-Turing Thesis – Breaking the Myth – *New Computational Paradigms – Lecture Notes in Computer Science*, 2005, 3526, 152-168. [*Online*] SpringerLink (25.11.2012)

[18] Hayes-Roth, B. An Architecture for Adaptive Intelligent Systems – Artificial Intelligence, 1995, 72 (1-2), 329-365.

[19] Hewitt, C. Viewing Control Structures as Patterns of Passing Messages – *Artificial Intelligence*, 1977, 8 (3), 323-364.

[20] IBM, Autonomic Computing: IBM's Perspective on the State of Information Technology. [WWW] http://www.research.ibm.com/autonomic/manifesto /autonomic_computing.pdf (19.11.2012)

[21] Jennings, N. R. On Agent-Based Software Engineering – *Artificial Intelligence*, 2000, 117 (2), 277-296.

[22] Jennings, N. R., Sycara, K., Wooldridge, M. A Roadmap of Agent Research and Development – *Autonomous Agents and Multi-Agent Systems*, 1998, 1 (1), 7-38.

[23] Knowles Acoustics, Enhanced RF Protected "Mini" SiSonic Microphone Specification [WWW] https://www1.elfa.se/data1/wwwroot/assets/datasheets/ 03010451.pdf (03.12.2012)

[24] Lamport, L., Melliar-Smith, P. M. Synchronizing Clocks in the Presence of Faults – *Journal of the ACM*, 1995, 32 (1), 52-78.

[25] Lenzen, C., Sommer, P., Wattenhofer, R. Optimal Clock Synchronization in Networks – *SenSys '09 Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, Berkeley, California, USA, November 4-6 2009*, 2009, 225-238.

[26] Lundelius, J., Lynch, N. An Upper and Lower Bound for Clock Synchronization – *Information and Control*, 1984, 62 (2-3), 190-204.

[27] Maes, P. Artificial Life Meets Entertainment: Lifelike Autonomous Agents – *Communications of the ACM*, 1995, 38 (11), 108-114.

[28] Maroti, M., Kusy, B., Simon, G., Ledeczi, A. The Flooding Time Synchronization Protocol – *SenSys '04 Proceedings of the 2nd international conference on Embedded networked sensor systems, Baltimore, MD, USA, November 3-5 2004*, 2004, 39-49.

[29] Milner, R. A Calculus of Communicating Systems. Berlin : Springer, 1980 [29]

[30] Milner, R. Communicating and Mobile Systems: the π-calculus. Cambridge : Cambridge University Press, 2001

[31] Mõtus, L. ISP0012 Software Dynamics : Tarkvara dünaamika [WWW] http://www.dcc.ttu.ee/LAP/ISP0012/ISP_0012_part%201_uus.pptx (20.11.2012)

[32] Mõtus, L., Meriste, M., Dosch, W. Time-Awareness and Proactivity in Models of Interactive Computation – *Electronic Notes in Theoretical Computer Science*, 2005, 141 (5), 69-95. [*Online*] ScienceDirect (25. 11 2012)

[33] Mõtus, L., Meriste, M., Preden, J.-S., Pahtma, R. Self-aware Architecture to Support Partial Control of Emergent Behaviour – *Proceedings of IEEE 7th International Conference on System of Systems Engineering, 2012 Genoa, Italy, July 16-19*, 2012, 422-427.

[34] Mõtus, L., Meriste, M., Preden J. Towards Middleware Based Situation Awareness – *IEEE Military Communications Conference, Boston, Massachusetts October 18-21 2009*, 2009, 1-7. [*Online*] IEEE Xplore (19.11.2012)

[35] Mõtus, L Research into Cognitive Artificial Systems in Estonia – *Research in Estonia: Present and Future*. Tallinn : Estonian Academy of Sciences, 2011, 168-183.

[36] Mõtus, L., Rodd, M. G. Timing Analysis of Real-time Software. Oxford : Pergamon, 1994.

[37] Odell, J. J., Parunak, H. V. D., Fleischer, M., Brueckner S. Modeling Agents and Their Environment – *Agent-Oriented Software Engineering III – Lecture Notes in Computer Science*, 2003, 2585, 16-31. [*Online*] SpringerLink (19.11.2012)

[38] Omicini, A., Ricci A., Viroli, M., Castelfranchi, C., Tummolini, L. Coordination Artifacts: Environment-based Coordination for Intelligent Agents - *AAMAS '04 Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, New York, 2004 19-23 July*, 2004, 286-293. [*Online*] ACM Digital Library (29.11.2012)

[39] Pena, J., Levy, R., Corchuelo, R. Towards Clarifying the Importance of Interactions in Agent-Oriented Software Engineering – *Inteligencia Artificial*, 2005, 9 (25), 19-28. [*Online*] Inteligencia Artificial (06.12.2012)

[40] Peterson, J. L. Petri Net Theory and the Modeling of Systems. New Jersey : Prentice Hall, 1981

[41] Preden, J., Pahtma R. Exchanging Situational Information in Embedded Networks – *2nd International Conference on Adaptive Science & Technology, Accra, Ghana, December 14-16 2009*, 2009, 265-272. [*Online*] IEEE Xplorer (19.11.2012)

[42] Preden, J. Situation Awareness for Networked Systems – *IEEE First International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), Miami Beach, Florida, February 22-24 2011*, 2011, 123-130. [*Onlne*] IEEE Xplore (19.11.2012)

[43] Research Laboratory for Proactive Technologies, Department of Computer Control, Tallinn University of Technology Annual Report 2008 / L. Mõtus, M. Meriste, J.-S. Preden, Tallinn : Tallinn University of Technology, 2009.

[44] Russel, S. J., Norvig P. Artificial Intelligence : A Modern Approach. 2nd ed. New Jersey : Prentice Hall, 1995.

[45] Schilit, B., Adams, N., Want, R. Context-Aware Computing Applications – *First Workshop on Mobile Computing Systems and Applications, Santa Cruz, California December 8-9 1994, Proceedings*, 1994, 85-90. [*Online*] IEEE Xplore (19.11.2012)

[46] Schmeck, H. Organic Computing – A New Vision for Distributed Computing Systems - *ISORC 2005. Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Seattle, USA, May 8-20 2005*, 2005, 201-203. [*Online*] IEEE Xplore (01.12.2012)

[47] Serugendo, G. D. M., Foukia, N., Hassas, S., Karageorgos, A., Mostéfaoui, S. K., Rana, O. F., Ulieru,M., Valckenaers, P., Van Aart, C. Self-Organisation: Paradigms and Applications – *Engineering Self-Organising Systems – Lecture Notes in Computer Science*, 2004, 2977, 1-19. [*Online*] SpringerLink (19.11.2012)

[48] Serugendo, G. D. M., Gleizes, M.-P., Karageorgos, A. Self-Organisation and Emergence in MAS: An Overview – *Informatica*, 2006, 30 (1), 45-54. [*Online*] Informatica (06.12.2012)

[49] Sierra, C., Jennings, N. R., Noriega, P., Parsons, S. A Framework for Argumentation-Based Negotiation – *Intelligent Agents IV Agent Theories, Architectures, and Languages – Lecture Notes in Computer Science*, 1998, 1365, 177-192. [*Online*] SpringerLink (21.11.2012)

[50] Sierra, C., Noriega, P. Agent Mediated Interaction. From Auctions to Negotiation and Argumentation – *Foundations and Applications of Multi-Agent Systems – Lecture Notes in Computer Science*, 2002, 2403, 27-48. [*Online*] SpringerLink (28.11.2012)

[51] Sommer, P., Wattenhofer, R. Gradient Clock Synchronization in Wireless Sensor Networks – *International Conference on Information Processing in Sensor Networks, 2009, San Francisco, California, USA, April 13-16 2009*, 2009, 37-48. [*Online*] IEEE Xplore (20.11.2012)

[52] 52, T. K., Toueg, S. Optimal Clock Synchronization – *Journal of the ACM*, 1987, 34 (3), 626-645.

[53] Stepneya, S., Braunstein, S. L., Clark, J. A., Tyrrell, A., Adamatzky, A., Smith, R. E., Addis, T., Johnson, C., Timmis, J., Welch, P., Milner, R., Partridge D. Journeys in Non-Classical Computation: A Grand Challenge for Computing Research – *International Journal of Parallel, Emergent and Distributed Systems*, 2005, 20 (1), 5-19.

[54] Sterritt, R. Autonomic Computing – *Innovations in Systems and Software Engineering*, 2005, 1 (1), 79-88.

[55] Sterritt, R., Hinchey, M. Engineering Ultimate Self-Protection In Autonomic Agents for Space Exploration Missions – *12th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, Maryland, USA, April 4-7 2005*, 2005, 506-511. [*Online*] SpringerLink (19.11.2012)

[56] Stümpel, A Stream Based Design of Distributed Systems through Refinement. Berlin : Logos Verlag, 2003

[57] TinyOS, TinyOS [WWW] http://www.tinyos.net/ (20.11.2012)

[58] Wegner, P. Why Interaction is More Powerful than Algorithms – *Communications of the ACM*, 1997, 40 (5), 80-91. [*Online*] ACM Digital Library (28.11.2012)

[59] Werner-Allen, G., Tewari, G., Patel, A., Welsh, M., Nagpal, R. Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects – *SenSys '05 Proceedings of the 3rd international conference on Embedded networked sensor systems, San Diego, California, USA, November 2-4 2005*, 2005, 142-153.

[60] Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T., Ferber J. Environments for Multiagent Systems State-of-the-Art and Research Challenges – *Environments for Multi-Agent Systems – Lecture Notes in Computer Science*, 2005, 3374, 1-47. [*Online*] SpringerLink (19.11.2012)

[61] Wikipedia, TinyOS [WWW] http://en.wikipedia.org/wiki/TinyOS (20.11.2012)

[62] Wolf, T. D., Holvoet T. Emergence and Self-Organisation: A Statement of Similarities and Differences [WWW] https://trac.v2.nl/export/7711/rui/projects/UnleashCulture/UnleashCulture3/Emergence%20and%20Self-Organisation,%20similarities%20and%20differences.pdf (01.12.2012)

[63] Wooldridge, M. An Introduction to Multiagent Systems. Chichester : John Wiley & Sons, 2002.

[64] Wooldridge, M. J., Jennings N. R. Agent Theories, Architectures, and Languages: A Survey – *ECAI-94 Workshop on Agent Theories, Architectures, and Languages Amsterdam, The Netherlands August 8–9, 1994 Proceedings. – Lecture Notes in Computer Science*, 1995, 890, 1-39. [*Online*] SpringerLink (19.11.2012)

APPENDIX

Appendix A1 - Pilot system source code

Source code includes 3 TinyOS based configuration files

- miDemoC.nc
- MediwareC.nc
- OperatorC.nc

and 3 TinyOS based module files

- miDemoP.nc
- MediwareP.nc
- OperatorP.nc

The files are included on a CD affixed to the thesis.