

Towards a Full Life-cycle Methodology for Engineering Decentralised Multi-Agent Systems

Tom De Wolf and Tom Holvoet

AgentWise@DistriNet Research Group
Department of Computer Science, KULeuven
Celestijnenlaan 200A, 3001 Leuven, Belgium
{Tom.DeWolf, Tom.Holvoet}@cs.kuleuven.be
<http://www.cs.kuleuven.be/~tomdw/>

Abstract When designing decentralised multi-agent systems (MASs), a fundamental engineering issue is to achieve a macroscopic behaviour that meets the requirements and emerges only from the self-organising behaviour of locally interacting agents. Agent-oriented methodologies today are mainly focussed on engineering the microscopic issues, i.e. the agents, their rules, how they interact, etc, without explicit support for engineering the required macroscopic behaviour. As a consequence, the macroscopic behaviour is achieved in an ad-hoc manner. This paper proposes a way to define a full life-cycle methodology based on an industry-ready software engineering process, i.e. the Unified Process, which is customised to explicitly focus on engineering macroscopic behaviour of decentralised MASs. As such, the MAS paradigm is integrated into an existing and widely accepted methodology and we can systematically develop a solution that exhibits the required macroscopic behaviour.

1 Introduction

Modern distributed systems exhibit an increasingly interwoven structure [1] (e.g. ad-hoc networks, transportation systems, etc.). Different subsystems depend on and interact with each other in many, often very complex, dynamic, and unpredictable ways. Also, more and more systems require and exhibit a completely decentralised structure (e.g. ad-hoc networks) and need to achieve their requirements autonomously [2].

A promising approach is to use a group of agents that cooperate to autonomously achieve the required system-wide or macroscopic behaviour using only local interactions, local activities of the agents, and locally obtained information. Such decentralised multi-agent systems (MASs) exhibit *self-organising emergent behaviour* [3]. It is a commonly perceived problem that the overall system tends to exhibit *macroscopic properties*, i.e. properties of the system as a whole, for which it is practically infeasible to derive them when analysing the individual entities in isolation (e.g. pheromone paths in ant optimisation algorithms [4], degree of connectivity or service success ratio in mobile ad-hoc networks [5]). A fundamental problem is the lack of a step plan that allows to systematically specify desirable macroscopic properties, map them to the behaviour of individual agents, build the system, and verify it to guarantee the required macroscopic properties, i.e. *a full life-cycle engineering process or methodology*.

Agent-oriented methodologies today are mainly focussed on engineering the microscopic issues, i.e. the agents, their rules, how they interact, etc., without explicitly engineering the required macroscopic behaviour [6]. Engineering the macroscopic behaviour is currently done in an ad-hoc manner because there is no well-defined process that guides engineers to address this issue. This paper proposes a way to define a full life-cycle methodology based on an existing industry-ready software engineering process, i.e. the Unified Process [7]. The UP process is customised to explicitly focus on engineering macroscopic behaviour of decentralised MASs. As such, we are not reinventing the wheel by proposing a completely new agent-oriented methodology. MAS should be allocated a correct role within mainstream software engineering, rather than positioning MASs as a radically new approach [8]. The MAS paradigm is integrated into an existing, widely accepted methodology and the research focus can be on issues specific for developing a MAS solution with desired macroscopic properties.

This paper has no intention to describe a full-fledged completed methodology. Instead, the paper is a starting point in which future work can be integrated to reach a usable full life-cycle methodology. The paper is structured as follows. Section 2 discusses why existing agent-oriented methodologies only focus on the microscopic issues of a MAS. Then section 3 describes the Unified Process and where customisation is needed to explicitly address the macroscopic behaviour of decentralised MASs. The following sections discuss in detail the different customisations and section 7 concludes.

2 A Shortcoming of Today's Agent-Oriented Methodologies

This section describes a shortcoming of today's agent-oriented methodologies with respect to engineering the macroscopic behaviour of decentralised MASs. The authors of [6] make a distinction of methodologies based on different scales of observing a system:

- *Micro-scale*: The focus here is on systems of a limited number of agents and the engineer details the features of each agent (i.e. agent action-selection architectures, the rules of each agent, knowledge representation), the mechanisms underlying each agent's interaction, and each interaction of the agents with their environment (i.e. protocols, indirect/direct coordination, action models), etc.
- *Macro-scale*: At this scale, the collective behaviour of the system is what matters. The focus is on understanding and controlling the macroscopic behaviour of systems with a very large number of interacting agents, possibly distributed over a network in dynamic and uncontrollable environments.
- *Meso-scale*: When micro-scale systems are used in a larger macro-scale system, two new issues are important: (i) the impact on the micro system of being used in an open and complex scenario and (ii) the impact on the macro-scale system of incorporating the micro-scale system.

As argued in [6], most current approaches to agent-oriented software engineering mainly focus on, and were intended for, development of small-size MASs (micro), and on the definition of suitable models, methodologies, and tools for these kind of systems. Meso-scale and especially macro-scale issues are mostly disregarded.

For example, in Gaia v.2 [9] models are constructed for the individual entities in the system, i.e. environmental model, role model, and interaction model. Also, the organisational model, intended to cover the structure of the system as a whole, is only concerned with relations between individual roles. The performance of the organisation of agents with respect to certain macroscopic properties is not considered explicitly. Also in MaSE [10] solely micro-scale issues are addressed: capturing goals, capturing use cases and refining roles with tasks, creating agent classes, constructing conversations, specifying how agents are deployed, etc. And as a final example, Tropos [11] uses actors, goals, tasks, resources, organisational theory from analysis through design which implies a strong focus on the micro-scale and less on the macro-scale issues. Although this methodology also emphasises the importance of social design patterns, those patterns are also focussed on how individual agents interact and relate and not on what macroscopic property should result from a certain social pattern.

Micro-scale issues are nonetheless important. However, for decentralised MASs the lack of explicitly dealing with macro-scale issues is a serious shortcoming. Until now the desired macroscopic behaviour has been engineered in an ad-hoc way because no support exists to systematically engineer it. To address this problem, the remainder of this paper focusses on how and where existing techniques and knowledge on macro-scale issues can be integrated into an existing software engineering process.

3 A Customised Unified Process for Decentralised MAS

Typically, engineering MASs means having 99% of the effort go to conventional computer science and only 1% involves the actual agent paradigm [12]. Therefore, to engineer decentralised MASs developers should exploit conventional software technologies and techniques wherever possible [8]. Such exploitation speeds up development, avoids reinventing the wheel, and enables sufficient time to be devoted to the value added by the multi-agent paradigm [13]. Instead of devising new entire methodologies, which was the trend until now [8], it is a good start to follow existing traditional software engineering processes, such as the Unified Process (UP) [7], as much as possible.

A key practice in both the UP and most other modern processes is iterative development [14]. Development is organised into a series of short mini-projects called iterations. The outcome of each iteration is a tested, integrated, and executable partial system. Each iteration typically includes a number of disciplines: requirements analysis, design, implementation, and testing-and-verification. Each discipline consists out of a number of activities that result in artifacts (documents, code, models, etc.). As such, a solution is successively refined, with cyclic feedback from verification to design and adapting the solution accordingly as core drivers to converge upon a suitable solution.

The idea is to follow this iterative process, however, customisation is needed to cope with the specific issues of constructing a coherent macroscopic behaviour that is autonomously maintained by a decentralised MAS. Therefore, in each discipline of the engineering process one should focus on how to *address the desired macroscopic properties* of the system. Of course, also other agent specific issues are important such as the micro-scale issues described in section 2. Most of them are to be incorporated into the design. For this paper, the emphasis is on addressing the macro-scale issues.

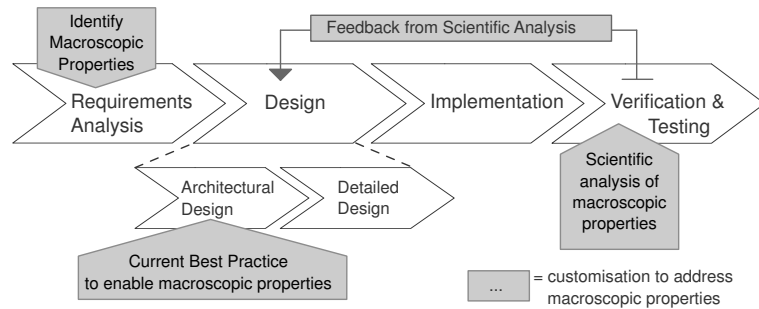


Figure 1. A traditional engineering iteration annotated with customisations for macro-scale issues of decentralised MASs.

The following sections outline the most important disciplines in one iteration of the UP process and indicate which customisations are needed that address the macroscopic behaviour. A schematic overview is given in figure 1. After that, starting from section 4, each customisation is covered in more detail.

3.1 Requirements Analysis

Requirements analysis emphasises an investigation of the problem with functional and non-functional requirements, rather than the solution [14]. This is achieved using standard techniques such as use cases, feature lists, and a domain model that reflects the problem domain. It is important to realise that after the requirements are known it is not always useful to use a decentralised MAS. The decision to use such a solution is made in the design based on the fact that (mostly non-functional) requirements indicate that a decentralised MAS solution is suitable and promising. Section 4 gives an overview of the typical characteristics of the problem that ask for a decentralised MAS. To facilitate that decision, the requirements analysis can be customised to give *special attention to issues that typically lead to decentralised MASs with macroscopic behaviour*:

- Explicitly check if the characteristics that call for decentralised MAS solutions (see section 4) are present or not.
- A decentralised MAS typically ‘maintains’ certain macroscopic properties. Therefore, explicitly identifying those requirements that are ‘ongoing’, i.e. that have to be adaptively maintained, is important. Typical examples are the so-called self-X properties: self-optimising, self-healing, self-configuring, etc [2].
- How is the performance of the system is measured? Typically, these measurements are closely related to macroscopic properties of the system that have to be maintained and explicitly engineered (e.g. the throughput in many systems is a performance measure and a certain throughput level has to be maintained)

3.2 Design

The design emphasises a conceptual solution (in software and hardware) that fulfills the requirements, rather than its implementation [14]. There are two design phases:

Architectural Design. In early iterations of the engineering process the design focusses more on coarse-grained structural decisions that determine the software architecture. The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [15]. The architectural design is mainly driven by non-functional requirements because they apply on the system as a whole. The overall structure of a system influences significantly how well these requirements are achieved. However, functional macroscopic properties of a decentralised MAS are also system-wide and can significantly influence the system architecture.

The author of [14] states that architectural design is partially a science and partially an art. The *science* of architecture is the collection and organisation of information about the architectural significant requirements (e.g. non-functional and macroscopic functionality). The *art* of architecture is making skillful choices and constructing a solution that meets these requirements, taking into account trade-offs, interdependencies, and priorities. The ‘art’ of architectural design is the creative step where designers use knowledge from a variety of areas (e.g. architectural styles and patterns, technologies, pitfalls, and trends) as a guide to reach a suitable solution.

As mentioned in section 3.1 the first decision to be made is if decentralised MAS is a suitable solution. If that is the case, then *knowledge and experience from current best practice* in engineering macroscopic behaviour of decentralised MASs is to be integrated into the architectural design of the engineering process in order to creatively address the required macroscopic properties. Section 5 elaborates on this.

Detailed Design. In later iterations the architecture converges more and more to a fixed structure and the more fine-grained design issues are resolved, i.e. mostly micro-scale issues. For example, with respect to decentralised MASs the choice of the internal action-selection architecture [16] of the agents can be addressed.

3.3 Implementation

The implementation realises the design in code of a specific language and on hardware. No customisations for macro-scale issues are needed because the implementation is completely microscopic. However, existing frameworks, tools, and middleware [17,18,19] specific for building decentralised MASs can support the implementation.

3.4 Testing-and-Verification.

Decentralised MASs will only be acceptable in an industrial application if one can give *guarantees about the macroscopic behaviour*. In other words, when verifying whether the system meets the requirements it is important to explicitly verify if the macroscopic behaviour evolves as required. Such a behaviour is typically system-wide and required to ‘dynamically’ maintain certain macroscopic properties over a long period in time.

In traditional software engineering the testing-and-verification discipline is typically unit-based [14], i.e. one formally proofs or verifies that the behaviour of a single

unit in the system meets the specifications of that unit. It is clear that this is not suitable for verifying system-wide behaviour. In addition, so called scenario tests verify if for a certain trigger, a chain of successive unit executions ends in the required post-conditions. This is also not suitable because, like unit-tests, they only test whether a particular result is obtained at a particular moment in time. Because macroscopic behaviour needs to be verified over a period in time, another approach is needed.

Although, the macroscopic behaviour can still be specified formally, it is practically infeasible to formally verify or proof the correctness of the macroscopic behaviour. The existing unit-tests and scenario-tests are still useful but need to be complemented by an empirical approach that allows to verify the required evolution of the macroscopic properties of the system. It is important that such an empirical approach allows to systematically analyse the behaviour and obtain objective results, i.e. more than subjective interpretation of simulation measurements. The results are used as feedback for the design. Section 6 discusses why formal approaches are practically infeasible and gives an example of an empirical approach and the kind of feedback that can result from it.

4 When to Use a Decentralised MAS?

In the design one decides if a decentralised MAS solution is suitable. Agents have been used in a wide range of applications, but are not a universal solution. For many applications, conventional software development paradigms are enough and more appropriate [13]. The requirements should indicate if a decentralised MAS is a promising solution or not. First of all, there should be a need for a certain **autonomous behaviour** otherwise agents in general are not suited. Based on literature [20,21,6,8,22], the following list gives an tentative overview of what we believe to be the most important problem characteristics that promote the use of decentralised MASs in particular:

1. **Decentralised or Distributed Information Availability.** The information that is needed for automated decision making is only available in a decentralised manner, i.e. no central aggregation is possible. Such problems tend to force that only local interactions are possible. Typical application scenarios in which this occurs are: restrictions on information sharing in competitive and no-cooperative domains (e.g. supply chain management), geographical distribution of the information (e.g. logistics, mobile and collective robotics, ...), and partial or temporal communication inaccessibility (e.g. ad-hoc networks).
2. **High Dynamics: failures and frequent changes.**
 - *Failures: Robustness.* A robustness requirement can impose that a central planner or controlling entity should be avoided as much as possible to eliminate a single point of failure (e.g. military mission-critical applications). Or a decentralised coordination mechanism should ensure that a failing planner is immediately replaced. Robustness also implies that the desired macroscopic behaviour has to be maintained while changes occur.
 - *Frequent Changes: Flexibility and Adaptability.* The flexible way in which agents operate and interact (both with each other and the environment) makes them suitable for dynamic and unpredictable scenarios. Because agents are typically embedded or situated in the environment where all the dynamics take

place, agents are able to rapidly respond to local stimuli to handle problems instantly. Such local reconfigurations enable constant replanning in the face of frequent changes.

There are other characteristics mentioned in literature that promote the use agents: simulation and modelling, openness, resource-constrained domains, complex problems, discrete domains, etc. We refer to [20,21,6] for a description of these characteristics and for a discussion on which of them indeed call for agents and which of them are not really that essential (e.g. complex problems are equally hard to solve with agents).

5 Design: Exploit Current State-of-the-art

After deciding if a decentralised MAS solution will be used, the creative design can explicitly address the development of macroscopic behaviour by exploiting existing best practice and experience. There exists a lot of work that indicates how macroscopic properties, that arise solely from local interactions between agents, can be achieved:

- On the one hand there are **general guidelines or ‘design principles’** such as “Think in terms of Flows rather than transitions”, “Keep agents small, focus on interactions”, and “Provide a local fitness measure for agents to react on that is correlated with the global situation” [21].
- Then there are more concrete guides like a **reference architectures** for autonomic computing [23] or specific MASs [22]. Such architectures embody know-how on how a MAS should be structured in terms of software elements.
- There is also a body of **decentralised mechanisms that allow coordination** between a group of agents to achieve desirable macroscopic properties. Often these mechanisms are inspired by biology [24]. For example, a commonly used coordination mechanism is a “digital pheromones infrastructure” [25] in which a group of agents coordinate by dropping synthetic pheromones (i.e. data that evaporates over time) in the environment for others to find and react on. The inspiration comes from ant colonies that forage for food based on trails of pheromones. “Co-fields” [26] are another example which put a kind of gradient map into the environment for agents to perceive and follow up or down hill according to the coordination goal. All these mechanisms have their advantages and disadvantages. According to the macroscopic properties that have to result from the coordination between the agents one can use these ‘patterns’ as a guide for building a suitable coordination architecture. We refer to [27,24] for a literature study of such decentralised mechanisms.

A lot of work has to be done to actually integrate and exploit these guidelines and mechanisms. For example, translating guidelines into concrete engineering support and providing tools and specific guidelines for using each decentralised coordination mechanism are possibilities.

6 Test-and-Verification: Guaranteeing the Desired Macroscopic Behaviour

The goal is to verify if the desired macroscopic behaviour, identified during the requirements analysis, is achieved. This section first discusses why formal approaches are not

suites to guarantee the macroscopic behaviour and then gives an example of an empirical approach that allows to verify the evolution of the macroscopic behaviour of decentralised MASs.

6.1 Formal versus Empirical Verification

The main question here is how to know if a certain decentralised MAS maintains the required macroscopic behaviour. Firm guarantees could be obtained if the system is modelled formally and the required macroscopic behaviour is proofed analytically. However, constructing a formal model and correctness proof of a complex interacting computing system is infeasible. Wegner [28] proves this based on the fact that computing systems using interaction are more powerful problem solving engines than mere algorithms. Wegner shows that one cannot model all possible behaviour of an interaction model and thus formally proving correctness is not merely difficult but impossible.

The alternative to formal proof is to use an empirical method to verify the macroscopic behaviour. Macroscopic properties are typically quantified with measurable variables which we define as *macroscopic variables*. The variables are then measured from simulations and the measurements are used to obtain statistical results. Although such results remain useful, they suffer from the fact that the interpretation of these results is very subjective, i.e. strongly depends on what is observed. This is not an objective and scientifically founded method. As advocated in [29], we need an empirical but scientifically founded method to verify the macroscopic behaviour. Then, more valuable and advanced verification results can be obtained objectively, supported by scientific algorithms. In section 6.2 such a method is described. Note that this does not exclude formal models completely. Formal models are less suited to verify the required behaviour, but are still useful to unambiguously specify the macroscopic behaviour and acquire more understanding on how decentralised MASs work (e.g. [30] and [31]).

6.2 Example Verification Approach: Equation-Free Analysis

To achieve objective and scientifically founded results, it is often a good idea to use mathematical methods. There exist a lot of numerical algorithms that allow to analyse the system dynamics. For example, a bifurcation algorithm allows to analyse if there are qualitative changes in the behaviour of the system as a result of a changing parameter. For example, consider a mobile ad-hoc network. The communication range used by each node is an important parameter. For certain values of that parameter the network can maintain itself as one connected network. For other values, the network can be constantly partitioned or completely disconnected. Bifurcation analysis allows to detect which type of behaviour is achieved for which communication ranges. Such numerical algorithms are typically applied on (differential) equation models that model the evolution of the macroscopic variables. This is illustrated in the left of figure 2. The analysis algorithm repeatedly evaluates the equation for certain initial values (X_i in figure 2) for the variables to obtain the value some time steps later (X_{i+1} in figure 2). As such the behaviour evolution is analysed on the fly by the algorithm.

However, as discussed in section 6.1, in complex interacting systems, deriving an equation, which is a formal model, is often not possible. Even if one derives an equation,

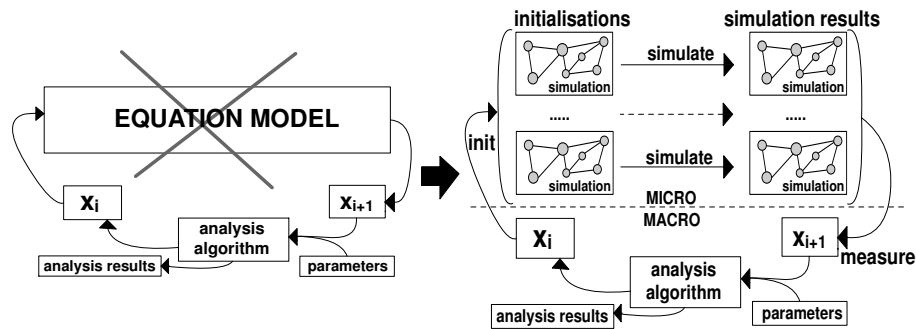


Figure 2. Equation-Free Accelerated Simulation guided by the analysis algorithm

one risks results that differ significantly from the real system behaviour (e.g. [32]). An approach that solves this problem is called “*equation-free*” *macroscopic analysis* [33]. The observation is that most numerical algorithms have no need for the equation itself; they only need a routine that *evaluates* the equation for a given value in order to return a value some time steps later. The idea is to replace the equation evaluations with realistic agent-based *simulations* (see figure 2). Simulations are initialised to reflect the given values (X_i) of macroscopic variables (*init* in figure 2). After simulating for the requested time (*simulate* in figure 2), the new values (X_{i+1}) are measured from the simulation state (*measure* in figure 2). The results no longer depend on the subjective interpretation of the observer because mathematical algorithms decide on the accuracy of the result and on how the behaviour is analysed. As such all numerical analysis algorithms can be applied (e.g. stability analysis, newton steady state detection, etc.) and useful feedback is obtained for the next design iteration of the engineering process.

A big challenge for this approach is to find a suitable set of macroscopic variables that reflects the evolution of the macroscopic properties under study, and an initialisation operator to initialise simulations based on these variables. We refer to [34] for more details, some results on an Automated Guided Vehicle transportation system, and a discussion about the challenges, the advantages, and the disadvantages of the approach.

6.3 Iterative Development with Feedback

The required macroscopic properties, identified during requirements analysis, are engineered in an iterative process. This means that the design discipline uses the test-and-verification results to get feedback on the performance with respect to those macroscopic properties and adapt the design to steer the solution towards the required behaviour. The design typically has a different focus in early iterations (architecture design) compared to later iterations (detailed design). As a consequence, to have useful feedback for the design, there should be a varying focus in the test-and-verification discipline also (illustrated in figure 3), some possibilities:

- **Non-functional and macroscopic performance:** How good is the performance of the solution for non-functional requirements and macroscopic properties?

- **Comparison of coordination mechanisms:** Which of the different mechanisms performs best with respect to the functional/non-functional requirements?
- **Parameter-tuning:** Which range of values for a certain parameter allows to achieve the requirements?
- **Characterisation of macroscopic behaviour:** When a solution is (nearly) completed, it is important to characterise that solution for different possible deployment scenario's: performance for a heavy utilisation load, what utilisation load can it cope with, etc.

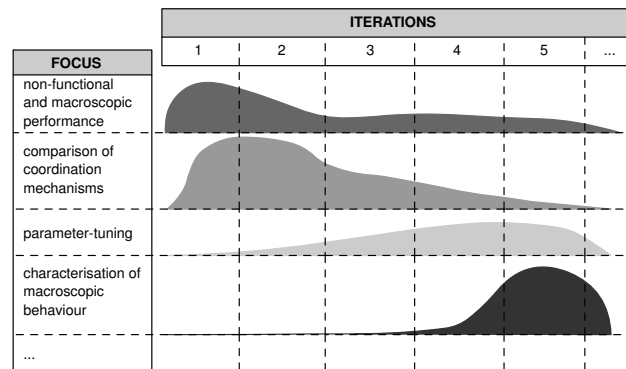


Figure 3. Scientific analysis: different focus in early iterations versus later iterations.

In *early iterations* the focus is more on the ‘coarse grained’ decisions with respect to the architecture of the solution. The influence of certain decisions on the performance of non-functional properties is analysed. Also, deciding which coordination mechanism will be used in each coordination scenario can influence the performance of macroscopic properties significantly. A comparison of different alternatives for coordination and other architectural decisions is necessary in the early iterations [14].

In *later iterations* one analyses the solution in more detail. For example, a decentralised multi-agent solution typically consists of a lot of parameters that can be set or changed online to adaptively react to changing situations (e.g. adapting the communication range in ad-hoc networks to react on an increase in node density). Analysing which parameter value set is most suitable in which situation serves as important feedback to know how to adapt. Also, when a solution is completed and has to be deployed in different operational conditions it is important to know how it will behave. For example, knowing what the minimum number of network nodes should be in a mobile ad-hoc network in order to still reach the required degree of connectivity determines where the boundaries are of that solution.

As such, an integration of a scientifically founded empirical verification approach into an iterative engineering process enables to systematically develop a decentralised MAS solution based on feedback about the macroscopic behaviour.

7 Conclusion and Future Work

Today's agent-oriented methodologies have a serious shortcoming. They do not deal explicitly with a fundamental issue when building decentralised MASs, i.e. engineering the desired macroscopic behaviour. To come to a full life-cycle methodology that allows to systematically engineer the macroscopic behaviour, one should not re-invent the wheel. One should follow a conventional software engineering process, i.e. UP, as much as possible. This paper proposes to customise the UP process by *explicitly addressing the macroscopic behaviour* throughout the process.

This position paper is only a starting point. A lot of future work is needed to achieve a full-fledged and usable methodology. Future work can study each guideline and coordination mechanism in detail in order to provide concrete support to exploit them (e.g. tools, design models, etc.). Also refinement of the empirical verification approach (e.g. a guide for choosing macroscopic variables) and applying it extensively is important.

Acknowledgements. This work is supported by the K.U.Leuven research council as part of the Concerted Research Action on Agents for Coordination and Control - AgCo2.

References

1. Herrmann, K., Mhl, G., Geihs, K.: Self-Management: The Solution to Complexity or Just Another Problem? IEEE Distributed Systems Online **6** (2005)
2. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer Magazine **36** (2003) 41–50
3. De Wolf, T., Holvoet, T.: Emergence and Self-Organisation: a statement of similarities and differences. In: Proc. of the 2nd Int. Workshop on Engineering Self-Organising App. (2004)
4. Dorigo, M., Maniezzo, V., Colnari, A.: Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B **26** (1996) 29–41
5. Parunak, H.V.D., Brueckner, S.A.: Analyzing Stigmergic Learning for Self-Organizing Mobile Ad-Hoc Networks (manet's). In: Proc. of the 2nd Int. Workshop on Engineering Self-Organising Applications. (2004)
6. Zambonelli, F., Omicini, A.: Challenges and Research Directions in Agent-Oriented Software Engineering. Autonomous Agents and Multi-Agent Systems **9** (2004) 253–283
7. Jacobson, I., Booch, G., Rumbaugh, J.: The unified software development process. Addison Wesley (1999)
8. Weyns, D., Helleboogh, A., Steegmans, E., De Wolf, T., Mertens, K., Bouck, N., Holvoet, T.: Agents are not part of the problem, agents can solve the problem. In: Proc. of the OOPSLA Workshop on Agent-Oriented Methodologies. (2004)
9. Zambonelli, F., Jennings, N., Wooldridge, M.: Developing multiagent systems: The Gaia methodology. ACM Trans. Software Eng. Meth. **12** (2003) 417–470
10. Wood, M.F., DeLoach, S.A.: An Overview of the Multiagent Systems Engineering Methodology. In: Agent-Oriented Software Engineering, Volume 1957 of LNCS. Springer (2001)
11. Giorgini, P., Kolp, M., Mylopoulos, J., Pistore, M.: The Tropos Methodology: An Overview. In: Methodologies And Software Engineering For Agent Systems. Kluwer (2004)
12. Etzioni, O.: Moving Up the Information Food Chain: Deploying Softbots on the World Wide Web. In: Proc. of the 13th Int. Conf. on Artificial Intelligence. (1996)

13. Wooldridge, M., Jennings, N.R.: Software engineering with agents: Pitfalls and pratfalls. *IEEE Internet Computing* **3** (1999) 20–27
14. Larman, C.: *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. 3rd edn. Prentice Hall (2005)
15. Bass, L., Clements, P., Kazman, R.: *Software Architecture in Practice* (2nd ed). (2003)
16. Tyrrell, T.: *Computational Mechanisms for Action Selection*. PhD thesis, University of Edinburgh (1993)
17. Bellifemine, F., Poggi, A., Rimassa, G.: Jade, A FIPA-compliant agent framework. In: Proc. of PAAM'99, London (1999)
18. Repast: <http://repast.sourceforge.net> (2003)
19. Schelfhout, K., Holvoet, T.: An environment for coordination of situated multi-agent systems. In: 1st Int. Workshop on Environments for Multi-Agent Systems (E4MAS). (2004)
20. Michal Pechoucek, Martin Rehak, V.M.: Expectations and deployment of agent technology in manufacturing and defence: Case studies. In: Proc. AAMAS'05 – Industry Track. Volume 3., Utrecht, Netherlands, ACM (2005)
21. Parunak, H.V.D., Brueckner, S.A.: Engineering Swarming Systems. In: *Methodologies and Software Engineering for Agent Systems*. Volume 11 of Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer (2004)
22. Weyns, D., Schelfhout, K., Holvoet, T., Lefever, T.: Decentralized Control of EGV Transportation Systems. In: Proc. AAMAS'05 – Industry Track. (2005)
23. IBM: An architectural blue-print for autonomic computing. Technical report, IBM (2004)
24. Nagpal, R.: A Catalog of Biologically-inspired Primitives for Engineering Self-Organization. In: *Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering*. Volume 2977 of LNCS., Springer (2004) 53–62
25. Brueckner, S.: *Return From The Ant - Synthetic Ecosystems For Manufacturing Control*. PhD thesis, Humboldt-Universitt, Berlin (2000)
26. Mamei, M., Leonardi, L., Zambonelli, F.: Co-fields: Towards a unifying approach to the engineering of swarm intelligent systems. In: Proc. of the 3rd Int. Workshop on Engineering Societies in the Agents World (ESAW) 2002. Number 2577 in LNAI, Springer (2003)
27. De Wolf, T., Holvoet, T.: Towards Autonomic Computing: agent-based modelling, dynamical systems analysis, and decentralised control. In: *Proceedings of the First International Workshop on Autonomic Computing Principles and Architectures*. (2003)
28. Wegner, P.: Why Interaction is More Powerful than Algorithms. *Commun. of the ACM* **40** (1997) 80–91
29. Edmonds, B., Bryson, J.J.: The Insufficiency of Formal Design Methods - the necessity of an experimental approach for understanding and control of complex MAS. In: Proc. of the AAMAS'04 Conference, New York, ACM Press (2004) 938–945
30. Gardelli, L., Viroli, M., Omicini, A.: On the Role of Simulations in Engineering Self-Organizing MAS: the Case of an Intrusion Detection System in TuCSon. In: 3rd International Workshop “Engineering Self-Organising Applications” (ESOA). (2005) 161–175
31. Sudeikat, J., Renz, W.: Mesoscopic Modeling of Emergent Behavior - A Self-Organizing Deliberative Minority Game. In: *The Third International Workshop on Engineering Self-Organising Applications (ESOA'05)*. (2005)
32. Wilson, W.: Resolving Discrepancies between Deterministic Population Models and Individual-Based Simulations. *American Naturalist* **151** (1998) 116–134
33. Kevrekidis, I.G., Gear, C.W., Hyman, J.M., Kevrekidis, P.G., Runborg, O., Theodoropoulos, C.: Equation-free, coarse-grained multiscale computation: enabling microscopic simulators to perform system-level analysis. *Comm. in Mathematical Sciences* **1** (2003) 715 – 762
34. De Wolf, T., Samaey, G., Holvoet, T., Roose, D.: Decentralised Autonomic Computing: Analysing Self-Organising Emergent Behaviour using Advanced Numerical Methods. In: *Proceedings of IEEE Int. Conf. on Autonomic Computing (ICAC'05)*, Seattle, USA (2005)