Information Sciences 180 (2010) 132–155

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

On model design for simulation of collective intelligence

Martijn C. Schut

VU University, Computer Science, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 13 October 2008
Received in revised form 17 July 2009
Accepted 4 August 2009

Keywords:
Model design
Artificial intelligence
Multi-agent systems
Self-organising systems
Collective intelligence

ABSTRACT

The study of collective intelligence (CI) systems is increasingly gaining interest in a variety of research and application domains. Those domains range from existing research areas such as computer networks and collective robotics to upcoming areas of agent-based and insect-based computing; also including applications on the internet and in games and movies. CI systems are complex by nature and (1) are effectively adaptive in uncertain and unknown environments, (2) can organise themselves autonomously, and (3) exhibit 'emergent' behaviour. Among others, multi-agent systems, complex adaptive systems, swarm intelligence and self-organising systems are considered to be such systems. The explosive wild growth of research studies of CI systems has not yet led to a systematic approach for model design of these kinds of systems. Although there have been recent efforts on the issue of system design (the complete design trajectory from identifying system requirements up to implementation), the problem of choosing and specifying a good model of a CI system is often done implicitly and sometimes even completely ignored. The aim of this article is to bring to the attention that model design is an essential as well as an integral part of system design. We present a constructive approach to systematically design, build and test models of CI systems. Because simulation is often used as a way to research CI systems, we particularly focus on models that can be used for simulation. Additionally, we show that it is not necessary to re-invent the wheel: here, we show how existing models and algorithms can be used for CI model design. The approach is illustrated by means of two example studies on a (semi-automated) multi-player game and collaborative robotics.

© 2009 Elsevier Inc. All rights reserved.

1. Introduction

There is a growing interest in the research of complex systems that (1) are effectively adaptive in uncertain and unknown environments, (2) can organise themselves, and (3) exhibit so-called 'emergent' behaviour. This interest can be seen across a wide range of research disciplines: computer science, social science, economics, biology, physics and others. Within the computer sciences, the possibility of *engineering* complex systems becomes crucial, both from a pull-side (i.e., application requirements), and from a push-side (i.e., natural complex systems are being increasingly better understood).

This article looks at an important part of the methodological ('how') aspect of designing complex systems. Di Marzo Serugendo [66] phrases the design of such systems as follows:

> The challenges to take up in this field relate to the design, and the development of applications which "work by themselves": how to define a global goal; how to design the components and their local functionality knowing that the global goal is not a local sum of functionality; which will be the interactions between components, and how to check that the desired result will emerge during the execution. (From Di Marzo Serugendo [66].)

E-mail address: mc.schut@few.vu.nl

0020-0255/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2009.08.006

Although there has been a recent series of developments in the work on design methodologies for *multiagent systems*, the range of systems to be designed that we have in mind is not restricted to multi-agent systems, but also includes systems that would be considered, for example, distributed problem solving, 'closed' systems, complex adaptive systems or self-organising systems. We call these kinds of systems *collective intelligent* (CI) systems. What binds all these systems together is what Ottino [57] has identified as the hallmarks of complex systems: adaptation, self-organisation and emergence.

One particular type of system deserving special attention and that we, and many with us, consider as CI, are the human–machine 'systems' emerging on the Social and Semantic Web: "Ecosystems of participation of humans and machines [...] where the involvement of human beings as creators and consumers in problem solving and learning tasks as well as of data and knowledge has been of paramount importance" (from Kapetanios and Koutrika in the Introduction of this Special Issue). Whereas these systems have in the last decade been rather 'emergent' phenomena (e.g., *Facebook, MySpace, Wikipedia*), it is now the time at which we want to be able to understand and design such systems in a systematic way instead of the current working practice of trial-and-error: we are looking for a new computing paradigm. This paradigm should be useful for computational environments that are strongly embedded in a social environment (i.e., with many-to-many users). Thus these environments are not only virtual (like in [38]), but also involves real human social interaction. One of the included case studies in this article focuses particularly on the design of such human–machine models of a computational-social system.

The focus of this article is on designing CI systems – as explained in the aforementioned quote from Di Marzo Serugendo. We are well aware that we do not achieve to address the proposed challenge completely in a single article. The particular aim of this article is to bring *model design* to the attention as an essential as well as an integral part of system design. Although there have been recent efforts on the issue of system design (the complete design trajectory from identifying system requirements up to verification and maintenance), the problem of choosing and specifying a good model of a CI system is often done implicitly and sometimes even completely ignored. Especially for the development of human–machine systems, systems and models from the 'engineering'-oriented sciences cannot be readily used. Therefore, we aim here to provide directives (in terms of stepwise diagrams and modelling recipes) that can be used by a very wide range of researchers (e.g., social/biological/economic scientists) that are interested in developing models of CI systems, not only computer/engineering/software scientists and developers.

In the remainder of this Introduction, we give a brief overview of what are commonly understood as the identifying properties of CI systems and we present a distinction between design ("how do we build CI systems?") and analysis ("how can we better understand CI as observed in nature, including human nature?") studies of CI.

1.1. Properties of collective intelligence

What distinguishes collective intelligent systems from other systems? In the literature, one can find various answers to this question. For example, Bonabeau et al. [11] identify four basic ingredients, based on researching self-organisation in social insects: positive feedback (simple behavioural "rules of thumb" promoting the creation of structures; examples are recruitment and reinforcement), negative feedback (counterbalances positive feedback and helps to stabilise the collective pattern), amplification of (random) fluctuations (randomness enables the discovery of new solutions), and multiple interactions (e.g., directly or through the environment). Heylighen [41] enumerates the following properties: global order, distributed control, robustness, feedback, bifurcations, adaptation and optimality through perturbations. Holland [43] identifies properties and mechanisms of complex adaptive systems: aggregation, tagging, nonlinearity, flows, diversity, internal models and building blocks. Mondada et al. [54] identify robustness, flexibility, scalability and performance as the key characteristics of socially inspired computing.

Based on our experience with a number of research studies that we conducted and the study of lists like the ones above, we have compiled the following list of properties of CI. In this list, we distinguish *enabling* and *defining* properties. The enabling properties are those that if you have them, it is possible for CI to emerge; the defining ones are those that if you observe these properties, the system is a CI one. The former are on the local (agent) level, whereas the latter are on the global (system) level.

1.1.1. Enabling CI properties

1.1.1.1. Adaptivity. To adapt literally means "to fit to", usually referring to (necessarily) changing one's own structure to deal with one's environment. In our context, this means that either an individual changes itself if necessary or the whole system changes itself. The former implies the latter, but not necessarily the other way around. On an individual level, it means that an individual changes its own behaviour. If this behaviour is specified by, for example, rules (see below), than (parts of) these rules are subject to change. But even if the individuals do not change, the system as a whole may still be adaptive. Consider, for example, an ant colony in which the ants do not change their own behaviour (e.g., for food foraging), but where the whole colony can change itself if so required: experiments have shown that a colony forages efficiently even when the food sources change over time [37].

1.1.1.2. Interaction. Systems with many (thousand or millions of) individuals thrive on the way that these individuals interact with each other. For instance, how much do they interact, what do they interact about, do we only consider communication or are there other kinds of interactions? In a complex system, one cannot isolate a single part and analyse its behaviour to find out about the system behaviour; instead, one has to consider the interactions as well as the individual

behaviours [79]. Other lists of properties explicitly consider *flows* to be a key property. This means that information flows through the system and some small cause somewhere in the system may eventually have very large consequences. We consider these flows as characteristics of interaction, but not separately. In combination with the properties of emergence and randomness (see below), we believe that the concept of flows is covered.

1.1.1.3. Rules. The most fundamental form of describing the behaviour of an individual (or whole system) is to use rules. Such rules are implications between inputs (observations) and outputs (actions). Such implications can be represented in many different ways, e.g., if/then rules, trees or networks. The best representation depends on the particular domain and problem.

1.1.2. Defining CI properties

1.1.2.1. Global-local. We distinguish two aggregation levels: the *local* level that concerns the individuals in the systems, and the *global* level concerning the system as a whole. We need this distinction to position, for example, adaptivity and emergence. As mentioned above, adaptivity can happen on the local and/or global level. For the former, individuals may have adaptive behaviour, as that they adapt themselves to changing circumstances. For the global level, the system as a whole is able to adapt itself if responsive to changes in its surroundings. Emergence is a concept, as explained below, that concerns going from the local to global aggregation level: at some point if you are going up the levels, "the system becomes more than its individual parts". Moving between these aggregation levels is a very fundamental scientific concept and subject of many research studies, for example Yamins who constructs a local to global theory for multi-agent systems [85].

1.1.2.2. Randomness. Complex systems typically have some element of randomness in it that causes the system to show self-organised critical behaviour: these systems exist somewhere on the edge of chaos where on one side there is chaos and disorder, and on the other side there is structure and order. Without an "external spark" (or: an infinitesimal driving rate [2]), such systems would be in balance: (1) little disturbances have no consequences, (2) the system's response is proportional to the impact (action = reaction), and (3) only dramatic disturbances can cause state transitions. But with it, these systems get out of balance, moving over the edge of chaos; for example, the so-called "butterfly effect" can be observed in these systems [51].

1.1.2.3. Emergence. This is arguably the most convoluted property in this list; its simple meaning is "the whole is more than the sum of the parts" [18]. In complex systems, *something* happens when you move from the lowest abstraction level (individual) to the highest abstraction level (system) – and this something is often surrounded by "mystical overtones" [18]. And so a scientific dilemma emerges: when you have discovered the science of this something, the mystery disappears. We can use the following working definition, coined by de Wolf and Holvoet: "A system exhibits emergence when there are coherent emergents at the macro-level that dynamically arise from the interactions between the parts at the micro-level. Such emergents are novel with respect to the individual parts of the system." [22].

1.1.2.4. Redundancy. This means that the same knowledge is represented in a number of different places in a system. This knowledge may be actual knowledge (e.g., rules), but it might also be information itself. This may be very straightforward: every individual works with the same rule set. In this way, the rules (considered to be knowledge) are everywhere in the system. It may also be more complex, in that there is knowledge or information that a number of individuals (but not everybody) share.

1.1.2.5. Robustness. Closely linked to redundancy is that the system is robust against malfunctioning. If the knowledge is represented at multiple places in the system, it means that if the system loses parts, chances are that the knowledge is still maintained somewhere in the system – making the system robust in case of damage or attack.

1.2. Studying collective intelligence

It is important to realise that there are two different ways to study CI: for the purpose of *design* or for *analysis*. We briefly explain these different ways, but first we show that this distinction has been made previously in related work.

Holland thought up the idea of *reproductive adaptive plans* which were structures that adapt themselves by means of some evolutionary mechanism [42]. Holland also suggested that these plans were actually hypotheses if considered in the context of natural systems, and algorithms in artificial systems. Dorigo makes a similar distinction: swarm intelligence exists in a natural form (where a natural phenomenon is of concern) and in a computational form (where one programs algorithms). Finally, in an attempt to categorise the research on artificial life, Bullock [13] identifies High Class, Model Class and Work Class Artificial Life (AL). High Class AL is concerned with the philosophy of artificial life; Model Class AL is concerned with the validity of scientific theory; and Work Class AL involves the production of useful artifacts, most naturally understood as engineering. It is interesting that people may feel strongly about these different types, as Bullock demonstrates here:

> Attempting to discover the nature of life (or some other biological phenomenon) through digital naturalism is analogous to attempting to discover the laws of aerodynamics which govern flight through drawing many different pictures of

There are three simple steering behavior parameters in the well-known BOIDS simulation of bird flocking [61]: separation, alignment and cohesion. With specific settings of these parameters, flocking, as observed in the real, world can be simulated. The design-step that we propose means that we search for an appropriate parameter setting in order to get some desired group behaviour. For example, we may want to have the flock moving in a straight line, a circle, or a square. This design problem raises a number of questions. In increasing order of complexity, these are some questions. How do we tune separation, alignment and cohesion to get this behaviour? Can we even get this behaviour with only tweaking the given parameters? If not, would it work to consider a more heterogeneous population where the parameters can vary per individual? If not, will it work to introduce new parameters (which ones?) With which we can get the desired behaviour? Currently, it is unknown how to define this problem, let alone to solve it.

Fig. 1. Towards designing collective intelligence. (See above-mentioned reference for further information.)

> imaginary birds, imaginary flying insects, etc. Once many such pictures of "flight-as-it-could-be" have been rendered, one might spend years searching for principles which reveal the nature of flight. (From Bullock [13, p. 22].)

It is important for us to also make such a distinction when it concerns collective intelligence. A good reason for this is that studying collective intelligence is versatile and interdisciplinary, and different researchers from different disciplines may have different study objectives.

1.2.1. Analysis studies

The purpose of analysis studies is to learn about and to get better understanding of phenomena as observed in nature, including human nature. Some of the scientific disciplines undertaking such studies, in the context of collective intelligence, are the social sciences (e.g., [27,36]), biological sciences (e.g., [15]), and economic/political sciences (e.g., [5]).

1.2.2. Design studies

The purpose of design studies is problem solving (in its broadest meaning). This can mean to find a solution in the search space of an abstract formal problem[1] or to find a coordination protocol that lets a number of individuals cooperate with each other (for example, a set of robots that has to perform some collective task.) Design studies are mostly the terrain of computer scientists, artificial intelligence researchers and engineers.

In the context of collective intelligence, it is sometimes difficult to distinguish between analysis and design. Consider evolution for example. You may implement a model that has evolution in it for some purpose that is not explicitly clarified. A biologist using this model may want to study the mechanism of evolution itself, while a computer scientist is interested in using the mechanism as a heuristic within some optimisation process. For the biologist, it is important that the way evolution is modelled is as realistic as possible: the model is the means and evolution is the ends. For the computer scientist, this is exactly the opposite: he can tweak and change the mechanism as much as he wants, because evolution is the means and the model (here: problem/solution) is the ends. However, the computer scientist may be solving a particular problem for which it is important that the *simulation* is realistic (even though the mechanism itself not). For example, if you want to develop robot controllers that are eventually implemented in real robots, the simulation should model the environment as good as possible. Example design studies are: collective robotics, peer-to-peer network protocols, data clustering.

Finally, we want to give an illustrative example of what "design of collective intelligence" encompasses. In the example in Fig. 1, we show what we eventually aim for: some kind of engine that takes in some collective task description and outputs a solution that prescribes the (adaptive) behaviour of the involved individuals. As the example says: we do not even know how to define (represent) this problem, never mind how to solve it – this is currently cutting-edge research on designing collective intelligence.

1.3. Overview

This article is structured as follows. In the following Section, we provide background information on the design of CI systems. We look at a number of closely related research streams, as well as relevant (model and system) design approaches. In Section 3 we present a systematic approach for the designing of CI system models. Section 4 contains two case studies to illustrate the methodology. Finally, Section 5 concludes and gives pointers for future work.

2. Background

This Section describes the related literature on, firstly, CI itself, and, secondly, model and system design approaches. For these approaches, the ones on model design have a rather *numerical* nature and the system design ones are rather *conceptual*.

[1] For example, solving the Traveling Salesman Problem by means of a self-organising neural network [52].

Numerical approaches include, for example, utility-based reinforcement learning methods, and have a more theoretical nature (although also widely used in practice). Conceptual approaches include, for example, multi-agent systems and agent-oriented software engineering, and have a more practical nature. It is best to consider model design as a part of system design – when you have to design a system, you have to spend considerable effort and time on designing a good model. We consider this a worthwhile distinction in general as the design of collective intelligence is concerned.

2.1. Collective intelligence

The big picture of collective intelligence includes fundamental ideas like evolution [19], culture [20] and order [45]. Understandably, there are thus many different terms, concepts, frameworks and processes that are closely related to collective intelligence as defined above. Of the related research streams, we briefly touch upon a number of them: self-organisation, complex adaptive systems, multi-agent systems, population-based adaptive systems, swarm intelligence and swarm engineering. Our aim here is not to give a comprehensive and detailed overview of these streams but rather present a short-list of related streams with some main references for the interested reader.

2.1.1. Self organisation

According to Farley and Clark [28], "A self-organising system changes its structure, depending on its experiences and its environment". The organisation process itself is autonomous, thus there is no central coordination that manages the organisation. A working definition of self-organisation (SO) is suggested by de Wolf and Holvoet: "Self organisation is a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control." [22]. Recently, the concept of *self-star* (or: self-*) emerges as an umbrella-term covering self-organising, self-managing, self-adaptive, self-etcetera. This concept is mainly used within the context of designing large computer systems and networks that are able to do most of their work autonomously – for example, the concept of self-adaptive software is explained in [64].

2.1.2. Complex adaptive systems

A complex adaptive system (CAS) is a complex, self-similar collection of interacting adaptive agents. Moreover, it is a key notion in complexity science. John Holland [42] observed CAS in many different contexts (for example, economics, biology, society) that eventually resulted in the theory and practice of evolutionary computation. Holland also set up the ECHO simulation tool[2] to investigate CAS behaviour. Researchers have looked at properties of such complex systems in order to engineer artificial ones [17], like the design of a complex automotive manufacturing plant.

2.1.3. Multi-agent systems

The field of artificial intelligence had throughout the 1960s till the 1980s developed the idea of a knowledge-based system (or: expert systems) in which expert knowledge was formalised and further used for decision support, consultation, etcetera. The systems worked in isolation and were passive with respect to the one operating them. When a number of these systems were combined with each other, a *multi-agent system* (MAS) came into being. Such a system is "composed of multiple interaction software components (or: agents), which are typically capable of cooperating to solve problems that are beyond the abilities of any individual member" [84].

2.1.4. Population-based adaptive systems

Population-based adaptive systems (PAS) are systems that consist of many (thousands or millions) units that have adaptive, reactive, and self-organising abilities. These systems resulted from the synergy of two research areas, namely intelligent multi-agent systems (as explained above) and evolutionary methods [25]. Both research fields contribute to each other in the context of PAS: the pro-activeness of intelligent agents is used in evolutionary systems (where the individuals typically are only passive) and the individual adaptive capabilities in evolutionary systems can be used in intelligent agents, allowing these to learn and adapt when necessary.

2.1.5. Swarm intelligence

Swarm intelligence (SI) is an AI technique for search/optimisation and based on the idea of "swarming" (as observed in bird flocking or fish schooling). For example, the flocking of birds or the way insects use pheromones can be used as inspiration for new ways to improve data clustering algorithms [29,34]. The term SI was initially coined by Beni and Wang [7] in the area of cellular robotics (i.e., robotics based on the theory of cellular automata) and has since then been worked out by Marco Dorigo [11], inventor of ant colony optimisation (ACO) – a metaheuristic for combinatorial optimisation problems. In SI, the intelligence of swarming as observed in nature is used for solving practical coordination problems. It is extremely successful in the area of intelligent robotics; and the related ACO has also been used fruitfully for more abstract problems, like the Traveling Salesman Problem and peer-to-peer computing [47]. Closely related to ACO is the work on particle swarm optimisation (PSO) [26]: an optimisation technique where particles form a swarm that flies through a (multi-dimensional) problem space, whereby the positions of the particles are updated each generation based on the particle's experience and

[2] http://www.santafe.edu/projects/echo/.

that of its neighbours. PSO has been used for many optimisation problems, like improving compressed image quality [50] and multi-objective optimisation [75,80]. In general, bio-inspired computing (including ACO and PSO) has recently seen a rapid increase in research attention (e.g., [6,58]).

2.2. Model design

We present a number of design models that are *numerical* in nature (as opposed to the conceptual approaches presented in the next Section). These models are, in order of presentation, swarm engineering, local to global, product distributions and desired collectives.

2.2.1. Swarm engineering

Swarm engineering (SE) is "a process where one creates a swarm of agents which complete a predefined task" [46]. The process consists of two steps: firstly, one generates a swarm condition (guiding the generation of a suitable swarm); secondly, a behaviour (set) must be generated that satisfy the given swarm condition. Kazadi, in [46], implemented this process and applied it to two robot applications (plume tracking and object clustering), and the Traveling Salesman Problem.

2.2.2. Local to global

In an ambitious attempt to come up with a "local to global" theory for distributed multi-agent systems, Yamins [85] presents a formal model to systematically describe and analyse a variety of problems within the context of self-organisation. The problems are scoped to so-called *spatial* multi-agent systems, which are decentralised systems composed of identically programmed agents that form or are embedded in some geometric space. One particular problem, the "equigrouping problem" is studied in detail: a one-dimensional problem in which multiple identical agents organise themselves in equally-sized groups. For this problem, two algorithms are presented that work on the local level (look at your neighbours within some radius and determine to move left or right) and achieve goals on the global level (groups of equal size). The philosophy of the work is a change in perspective on designing multi-agent systems: instead of describing and verifying a single solution, analysing a solution space might be more effective. Hence, based on the two algorithms, the solution space of the equigrouping problem is analysed, i.e., characterising parts of this space and see whether the algorithms can reach these parts.

This leads to a so-called "global to local" compiler, that takes in a target pattern and set of resource limits, and generates optimised local rules able to produce the pattern given the limited resources. The compiler is a suite of mathematical techniques that together can (1) characterise patterns that are self-organisable (based on a "necessary" condition), (2) algorithmically generate local rule solutions to any solvable target pattern, and then (3) analyse the resource usage and runtime properties of these rule solutions. Parts of the compiler have been implemented by means of a self-organising Turing machine.

To place the work of Yamins in the context of our article, we see that it consists mainly of algorithms, corollaries, propositions and proofs. Simulations are used to illustrate parts of the theory, but more as supporting acts rather than first-class citizens. We consider Yamin's work a valuable theoretical contribution rather than a practical design methodology to engineer collective intelligence.

2.2.3. Product distributions

Wolpert et al. [77] present a distributed optimisation algorithm based on *collectives*. Such collective is a "multi-agent system (1) where each agent is self-interested and capable of learning, and (2) that has a specified system objective which rates the performance of the joint actions of the agents" [10]. The introduced Probability Collectives (PC) Theory[3] is a formalisation and extension of the earlier COIN framework of Wolpert and Tumer [83]. The key insight of PC theory is to look at how agents update the probability distribution over the possible actions rather than the joint action resulting from sampling those distributions (as happens in reinforcement learning, the key mechanism of COIN). This is so fundamental that the theory has deep connections with game theory and statistical physics. The most relevant feature of PC within the context of this article is that it can be used to control multi-agent systems.

2.2.4. Desired collectives

In 2004, Namatame coined the term *desired collectives* [55]. He explains it was inspired by a number of things: firstly, Surowiecki's book "The Wisdom of Crowds" [73] on how large groups of people can outsmart an elite few; secondly, Turner and Wolpert's aforementioned forward-inverse problem; and, finally, the so-called price of anarchy (selfish behaviour of agents) [63]. The idea of desired collectives to design a individual-based system that can induce desirable collective outcomes. The approach that Namatame takes is that he identifies a number of properties of such system (e.g., knowledge sharing, diversity, preferences, interaction, norms), runs simulations of agent systems with such properties within some social games (e.g., dispersion, transaction, iterated prisoner's dilemma, coordination, hawk/dove) and then analyses the results game-theoretically (e.g., equilibria, micro-macro dynamics, evolutionary dynamics). An examplary insight yielded by such study is that the gain from diversity depends on the type of interaction and location of heterogeneous agents.

[3] Versions of PC include Product Distributions and Product Distribution Theory.

Namatame's particular mechanism that is able to produce desired collective outcomes is to let agents evolve so-called *coupling rules* [55, Chapter 9]. This mechanism works by letting each agent have a number of bit-string rules consisting of (1) a number of bits encoding an action history and (2) a number of bits encoding the agent's strategy ("if my opponent does this, I do that") and evolving these rules by means of evolutionary learning. Simulation results of a generalised rock-scissors-paper scenario shows that the mechanism yields high individual pay-offs for the agents.

To link Namatame's work on desired collectives to our article, we mainly see that our objective is different. Whereas the game-theoretical analysis of social games in combination with a number of collective system properties (e.g., diversity) yields valuable insights on designing collective systems, it does not provide a design methodology. On the other hand, the coupling-rules mechanism is a well suited instantiation of our general agent adaptation mechanism (as presented later).

2.2.5. Discussion

Although the presented approaches are closely related to the design methodology that we introduce in this article, we can point out some important aspects from which it becomes clear that our approach is different. Firstly, for the local to global and product-distribution approaches, we consider that they are too theoretical in nature and too far away from the intended use in practice as we envision. Interestingly, Yamins points out in [85, p. 19] how his work relates to the theory and practice of Cellular Automata [82]. This explanation further emphasises the theoretical nature of the approach. Also, for this approach, Yamins clearly identifies the local and global level but is rather unclear about the 'to' part. In our methodology, we pay particular attention to moving from one level to another. Finally, concerning the desired-collectives approach, it is not clear to us how the *coupling rules* in this approach relate to so-called *learning classifier systems* (LCS), which are introduced below. This needs further investigation.

2.3. System design

In this section, we describe work of others relating to system design modelling of CI, e.g., development methodologies for agent-oriented programming platforms, design methodologies for multi-agent societies and engineering methodologies for adaptive multi-agent systems. We want to mention explicitly that it is not our aim to present an all-inclusive overview of all such modelling approaches. Rather, we concentrate on those that in our opinion are most closely related to the ideas presented in this article (i.e., the system design methodologies for *self-organising* multi-agent systems).

2.3.1. Self-organisation in multi-agent systems

Within the AgentLink-III Co-ordination Action,[4] the *Technical Forum Group Self-Organisation in MAS* (TGFSO) researched approaches to incorporate self-organisation mechanisms in the design of multi-agent systems.

The contributions[5] of TGFSO to the question how to engineer emergence can be summarised as follows: (1) an approach for selecting self-organisation mechanisms; (2) categorisation of mechanisms leading to self-organisation; and (3) identification of criteria to compare self-organised systems. We briefly go into each of these contributions.

For (1), Di Marzo Serugendo et al. present in [69] the synthesis of the TGFSO regarding (among others) a framework for selecting self-organisation mechanisms by means of case studies and evaluation criteria. The case studies concerned managing computer networks, manufacturing control and space conformation of molecules. Here, we briefly enumerate the different steps in the framework.

1. *System description* – a description that contains purpose and function of the system, global behaviour, and the system inputs;
2. *Environment description* – the system is situated in;
3. *Perturbation* – stimuli the system can receive and how they vary over time;
4. *Entity Description* – characteristics of the system components, should include (but not limited to): goals, capabilities and beliefs (about others, itself, and the environment);
5. *Self-organisation engine* – details the mechanism of self-organisation process: what triggers re-organisation, internal of external mechanism, etcetera.

The authors further work out the case studies based on this framework. For each case study, they study a number of different approaches which are all analysed by means of the framework. For example, for the computer network management, they discuss approaches based on holonic systems, peer-to-peer systems, routing and delegation concepts and stigmergy mechanism.

For (2), Bernon et al. [8] overviewed a number of examples of multi-agent systems in which self-organisation was used to solve complex problems; ranging from cellular radio networks to traffic simulation. Each system was explained by identify-

[4] 2004–2006, FP6-IST-1-002006-CA.

[5] Note that this shortlist does not include *everything* that was done by this group. The group also looked at, for example, bio-inspired mechanisms for artificial self-organised systems, self-organising mechanisms from social, business and economic domains, looked at emergence vs. self-organisation, and many other things. We selected this shortlist from the TGFSO literature based on what we thought to be most relevant to specifically the engineering of emergence.

ing (1) the environment, (2) the agents and (3) the behavioural model. The work resulted in a list containing two types of criteria for comparison between the different applications Firstly, these criteria can be evaluated by looking at the description of the system: absence of external or of centralised control; dynamic operation; emergence properties; simple local rules; and reusability. Secondly, the following criteria can be tested by running the systems: anytime property; instability; and adaptation.

For (3), Di Marzo Serugendo et al. [67] have categorised different mechanisms leading to self-organisation in five different classes: direct interaction (e.g., broadcast, localisation); indirect interaction (e.g., stigmergy); reinforcement (e.g., behavioural rewards); cooperation; and generic architectures (e.g., holarchies).

2.3.2. METASELF

In recent work [68], di Marzo Serugendo works out the idea of a design framework, called METASELF, that follows a service-oriented architecture, combines design-time and run-time features and that can adaptively coordinate between system metadata about the functionality and performance characteristics (self-descriptions of components, coordination space environment, and self-* properties) and execution policies (guiding, coordination, bounding and sensing/acting). The perspective underlying the framework is concentration on *threats* (e.g., networks failures, deviation of environment from anticipated behaviour) that compromise desired system behaviour. The framework follows the traditional steps of an engineering process: *analysis* – where desired properties of the system are identified; *design* – where architectural patterns are selected and suitable adaptation mechanisms (e.g., stigmergy or field-based) are identified; and *implementation* – where the run-time architecture is developed (including metadata and policies). The METASELF framework is illustrated in [30] by means of a case study on industrial manufacturing assembly systems (an adhesive tape roller dispenser assembly system). The paper presents the case study in the context of the concept of *self-organising evolvable assembly systems* (EAS). An EAS is an assembly system which can co-evolve together with the product and the assembly process.

2.3.3. Adaptive multi-agent systems

Capera et al. [16] provide the Adaptive Multi-Agent System (AMAS) theory for the design of complex adaptive systems. The paper describes (1) that the system organisation is essential for its global function, and (2) that there can be an "interdependence between a cooperative local behaviour and the functional adequacy of the collective function". For (1), assuming that each system part brings forth a partial function and that these partial functions can be combined in some way, then by transforming the organisation and thereby the *order* in which the partial functions are combined, one can change the global function. For (2), the authors demonstrate the theorem that any functional adequate system (i.e., those that adequately carry out for which they were conceived), has a counterpart (called "internal cooperative medium") that fulfills an equivalent function. This counterpart contains all parts of the system itself and the physical support needed for exchanges between the parts. The (cooperative) exchanges between the system and the environment are then a subset of its parts interaction. The main consequence of the theorem is that the application of the AMAS theory gives systems that do not need to know their own global function or to have a goal. In the designed AMAS systems, agents realise their partial functions and simultaneously act on the internal system organisation if they detect *non-cooperative situations*. A system designer has to describe how agents fulfill their function, as well as which situations must be avoided and how to suppress these. The idea is similar to using "exception handlers" in classical programming.

ADELFE[6] [9] is a software engineering methodology suited for AMAS design. It largely follows the Rational Unified Process (RUP), which is a widely used object-oriented development methodology [48]. In addition to the 'traditional' RUP activities during development, ADELFE extends the first three classic RUP phases as follows:

1. *Requirements* – development of an *environment model* consisting of actors, context and characteristics of the environment;
2. *Analysis* – (1) the agents are identified and (2) it is decided if AMAS is adequate to use (e.g., if the task is not complex, then AMAS is not adequate but rather overkill);
3. *Design* – (1) an agent model of *cooperative* agents describing their components realising their behaviours, (2) NCS model (Non-Cooperative Situation) for an agent to detect cooperative failures.

2.3.4. Agents and artefacts

Gardelli et al. [32] present a methodological approach for designing self-organising multi-agent systems, illustrated by a case study in collective sorting of tuple spaces. It is particularly aimed at the early system design stages and consists of three steps: modelling, simulation and tuning. The foundations of the approach are: (1) the Agents & Artefacts (A&A) metamodel (which was developed earlier by the authors), and (2) identification of 'environmental' agents whose goal is to achieve some self-organising property. The idea that "intelligence is in the environment" is much in line with biologists observing self-organisation in nature [15] and self-organising multi-agent system developers who treat the environment as a first-class abstraction [81]. In the A&A metamodel, agents are pro-active goal-driven entities and artefacts are reactive entities providing services and functionalities. Gardelli et al. [32] present two main ideas: firstly, the inclusion of environmental agents

[6] http://www.irit.fr/ADELFE/.

(additional to the user-agents that are the 'standard' agents) that manage the artefacts in order to achieve a target self-organising property; and, secondly, the recognition of simulation as a way to enhance self-organising systems design.

As mentioned, the approach consists of three different steps. We briefly describe these steps here. Firstly, *modelling* involves the characterisation of user agents, artefacts and environmental agents. Here, the environmental agents interface between the user agents (who require services) and artefacts (that provide services). The environmental agents have to do this interfacing as such that it generates a feedback loop between user agents and artefacts (analogous to, for example, pheromones in an ant foraging scenario). Potential candidates for these environmental agent models (and thus the feedback loop) are known self-organising design patterns. While such patterns are not readily available now, the paper mentions several research efforts to come up with these (that are not repeated here for reasons of space). Secondly, *simulation* is the phase in which the formal specification that resulted from the modelling phase is extended with a number of model parameters. Having defined these parameters, the function of the simulation is to blueprint proper system behaviours and valid ranges for the system parameters. Finally, by means of *tuning*, the environmental agent's behaviours and system parameters are made such that desired dynamics are observed. Depending on the dynamics of the tuning process itself, wider parameter ranges may be considered or domain information may already be used to steer the tuning somewhat. The outcome of this last step (that finalises the approach as a whole) is an accurate statistical characterisation of system behaviours.

2.3.5. Self-organising traffic lights

Also starting from the *agent* methapor, Gershenson presents a conceptual framework[7] SOTL for the design of self-organising systems [33]. The main idea of the SOTL framework is that by reducing the *friction* between elements, a higher system *satisfaction* can be reached. The design methodology of the framework contains the following phases: representation, in which a specification of the system components is developed, guided by system requirements and constraints; *modelling*, in which a control mechanism is specified that ensures the system does with it is required to do; *simulation*, to build a computer simulation implementing the model(s) developed at the previous stage; *application*, to use the developed and tested model(s) in a real system; and *evaluation*, in which the performance of the system is compared to previous systems.

The core part of this methodology is the specification of the control mechanism in the modelling step. This specification is based on the earlier mentioned "minimise local friction, maximise global satisfaction" idea. This is imposed on the system by means of *mediators*. Such mediators are constraints with respect to what elements are allowed to do and what they cannot do. A simple analogy that Gershenon presents is the presence of traffic lights (i.e., the mediators) for vehicular traffic (i.e., the elements): these are constraints as to what vehicles may and may not do. However, differently from traffic lights, a control mechanism for a self-organising systems should be adaptive and active. The choice of control mechanism is then deciding which mediator is best for the design problem at hand. Gershenson overviews a number of different ways how to reduce friction (e.g, tolerance, courtesy, compromise) and increase synergy (cooperation, altruism, exploitation).

The working of the framework is illustrated by a number of case studies: traffic lights that can adapt to changing traffic conditions, reduce waiting times, number of stopped cars, and increase average speeds; bureaucracies, where communication within bureaucracies are improved, public satisfaction are sensed, hierarchies can be modified dynamically and procedures are contextualized; and common-use artefacts (PDAs, mobile phones) that respond to user's desires without explicit programming or commands.

2.3.6. Customised unified process

In his Ph.D. Thesis [21], de Wolf presents an innovative design approach based on a unified process[8] for engineering self-organising emergent applications – the Customised Unified Process (CUP). Also, the applicability of a numerical verification technique, i.e., Equation-free Macroscopic Analysis which was developed in the area of physics and chemistry, was studied in the context of realistic and engineered self-organising emergent systems. Here, we do not go into the details of this study, but focus on CUP as it is closest to the other system design methods discussed in this Section. The CUP approach is illustrated by means of two case studies: data clustering and automated guided vehicles. The design is supported by two techniques: firstly, a design pattern catalogue of decentralised coordination mechanisms, and secondly, a modelling approach for designing coordination through correct distribution information with information flows. The motivation for introducing the CUP approach is twofold: (1) an observed shortcoming of agent-oriented methodologies to focus on the system micro-level without explicitly taking the macro-level into consideration, and (2) the UP is a well-known design process that shares steps with many other processes (e.g. extreme programming) and it allows for *iterative development*. The latter is a very welcome property for designing self-organising, since it is not possible to develop the whole system in one go (which was also noticed by Gershenson, as mentioned above).

Each iteration in the development approach consists of requirements analysis, design, implementation, and testing-and-verification. In the CUP approach, each of these steps is customised in order to *address the desired macroscopic properties*. In the *requirements analysis*, the macroscopic properties are identified; *design* consists of an architectural design (making use of best-practice and experience as design patterns; identifies information flows) and detailed design; *implementation* is not customised; and *verification and testing* includes the scientific analysis of the macroscopic properties, generating feedback to the

[7] Because the methodology of Gershenson does not have a name in its entirety, we refer to this methodology by the earlier used acronym SOTL (self-organising traffic lights) that Gershenson gave to a specific instance of the methodology in the domain of traffic control.

[8] The *Unified Process* is a well-known industry-ready software engineering process.

(next) design step. As mentioned, two essential concepts of the approach are information flows, and design patterns. For the former, these enable the necessary feedback-loops for self-organisation to emerge, and they are introduced as a design abstraction for self-organising emergent solutions (as opposed to, for example, transition as abstraction). The latter (design patterns) are also a well known design abstraction in mainstream software engineering and here means: a catalogue of decentralised coordination mechanisms that says when to best use which mechanism. Examples of such mechanisms are digital pheromones, gradient fields, market-based control, and tags. In [21], an extensive overview is given of the catalogue in terms of problem/solution pairs for each pattern.

2.3.7. Discussion

We looked at related system design methodologies. As explained above, we consider model design to be an essential and integral part of any system design approach – in that respect, this article is complementary to the approaches described in this section. All presented methodologies follow (more or less) the general accepted steps of a traditional software engineering process: analysis, design, implementation, and testing and verification (although these terms do not always refer to exactly the same activities in the various methodologies). The METASELF, ADELFE, SOTL and CUP methodologies follow these steps quite closely (with all still contributing much over the traditional approach as this was required to make it fit for the design of self-organising, emergent systems). The TFSGO and A&A methodologies are somewhat different: TFSGO looks at self-organising dynamics of entities in combination with the environment that they are situated in; and A&A takes simulation and tuning as the way to come to a desired self-organising system. Considering all mentioned approaches, the work that we present here for model design can be incorporated and offer some structure for building a model of the system to be designed. For each approach, we have summarised in which phase this can be done best in Table 1.

In addition to this Table and these specific approaches, we would like to remark that design methodologies for multi-agent-based systems *in general* (not only for complex, adaptive and/or self-organising systems) are often intended for the design of 'open' systems, i.e., systems in which agents can join and leave the system at their own will, e.g., semantic web, grid computing. For designing CI, it is not necessarily the most important property of the system that it is open in this way. For some systems (file sharing, distributed computing), this may indeed be an important issue, but there are many other applications (e.g., optimisation, swarm robots) that do not need to be this open. In this way, CI design can better be considered as a kind of distributed problem solving (DPS) rather than open system design [86, p.320]. Still, it something new compared to DPS, because there is typically no centralised coordination. While DPS is often used in the context of coordinating how to distribute a complex and large computational process over a number of units, CI systems do not necessarily have such centralised coordination or are used in such contexts.

Finally, we point out a number of significant differences between the model design approach presented in our article and the way that the discussed approaches go about model design in the respective development phase (exactly which phase is shown in Table 1). We enumerate these differences here and go into details of each in the next Section. Firstly, we make no explicit distinction between an agent and the environment as different 'entities' in a model. Instead, we model the environment as a *function* that takes in the output (actions) of agents and generates new input (observations) for the agents. Secondly, we identify a number of notions that are recurrent in the considered types of systems (e.g., diversity, non-determinism, adaptivity) and allow for the explicit modelling of these notions. Next, we give an overview of existing models and algorithm that are useful for analysing or achieved desired dynamics of a complex system, e.g., cellular automata, boolean networks, evolutionary algorithm and classifier systems. In the engineering literature on system design, these models and algorithms are usually not addressed or considered. Finally, we aim for our approach not only to be useful in the context of design studies, but also for building models for analysis studies (the distinction between these two study types was made above). The discussed approaches are specifically meant for system design and cannot easily be used for analysis studies. An important reason for this is that these analysis models are to be built by domain experts who are typically not computer scientists (but, for example, biologists, social scientists or cultural anthropologists). Our approach facilitates such interdisciplinary usage.

2.4. Summary

This Section worked out the concept of CI further as an umbrella term covering systems ranging from open multi-agent and self-organising system to swarm intelligence and complex adaptive systems. We gave an overview of related research

Table 1
Overview of the phase(s) for each discussed system design approach in which our approach for model design can be incorporated.

Approach	Phase
TFSGO	System/entity description
METASELF	Design
AMAS + ADELFE	Requirements and analysis
A&A	Modelling
SOTL	Modelling
CUP	Design

streams, including, among others, self-organisation, multi-agent systems and swarm intelligence. Also, we presented an overview of models and system design approaches (numerical and conceptual in nature, respectively) for designing CI systems.

3. Model design

An essential part in any system design process is the model design phase: the "iterative decision-making process in which alternative abstract forms to represent a real-world system or situation are evaluated, compared and selected" [78]. Here, abstraction refers to selecting prominent features by which a real-world system can be represented in a meaningful way. That there can be different abstract forms comes from the particular purpose of a specific modelling process: descriptive, explanatory, hypothetical, experimental, predictive, innovative or epistemological. The overall decision-making process then means that specific abstract forms of the real-world system or situation are evaluated (how good is the chosen form), compared (how good are different forms relative to each other) and selected (based on the previous evaluations and comparisons). In this section, we show how to design CI system models.

Fig. 2 shows that model design is part of the generic system development cycle (system design). In this section we first describe the different phases in the model design process (second column in Fig. 2) and secondly, we go into more detail on the 'model' phase, consisting of generating generic, specific and computer models (third column in Fig. 2).

3.1. Systematic approach

Model design is thus one particular phase in system design. We explain the different phases of a systematic approach to model design as shown in Fig. 2 in the second column.

3.1.1. Problem assessment

In this phase, the developer: (1) identifies the involved entities, (2) describes the input data; (3) identifies performance measures; and (4) describes expected results (if possible). If the developer has identified (functional) requirements in the system design process, then these requirements can also be taken into account for the problem assessment.

3.1.2. Model

The purpose of this phase is to come from an informal description (in natural language) to a (semi-)formalised system design model. As in the previous phase it was decided what were 'entities' in the systems, in this phase these are given eyes (observations) and hands (actions). In multi-agent system development, there are often *organisational abstractions* used for modeling, like the concept of roles. As mentioned, after discussing the general research methodology, we explain our and existing approaches for design modelling in detail.

3.1.3. Simulation

An often used way of finding out about the functionality and behaviour of a system is by means of simulation. Many collective intelligent systems are researched as such, and in this article we accordingly give it much attention. However, there has been some scientific debate about simulation as a vehicle of investigation. Regarding this, we follow up on Axelrod's argument for the use of simulation as a third way to conduct scientific inquest (in the social sciences) besides induction and deduction:

> Simulation is a third way of doing science. Like deduction, it starts with a set of explicit assumptions. But unlike deduction, it does not prove theorems. Instead, a simulation generates data that can be analyzed inductively. Unlike typical

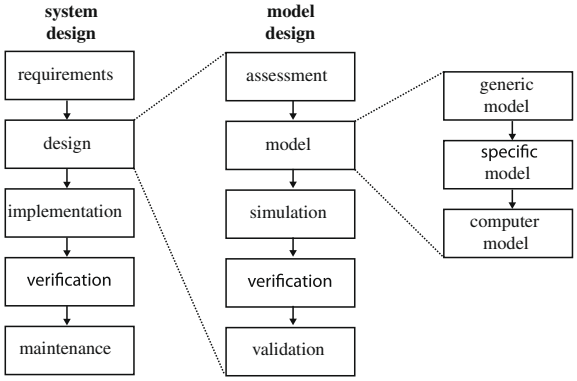


Fig. 2. Overview of different phases in system design (left column) and model design (right columns).

Table 2
Overview of simulation platforms.

Category	Platform	http://
Programmable modelling environment	Starlogo	education.mit.edu/starlogo
	Netlogo	ccl.northwestern.edu/netlogo
	Breve	spiderland.org/breve
	Newties	newties.org/
Libraries	Ascape	www.brookings.edu/es/dynamics/models/ascape/
	Swarm	swarm.org/
	MASON	cs.gmu.edu/~eclab/projects/mason/
	RePast	repast.sourceforge.net/
3D/Physical/ALife	Webots	www.cyberbotics.com/products/webots/
	FramSticks	www.frams.alife.pl/
Specialised	CORMAS	cormas.cirad.fr/
	MOISE+	www.lti.pcs.usp.br/moise/
	AgentSheets	www.agentsheets.com/
	LEADSTO	www.cs.vu.nl/~wai/TTL/
	SDML	sdml.cfpm.org/

> induction, however, the simulated data comes from a rigorously specified set of rules rather than direct measurement of the real world. While induction can be used to find patterns in data, and deduction can be used to find consequences of assumptions, simulation modeling can be used as an aid intuition. (From Axelrod [5].)

In the context of this article, Axelrod's statement is most applicable to what-we-call analysis studies. For design studies, simulation is more than an intuition aid: for these, there are two other practical reasons to include simulation. Firstly, pragmatically speaking, the type of systems to be designed have such a degree of complexity that simulation is the most accessible technique (compared to heavy theoretical frameworks). Secondly, simulation is widely used in the community as a tool to design collective intelligent systems.

There are many platforms available for simulation of CI systems. A number of surveys have evaluated such platforms [65,35,74]. We can roughly categorise the platforms as shown in Table 2: programmable modelling environments,[9] libraries, 3D/Physical/ALife and specialised. These platforms should be considered worthwhile for prototyping a system and performing simulations in order to get "finger-tip feeling" for the behaviour of your system. For further development and implementation, other software is more suited, e.g., that is particularly used for system design (software for this can be found by following the references in Section 2).

3.1.4. Verification

Rigorous analysis of obtained results is crucial for establishing the scientific worth of the conducted study. The two involved activities are *output analysis* and *validation*. In the next Section, we discuss validation. With output analysis, we mean the *statistical verification of the hypotheses*. Many existing statistical methods and techniques can be used for this purpose, e.g., two-sample *t*-test and Analysis of Variance (ANOVA). We do not go into more detail here. However, we would like to stress the importance of this step and mention that in good working-practice it should not be ignored.

3.1.5. Validation

Validation is an essential step: determining whether the model and solution fit reality. Unfortunately it is often neglected. For the design of collective intelligent systems, to our best knowledge there is no single method or technique for validation. In contrast, if one considers the available literature about validation and verification of (simulation) models in general, there is an abundance of methods and techniques that can be used [49]. At this point, we can only give the strong recommendation to check the results with a domain expert. More general methods and techniques need to be developed in future research.

3.2. Designing the generic model

As shown in the previous Section, design modeling is the second step in the overall research methodology. In this section, we present the two stages that together form the design modeling part: generic and specific modeling – see also the third column in Fig. 2. For the generic model, we give some fundamental recipes (or: algorithms) to be used as a starting point for modeling collective intelligent systems. In the next Section, we show how to *instantiate* these generic models based on existing models and algorithms.

[9] These software programs offer a complete programming environment for creating a simulation, including a visualisation of the simulation and often their own programming language.

Table 3
The control loop.

Control loop	
1:	initialise individuals
2:	initialise world
3:	while true
4:	for each individual
5:	observe the world
6:	perform an action
7:	end for
8:	world: determine new observations
9:	world: process costs and benefits for all individuals
10:	end while

Table 4
The BASIC recipe.

Recipe: **BASIC**	
1:	Determine action set for all individuals
2:	Determine observation set for all individuals
3:	Determine action → observation methods
4:	Determine costs for individuals for methods from 3
5:	Determine benefits for individuals for methods from 3
6:	Determine observation → action methods for all individuals

Let us first make a conceptual distinction between the *world* (or: environment) and *individuals* inhabiting that world. Sometimes these terms are really as straightforward as that, for example, an ant colony that learns to live in some hostile environment. In other, more abstract, studies, the world may actually be some search space defined by some optimisation criteria and the individuals inhabiting this world may be solutions for value optimisation within this search space.

The very first (basic) recipe that we present lets you set up a simple world and simple individuals. After that, we present a number of extensions on this basic recipe that allow you to model: internal states, diversity, non-determinism and adaptivity. Although we present each as a separate extension of the basic model, it is possible to combine all extensions in one model. The recipes will deliver you the *data sets* necessary to execute the loop in Table 3, e.g., a set of actions that an individual may choose to execute at some suitable point. The data sets alone do not give you an *operational model* yet, i.e., something in which you can press a Run button in order to execute the model. Therefore, we assume a control loop as shown in Table 3. This loop basically says that, after initialisation, the execution (or: simulation) runs forever, and every iteration the individuals look around and perform some action, the world is updated and individuals are re-evaluated. The control loop in combination with one of the recipes presented below, give you a model that you can execute – for example, a so-called microworld [60].

This control loop is definitely not the only or always best way to execute your model. Neither is it ready-to-use. For example, one must assure, when desired, that the individuals do-their-thing in parallel. Also, one must arrange for the world to be updated correctly and timely such that one individual does not see a different world as another individual in the same iteration. Or, your individuals may perform multiple actions per iteration. It may even be that an iteration-based control loop is not the correct model for your particular study (however often it is used in similar studies). Thus, we take this control loop such that we have something to refer to when presenting the recipes, but be aware that it may take some playing around with before it suits your needs.

3.2.1. BASIC

The basic recipe for modelling CI is shown in Table 4. It concerns 6 steps that take you from determining what individuals can do (actions, line 1) and can see or hear (line 2) up to deciding what they should do when (line 6).

Firstly, one decides what actions an individual can undertake. An action is something that causes a change in the environment. This may be non-trivial; for example, if you move one step forward, then this is considered to be an action because it changes your position in the world. In this first modelling step, the only thing you actually do is *naming* the actions: you do not need to specify results or costs. You will do this later, in the other steps. However, you may want to parametrise your actions; that is, extending them with variables, for example, `move(`s`)`, where s denotes the speed at which one moves.

Secondly, you determine the observations of the individuals. This is the available information that an individual may receive from the world (including other individuals). It may be that your individuals actively look around themselves – in which case observation (or: observe) must be an explicit action. It may also be that your environment supplies your individuals (each iteration) with fresh information – then it is not necessarily an explicit action, but rather a step in the control loop of your world. Like with actions, the only thing you do here is specifying a set of observations by *naming* them. Just as for actions, it may be useful to parametrise observations: for example, `position(`x`,` y`)`, where x and y are coordinates.

Table 5
The BASIC + internal states recipe.

Recipe: **BASIC + internal states**	
1:	Determine action set for all individuals
2:	Determine observation set for all individuals
3:	Determine action → observation methods
4:	Determine costs for individuals for methods from 3
5:	Determine benefits for individuals for methods from 3
6:	Determine (internal state) set for all individuals
7:	Determine observation → (internal state) methods for all individuals
8:	Determine (internal state) → action methods for all individuals

Thirdly, we model the world – however, this might be somewhat counter-intuitive. We define a set of methods that map actions to observations.[10] Consider the `move-forward` action. Assuming that you can observe your own position in the world, the result of this action is that in the next iteration you perceive an observation with updated information about your position in the world. At this point is an opportunity to decide whether your individuals may perform multiple actions per iterations or not. Also, by means of these methods you can let an attempt to perform an action fail or succeed. For example, you may decide to `move-forward`, but the method that maps this action to a new observation may let it fail – your position is still the same in the next iteration. Thus, in general, individuals attempt actions, and the world lets these actions fail or succeed.

Fourthly, doing something always comes with a cost. Thus the methods that were just defined, are assigned *costs*: an individual has to pay when performing an action. Therefore, an individual is assumed to have some property that expresses its *fitness*. Costs are deducted from this fitness. Fitness is a difficult concept. One might model it rather straightforward as it being energy that is accumulated and used throughout the individual's life. But, in evolutionary sense, it may also be the number of offspring an individual brings forth (as suggested by Holland [42]). This is more difficult to model, but sometimes necessary. Costs influence fitness negatively (and possibly indirectly).

Fifth, successful actions are beneficial for you – otherwise you would not undertake them. This means that you earn when an attempted action is successful. Therefore, the methods from line 3 are associated with benefits. However, if this is not necessary, then the benefit can be 0. It may, for example, not make sense to assign benefits to moving. Benefits influence the fitness positively.

Sixth, you specify the behaviour of the individuals. This is done by means of a number of methods mapping observations to actions. You can understand these as being simple `if-then` rules. But actually, this can be any representation as desired, for example, a decision tree or Bayesian network – as long as observations come in and actions come out. Since the type of systems under consideration here usually consist of many simple and reactive (as opposed to pro-active) individuals, the suggested condition-action rules are well suited to represent such individuals. However, if the system require more 'intelligent' agent that are also pro-active (or: goal-oriented), the addition of a rule inference engine (as in knowledge-base systems) or utility-labeled rules can produce this desired behaviour. As mentioned, since we here mainly consider simple individuals, we do not go into details of (utility-based) rule inferencing.

In these BASIC models, it is only possible to model the *outside* (or: behaviour) of individuals. Because we only allow actions to be caused by observations (line 6), this prevents you from modelling individuals with a memory or internal state (e.g., hunger, anxiety, and so on). This modelling is the simplest (scientifically speaking), you often want to model the *inside* of your individuals too. Therefore, you can extend your individual with internal states, which we explain next.

3.2.2. BASIC + internal states

You may notice quite quickly, that you run into trouble with the basic model. It is too basic. For example, sometimes you just need your individuals to *remember* something, or the individuals may not always behave the same *all the time*, or they may have *roles* that determines their behaviour and it is troublesome to model these enumeratively in single observation-action mappings. Therefore, this extension allows for the system's individuals to have an *internal state* – see Table 5. Although such a state (in its simplest form) may only hold one or more variables, it can also contain a complete *internal model*. This term was used by Holland [42] and denotes a model that takes some input, filters it into patterns which can be used to change itself. An individual that has such a model can anticipate results when it receives input, which had been received before.

In Table 5, we have replaced line 6 from the basic model with lines 6–8. Basically, we have created an intermediate state, called *internal state*, that is in between observations and actions. Such a state is something *inside* the individual. For example, it may represent a memory or some specific role. The implementation of this concept is actually not as straightforward as with observations and actions. It depends very much on the specific goal that you have in mind. In case of a memory, it might be a data structure that is gradually filled when observations come in; for different roles, it may be a container with different role descriptions, which are activated based on particular observations.

[10] This renders the world 'stateless', i.e., we do not explicitly define the state of the world, but we do this implicitly via this set of methods.

Table 6
The BASIC + diversity recipe.

Recipe: **BASIC + diversity**	
1:	Determine action set *for each individual*
2:	Determine observation set *for each individual*
3:	Determine action → observation methods
4:	Determine costs for individuals for methods from 3
5:	Determine benefits for individuals for methods from 3
6:	Determine observation → action methods *for each individual*

Table 7
The BASIC + non-determinism recipe.

Recipe: **BASIC + non-determinism**	
1:	Determine action set for all individuals
2:	Determine observation set for all individuals
3:	Determine action → observation methods
4:	*Assign probabilities to each of the methods from 3*
5:	Determine costs for individuals for methods from 3
6:	Determine benefits for individuals for methods from 3
7:	Determine observation → action methods for all individuals
8:	*Assign probabilities to each of the methods from 7*

3.2.3. BASIC + diversity

The basic recipe assumes that all individuals behave the same: all individuals have the same action set, observation set and policy methods. But this may not be correct for your study. Hence, you may want to model *diversity*. This requires to identify possibly dissimilar action and observation sets and policy methods for the (subsets of) individuals. The resulting recipe is shown in Table 6. Here, lines 1, 2 and 6 are not defined for all individuals (like in the basic recipe), but *for each individual*.

3.2.4. BASIC + non-determinism

Nothing in the real world is certain – and that is for certain. Therefore, making models of the real world requires modelling uncertainty. We can make our models to allow for this uncertainty by including *non-determinism*: not every state of affairs, including human event, act, and decision is the inevitable consequence of antecedent states of affairs.

Table 7 shows that we can model non-determinism by assigning probabilities to the methods that we have identified for mapping actions to observations and vice versa. We are aware of the fact that this is a very pragmatic approach and can be scientifically argued. The idea is that on the environment level (lines 3 and 4) uncertainty exists as such that undertaken action may not always have the intended results. For example, a robot attempts to `move-forward`, but some slippage on the surface causes the robot to move somewhat more to the right rather than forward. This can be modelled by means of assigning a probability to the (world) method that takes `move-forward` as input. On the individual level (lines 7 and 8), uncertainty manifests itself if individuals do not always undertake the same actions in the same situation – even without allowing for internal states (discussed above) or adaptivity (discussed below).

We do not prescribe the origins of these probabilities. The probabilities on the environment level may have come from domain expert knowledge; or, your model may be directly coupled to physical environment (for example, in case of collective robotics) in which case these probabilities may fluctuate over time. On the individual level, if your individuals are non-adaptive, then these probabilities are hardwired (determined beforehand) into the system. They may also come from domain knowledge. Finally, be aware of the fact that non-determinism introduces inherent complexity in your model, but which is sometimes necessary.

3.2.5. BASIC + adaptivity

Adaptivity is the most complicated extension that we discuss here. It concerns the ability of individuals to change themselves if the environment requires so. Table 8 shows the recipe for adaptivity: we have replaced line 6 of the basic recipe. Now, line 6 says that you have to choose and implement a mechanism that actually *generates* the methods that map observations to actions. Example mechanisms are evolutionary computation, neuro-evolution, and co-evolution, among others. Some mechanisms already specifically take collectives into account: the collective intelligence (COIN) framework of Tumer and Wolpert or the Particle Swarm Optimisation method [24].

There may be many reasons to include adaptivity in your model. For one, you may have observed it in nature (e.g., an adaptive ant colony) that you want to model in order to understand it better. Or you wonder how some rule-sets that you discovered (about how birds flock, for example) have evolved as to what they are now.

Table 8
The BASIC + adaptivity recipe.

Recipe: **BASIC + adaptivity**	
1:	Determine action set for all individuals
2:	Determine observation set for all individuals
3:	Determine action → observation methods
4:	Determine costs for individuals for methods from 3
5:	Determine benefits for individuals for methods from 3
6:	Choose and implement mechanism that generates observation → action methods for all individuals

Table 9
Overview of models that can be use for the specific model.

Name	Description
Models	
Cellular automata (CA) [82]	A CA is "an array of identically programmed automata, or cells which interact with one another in a neighbourhood and have definite state". They can be complex systems themselves, and offer good ways of studying the behaviour of such systems
Multi-Agent Systems (MAS) [84]	The MAS paradigm involves the investigation and design of situated agents that exhibit autonomous and rational behaviour with respect to their environment and to each other
Boolean Networks (BN) [44]	The basis of a BN is a set of logical propositions, which can be evaluated true or false and are combined by logical operators (OR, AND, XOR, etcetera) in some networked fashion
NK-Model [45]	A *random Boolean network* (RBN) is a network is a Boolean network of which the configuration (defines how the propositions are connected to each other) is unknown, and where the gate types and their connections are randomly initialised. The number of gates is denoted by *N* and the number of inputs to each node by *K*, hence the name of this model
Particle-Based Models [40]	Farkas, Helbing and Vicsek have successfully modelled human panic behaviour in case of a crowd stampede based on a physical force model for many-particle systems
Game and Decision Theory [59]	A game involves all those situations where "a number of players or decision makers interact, possibly threaten each other and form coalitions, take actions under uncertain conditions, and finally receive some benefit or reward or possibly some punishment or monetary loss". The study of such games is the area of *game theory*
Formal Logics [4,23]	Many AI researchers have used formal logical models, e.g. agent societies have been formalised using logic; agent-based social simulation has been compared with formal systems, and human organisations have been formally modelled
Knowledge Systems [56,72]	A knowledge system is a computer system which represents and uses knowledge to accomplish a task. The Physical Symbol System Hypothesis says that "physical symbol systems are necessary and sufficient for general intelligent action". This hypothesis, although debatable, justifies the interest in knowledge systems

In this way, the mechanisms mentioned above are generally used for completely different purposes: mostly in the context of search or optimisation. For example, evolutionary methods are used as search meta heuristic to find optima in very large search spaces. Thus although the inspiration may have come from nature (evolution), the mechanism is not implemented/simulated to find out more about this phenomenon; rather, it is used as a technique for exploring large abstract search spaces in order to solve some problem. This brings us to the above-mentioned distinction between analysis and design (problem solving studies).

3.3. Designing the specific model

When the generic model has been designed, one does not have to re-invent the wheel in order to further specify the model: there are many existing *models* that can instantiate such a generic model. We here give a concise overview[11] of these models. Moreover, especially for adaptation, there are many *algorithms* available to accomplish this – these will also be overviewed here. Tables 9 and 10 contain the overviews of models and algorithms, respectively. The descriptions of the models and algorithms are on purpose kept short: with the included references, the reader can orientate him/herself in more detail on the specific model of his/her interest.

When we have finalised the specific model, the final step to complete the design of the model is to transform this model into a computer model. In other words, while the generic and specific models were still "designed with pen and paper", the computer model is programmed into the computer and used for further simulation and experimentation. The choice of

[11] An alternative way of looking at this overview is to consider the presented models and algorithms as a collection of tools that together form a kit to research collective intelligence – just like chemists and biologists use microscopes, vats, Petri dishes, test tubes, etcetera.

Table 10
Overview of algorithms that can be use for the specific model.

Name	Description
Algorithms	
Evolutionary Methods [25]	An evolutionary method (or: algorithm) is "a generic population-based metaheuristic optimisation algorithm". The heuristics that it uses are inspired by biological evolution: reproduction, mutation, recombination, survival of the fittest, and natural selection
Co-evolution [25, Chapter 13]	If a natural environment does not give any information about the quality of individuals, then evolution is impossible. However, if the environment is *informative*, then we can define fitness on the basis of *co-evolution*. A typical setup of a basic co-evolutionary algorithm then consists of two populations with first and second player strategies, respectively
Learning Classifier Systems (LCS) [14]	LCS exploit the combination of evolutionary computation and reinforcement learning to develop sets of condition-action rules. They exploit "possibilities of general-purpose learning algorithms applied to parallel rule-based systems"
Neuro-Evolution (NE) [71]	NE is a technique that evolves artificial neural networks with evolutionary methods. The evolutionary operators of NE work on the weight space of the network: recombination combines weight vectors of complete networks; mutation works on single networks' weights
COllective INtelligence (COIN) [76]	A COIN is a multi-agent system where: (1) the agents each run reinforcement learning (RL) algorithms; (2) there is little to no centralised communication or control; and (3) there is provided a *world utility function* that rates the possible histories of the system
Particle Swarm Optimisation (PSO) [26]	In a PSO algorithm, particles form a swarm that flies through a (multi-dimensional) problem space, whereby the positions of the particles are updated each generation based on the particle's experience and that of its neighbours

computer model depends on the used software platform (i.e., from Table 2) – we do not go into further detail on this here, but refer the interested reader to the documentation of the mentioned platforms.

3.4. Summary

This Section presented a systematic approach to investigate CI systems. We mainly looked at the design modeling phase, consisting of a generic model stage (for which a number of modelling 'recipes' were presented) and a specific model stage (where off-the-shelf models and algorithms are used).

4. Case studies

In this section, we give two examples of the use of the introduced methodology. The first example introduces an online variant of the *Chinese Whispers* (CW) game, where different persons play the Chinese Whispers game in different languages on the Internet. The idea of the game is to iteratively learn each other's languages by playing many consecutive CW games. The example shows that it is possible to use our approach to build human–machine models of CI systems. The second example considers a collective of physical robots that have a collaborative task to surveil a specific area. To do this, they can exchange messages with each other about their whereabouts in order to perform the task more effectively.

These specific case studies were chosen with two explicit reasons in mind. Firstly, to show that the same methodology can easily be used for two quite distinct problems (information search in the internet on the one hand, and coordination of physical robots on the other hand). Secondly, to show that with the conceptual models that come out of this process, one is able to further build implementations that actually solve the problem at hand (we later show how the CW game is used for the problem of 'semantic interoperability' to let different knowledge bases talk to each other). Regarding the latter, the formal or conceptual models normally presented in papers are specifically suited for a single problem and are more ad hoc than the models presented here. The conceptual models are also very useful in an interdisciplinary environment, where establishment and definition of terms is often the first hurdle to take.

4.1. The chinese whispering room

Consider an online variant of the well-known *Chinese Whispers* (CW) game. We call this game the *Chinese Whispering Room* (CWR) after the combination of the CW game and the famous 'Chinese Room' thought experiment of Searle (1980). To recall, the original CW game is one where a group of persons whispers a sentence from person to person; the last person says the sentence out loud; after which the initiator can compare this with its original sentence. The extension we propose is one where the participants communicate via Internet and speak different languages. The aim of the game is to iteratively learn each other's language (only on word level, not grammar) by playing many consecutive CW games and in that way evolve to a *lingua franca (universal language) – hence the reference to Searle's 'Chinese Room'. Tackling this issue is a typical*

Table 11
CWR – Overview of *sets* for the generic model.

Action	Description
(a) Overview of actions (S1) that the CWR participants can undertake	
NETWORK	
`send-ping`	Send network echo request to a participant
`send-pong`	Send reply to a ping message (including language)
DATA	
`return-results`	Return results of the query to user or peer
`forward-query-to-Ne`	forward a query to neighbour with same language
`forward-query-to-Nd`	Forward a query to neighbour with different language
Observations	**Description**
(b) Overview of observations (S2) that the CWR participants can make	
NETWORK	
`received-ping`	Reception of a ping message
`received-pong`	Reception of a reply on a ping message (including language)
DATA	
`received-query`	Query received from a participant
`received-results`	Reception of results of a query
Internal state	**Description**
(c) Overview of Internal States (S6) that the CWR participants have	
DATA	
`(query,database)`	Tuple containing a query and the participant's database
`(query,translator)`	Tuple containing a query and the participants's translation dictionary
`(results,criteria)`	Tuple containing the results of a query and evaluation criteria

example of collective intelligence design: the overall goal of a 'lingua franca' can only be defined on the global level, while we can only 'control' the local behaviours of the involved participants in order to achieve this goal.[12]

Let us illustrate the CW case study with the following example. A possible starting position of the game is that there are 6 persons $p_1, \ldots, p_6$, each speaking their own language (persons 1 and 2 speak the same language, so do 5 and 6). Furthermore, let assume that the vocabulary of each person consists of a fixed number of words and that there are *complete* mappings between the languages (each word in each language can be mapped into a word in any other language – although strictly speaking this is not necessary, since the way that the game is played can deal with infinite numbers of words and incomplete mappings result in not achieving a *lingua franca).* The game works as follows. Each participant can generate a sentence in his/her own language and send this to another person with the request to translate it into a particular language. The sender attaches his/her name to the sentence. The receiving person attempts to translate as much as possible of the sentence and then either returns the sentence back to the original sender (if completely translated), or forwards it to another person (if not completely translated) and attaches his/her name. In case of forwarding, participants always try to forward a sentence to the person that they think can best translate the rest of the sentence. When the original sender receives back his (translated) sentence, we assume that (s)he can check the quality of the translation – not by means of direct translation, which would render the whole framework redundant, but, for example, by means of checking implicitly with a speaker of the language (i.e., "if you think that you order a drink, and you get a drink, then the translation was good; if you do not get a drink, then it was not"). Such a quality assessment allows him/her to evaluate the chain over which the translation travelled.

4.1.1. Model specification

We want to build a model of the CWR game. There can be several reasons to do this, e.g., we may be interested in analysing how quickly a *lingua franca* emerges for a certain set of players, or we want to put computers in to take up the roles of participants. Following our approach, we first decide on which recipes to use (choosing from a combination of basics plus internal states, diversity, non-determinism and/or adaptivity), then go through the recipes to obtain a generic model, and finally instantiate this in a specific model.

4.1.1.1. Used recipes. For this study, we need the agents to have internal states (for example, to store the peer's *neighbourhood*). For sake of simplicity, we do not model diversity, non-determinism or adaptivity. The recipes that we use for the generic model are shown in Table 4 (basic) and Table 5 (internal states) from Section 3.

[12] Also, the relevance of this game is the issue of *semantic interoperability* on the semantic web [1,3], which is about enabling different knowledge bases talk to each other (i.e., answering each other's queries) while they speak different 'languages' (i.e., use different ontological schemas). Aberer et al. propose the *semantic gossiping* mechanism to achieve semantic interoperability [1]. The mechanism works similar to the CW game, as peers are able to detect network cycles and can compare their own query with the transformed one (resulting from the cycle), based on which semantic agreement can be adjusted locally and ultimately achieved globally. Additionally, the issue of peer-exchange of data in general is a huge topic of research and application in the development of the Internet, e.g., [31,39,62].

Table 12
CWR – overview of *methods* for the generic model.

Method	Description
(a) Overview of (action → observation) methods (M3)	
NETWORK	
`i.send-ping → j.received-ping`	Peer *i* sends a ping message to participant *j*
`i.send-pong → j.received-pong`	Peer *i* sends a pong message to participant *j*
(b) Overview of (observation → action) methods (M6)	
NETWORK	
`received-ping → send-pong`	Send a pong reply to a ping message
(c) Overview of (observation → (internal state)) methods (M7)	
DATA	
`received-query → (query,database)`	Look up a query in own database
`received-query → (query,translator)`	Look up the translation of a query
`received-results → (results,criteria)`	Update the criteria based on received results
(d) Overview of ((internal state) → action) methods (M8)	
DATA	
`(query,database) → return-results`	Return results of the query to participant
`(query,database) → forward-query-to-Ne`	Forward query to neighbour with same language
`(query,translator)` AND `(query,criteria)` `→ forward-query-to-Nd`	Forward query to neighbour with different language

4.1.1.2. Generic model. Since we are using the basic and internal states recipes, for the generic model we need to define a number of sets and methods.[13] Regarding the basic part of the model, we define actions, observations, and action → observation. For the part with internal states, we define internal states, observation → (internal state), and (internal state) → action. The sets are shown in Table 11, and the methods are shown[14] in Table 12.

Note that we defined all the sets and methods on two levels: network and data. This is to distinguish the 'network discovery capabilities' of the persons (i.e., by means of ping/pong messages participants can find out information *about* the network) from the data exchange (i.e., the game allows for participants to send translation requests to each other in order to obtain information *from* the network.).

4.1.1.3. Specific model. The choice for a specific model depends on the objective of the research. We consider and work out two possible objectives: first, we want to know how fast a *lingua franca* emerges under specific circumstances; second, we contemplate the addition of computers as participants in the game (for example, to speed up the convergence to a *lingua franca).*

For the first objective, wanting to know how fast convergence happens translates into analysing the *dynamics* of the information flow in the network. For example, considering that we have some given *begin* situation (e.g., two Dutch, three English and four Germans), we could count the number of iterations up to which there is no participant anymore that needs to contact another participant upon reception of a query for translation. While the first model that comes to mind in this situation may be a multi-agent system, there are actually other existing models in which we can investigate exactly these kinds of objectives on the system dynamics. The cellular-automata model (or its bigger brother, a boolean network) is a suitable model to research dynamics of information flows. For a CA model, we have to identify the entities, establish the interactions that they can engage in with their neighbours and then 'run' the model. We can choose from a plentitude of methods for analysing the dynamics of CA models. And with the generic model specification readily available (including specification of the entities and their interactions), a CA model can be rapidly constructed to research this objective.

For the second objective, we thus consider including computerised participants in the network, for example to speed up the convergence towards a *lingua franca.* Then we are thus interested in a more 'active' adaptation of the participants. We may then consider optimisation models and machine learning algorithms like particle swarm optimisation, evolutionary methods or learning classifier systems. Regarding the earlier mentioned network/data levels, it is interesting to see what effect computers may have on both levels. It is expected that they can discover information *about* the network much faster (i.e., sending/receiving ping/pong messages), but may be much less creative when it comes to obtaining and processing information *from* the network (i.e., a human participant may quickly realise that 'quase' in Portugese and 'casi' in Spanish are similar in meaning ('almost' in English), but a computer may not).

4.2. Collective robotics: Braitenberg collectivae

Based on the well-known *Braitenberg vehicles* [12], we present the *Braitenberg Collectivae* (BC) scenario: a collective of vehicles that have to perform a collaborative task. In this particular case study, shown in Fig. 3, the vehicles have the

[13] Here, we use the notation S*i* and M*i*, where S and M refer to set and method, respectively, and index *i* corresponds to the line number from Table 5 from Section 3.

[14] We have added a Table that contains the (observation → action) methods that do not need internal states.

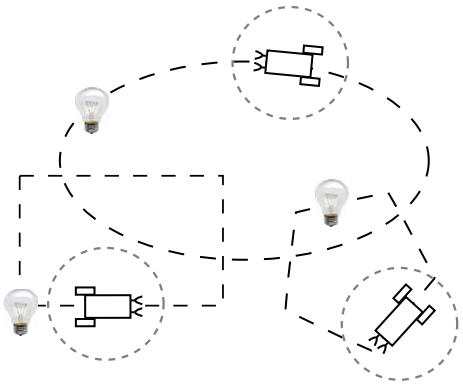


Fig. 3. The *Braitenberg Collectivae* scenario, where charging areas are indicated by a light bulb, routes of the vehicles are indicated by the dark-colored dotted lines and communication ranges are indicated by light-colored dotted lines.

continuous (individual) task to keep their batteries charged and the collaborative task to surveil as much area as possible. For the application of this scenario, one can think of an industrial terrain with robot surveillance to spot intruders. Concerning the capabilities of the vehicles, in comparison with the original ones, the BC vehicles can *communicate* with each other. The specific information they exchange depends on the collaborative task. In this case, the vehicles exchange information about where they have been. As the vehicles move around charging and surveilling, each one will ultimately travel one static route – this is shown in Fig. 3.

The original study by Braitenberg demonstrated that simple agents can produce complex behaviour by exploitation of the environment's complexity. Although our BC vehicles are a bit more complex (they are able to communicate and are aware of where they are), their individual intelligence is still quite close to zero – the aim is to show that they can achieve some reasonable degree of *collective* intelligence. We take vehicle II.(b) from the original study as our starting point [12, p. 7]: a vehicle with a left and right sensor and two wheels, and where the left wheel is connected to the front-right sensor and the right wheel is connected to the front-left sensor. These connections function as reinforcers between the sensors and wheels: the level of excitement of the sensor makes the connected wheel turn either faster or slower. A BC vehicle exhibits such behaviour if it has enough energy; otherwise, it tries to move back to the light to recharge its battery.

The BC scenario is a design study, because we have to design the individual behaviours of the vehicles as such that they can perform their collaborative task as good as possible. For this case study, we demonstrate the design modeling step yielding a model specification which one can use further to implement a simulation model. We have aimed to keep the resulting system as simple as possible, without trivializing it. The main aim of presenting this case study is to demonstrate how our methodology works. Note that at some points we have made certain design choices for the sake of simplicity of the model. Also, we have made no attempt to make the model compact for the purpose of clarity. Hence, all sets and methods are described extensively, while we are aware that intensive description make the model much more compact. Finally, we do not address the other steps of the overall research methodology from Section 3, i.e., we ignore the steps before and after the design modeling.

4.2.1. Model specification

We have structured the model specification the same way as the previous case study: used recipes, generic model and specific model.

4.2.1.1. Used recipes. Here, we also need the agents to have internal states (to store the communicated information). Again, we do not model diversity, non-determinism or adaptivity. Thus we use the basic and internal state recipes.

4.2.1.2. Generic model. We present the generic model specification in the same way as the previous case study. We present the necessary sets (actions, observations, internal states) in Table 13 and the methods in Table 14.

4.2.1.3. Specific model. As mentioned before, the first step in developing a specific model is deciding on which models and/or algorithms to use. In Section 3.3 we gave an overview of the existing models and algorithms of which we can choose. In this case study, a vehicle cannot change its behaviour itself, they merely react to incoming information. Thus, we do not have to include *adaptivity*, and there is no need to include any learning algorithm, from Table 10, here. For the models, the *Multi-Agent-Based Systems* model seems to be the most appropriate one here. For such a MA(B)S model, we can use any the well-known models (or: standards/platform/toolkits) to further specify the BC model, e.g., FIPA, KIF, JADE, JACK [70]. Also, the frameworks from Section 2 can be used in this case. We do not do this modelling exercise here.

In many situations, the MA(B)S model qualifies as an appropriate model. However, the other presented models should not be ignored and can be very useful *depending on the particular research objectives*. We briefly go through the list of models in

Table 13
BC – Overview of *sets* for the generic model.

Action	Description
(a) Overview of Actions (S1) that the Braitenberg vehicles can perform	
`left-wheel-speed-increase`	Increase speed of the left wheel
`left-wheel-speed-decrease`	Decrease speed of the left wheel
`right-wheel-speed-increase`	Increase speed of the right wheel
`right-wheel-speed-decrease`	Decrease speed of the right wheel
`broadcast-message`	Broadcast message (here: `visited-locations`)
`correct-steering`	Deviate from current heading
Observations	**Description**
(b) Overview of Observations (S2) that the Braitenberg vehicles can make	
`left-intensity-increase`	Increased intensity in left sensor
`left-intensity-decrease`	Decreased intensity in left sensor
`right-intensity-increase`	Increased intensity in right sensor
`right-intensity-decrease`	Decreased intensity in right sensor
`incoming-messages`	Contains the messages of other agents
`my-location`	Current location of this agent, e.g., (x, y)
`my-energy`	Current energy level of this agent, e.g., 0–100%
`critical-energy-level`	Minimal required energy to return to light (constant)
Internal State	**Description**
(c) Overview of Internal States (S6) that the Braitenberg vehicles have	
`visited-locations`	List of last X locations of the agent
`unvisited-locations`	List of locations considered unvisited

Table 14
BC – overview of *methods* for the generic model.

Method	Description
(a) Overview of (action → observation) methods (M3)	
`right-wheel-speed-decrease` → `my-location`	Moving should change agent's location
`right-wheel-speed-increase` → `my-location`	Moving should change agent's location
`left-wheel-speed-decrease` → `my-location`	Moving should change agent's location
`left-wheel-speed-increase` → `my-location`	Moving should change agent's location
`right-wheel-speed-decrease` → `my-energy`	Moving decreases agent's energy
`right-wheel-speed-increase` → `my-energy`	Moving decreases agent's energy
`left-wheel-speed-decrease` → `my-energy`	Moving decreases agent's energy
`left-wheel-speed-increase` → `my-energy`	Moving decreases agent's energy
`TRUE` → `critical-energy-level`	Always observe this value by default
(b) Overview of (observation → action) methods (M6)	
(`my-energy` > `critical-energy-level`) AND	
`left-intensity-increase` → `right-wheel-speed-decrease`	Vehicle II.(b) behaviour
`left-intensity-decrease` → `right-wheel-speed-increase`	Vehicle II.(b) behaviour
`right-intensity-increase` → `left-wheel-speed-decrease`	Vehicle II.(b) behaviour
`right-intensity-decrease` → `left-wheel-speed-increase`	Vehicle II.(b) behaviour
(`my-energy` ⩽ `critical-energy-level`) AND	
`left-intensity-increase` → `right-wheel-speed-increase`	Steer towards light
`left-intensity-decrease` → `right-wheel-speed-decrease`	Steer towards light
`right-intensity-increase` → `left-wheel-speed-increase`	Steer towards light
`right-intensity-decrease` → `left-wheel-speed-decrease`	Steer towards light
(c) Overview of (observation → (internal state)) methods (M7)	
`my-location` → `visited-locations`	Keep track of visited locations
`my-location` AND `incoming-messages`	
→ `unvisited-locations`	Keep track of unvisited-locations
(d) Overview of ((internal state) → action) methods (M8)	
`visited-locations` → `broadcast-message`	Broadcast visited locations
(`my-energy` > `critical-energy-level`) AND	
`unvisited-locations` → `correct-steering`	Steer towards unvisited-locations

Table 9 and explain in what situation one could use each model. For *cellular automata*, *boolean networks* and *NK-models*, there is one important reason to use these: to analyse the *dynamics* of the system. These models belong to the same model class, where a cellular is the most restricted one (a grid of cells where all neighbours are decided beforehand and also the behaviour of each cell) and the *NK*-model is the most generic one (a graph where the behaviour of the cells can change over time

and where it is not decided beforehand who are neighbouring nodes). In the BC-scenario, would we, for example, be interested in the effect of one or more parameters on the system's behaviour over time, we can use these models for this. It could be that making the `correct-steering` method too 'strong' (so that it often triggers), the system will never converge to stable routes. A *particle-based model* is useful when there is an analogy in Nature for your design problem. If we take the example with the pedestrians (where each pedestrian is represented by a particle and a model from physics is used to model realistic human behaviour), your vehicles may benefit from knowing about how people move about in a crowded room. In that case, one can represent the vehicles as particles and take the physical model to investigate steering effective strategies. If we are to use a *game and decision theoretic* framework, we can investigate how the vehicles react to each other. Viewing the interactions between the vehicles (i.e., seeing and reacting to other vehicles) as moving about (or: taking turns) in a game, we can use game-theory to research effective strategies taking into account the behaviours of other vehicles. There are many reasons to *formalise* your model further and represent/analyse it logically. With such a model, you may try to prove (instead of discovering experimentally) that, for example, the vehicles will converge to stable routes [53]. Alternatively, one may want to formally analyse the different aggregation levels (from local individual level of each of the vehicles to the global system level in terms of total surveilled area). Finally, we can represent the controllers of vehicles as *knowledge systems* (which will usually happen at the knowledge level, hardly the symbol level). Although such systems were originally meant for 'isolated' knowledge processors, there are many situations in which we can use this body of work when designing collective intelligence. For example, would the vehicles be adaptive and the adaptation mechanism is a *Learning Classifier System*, then your controllers will necessarily be represented as knowledge systems, albeit rather simplistic systems.

4.3. Summary

In this section, we presented two case studies to illustrate the introduced methodology. The first case study concerned self-organising knowledge systems, where a network of nodes tries to achieve global semantic agreement on an individual-based manner. The second study concerned a set of robots that have a collective surveillance task. Both are examples of design problems where there is an explicit global aim, and where the only design parameters are local.

5. Conclusions

In this article, we have made an elaborate effort to break a lance for the process of model design for *collective intelligent* (CI) systems. We introduced CI systems as the umbrella term covering many existing systems as multi-agent, swarm, complex, adaptive and self-organising systems. The application range of this system is very wide, of which we gave some examples in this article (e.g., collective robotics, computer networks, and applications in games and movies). We consider this article an opportunity to bring together different research communities by pointing out various similarities, and using the presented ideas as a potential research vehicle.

The main contribution is that we present a systematic approach for CI model design that is closely embedded in the generic system design development process (consisting of requirement, design, implementation and maintenance phases). The model design approach consist of three particular steps[15]: firstly, generic modeling, for which we provide a number of recipes modeling the different characteristics of CI systems (internal models, diversity, non-determinism, and adaptivity); secondly, specific modeling, for which we present an overview of existing models and algorithms which can be considered a researcher's toolkit for investigation of CI systems; finally, computer modelling, where the specific model is transformed into a computer model on which simulation experiments can be conducted.

In addition to the main contribution, this article also contributes to the general issue of designing self-organising systems: the explicit distinction of different notions that are recurrent in the considered types of systems (e.g., adaptivity, emergence); the distinction between analysis studies (understanding phenomena) and design studies (building systems); a set of useful models and algorithms in Tables 9 and 10 (and that are not usually mentioned in the context of system engineering); and a list of 'design' recipes to model some particular (but still unknown) type of systems.

This research field is in its infancy, although many aspects have been looked at in different disciplines and across different research topics. Still, there are many open issues for future research, of which we mention two major ones that need to be addressed to move the field forward. Firstly, in general, this research area needs *theory*. As mentioned in the example about designing flocks of birds: "we do not even know how to define (represent) this problem, never mind how to solve it". Secondly, regarding the empirical part of the methodology, we need to work on methods and techniques for *validation*. The best that we can do now is to talk to a domain experts to clarify of the obtained results are plausible and useful. Inspiration for such methods may come from areas where simulation is a well-established means of investigation, e.g., operation research.

References

[1] K. Aberer, P. Cudré-Mauroux, M. Hauswirth, The chatty web: emergent semantics through gossiping, in: Proceedings of the 12th International World Wide Web Conference, 2003, pp. 197–206.

[15] We want to mention explicitly that the design modeling stage cannot be seen in isolation of the other phases: modeling should be considered as a means to an end, not an end itself.

[2] C. Adami, Introduction to Artificial Life, Springer, 1999.
[3] G. Antoniou, F. van Harmelen, A Semantic Web Primer (Cooperative Information Systems)), The MIT Press, 2004.
[4] A. Artikis, J. Pitt, A formal model of open agent societies, in: E. Andre, S. Sen, C. Frasson, J. Mueller (Eds.), Proceedings of the Fifth International Conference on Autonomous Agents, 2001, pp. 192–193.
[5] R. Axelrod, R.A. Hammond, The evolution of ethnocentric behavior, in: Paper Presented at the Midwest Political Science Convention, April 2003, Revised Version.
[6] S. Bandini, S. Manzoni, G. Vizzari, Multi-agent modeling of the immune system: the situated cellular agents approach, Multiagent Grid System 3 (2) (2007) 173–182.
[7] G. Beni, J. Wang, Self organizing sensory systems, in: J. Tou, J. Balchen (Eds.), Highly Redundant Sensing in Robotic Systems: Proceedings of NATO Advanced Workshop on Highly Redundant Sensing in Robotic Systems, Springer-Verlag, 1990, pp. 251–262.
[8] C. Bernon, V. Chevrier, V. Hilaire, P. Marrow, Applications of self-organising multi-agent systems: an initials framework for comparison, Informatica 30 (2006) 73–82.
[9] C. Bernon, M.-P. Gleizes, S. Peyruqueou, G. Picard, Adelfe: A methodology for adaptive multi-agent systems engineering, in: P. Petta, R. Tolksdorf, F. Zambonelli (Eds.), Proceedings of the Third International Workshop on Engineering Societies in the Agents World, vol. 2577 of Lecture Notes in Artificial Intelligence, Springer, 2002, pp. 70–81.
[10] S. Bieniawski, D.H. Wolpert, I. Kroo, Discrete, continuous, and constrained optimization using collectives, in: Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2004.
[11] E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, 1999.
[12] V. Braitenberg, Vehicles: Experiments in Synthetic Psychology, The MIT Press, 1986.
[13] S. Bullock, Evolutionary Simulation Models: on their Character and Application to Problems Concerning the Evolution of Natural Signalling Systems, Ph.D. Thesis, University of Essex, 1997.
[14] M.V. Butz, Rule-Based Evolutionary Online Learning Systems, Springer, 2006.
[15] S. Camazine, J. Deneubourg, N. Franks, J. Sneyd, E. Bonabeau, G. Theraulaz, Self-Organization in Biological Systems, Princeton University Press, 2001.
[16] D. Capera, J.-P. George, M.-P. Gleizes, P. Glize, The AMAS theory for complex problem solving based on self-organizing cooperative agents, in: Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003, p. 383.
[17] M. Cossentino, N. Gaud, V. Hilaire, S. Galland, A. Koukam, Aspecs: an agent-oriented software process for engineering complex systems, Autonomous Agents and Multi-Agent Systems (2009).
[18] R. Damper, Emergence and levels of abstraction, International Journal of Systems Science 31 (7) (2000) 811–818.
[19] C. Darwin, The Origin of Species, John Murray, 1859.
[20] R. Dawkins, The Selfish Gene, Oxford University Press, 1976.
[21] T. de Wolf, Analysing and Engineering Self-organising Emergent Applications, Ph.D. Thesis, Department of Computer Science, K.U. Leuven, Leuven, Belgium, 2007.
[22] T. de Wolf, T. Holvoet, Emergence versus self-organisation: different concepts but promising when combined, in: S. Brueckner, G.D.M. Serugendo, A. Karageorgos, R. Nagpal (Eds.), Proceedings of the Workshop on Engineering Self Organising Applications, vol. 3464 of Lecture Notes in Computer Science, Springer, 2005, pp. 1–15.
[23] V. Dignum, F. Dignum, V. Furtado, A. Melo, L. Sonenberg, Towards a simulation tool for evaluating dynamic reorganization fo agents societies, in: Proceedings of the Workshop on Socially Inspired Computing at AISB Convention, 2005.
[24] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995, pp. 39–43.
[25] A. Eiben, M. Schut, T. Toma, Comparing communication protocols in evolving agent societies: cell phones versus shouting, in: P. McOwan, K. Dautenhahn, C. Nehaniv (Eds.), Evolvability and Interaction: Evolutionary Substrates of Communication Signalling and Perception in the Dynamics of Social Complexity, vol. TR-393, University of Hertfordshire, 2003, pp. 22–28.
[26] A.P. Engelbrecht, Fundamentals of Computational Swarm Intelligence, Wiley, 2005.
[27] J.M. Epstein, Generative Social Sciences, Princeton University Press, 2006.
[28] B. Farley, W. Clark, Simulation of self-organising systems by digital computer, Institute of Radio Engineers Transactions on Information Theory 4 (1954) 76–84.
[29] G. Folino, A. Forestiero, G. Spezzano, An adaptive flocking algorithm for performing approximate clustering, Information Sciences 179 (18) (2009) 3059–3078.
[30] R. Frei, G.D.M. Serugendo, J. Barata, Designing self-organization for evolvable assembly systems, in: Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, 2008, pp. 97–106.
[31] A. Fuxman, P.G. Kolaitis, R.J. Miller, W.-C. Tan, Peer data exchange, ACM Transactions on Database System 31 (4) (2006) 1454–1498.
[32] L. Gardelli, M. Viroli, M. Casadei, A. Omicini, Designing self-organising environments with agents and artefacts: a simulation-driven approach, International Journal of Agent-Oriented Software Engineering (IJAOSE) 2(2) (2008) 171–195 (Special Issue on Multi-Agent Systems and Simulation).
[33] C. Gershenson, Design and Control of Self-organizing Systems, Ph.D. Thesis, Vrije Universiteit Brussel, 2007.
[34] A. Ghosh, A. Halder, M. Kothari, S. Ghosh, Aggregation pheromone density based data clustering, Information Sciences 178 (13) (2008) 2816–2831.
[35] N. Gilbert, S. Bankes, Platforms and methods for agent-based modeling, Proceedings of the National Academy of the United States of America 99 (3) (2002) 7197–7198.
[36] N. Gilbert, P. Terna, How to build and use agent-based models in social sciences, Mind & Society 1 (1) (2000) 57–72.
[37] D.M. Gordon, Ants at Work: How an Insect Society is Organized, W.W. Norton, 1999.
[38] F. Grimaldo, M. Lozano, F. Barber, G. Vigueras, Simulating socially intelligent agents in semantic virtual environments, Knowledge Engineering Review 23 (4) (2008) 369–388.
[39] A.Y. Halevy, Z.G. Ives, D. Suciu, I. Tatarinov, Schema mediation in peer data management systems, in: ICDE, 2003, pp. 505–516.
[40] D. Helbing, I. Farkas, T. Vicsek, Simulating dynamical features of escape panic, Nature 407 (2000) 487.
[41] F. Heylighen, The science of selforganization and adaptivity, 2001. <http://pespmc1.vub.ac.be/Papers/EOLSS-Self-Organiz.pdf>.
[42] J. Holland, Adaptation in Natural and Artificial Systems, The MIT Press, 1975.
[43] J. Holland, Hidden Order: How Adaptation Builds Complexity, Perseus Books (1995).
[44] S. Kauffman, Metabolic stability and epigenesis in randomly constructed genetic nets, Journal of Theoretical Biology 22 (1969) 434–467.
[45] S.A. Kauffman, At home in the universe: the search for laws of complexity, Penguin Group (1995).
[46] S. Kazadi, Swarm Engineering, Ph.D. Thesis, California Institute of Technology, 2000.
[47] S.Y. Ko, I. Gupta, Y. Jo, A new class of nature-inspired algorithms for self-adaptive peer-to-peer computing, ACM Transactions on Autonomous Adaptive System 3 (3) (2008) 1–34.
[48] P. Kruchten, The Rational Unified Process – An Introduction, Addison-Wesley Professional, 2003.
[49] A.M. Law, W.D. Kelton, Simulation Modeling and Analysis, third ed., McGraw-Hill, 2000.
[50] X. Li, J. Wang, A steganographic method based upon jpeg and particle swarm optimization algorithm, Information Sciences 177 (15) (2007) 3099–3109.
[51] E. Lorenz, Deterministic nonperiodic flow, Journal of Atmospheric Science 357 (1963) 282–291.
[52] T.A.S. Masutti, L.N. de Castro, A self-organizing neural network using ideas from the immune system to solve the traveling salesman problem, Information Sciences 179 (10) (2009) 1454–1468.
[53] J. McLurkin, Analysis and Implementation of Distributed Algorithms for Multi-Robot Systems, Ph.D. Thesis, M.I.T., 2008.

[54] F. Mondada, G. Pettinaro, A. Guignard, I. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. Gambardella, M. Dorigo, Swarm-bot: a new distributed robotic concept, Autonomous Robots 17 (2–3) (2004) 193–221.
[55] A. Namatame, Adaptation and evolution in collective systems, Advances in Natural Computation, vol. 5, World Scientific, 2006.
[56] A. Newell, H. Simon, GPS: a program that simulates human thought, in: E. Feigenbaum, J. Feldman (Eds.), Computers and Thought, McGraw-Hill, New York, 1963.
[57] J. Ottino, Engineering complex systems, Nature 427 (2004) 399.
[58] R. Pasti, L.N. de Castro, Bio-inspired and gradient-based algorithms to train mlps: the influence of diversity. Information Sciences, 179(10) (2009) 1441–1453 (Including Special Issue on Artificial Immune Systems).
[59] E. Rasmusen, Games and Information: An Introduction to Game Theory, fourth ed., Wiley–Blackwell, 2006.
[60] M. Resnick, Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds, The MIT Press, 1997.
[61] C.W. Reynolds, Flocks, herds, and schools: a distributed behavioral model, Computer Graphics 21 (4) (1987) 25–34.
[62] P. Rodríguez-Gianolli, A. Kementsietsidis, M. Garzetti, I. Kiringa, L. Jiang, M. Masud, R.J. Miller, J. Mylopoulos, Data sharing in the hyperion peer database system, in: VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB Endowment, 2005, pp. 1291–1294.
[63] T. Roughgarden, Selfish Routing and the Price of Anarchy, MIT Press, 2005.
[64] M. Salehie, L. Tahvildari, Self-adaptive software: landscape and research challenges, ACM Transactions on Autonomous Adaptive System 4 (2) (2009) 1–42.
[65] A. Serenko, B. Detlor, Agent Toolkits: A General Overview of the Market and an Assessment of Instructor Satisfaction with Utilizing Toolkits in the Classroom, Technical Report 455, Michael G. DeGroote School of Business, McMaster University, Hamilto, Ontario, 2002.
[66] G.D.M. Serugendo, Engineering emergent behaviour: a vision, in: D. Hales, B. Edmonds, E. Norling, J. Rouchier (Eds.), Proceedings of the 4th International Workshop Multi-Agent-Based Simulation, number 2927 in Lecture Notes in Artificial Intelligence, Springer, 2003, pp. 1–7.
[67] G.D.M. Serugendo, Autonomous systems with emergent behaviour, in: J.-P. Rennard (Ed.), Handbook of Research on Nature Inspired Computing for Economy and Management, Idea Group Inc., 2006, pp. 429–443.
[68] G.D.M. Serugendo, J. Fitzgerald, A. Romanovsky, N. Guelfi, Metaself – a framework for designing and controlling self-adaptive and self-organising systems, Technical Report BBKCS-08-08, School of Computer Science and Information Systems, Birkbeck College, London, 2008.
[69] G.D.M. Serugendo, M.-P. Gleizes, A. Karageorgos, Self-organisation in MAS, Knowledge Engineering Review 20 (2) (2005) 165–189.
[70] E. Shakshuki, Y. Jun, Multi-agent development toolkits: an evaluation, in: Innovations in Applied Artificial Intelligence, vol. 3029 of Lecture Notes in Computer Science, Springer, 2004, pp. 209–218.
[71] K. Stanley, B. Bryant, R. Miikkulainen, Real-time neuro-evolution in the nero video game, IEEE Transactions Evolutionary Computation 9 (6) (2005) 653–668.
[72] M. Stefik, Introduction to Knowledge Systems, Morgan Kaufmann, 1995.
[73] J. Surowiecki, The Wisdom of Crowds, Anchor, 2004.
[74] R. Tobias, C. Hofmann, Evaluation of free java-libraries for social-scientific agent based simulation, Journal of Artificial Societies and Social Simulation 7 (1) (2004) 1–27.
[75] P.K. Tripathi, S. Bandyopadhyay, S.K. Pal, Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients, Information Sciences 177 (22) (2007) 5033–5049.
[76] K. Tumer, D. Wolpert, Coordination in large collectives, in: Y. Bar-Yam, (Ed.), Fifth International Conference on Complex Systems, Chapter 1, Perseus Books, 2006.
[77] K. Turner, D. Wolpert (Eds.), Collectives and the Design of Complex Systems, Springer, 2004.
[78] J.P. van Gigch, System Design Modeling and Metamodeling, Springer, 1991.
[79] T. Vicsek, The bigger picture, Nature 418 (2002) 131.
[80] Y. Wang, Y. Yang, Particle swarm optimization with preference order ranking for multi-objective optimization, Information Science 179 (12) (2009) 1944–1959.
[81] D. Weyns, A. Omicini, J. Odell, Environment as a first-class abstraction in multiagent systems, International Journal on Autonomous Agents and Multi-Agent Systems 14 (1) (2007) 5–30. February.
[82] S. Wolfram, A New Kind of Science, Wolfram Media, Inc., 2002.
[83] D.H. Wolpert, K. Tumer, J. Frank, Using collective intelligence to route internet traffic, Advances in Neural Information Processing Systems 11 (1999) 952–958.
[84] M.J. Wooldridge, An Introduction to Multi-Agent Systems, John Wiley & Sons, 2002.
[85] D. Yamins, A Theory of Local-to-Global Algorithms for One-Dimensional Spatial Multi-Agent Systems, Ph.D. Thesis, Harvard School of Engineering and Applied Sciences, 2007.
[86] F. Zambonelli, N. Jennings, M. Wooldridge, Developing multiagent systems: the gaia methodology, ACM Transactions on Software Engineering and Methodology 12 (3) (2003) 317–370.