Investigation of Generic Observer/Controller Architectures in a Traffic Scenario

Emre Cakar, Jörg Hähner and Christian Müller-Schloer*

Abstract:
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 [RMB+06] in order to establish the 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.

1 Introduction

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 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 behavior 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. The generic o/c architecture has been proposed in [RMB+06] in order to establish the controlled self-organisation in technical systems. 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 expect an advantage of a centralized architecture in case of a low-conflict scenario whereas a distributed architecture should performe better in more complex scenarios.

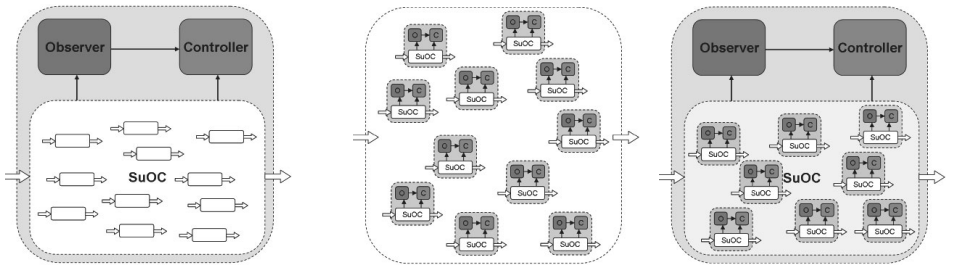
2 The Generic Observer/Controller Architecture

There are architectures presented in the literatur, which exibit promising results in collaborative problem solving in multi-agent systems. However, they are either specialized in solving specific problems [CM06, TLG+02, KOV+03] or are used in specific problem domains [CTRC03, LGMV05]. Generic concepts and methodologies related to the observation and control of collaborative systems are investigated only in [RMB+06, BMMS+].

*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".

The generic o/c architecture [RMB+06] has been proposed in this context. 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. 1(a)), whereas in the latter case there is one observer and one controller for each agent (see Fig. 1(b)) in the system. 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. 1(c)).



(a) Central (b) Distributed (c) Multi-level

Figure 1: Distribution possibilities of the generic observer/controller architecture

Our goal is to investigate different distribution possibilities of the generic o/c architecture. In this context, we implemented a sample traffic scenario to explore different collaboration and communication patterns resulting from application of implemented o/c architectures to the system.

3 Problem Description

We used the agent-based modeling and simulation toolkit *"RePast"* to implement our scenario. RePast provides a scheduler, which triggers agents to perform their predefined behavior in each time step. A time step in a RePast simulation is called a *"Tick"*. In our experiments, we also used the notion of ticks in producing experimental results. Since we have a simulated traffic scenario, we can adopt metrics that are already used in the literature to measure the system performance in our scenario. Common metrics used in traffic scenarios are waiting time, the difference between the minimum possible travel time and actual travel time, percentage of stopped cars, density of cars and travel time. We used the mean waiting time in the intersection to measure the system performance. All cars in the intersection exhibit the same behaviour. A car (C1) first tries to move forward. If another car (C2) is in front of it, C1 tries to overtake C2 from the right. If the intended position is occupied by another car (C3), so C1 tries to overtake C2 from the left. C1 doesn't change its position in the case where all intended positions are already occupied. We first ran the simulation without an o/c architecture 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. 2). The problem defined 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 used a priority based scheduling algorithm, which is adopted from operating systems theory, in order to cope with the clustering problem in the critical area. Just as in the case of a scheduler in an operating system, we determine priorities for cars with respect to their waiting times in the intersection. A car (or a group of cars) with higher waiting time gets a higher priority. In our work we used this priority allocation mechanism on different distribution levels of the generic o/c architecture.

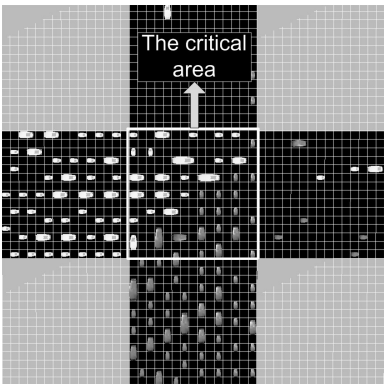


Figure 2: The critical area and the clustering behavior in this area

4 Observer/Controller Architectures

We first implemented a fully distributed o/c architecture, 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. 3 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 behavior of a car in the next simulation step. In the following, these local rules are explained in the sample situation given in Fig. 3. The observer creates a list of situation parameters considering the neighborhood of its corresponding car. These parameters include cars in the direct neighborhood, their waiting times and their positions relative to the car, to which the observer belongs. In the example given in Fig. 3, 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" 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. The controller first checks relative positions in the parameter list. If it encounters one of the relative positions "On_the_Right",

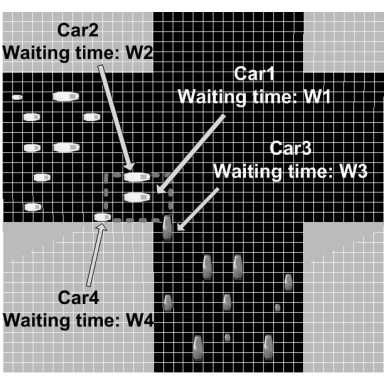


Figure 3: The direct neighborhood of Car1

"On_the_Left" or "Conflict", it compares the waiting time of its corresponding car with the waiting 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 waiting 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 behavior of its right neighbor.

We also implemented the priority allocation mechanism presented in section 3 on the central level. The central observer and the central controller can interact with all cars in the intersection, i.e. their view is not limited. Since we have a different view and granularity level than what we had in the fully distributed case, we can use the priority allocation mechanism to determine priorities for groups of cars on the central level. Thus, we assign priorities to cars in each traffic flow with respect to their cumulative waiting times (see Fig. 4). On the central level, the observer interacts with each car, which has not entered the critical area (see cars in red and blue circles in Fig. 4) in determining priorities for each traffic flow. After collecting waiting times, the observer calculates cumulative waiting 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 simulation 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 cars, which could enter the critical area in the next simulation step, but did not enter yet. After getting cumulative waiting times and a list of cars, the controller determines priorities for traffic flows and sends stop and go signals to corresponding cars.

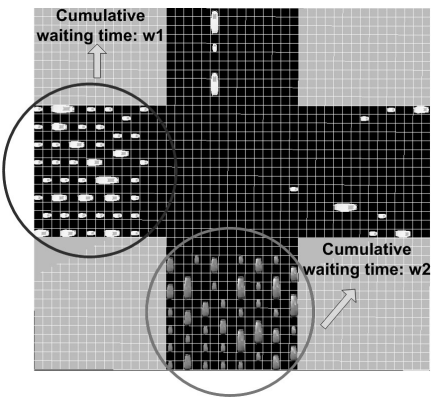
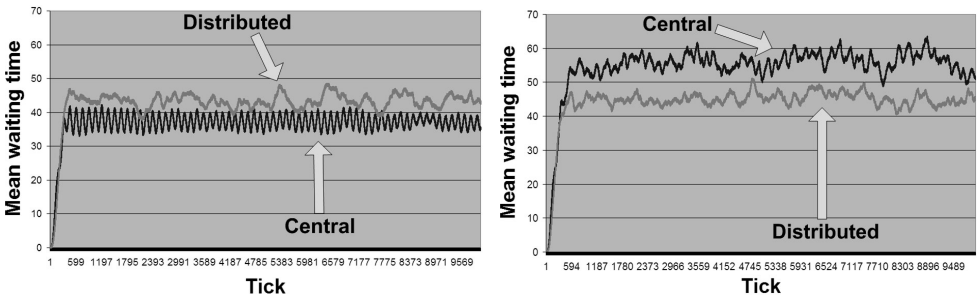


Figure 4: The cumulative waiting times of cars w1 and w2

5 Experimental Results

We used two different test scenarios to measure and compare system performances, which result from the application of each architecture to the system. In scenario A, cars cross over the intersection without changing their driving directions, whereas in scenario B 20% of all created cars try to change their driving directions after they get into the critical area. The experimental results are shown in Fig. 5. The central architecture provides a better system performance than the distributed one in scenario A (see Fig. 5(a)). The application of the distributed architecture leads to a better system performance in scenario B (see Fig. 5(b)). In scenario A, we have cars with orthogonal driving directions in the critical area at the same time only if we use the distributed architecture, whereas in scenario B this is always the case regardless of which architecture we use. But the central architecture does not



(a) Scenario A: Cars don't change their driving directions (b) Scenario B: 20% of cars change their driving directions

Figure 5: System performances resulting from the application of central and distributed o/c architectures to the system

	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 1: Communication costs in Kb/Tick

consider conflict situations of that kind since it determines priorities for groups of cars on the macro level before they enter the critical area. This leads to development of temporary clusters, which disperse after some time without an intervention from a controller, and this in turn increases the mean waiting time in the intersection. The distributed architecture on the other hand, can determine priorities for each car on the micro level and prevent the development of clusters in the critical area. Therefore, the distributed architecture provides a better system performance in scenario B depicted in Fig. 5(b). Thus, a central approach that tries to solve problems in a given system on a higher abstraction level doesn't scale with an increasing number of problem situations, which occur on a lower abstraction level. The distributed approach performs better in such a case, since it determines and solves problems effectively on the same abstraction level. Communication cost resulting from the application of both architectures to the system can be found in table 1. We considered only car-to-car comunication in the distributed case. That means, we omitted the 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 [1], 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 2

[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 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 2: Response times in ms

shows the resulting response times in the central and in the distributed cases.

6 Conclusion and Outlook

We have investigated different distribution possibilities of the generic o/c architecture in the case of an intersection without traffic lights. Our experiments show that a central controller, which works on a macroscopic level, doesn't scale with an increasing number of conflict situations that occur on a microscopic level. In other words, a central control mechanism with a larger view on the system, which accordingly makes decisions on a low granularity level, doesn't address the complexity that occurs on agent level. The results suggest an adaptive architecture switching between a centralized and a distributed o/c architecture depending on the current complexity domain. Future work will have to determine criteria for a transition between the domains.

References

[BMMS+] Jürgen Branke, Moez Mnif, Christian Müller-Schloer, Holger Prothmann, Urban Richter, Fabian Rochner, and Hartmut Schmeck. Organic Computing - Addressing Complexity by Controlled Self-organization. In *Proceedings of ISoLA 2006*.

[CM06] Nikolaus Correll and Alcherio 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.

[CTRC03] Steven A. Curtis, Walter F. Truszkowski, Michael L. Rilee, and Pamela E. Clark. ANTS for Human Exploration and Development of Space. In *Proceedings of IEEE Aerospace Conference*, volume 1, pages 1–261, 2003.

[KOV+03] Kurt Konolige, Charles Ortiz, Regis Vincent, Andrew Agno, Michael Eriksen, Benson Limketkai, Mark Lewis, Linda Briesemeister, Enrique Ruspini, Dieter Fox, Jonathan Ko, Benjamin Stewart, and Leonidas Guibas. CENTIBOTS Large Scale Robot Teams. 2003.

[LGMV05] 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.

[RMB+06] Urban Richter, Moez Mnif, Jürgen Branke, Christian Müller-Schloer, and Hartmut 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 Kllen Verlag, 2006.

[TLG+02] Vito Trianni, Thomas H. Labella, Roderich Groß, Erol Sahin, Marco Dorigo, and Jean-Louis Deneubourg. Modeling Pattern Formation in a Swarm of Self-Assembling Robots. (TR/IRIDIA/2002-12), 2002.