Creating Collaboration Patterns in Multi-Agent Systems with Generic Observer/Controller Architectures

Emre Cakar
Institute of Systems Engineering
Hanover, Germany
cakar@sra.uni-hannover.de

Jörg Hähner
Institute of Systems Engineering
Hanover, Germany
haehner@sra.uni-hannover.de

Christian Müller-Schloer*
Institute of Systems Engineering
Hanover, Germany
cms@sra.uni-hannover.de

ABSTRACT

Flexibility, robustness and adaptivity are key concepts in developing today's technical systems. Nowadays, systems that are developed with conventional design methodologies do not sufficiently meet the requirements of these concepts. An increasing number of system elements, their complexity and a dynamically changing environment often lead to an unexpected system behaviour, although all system elements are available and work correctly. The Organic Computing (OC) initiative deals with new design concepts, which facilitate developing technical systems with life-like properties such as self-organisation, self-optimisation and self-configuration in order to make them robust, flexible and adaptive. In this context, a generic observer/controller architecture[1] has been proposed in order to establish controlled self-organisation in technical systems. In this paper, we investigate different distribution possibilities of the generic o/c architecture and the resulting collaboration and communication patterns in a traffic scenario.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Coherence and coordination, Intelligent agents, Multiagent systems*

General Terms

Theory, Design, Algorithms

Keywords

Organic Computing, observer/controller architectures, design space exploration, self-organisation, multi-agent systems

*In close cooperation with Hartmut Schmeck und Urban Richter, KIT / University of Karlsruhe

[1]In the rest of the paper we use the abbreviation "o/c architecture".

AUTONOMICS 2008, September 23-25, Turin, Italy

DOI 10.4108/ICST.AUTONOMICS2008.4379

1. INTRODUCTION

Modern technical systems consist of a large number of elements that interact with each other in order to accomplish a given task. The state and configuration spaces of such systems become increasingly larger due to the growing number of system elements and their complexity. Thus, a system developer cannot design the behaviour of each system element for every arising state explicitly. This kind of design complexity often leads either to an unexpected system behaviour or to a system crash in runtime. The vision of OC is to endow technical systems with life-like properties such as self-organisation, self-optimisation and self-configuration to address this complexity. Self-organising systems can tolerate disturbances either from inside the system, e.g. in case of defective system elements, or from outside the system, e.g. in case of a dynamic environment, and continue working properly while adapting to changes in their environment. In order to design self-organising systems some degrees of freedom must be given to system elements so that they can adapt their behaviour and/or structure to new environmental situations. The self-organisation process on its own may lead a system to an undesirable state that does not conform with a system goal given by the developer. Hence, this process must be controlled in some way so that the system can adapt to its dynamically changing environment and work towards a predefined goal at the same time.

We proposed the generic o/c architecture in [12, 1]. The observer monitors the system state and dynamics, quantifies them and aggregates its observations as a vector of situation parameters. These parameters are sent to the controller. The controller evaluates the situation parameters and influences the system under observation and control (SuOC) with respect to the given goal by the user. An abstract illustration of the architecture can be seen in Fig. 1.

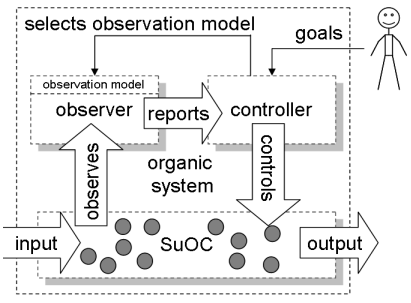
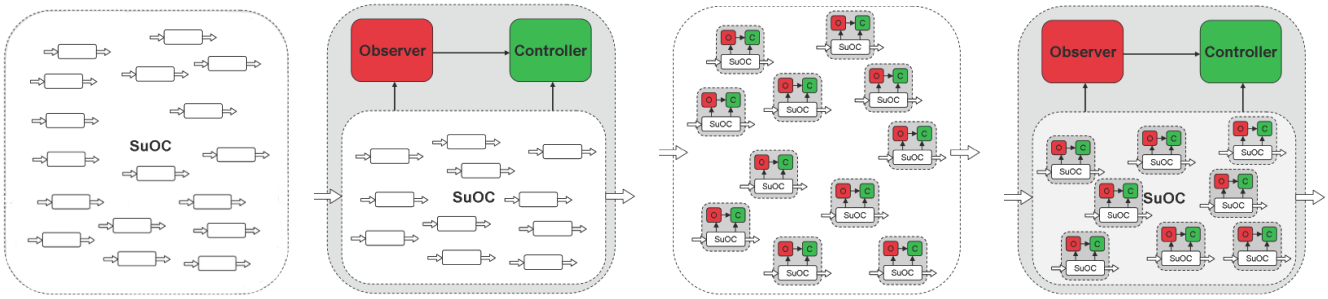


Figure 1: The observer/controller architecture



(a) System without an o/c architecture

(b) System with the central o/c architecture

(c) System with the distributed o/c architecture

(d) System with the multi-level o/c architecture

Figure 2: System without an o/c (SuOC) and some distribution possibilities of the generic observer/controller architecture on the SuOC

There are many distribution possibilities of the proposed architecture varying from fully central to fully distributed. In the former case, there is only one observer and one controller for the whole system (see Fig. 2(b)), whereas in the latter case there is one observer and one controller for each agent in the system (see Fig. 2(c)). The fully central and the fully distributed architectures define the two extreme points in the design space. So, there are also many other distribution possibilities like a multi-level architecture between these extreme points (see Fig. 2(d)).

In this paper, we investigate different distribution possibilities of the generic o/c architecture. As a test scenario we consider a resource sharing problem presented through an intersection without traffic lights. We first implement the intersection and the cars without an o/c (see Fig. 2(a)). A car without an o/c tries to maximize only its own gain by crossing over the intersection as soon as possible. This type of behaviour produces competition situations, since two (or more) cars may want to occupy the same position at the same time. Thus, the competition-based behaviour of agents leads to a traffic jam where cars with orthogonal driving directions block each other in the middle of the intersection. We implement a central (see Fig. 2(b)) and a fully distributed (see Fig. 2(c)) o/c architecture to create a collaborative group behaviour in order to avoid traffic jams. The realisation of the collaboration on different distribution levels should produce different system behaviours with corresponding collaboration and communication patterns. So, we expect an advantage of a centralized architecture in case of a low-conflict scenario whereas a distributed architecture should perform better in more complex scenarios. Therefore, we argue that the optimum collaboration strategy can be realised neither on the central nor on the fully distributed level, but rather somewhere in between those extremes.

This paper is organized as follows: Section 2 summarizes some related work concerning the architectures used in collaborative problem solving in multi-agent systems. Section 3 presents the experimental setup and describes the problem occuring in the system without an o/c together with the technique that is to be used to cope with the problem. Section 4 introduces different o/c architectures that implement the technique given in section 3. A classification of metrics, which can be used to measure the system performance, is discussed in section 5. Experimental results are given in section 6, followed by the conclusion and outlook in section 7.

2. STATE OF THE ART

There are different architectures proposed in the literature, which are applied to complex technical systems. An application of the generic o/c architecture to a robot swarm is proposed in [10]. In this scenario, the observer implements a technique based on Shannon's information theory, which is used to measure the unwanted emergent clustering behaviour of robots. The controller prevents this emergent behaviour by changing the environment and influencing the system indirectly.

The Swarm-Intelligent Systems (SWIS) group of the EPFL (Swiss Federal Institute of Technology) investigates new modelling, control and design methodologies for distributed and collectively intelligent systems. They proposed a distributed control architecture [2], which exploits principles based on swarm intelligence like redundancy and autonomy.

Distributed and collaborative problem solving is also investigated in the Swarm-bots project, which is funded by the Future and Emerging Technologies programme of the European Commission. A context specific control architecture is proposed in [13] that allows a group of robots to collaborate with each other in order to accomplish a given task, which originally cannot be accomplished by a single robot.

The Centibots project also aims at developing new methodologies, which facilitate a group of agents to solve a given problem collaboratively. Large-scale, fault-tolerant and adaptive robot systems are investigated in this context. A distributed, multi-level control architecture is proposed in order to produce an intended collaborative behaviour in a group of robots [7].

NASA investigates new techniques and methods for developing multi-agent systems for space missions. The autonomous nano-technology swarm (ANTS) architecture is proposed in this context, which is based on the principles of social insect colonies. The architecture supports task specialisation for the agents in the system and allows them to accomplish a given task collectively [6, 3, 4].

The distributed field robot architecture (DFRA), which is an extension of the Sensor Fusion Effects (SFX) architecture [11], is introduced in [8]. DFRA allows for a dynamic discovery and acquisition of robot resources and the integration of human and artificial agents in robot teams. The access to capabilities of a team of robots is realised in form of Jini services. An application of DFRA to a simulated demining task on a team of ground and aerial robots is presented in

[8].

Architectures presented above exhibit promising results in collaborative problem solving in multi-agent systems. However, they are either specialized in solving specific problems [2, 13, 7] or are used in specific problem domains [6, 4, 8]. Generic concepts and methodologies related to the observation and control of collaborative systems are investigated only in [12, 1, 10]. The investigation of the generic o/c architecture promises to provide a basic understanding for creating adaptive, flexible, robust and self-organising systems.

3. EXPERIMENTAL SETUP AND PROBLEM DESCRIPTION

We use the agent-based modeling and simulation toolkit *"RePast"* [9] to implement our scenario. RePast provides a scheduler, which triggers agents to perform their predefined behaviour in each time step. A time step in a RePast simulation is called a *"Tick"*. In our experiments, we also use the notion of ticks in producing experimental results presented in section 6.

A grid-based layout is used to model the intersection (see Fig. 3). Each car in the intersection is created with a random size and a random speed. There are three sizes defined for cars: A small size car covers one cell, a medium size car two cells and a large size car three cells in the grid environment. Each car obtains one of two predefined speeds. Slow cars can only make one move in a single time step (tick) whereas fast cars can make two moves in the same length of simulation time. A car without an o/c has a basic behaviour, which allows it to avoid collisions with other cars in the intersection. So, a car (C1) moves forward, if there is no car in front of it. Otherwise, if the position is occupied by another car (C2), C1 tries to overtake C2 on the right. If the intended position is occupied by another car (C3), C1 tries to overtake C2 on the left this time. C1 doesn't change its position in the case where all intended positions are already occupied. Each car has an internal counter to keep track of its own delay time while crossing over the intersection. The counter is increased each time the car cannot move and remains unchanged otherwise.

We use a Poisson distributed number generator to determine the number of cars, which enter the intersection at the same time in the appropriate direction. As can be seen in figure 3, we have two traffic flows with orthogonal directions. Yellow cars drive in west-east and red cars in south-north direction into the critical area. On average 4 cars enter the intersection at each tick in each direction. The dimensions of the environment are defined so that a maximum of 8 cars can drive into the intersection in one direction at the same time. Unless stated otherwise, we use the same environment and same parameters in all our experiments presented in the next sections.

We first ran the system without an o/c architecture (see Fig. 2(a)) to identify the system performance without an intervention. We observed that cars with different driving directions block each other in the intersection. We call the part of the intersection, where this happens, the "critical area". Some cars in the critical area either need much more time than others to cross over the intersection or in the worst case, they remain blocked in a large cluster over the whole simulation time (see Fig. 3).

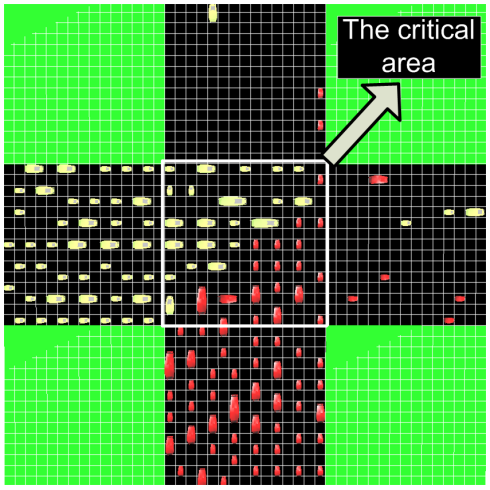


Figure 3: The clustering behaviour in the critical area. This is the system without an o/c illustrated in Fig. 2(a)

The problem presented here is very close to the scheduling problem known from operating system theory. In this context, cars in the intersection can be considered as different processes running on the same machine and the critical area can be considered as a shared resource, e.g. the CPU, for which processes compete. We use a priority-based algorithm to create a collaborative behaviour between agents and to cope with the clustering problem in the critical area. We determine here priorities for cars with respect to their delay times in the intersection. A car (or a group of cars) with a higher delay time gets a higher priority. In our work we use this priority allocation mechanism on different distribution levels of the generic o/c architecture. According to the implemented o/c architecture, the priority allocation takes place on different abstraction levels, i.e. on a macro level and a micro level.

4. OBSERVER/CONTROLLER ARCHITECTURES

We first implemented the fully distributed o/c architecture (see Fig. 2(c)), where each car is endowed with a pair of an observer and a controller. The view of each observer and controller is limited to the direct neighborhood of the corresponding car. As depicted in Fig. 4, only Car2, Car3 and Car4 are in the direct neighborhood of Car1. Considering this type of neighborhood, local rules are defined for the observer and controller, which determine the behaviour of a car in the next simulation step. In the following, these local rules are explained in the sample situation given in Fig. 4.

The observer creates a list of situation parameters considering the neighborhood of its corresponding car. These parameters include cars in the direct neighborhood, their delay times and their positions relative to the car, to which the observer belongs. In the example given in Fig. 4, the parameters created by the observer of Car1 look like [Car2 - W2 - On_the_left], [Car3 - W3 - Conflict] and [Car4 - W4 - Behind]. The observer of Car2 creates only the parameter [Car1 - W1 - On_the_Right], since there is only one car (Car1) in its direct neighborhood. The word "Conflict"

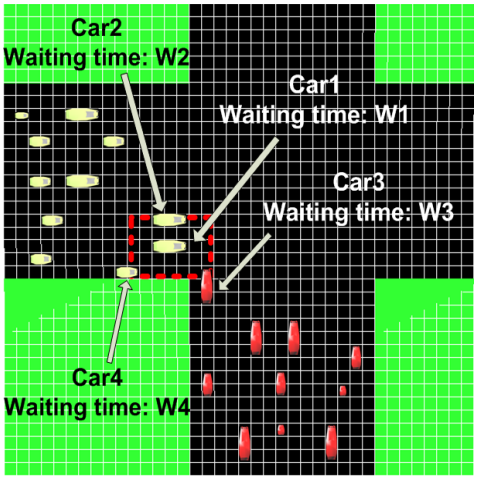


Figure 4: The direct neighborhood of Car1. The observer and the controller can only notice other cars in the direct neighborhood

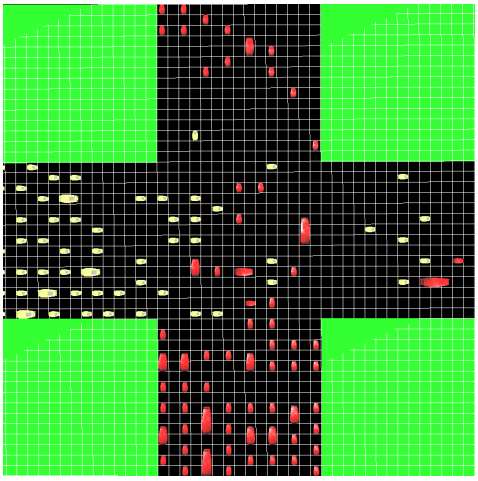


Figure 5: Two traffic flows with orthogonal directions passing through each other. This is the system with the distributed o/c illustrated in Fig. 2(c)

in the parameter list of Car1 indicates that either Car1 or Car3 can move forward in the next simulation step. After creating all situation parameters, the observer sends them to the controller. According to parameters from the observer, the controller sends either a "Stop" or a "Go" signal to its corresponding car.

In order to take a decision, the controller checks relative positions in the parameter list. If it encounters one of the relative positions "On_the_Right", "On_the_Left" or "Conflict", it compares the delay time of its corresponding car with the delay time of the car in the encountered relative position. In case of Car1, the controller sends a stop signal to Car1, if W3 > W1. That means Car3 has a higher priority than Car1 and Car1 yields right of way to Car3. In case of Car2, the controller sends a stop signal to Car2, if W1 $\geq$ W2 and Car1 is stopped by its own controller. That means there is a car (Car3), which is not in the neighborhood of Car2, but has a higher delay time than Car2 so that the right neighbor (Car1) stopped and yielded right of way to that car. So, Car2 makes a decision based on the behaviour of its right neighbor. The controller sends a go signal to its corresponding car, if it has stopped in the previous simulation step and the reason for the stop signal does not exist in the current step. If none of previously mentioned cases occurs, the controller does nothing and waits for the next simulation step.

The implementation of these rules in each o/c instance prevents the development of clusters in the critical area allowing cars with higher delay times first cross over the intersection. This leads to a lower traffic density in the critical area (see Fig. 5).

We also implemented the priority allocation mechanism presented in section 3 on the central level (see Fig. 2(b)). The central observer and the central controller can interact with all cars in the intersection, i.e. their view is not limited. In general, systems with a single controller instance, where the controller influences the behaviour of each agent explicitly, cannot scale with the increasing number of agents, if the controller has limited resources like for example the cpu or the memory that cannot be expanded accordingly. This type of limitation would after some time prevent the controller to influence the system as necessary so that the overall system could produce an unwanted behaviour. Thus, in such large-scale multi-agent systems the behaviour of each agent cannot be explicitly determined by a central controller. In such a case, a system developer can either implement a distributed control mechanism or modify the central controller so that it works on a higher abstraction level determining (or influencing) the behaviour of agent groups instead of the behaviour of every single agent. In this way, the controller can save resources and is able to work with a large number of agents. We have presented a distributed control mechanism above. In the following, we present the central approach using the priority allocation mechanism to determine the priority for a group of cars on the central level.

Since we have the advantage of a non-limited view, we assign priorities to cars in each traffic flow with respect to their cumulative delay times (see Fig. 6).

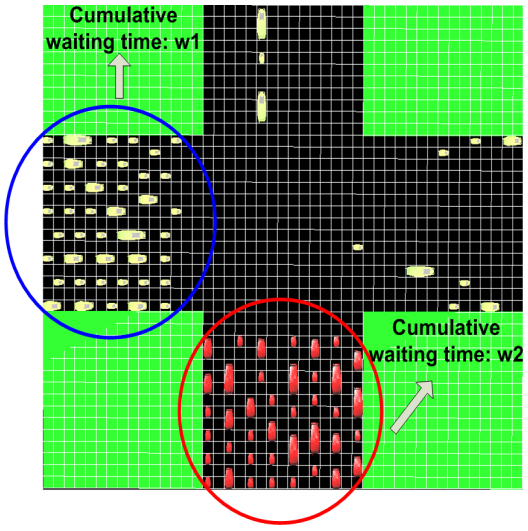


Figure 6: The cumulative delay times of cars w1 and w2. This is the system with the central o/c illustrated in Fig. 2(b)

On the central level, the observer interacts with cars, which have not entered the critical area (cars in red and blue circles in Fig. 6) in determining priorities for each traffic flow. After collecting delay times, the observer calculates cumulative delay times for each group of cars. This is the first parameter sent to the controller. The second parameter is related to cars, which could enter the critical area in the next time step. These are actually cars, which get a stop/go signal from the controller. That means, if the controller decides to stop a traffic flow, it only sends a stop signal to those cars, which could enter the critical area in the next time step, but did not enter yet. After getting cumulative delay times and a list of cars, the controller determines priorities for traffic flows and sends stop or go signals to corresponding cars.

The o/c here adapts itself to particular changes in a given traffic scenario due to the dynamic priority allocation mechanism presented above. Since we have no fixed green and red light phases, an increase in the arrival rate of cars in one traffic flow (A) and a decrease in the arrival rate of cars in the other traffic flow (B) with orthogonal direction would cause the controller to award the "green light" to traffic flow A more often than to traffic flow B.

5. SYSTEM PERFORMANCE METRICS

Before we compare the system performances resulting from the application of implemented architectures to the system, we first have to define the right metric to measure the system performance. In a system where agents compete for limited resources each agent maximizes its own profit. Thus, only its own gain is relevant for a competing agent but not the gain of the group. Since there is no collaboration between agents, the overall system performance depends alone on how effective an agent on average prevails in different encountered competing situations. Thus, we need to make an assessment on the agent level in order to determine the system performance. This can be done by calculating the average gain (or loss) of an agent in the system. For example, the success of a scheduling algorithm in an operating system is always measured with the average waiting time of a process on the system.

On the other hand, in a system where agents collaborate with each other, the gain of the overall group is relevant but not the gain of every single agent. Thus, we need a different approach to measure the system performance, since the average gain of an agent in the system doesn't really show how successful the "group" is. In such a case, some agents may temporarily abstain from increasing their own profit in behalf of the overall group. Thus, we cannot define here a utility function for an agent without considering the rest of the agents. Therefore, we define a utility function for the whole group, since the agents collaborate and the actions of agents are all linked with each other. This kind of utility function defines the gain of the group, which is composed of the contribution of every single agent, and can therefore be calculated using an accumulated value. So, to measure the system performance in collaborative agent groups and make an assessment on the group level, we use an accumulated value instead of an average. As an example consider the case where the number of agents increases more rapidly than the gain of the overall group. This would decrease the system performance measured with the average value, although the system increases its gain.

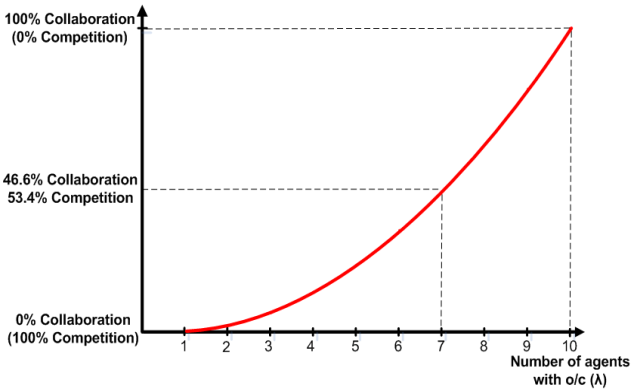


Figure 7: The change of the group behaviour with an increasing number of agents with o/c (λ) where the total number of agents is 10.

Since we want to facilitate a collaborative group behaviour in our scenario, we used an accumulated value instead of an average (e.g. the average delay of cars) to measure the system performance as the total amount of cars that leave the intersection, i.e. the traffic flow-rate. In the intersection, the competitive behaviour of cars leads to a traffic jam in the critical area and we get a poor system performance. The application of o/c architectures to the system facilitates the collaboration between the cars. According to the classification above we can define the goal of the implemented o/c arhitectures as minimizing the competition and maximizing the collaboration between the agents.

We can also calculate the degree of collaboration occurring in the system through the number of agents with an o/c. To demonstrate an example calculation assume that we have a system S with n agents where each agent interacts with all other agents for a given point of time. Let an interaction I be either a competition or a collaboration. The number of all interactions ω is then the sum of the number of competitions and the number of collaborations between agents. Let γ be the number of competing agents and λ the number of collaborating agents. Two agents can only collaborate with each other iff both parts have an observer and a controller. ω can easily be calculated through $\binom{n}{2}$ so that we have:

$$n = \lambda + \gamma$$

$$\omega = \binom{n}{2} = \binom{\lambda}{2} + \left(\binom{\gamma}{2} + \gamma \times \lambda\right)$$

To exemplify the formula given above we consider a system with 10 agents where only 7 of them have an o/c. So, n is 10, λ is 7 and γ is 3. In this case the number of all interactions ω would be $\binom{10}{2} = 45$. The number of all collaborations is then 21, which can be calculated through $\binom{7}{2}$. The number of all competitions would be $\binom{3}{2} + 3 * 7 = 24$. So, we have only $(21 \div 45) \times 100 = 46{,}6\%$ collaboration in the system. Fig. 7 demonstrates the change of the collaboration ratio with an increasing number of agents with o/c (λ) according to the formula given above.

Note that, in the example above, a static degree of collaboration is given assuming that each agent interacts with

all other agents in each time step. So, we determine the maximum possible (upper bound) number of competitions and collaborations and calculate accordingly the degree of collaboration. We also can determine the dynamic degree of collaboration where the number of all interactions and the number of agents with and without o/c that participate in these interactions change over time. We can then use the formula given above to determine the degree of collaboration for each sequential time step.

We give here a quantitative definition of collaboration, the quality of the collaboration is not considered. This approach can also be adapted to different types of systems where the behaviour of agents can be defined either as competitive or as collaborative. Until now we didn't measure the degree of collaboration occurring in our system. Our future work will investigate the effects of different degrees of collaboration on the system performance.

6. EXPERIMENTAL RESULTS

We used different test scenarios to measure and compare system performances, which result from the application of each o/c architecture to the system. We first implemented the basic configuration (see scenario I) that presents a low complexity situation in the intersection. We then used this configuration and increased the complexity in the system step by step in order to determine and compare the system behaviour with the particular o/c architecture in different scenarios. In order to avoid the possible side effects of the experimental setup on our assessment, we implemented three different test scenarios with increased complexity and evaluated the corresponding experimental results. In the following we introduce our test scenarios.

6.1 Scenario I

We demonstrate in this scenario the basic configuration that is used to explore the effects of central and distributed o/c architectures on the system. Here, we create cars only in south-north and west-east directions, which cross over the intersection without changing their driving directions. That means, for example, if a car drives in south-north direction into the intersection, it leaves the intersection in the same direction. A car can change from lane to lane on its trip depending on the particular traffic situation in its driving direction (see sec. 3 for the behaviour of a car). In this scenario, the system with the central o/c architecture provides a better system performance (flow-rate) than the system with the distributed architecture achieving a flow-rate of 3821 cars after 10.000 iterations. Only 3570 cars can leave the intersection in the distributed case in the same length of simulation time.

6.2 Scenario II

Scenario I is modified into the second scenario so that we have a specific subset of created cars that change their driving directions after they get into the critical area. In this scenario, a car in south-north direction can turn to the right changing its driving direction to west-east and a car in west-east direction can turn to the left changing its driving direction to south-north. We measure the system performance for the cases where 10%, 20%, 30% and 40% of all cars change their driving directions. Fig. 8 shows the system performances for each case with the central and distributed

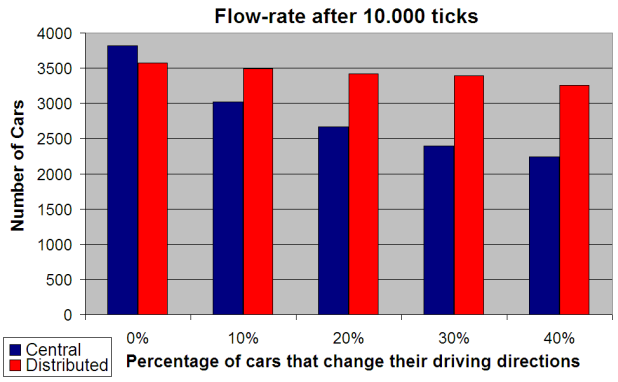


Figure 8: The traffic flow-rate after 10.000 ticks where cars change their driving directions in scenario II

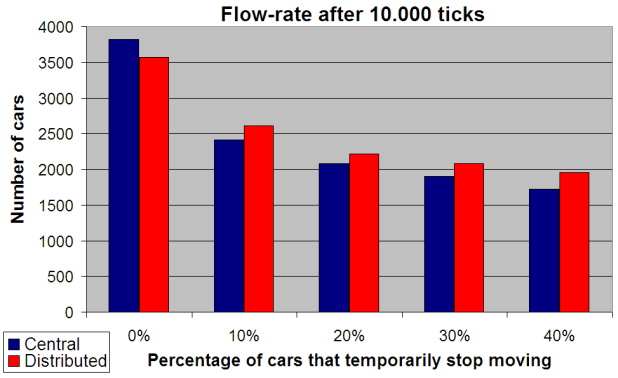


Figure 9: The traffic flow-rate after 10.000 ticks where cars temporarily stop moving in scenario III

o/c architectures. Note that the system performances for 0% correspond to the performances measured in scenario I.

6.3 Scenario III

Scenario I is modified into this scenario so that we can simulate a temporary deficiency of the system elements. In scenario III we have a number of defective cars, which temporarily stop moving and block other cars that come afterwards. A car on its trip can stop moving at a random position in the intersection. Since we want to simulate only a temporary deficiency, a defective car continues moving again after some time. This scenario demonstrates an internal disturbance in the basic system presented in scenario I. We investigate the effects of implemented architectures on the system for the cases where 10%, 20%, 30% and 40% of cars temporarily stop moving. The resulting performances can be seen in Fig. 9.

6.4 Scenario IV

After we demonstrate the effects of an internal disturbance on the system, we investigate here the system performance in the case of an external disturbance. In this scenario, our system consists of cars in south-north and west-east directions and the corresponding observers and controllers. As an external disturbance we create two additional traffic flows in

north-south and east-west directions so that we have a two-way traffic in both directions. Cars in new traffic flows don't have an o/c so that they do not collaborate with other cars. They only consider their own gain and try to cross over the intersection as soon as possible. Due to the aggressive and non-collaborative driving behaviour of cars in north-south and east-west directions, cars in south-north and west-east directions are blocked repeatedly while they cross over the intersection. The configuration of this scenario is shown in Fig. 10. Since cars in north-south and east-west directions

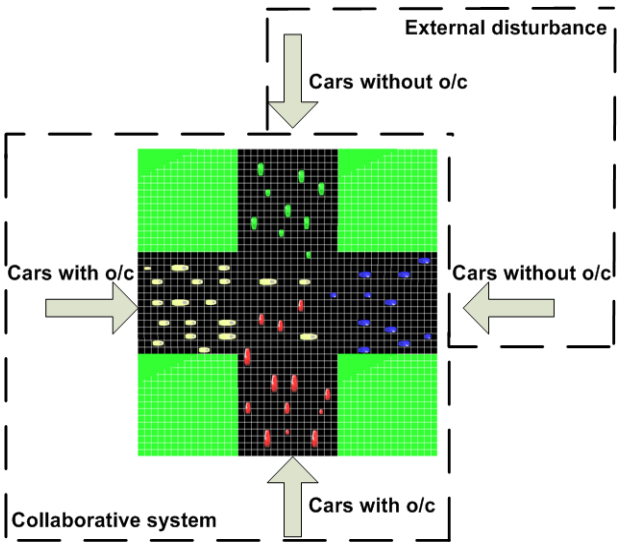


Figure 10: Cars without an o/c are considered as an external disturbance

don't have an o/c, they are not considered as part of the collaborative system. Thus, in measuring the system performance we only consider cars with an o/c. In this scenario, we measure the system performance for the cases where on average 0.5, 1.0, 1.5 and 2.0 cars per tick drive into the intersection in north-south and east-west directions. The results can be seen in Fig. 11.

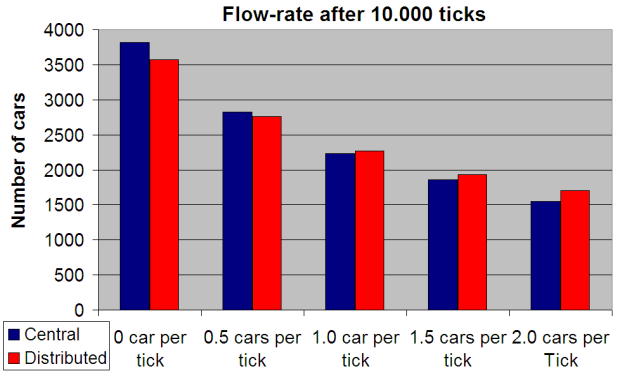


Figure 11: The traffic flow-rate after 10.000 ticks with two additional traffic flows in north-south and east-west directions in scenario IV

6.5 Evaluation of the system performance

The configuration of scenario I produces no disturbances and allows one type of agent behaviour, since cars don't change their driving directions. Thus, we have a low complexity in scenario I that can easily be handled by the central o/c. The non-limited view allows the central o/c to work on a higher abstraction level and implement a large-scale collaboration between cars providing a better system performance while saving the required system resources like the cpu and memory. On the other hand, the distributed o/c has a limited view so that a car cannot collaborate with another car outside its neighbourhood. Thus, a car (C1) with direction D_1 can block another car (C2) with direction D_2, if $D_1 \perp D_2$ and C2 was not in the neigbourhood of C1 in the previous time step. This produces additional delay for cars while crossing over the intersection, which eventually leads to a lower flow-rate.

In scenario I, we have cars with orthogonal driving directions in the critical area at the same time only if we use the distributed o/c. In scenario II, this is always the case regardless of which o/c we use, since cars can turn to the left or to the right in the critical area. This kind of agent behaviour produces additional competition situations in the critical area, if we use the central o/c. But the central o/c doesn't consider this kind of competition situations since it determines priorities for groups of cars on the macro (higher abstraction) level. This decreases the collaboration level (and at the same time increases the competition level) between agents, which eventually leads to the development of temporary clusters in the critical area. These clusters disperse after some time without an intervention from a controller, and this in turn decreases the flow-rate in the intersection. The distributed o/c, on the other hand, can determine priorities for each car on the micro (lower abstraction) level sustaining the collaboration level between agents. Therefore, the distributed o/c provides a better system performance in scenario II depicted in Fig. 8.

Scenario III and scenario IV increase the complexity in the intersection by creating internal and external disturbances respectively. In both scenarios we have cars that don't participate in the collaboration, which leads to a decrease of the system performance in central and distributed cases. The experiments show that the distributed o/c scales better than the central one in the case of an increasing complexity leading to the conclusion that the distributed o/c is more robust than the central one in both scenarios. The implementation of the collaboration between agents with the distributed o/c on a lower abstraction level prevents the propagation of disturbances more successfully isolating the agents affected by the corresponding disturbance from the rest of the system. On the other hand, on a higher abstraction level, we can more effectively determine bottlenecks in the system due to the non-limited view, but we cannot define the limits of the isolation precisely enough with the result that more collaborating agents get disadvantaged by non-collaborating agents than in the distributed case.

Although we expect that the central approach should work at least as well as the distributed one in scenarios with the increased complexity, the experiments show that the more complex the situation in the intersection is, the better works the system with the distributed o/c in spite of the limited view. Thus, a central o/c with a larger view on the system, which makes decisions on a high abstraction level, doesn't

implement the required collaboration in complex situations as successful as the distributed o/c. On the other hand, the central o/c works better in the low-complexity situation presented in scenario I. Thus, the design optimum for systems, which work in a dynamically changing environment, is located neither on the central nor on the fully distributed level. So, the optimum should be somewhere in between the central and distributed approaches.

The results suggest an adaptive architecture switching between a centralized and a distributed o/c architecture depending on the current complexity domain. Such an architecture can be realized by incorporating the presented central and the distributed o/c architectures in the system as shown in Fig. 2(d). Here, the central observer monitors continuously the system state and dynamics and give the corresponding information to its controller. According to the identified situation in the system, the central controller either sends a control signal directly to the agents so that the distributed controllers only execute actions, which they get from the central controller without "thinking" or lets the distributed controllers determine the next action of their cooresponding agents. This type of architecture increases the communication cost, since each distributed controller must communicate with the central controller in order to determine their work mode that is either executing the predetermined actions from the central instance or calculate the next action for its agent based on its limited view. Such a multi-level architecture covers all the scenarios with different complexity levels presented in this section providing the best possible system performance for each case. Since we can show that the system performance depends on the collaboration between the agents, this kind of switching between the central controller and the distributed controllers can be implemented according to the degree of collaboration occurring in the system. Thus, such an adaptive architecture can realise the collaboration between agents both on the macroscopic and the microscopic levels.

6.6 Evaluation of the communication cost and response time

We also compared the implemented architectures with respect to communication cost and response time[1]. Communication cost resulting from the application of both architectures to the system can be found in table 2. We considered only car-to-car communication in the distributed case. That means, we omitted the local information exchange between an o/c pair and their corresponding car in determining communication costs. Therefore, the amount of information sent to the system is 0, since a controller in the distributed case sends messages only to its own car. The corresponding observer on the other hand, communicates with all cars that are in the direct neighbourhood, and receives messages from them. That means that each car must send a message not only to a single observer as in the central case but to each observer in its neighbourhood, which in turn leads to a higher communication cost.

In determining response times, we assume that a single message has a maximum size of 1375 Byte and is sent via an IEEE 802.11 network with 5.5 MBit/s, which leads approximately to 2 ms message transmission time. Table 1 shows the resulting response times in the central and in the distributed cases.

[1]The response time of an o/c pair is the time between the moment when the observer retrieves information from the system according to its observation model and the moment when the corresponding controller sends appropriate signals to the system according to situation parameters from the observer.

	Observer	Controller	Response time
Central	25.58 ms	2.93 ms	28.51 ms
Distributed	12.43 ms	2.00 ms	14.43 ms

Table 1: Response times in ms

We have shown that the distributed o/c provides a better system performance in complex scenarios whereas the central o/c performs better in the low-complexity scenario. If we want to develop an adaptive architecture that can switch between the centralized and distributed architectures, we must consider the communication cost and response time resulting from the utilisation of each architecture. Therefore, when switching from the centralized to the distributed architecture, we must assure that the underlying communication network can scale with the increasing message transmission rate and perform acceptable with the growing size of exchanged messages between system elements per time unit (see Table 2). On the other hand, when switching from the distributed to the centralized architecture, we must consider that the decision making on the central level takes longer than on the distributed level (see Table 1) and assure that the underlying system can still perform in real-time after switching.

7. CONCLUSION AND OUTLOOK

We investigated the generic o/c architecture in the case of an intersection without traffic lights. We implemented different o/c architectures and applied them to the intersection in order to increase the system performance. In this context, we considered the clustering problem in the intersection as a resource sharing problem, which is a common problem in multi-agent systems, and presented a priority allocation mechanism to cope with that problem. We implemented this priority allocation mechanism on different distribution levels of the generic o/c architecture. We presented a fully distributed and a central o/c architecture and provided a comparison of both architectures. As comparison criteria we used the system performance, the communication cost and the response time resulting from the application of each architecture to the system. Our experiments show that the central o/c doesn't implement the required collaboration in complex situations as successful as the distributed o/c. The more complex the situation is, the better works the distributed o/c in comparison to the central one. The central o/c, on the other hand, provides a better system performance in the low-complexity scenario. This suggests the implementation of an adaptive architecture, which combines the advantages of both the central and the distributed o/c architectures. Such an architecture can switch between a centralized and a distributed o/c depending on the current complexity domain, which requires a specific degree of collaboration between the agents in the system. Future work will have to determine criteria for a transition between the domains.

Acknowledgment: This work has been done within the DFG Priority Program 1183 Organic Computing [5].

	Information retrieved from the cars (Observer)	Information sent to the cars (Controller)	Total
Central	261.04 Kb/Tick	0.42 Kb/Tick	261.46 Kb/Tick
Distributed	923.40 Kb/Tick	0 Kb/Tick	923.40 Kb/Tick

Table 2: Communication costs in Kb/Tick

8. REFERENCES

[1] J. Branke, M. Mnif, C. Müller-Schloer, H. Prothmann, U. Richter, F. Rochner, and H. Schmeck. Organic computing - addressing complexity by controlled self-organization. In T. Margaria, A. Philippou, , and B. Steffen, editors, *Proceedings of ISoLA 2006*, IEEE-ISoLA, pages 200–206, Paphos, Cyprus, NOV 2006.

[2] N. Correll and A. Martinoli. Collective Inspection of Regular Structures using a Swarm of Miniature Robots. In *The 9th Int. Symposium on Experimental Robotics (ISER 2006)*, Springer Tracts in Advanced Robotics, pages 375–385, 2006.

[3] S. A. Curtis, M. L. Rilee, P. E. Clark, and G. C. Marr. Use of swarm intelligence in spacecraft constellations for the resource exploration of the asteroid belt. In *Proc. of the 3rd Int. Workshop on Satelllite Constellations and Formation Flying*, 2003.

[4] S. A. Curtis, W. F. Truszkowski, M. L. Rilee, and P. E. Clark. Ants for human exploration and development of space. In *Proceedings of IEEE Aerospace Conference*, volume 1, pages 1–261, 2003.

[5] DFG Priority Program 1183 Organic Computing. Website. http://www.organic-computing.de/SPP, 2005. visited June 2007.

[6] M. G. Hinchey and R. Sterritt. 99% (biological) inspiration... In *Proc. of the 1st IFIP International Conference on Biologically Inspired Cooperative Computing (BICC 2006)*, volume 216. Springer, 2006.

[7] K. Konolige, C. Ortiz, R. Vincent, A. Agno, M. Eriksen, B. Limketkai, M. Lewis, L. Briesemeister, E. Ruspini, D. Fox, J. Ko, B. Stewart, and L. Guibas. CENTIBOTS large scale robot teams. 2003.

[8] M. Long, A. Gage, R. Murphy, and K. Valavanis. Application of the distributed field robot architecture to a simulated demining task. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3193–3200, 2005.

[9] N. C. J. V. M.J. North, T.R. Howe. The repast simphony development environment. In *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models and Mechanisms*, 2005.

[10] M. Mnif, U. Richter, J. Branke, H. Schmeck, and C. Müller-Schloer. Measurement and control of self-organised behaviour in robots. In *Proceedings of the 20th International Conference on Architecture of Computing Systems (ARCS 2007)*, volume 4415 of *LNCS*, pages 209–223. Springer, MAR 2007.

[11] R. R. Murphy. *Intro to AI Robotics*. MIT Press, 2000.

[12] U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, and H. Schmeck. Towards a generic observer/controller architecture for organic computing. In *INFORMATIK 2006 – Informatik für Menschen!*, volume P-93 of *GI-Edition - Lecture Notes in Informatics (LNI)*, pages 112–119. Bonner KÃűllen Verlag, 2006.

[13] V. Trianni, T. H. Labella, R. Groß, E. Sahin, M. Dorigo, and J.-L. Deneubourg. Modeling pattern formation in a swarm of self-assembling robots. (TR/IRIDIA/2002-12), 2002.