

Incremental design of organic computing systems - moving system design from design-time to runtime

Sven Tomforde, Jörg Hähner, Christian Müller-Schloer

Angaben zur Veröffentlichung / Publication details:

Tomforde, Sven, Jörg Hähner, and Christian Müller-Schloer. 2013. "Incremental design of organic computing systems - moving system design from design-time to runtime." In *Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2013), July 29-31, 2013, in Reykjavík, Iceland*, edited by Jean-Louis Ferrier, Oleg Gusikhin, Kurosh Madani, and Jurek Sasiadek, 185–92. Setúbal: SciTePress.
<https://doi.org/10.5220/0004457901850192>.

Incremental Design of Organic Computing Systems

Moving System Design from Design-time to Runtime

Sven Tomforde¹, Jörg Hähner¹ and Christian Müller-Schloer²

¹Organic Computing Group, University of Augsburg, Eichleitnerstr. 30, 86159 Augsburg, Germany

²System and Computer Architecture Group, Leibniz University Hannover, Appelstr. 4, 30167 Hannover, Germany

Keywords: Design Process, System Engineering, Organic Computing, Adaptivity, Intelligent System Control.

Abstract: System engineers are facing demanding challenges in terms of complexity and interconnectedness. Current research initiatives like Organic or Autonomic Computing propose to increase the freedom of the system to be developed using concepts like adaptivity and self-organisation. Adaptivity means that for such systems we defer a part of the design process from design time to runtime. Therefore, we need a runtime infrastructure which takes care of runtime modifications. This paper presents a meta-design process to develop adaptive systems and parametrise the runtime infrastructure in a unified way. To demonstrate the proposed design process, we applied it to a communication scenario and evaluate the resulting system in a realistic setting.

1 INTRODUCTION

Nowadays, the vision of Ubiquitous Computing (Weiser, 1991) becomes increasingly realistic. Technology has become a fundamental part of human lives and supports us embedded in the environments that we encounter on a daily basis. Engineers build technology-driven environments, where systems observe the conditions in the real world, derive plans of how to act best in this environment, try to draw conclusions from observed behaviour, and finally act proactively by applying actions and manipulating this environment. Recent research initiatives like *Organic Computing* (OC), cf. (Müller-Schloer, 2004), develop novel concepts to be able to handle the resulting complex systems. In this context, OC focuses on developing autonomous entities that are acting without strict central control and achieve global goals although their decisions are based on local knowledge. Due to the complexity of the particular tasks, not all possibly occurring situations can be foreseen during the development process of the system. Therefore, the system must be adaptive and equipped with learning capabilities, which leads to the ability to learn new strategies for previously unknown situations.

OC postulates to move design time decisions as typically taken by engineers to runtime and into the responsibility of the OC system itself. Hence, such an OC system requires a self-adaptation mechanism which is generic in the sense that it is not designed

especially for each application. Instead, this mechanism forms a runtime infrastructure which must be adapted to the particular problem using well-defined parametrisable steps. This leads to a meta-design process whose results provide the parametrisations of the online adaptation mechanism.

This paper is organised as follows. Section 2 describes the architectural concept of OC systems, the specific demands for an appropriate meta-design process and the difference to traditional concepts. Afterwards, Section 3 introduces the novel process in detail. This is evaluated using an example application from the data communication domain in Section 4. Finally, Section 5 summarises the paper and gives a short outlook on current and future work.

2 SYSTEM DESIGN

This section discusses the basic system design for OC according to the *Multi-level Observer/Controller* (MLOC) framework (Tomforde, 2012). MLOC is a three-layered framework that implements the desired self-adaptation mechanism. Thereby, each of the layers has a certain functionality that has impact on the design process introduced in this paper. Afterwards, we give a brief overview of design processes and explain why a novel meta-process to incrementally design adaptive systems is needed.

2.1 Observer/Controller Design

The MLOC framework for learning and self-optimising systems provides a unified approach to automatically adapt technical systems to changing environments, to learn the best adaptation strategy, and to explore new behaviours autonomously. Figure 1 illustrates the encapsulation of different tasks by separate layers. Layer 0 encapsulates the system's productive logic (the *System under Observation and Control* – SuOC) which is parametrisable in terms of variable configurations. Layer 1 establishes a control loop with safety-based on-line learning capabilities, while Layer 2 evolves the most-promising reactions to previously unknown situations. Layer 3 provides interfaces to the user and to neighbouring systems. Details on the design approach, technical applications, and related concepts can be found in (Tomforde, 2012).

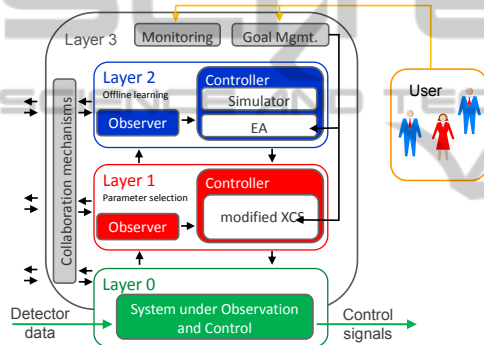


Figure 1: System Design.

2.2 Design Time to Runtime

OC systems are characterised by attributes like self-organisation, self-configuration, self-protection, or self-explanation. In consequence, this means that at least a part of the design effort moves from design time to runtime. OC therefore means to move design time decisions to runtime. This relocation of design activities comprises five aspects:

1) Runtime Exploration: Design space exploration in traditional system design discovers all possibly occurring situations during the design time, while OC systems explore the configuration space using its Layers 1 and 2 at runtime.

2) Runtime Optimisation: The control mechanism defined by Layers 1 and 2 of the MLOC framework continuously optimises the performance of the OC system. Thereby, especially the simulation-coupled optimisation component of Layer 2 has to work under certain constraints, since only limited computing resources and time are available.

3) Online Validation: Searching the configuration space for the optimal parameter setting in a certain situation relies on the possibility to validate candidate solutions. In OC systems, this validation takes place at runtime. Since approaches like trial-and-error mean that bad (or illegal) solutions are tried out in reality, OC proposes to use a sandbox approach where solutions are validated in a simulated environment (i.e. the simulation environment of Layer 2).

4) Continuous Revision: While classical design processes freeze the design at a certain point in time and the result goes into production, OC systems have to work without freezing. This leads to a continuous runtime reconfiguration process where all, even the higher-level, design decisions must be adaptive and changeable at runtime.

5) Runtime Yo-Yo Design: Current design and development processes use models at design time to validate the system before it is actually built. In OC, however, two different flavours of these models have to be distinguished: a) prescriptive models reflect the classical top-down enforcement and b) descriptive models reflect the actual system state. As both models are not necessarily always consistent, possible contradictions have to be resolved or at least minimised – which leads to a runtime version of Yo-Yo design. Runtime modelling is currently gaining high interest (e.g. for self-adaptive systems (Amoui et al., 2012)).

These five aspects of moving design time decisions into the responsibility of the self-adaptation mechanism of the organic system and into the runtime define the specific requirements of OC for an appropriate design process. Considering such a design process from a more general perspective, a dichotomy that partitions adaptive systems according to the targeted functionality can be observed. One part is (similar to traditional approaches) responsible for the productive logic part, while the other part implements the self-adaptivity aspects. Since traditional design processes cover only the productive part, these five aspects are novel and hence define the need of an OC-specific approach. This assumption is substantiated in the following by discussing the most prominent design and development processes as well as their applicability to the design of OC systems.

2.3 Related Work

The organisation and definition of design and development processes has gained a high degree of attention by research and industry since fast and successful projects are a key factor for controlling costs. As a result, approaches following different directions can be found in literature. Thereby, the most

prominent representatives belong to the field of *iterative and incremental development* (Larman and Basili, 2003). These are the *Waterfall* model (Royce, 1988), *V-model* (Forsberg and Mooz, 1991) (and its extension, the *Dual-Vee* model (Forsberg and Mooz, 1995)), the *Y-Chart* model (Gajski et al., 2003) or the *Spiral-model* (Boehm, 1986) that all distinguish between several consecutive development phases, e.g. describing a general concept of operations, refining the system description iteratively with decreasing abstraction until a detailed design exists, and finally implementing the description accordingly and testing it. Here, the overall process is organised according to the ongoing timeline of the project.

Besides these standard concepts, several more hardware-driven approaches like the *Design Cube Model* (Ecker and Hofmeister, 1992) or more recent developments like the *Chaos* model (Raccoon, 1995) and *agile methods* (Erickson et al., 2005) can be found. In addition, researchers focused on building reliable software in the sense of guaranteeing the system's correctness (see e.g. (Good, 1982), or (Nafz et al., 2011) with a certain OC background). A good overview of current design methodologies can be found in (Pressman, 2012). All these processes lack the possibility of applying them to the development of OC systems due to a variety of reasons. First, the most important OC aspect of moving the design time optimisation process into the responsibility of the system itself at runtime is not addressed. Instead, a generalised and time-line-based approach is followed in most of the classic approaches. OC's structure is aligned according to the capabilities instead of the time-line. In this context, the Yo-Yo approach is closest to what we want to achieve with OC systems in terms of changing between top-down and bottom-up constraints – but Yo-Yo design is meant as a design-time process only.

Furthermore, all (including higher-level) design decisions have to be revised continuously in OC systems (technically, this is limited by the constraints of runtime-reconfigurable software or hardware solutions). On the other hand, formal approaches are hardly applicable due to vast situation and configuration spaces and the entailed impossibility to anticipate all potentially occurring situations and best responses at design time. This leads to the insight that the general design of OC systems requires a meta-design process including runtime reconfiguration rather than a standard organisational process structure for the design time part. The architectural concept as illustrated by Figure 1 specifies three layers on top of the classical productive part of the system – comprising in total the application-independent self-adaptation mech-

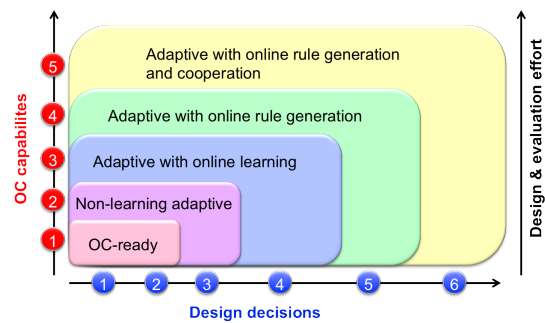


Figure 2: OC capabilities vs. design decisions.

anism. The task of an OC design process is to adapt this generic mechanism to the particular problem using well-defined parametrisable steps.

3 INCREMENTAL DESIGN OF ADAPTIVE SYSTEMS

The design of OC systems according to MLOC (see Figure 1) describes a canonical way of adding self-organised adaptivity and self-optimisation functionality to a productive system. MLOC's basic assumption is that the productive part of the system – the *SuOC* – is handled as an observable and parametrisable black box. Accordingly, the control mechanism defined by higher-layered OC components does not need detailed knowledge about what certain variable parameters mean. In this context, the control mechanism itself can be characterised by an increasing degree of sophistication – depending on the design effort: increasing degree means that an OC system has more possibilities to observe and analyse its state and the environmental conditions as well as reacting more appropriately to these. Considering the design process and the corresponding effort, a system with a lower degree of capabilities requires fewer design decisions (see Figure 2). The next part explains the different capabilities named in Figure 2 in detail (a set of six consecutive *design decisions* that have to be taken) and combines them to a meta-design process.

(1) Observation Model: The first design decision is concerned with the attributes to be observed. All internal and environmental attributes with impact on a) the adaptation or b) the process of measuring the system performance need to be available.

(2) Configuration Model: The second design decision defines the configuration interface to the productive system – which parameters of the SuOC can be altered at runtime in general and which are actually subject to control interventions?

(3) Similarity Metric: Due to the possibly vast

situation and configuration spaces, OC systems have to cope with an unbounded number of possibilities to configure the SuOC. In order to initially close the control loop by activating a rule-based reconfiguration mechanism, a quantification of similarity between situations is needed. This similarity serves as basis for choosing the best available configuration for the currently observed situation.

(4) Performance Metric: Classical system development is based on (mostly hard-coded) implicit goals. In contrast, OC systems have to deal with explicit goals that are user-configurable at runtime – the O/C component uses these goals to guide the system’s behaviour. Automated (machine) learning needs *feedback* to distinguish between good and bad decisions without the need of an external expert. In this context, feedback is a quantification method for evaluating the system performance at runtime. It is derived from observations in each step of the O/C loop.

(5) Validation Method: Automated machine learning has two severe drawbacks: a) it is strongly based on trial-and-error and b) it needs a large number of evaluations to find the best solution. MLOC handles this problem by dividing the learning task into two parts: a) online learning works on existing and tested rules only, while b) offline learning in an isolated environment explores novel behaviour without affecting the system’s productive part. The second aspect can be implemented using *computational* models, *approximation* formulas, or *simulation* (test actions under realistic conditions and therefore evaluate their behaviour in a specific situation). The former two are seldom available, but would be favoured over the latter one due to quality and time reasons.

(6) Cooperation Method: OC distributes computational intelligence among large populations of smaller entities. These entities cooperate to achieve common goals. In order to allow for such a division of work between a set of self-motivated elements, communication and social interaction are needed. Hence, cooperation methods and standardised communication schemes are needed.

The idea of OC is to move design time decisions to runtime. At runtime, the systems make their decisions autonomously. Depending on the application, this is potentially dangerous. This means that the design process has to be planned carefully such that the power of decision is transferred gradually from the designer to the O/C structure. Three main design phases are defined, with each phase resulting in a system with increasing autonomy and requiring more validation (see Figure 3).

Phase 1: Preparation (OC-ready). The first phase prepares a system for a later addition of higher-

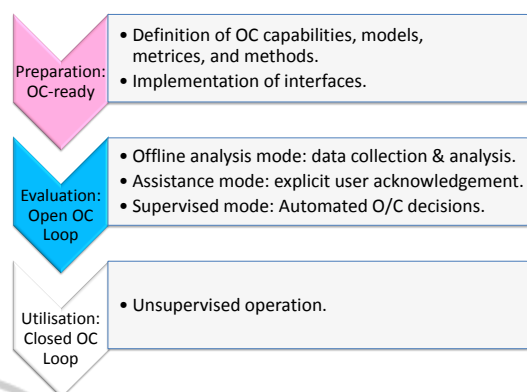


Figure 3: The three OC design phases.

level OC capabilities by defining observation and configuration models and interfaces. Building all new systems OC-ready might be advantageous, regardless whether these interfaces are later used or not.

Phase 2: Evaluation (Open O/C Loop). The second phase adds observer(s) and controller(s) to the system. This requires at least the definition of a similarity metric. For higher degrees of OC capabilities, further OC capabilities like a performance metric or an online validation method have to be defined. This phase stepwise closes the O/C loop. Initially, the evaluation begins by collecting and aggregating observation data and analysing them offline. In a second step, the O/C loop is open and works in assistance mode, i.e. the controller suggests certain control actions to the user who has to explicitly acknowledge them before enactment. In the third step (supervised mode), the O/C loop is closed but the situation observations and the according actions are logged for a later offline analysis in case of wrong decisions.

Phase 3: Utilisation (Closed O/C Loop). Finally in the third phase, the systems work with closed O/C loop and, at least in principle, without super-vision.

4 EVALUATION

The basic design approach as presented in Section 2.1 has been applied to various application scenarios, including vehicular traffic control, production, and mainframe systems (Tomforde, 2012). In the following, we demonstrate the OC design process by applying it to an example application from the data communication domain – the Organic Network Control (ONC) system (Tomforde et al., 2011). Thereby, the design decisions are discussed in detail and the behaviour of the resulting OC system is analysed using a simulation-based approach. ONC has been developed to dynamically adapt parameters of data communica-

tion protocols (i.e. buffer sizes, delays, or counters) in response to changing environmental conditions. It learns the best mapping between an observed situation of the environment and the most promising response in terms of a parameter configuration. ONC has been successfully applied to different types of protocols, including mode-selection in wireless sensor networks, BitTorrent as exemplary Peer-to-Peer client, and mobile ad-hoc networks (MANets) (Tomforde et al., 2011). In the context of this paper, we consider a reliable broadcast protocol for MANets (the R-BCast, details on the protocol can be found in (Kunz, 2003)) as application scenario for ONC.

4.1 Design Decisions for Organic Network Control

(1) **Observation Model:** In a MANet environment, the most important factor influencing the protocol's performance is the distribution of other nodes. Typically, the transmission range for Wifi-based MANets is about 250 meter (the *sending* distance, while the *sensing* distance is 500 meter). Therefore, a sector-based approach as depicted in Figure 4 has been developed. The radius of the outer circle is equal to the sensing distance of the node, as this is the most remote point where messages of this node can interfere with other ones. As nodes within the first circle are really close (50m), their exact position has no impact. In contrast, the direction of the neighbour becomes increasingly important for the second circle (125m – 4 sectors), the third circle (200m – 8 sectors), and the fourth circle (250m – 16 sectors). Within sensing range, two circles (375m and 500m) are introduced and divided into 32 sectors each. A node is assumed to be able to determine the current positions of its neighbours in sensing range relative to its own position, e.g. based on GPS, see (Pahlavan and Krishnamurthy, 2001). Additionally, the node's direction of movement is stored since it has high influence on the best parameter set. Due to the sector-based approach, situations are generalised, which is necessary to avoid evolving a rule for each situation.

(2) **Configuration Model:** The configuration model describes the possibilities of the OC system to automatically manipulate the broadcast algorithm at runtime. Here, we re-use the configuration possibilities provided by the author of the protocol (which he configured once at design time and kept static afterwards). Table 1 lists the parameters and their configuration in the static case.

(3) **Similarity Metric:** A rule-based selection of SuOC configurations has to discover the most promising rule in terms of a configuration set that is most-

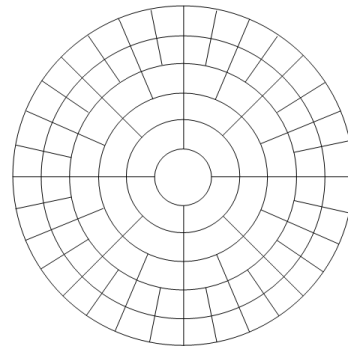


Figure 4: Environment representation.

Table 1: Variable parameters of the R-BCast protocol.

Parameter	Standard configuration
Delay	0.1 s
AllowedHelloLoss	3 messages
HelloInterval	2.0 s
δ HelloInterval	0.5 s
Packet count	30 messages
Minimum difference	0.7 s
NACK timeout	0.2 s
NACK retries	3 retries

related to the currently observed situation. Therefore, the system has to be able to compare situation descriptions which is done based on a similarity metric. In step 1, the situation description has been defined as sector-based encryption of occurring nodes in the neighbourhood. A measure for the similarity of two entities (A, B) to determine the distance, needs to deduct the possible influence of rotation and reflection, initially. Afterwards, the formula for the distance (δ) can be defined with $r \in RADII$ and $s \in SECTORS$ as follows:

$$\delta(A, B) = \sum_r \sum_s (A_{r,s} - B_{r,s})^2 / r.distance$$

The function $r.distance$ defines the radius size as introduced before (50m, 125m, ...). $A_{r,s}$ gives the number of neighbours within the sector s of radius r for the situation description A . This means that the importance of a node's neighbour decreases if it is situated within an outer radius.

(4) **Performance Metric:** The learning mechanism needs an online feedback to draw conclusions from its past actions. In the context of MANet broadcast algorithms, *Packet Delivery Ratio* and *Packet Latency* have to be considered. Both metrics are network-wide figures and cannot be used at each node locally. Nevertheless, the system aims at approximating both effects by reducing the number of forwarded broadcasts and simultaneously assuring the

delivery of each broadcast. To achieve this, the following fitness function ($Fit(x)$) has been developed: $Fit(x) = \frac{\#RecMess}{\#FwMess}$. Here, x represents the currently observed network protocol instance. Since a new parameter set has to be applied for a minimum duration to show its performance, *evaluation cycles* are used defining discrete time slots – the control loop consisting of Layers 0 and 1 is performed once every evaluation cycle. The duration of these cycles depends on how dynamic an environment is. The faster the environment changes, the shorter is the cycle (and the more often is the SuOC adapted). Thus, the formula above takes all messages sent and received within the last cycle into account. It divides the sum of all *received* messages ($\#RecMess$) by the sum of all *forwarded* messages ($\#FwMess$). As a result, high effort (unnecessary forwards) and low delivery rates (not successful broadcasts) are penalised.

(5) Validation Method: The online validation of the Layer 2 component re-uses the protocol's implementation within the network simulation tool *NS-2* (Fall, 1999). Thereby, the neighbouring nodes from the node encoding of step 1 are initialised in relation to the simulated node's position based on a randomised approach within the particular sector. Afterwards, all nodes besides the simulated one are kept static (i.e. they are not moving) which is a reasonable approximation depending on short cycle rates.

(6) Cooperation Method: Cooperation e.g. allows for knowledge sharing between nodes (Tomforde et al., 2011). Further collaboration mechanisms are part of current research initiatives.

4.2 Experimental Results

The experimental setup has been chosen as follows. The simulation environment is implemented in JAVA using the Multi-Agent Simulation Toolkit *MASON* (Luke et al., 2004). Within the simulated area of 1000 x 1000 meters, six agents are created at random positions and move according to a random-waypoint-model (Lawler and Limic, 2010). From these agents, one is equipped with the ONC system and the other five agents perform the standard protocol configuration. The sampling rate of ONC's Layer 1 is set to 1s. The simulation relies on pseudo-randomised movements by taking *seeds* into account – which makes them repeatable and comparable to the usage of the protocol's standard configuration. The investigated simulation period covers seven hours; the results are discretised to blocks of half an hour each. The evaluation compares an OC-ready variant (i.e. the standard protocol version equipped with interfaces to observe it), an Open O/C Loop variant, and

a Closed O/C Loop variant (with and without experience). Thereby, the Open O/C Loop variant is modelled by taking a pre-defined rule-base into account that covers randomly chosen 10% of the occurring situations. The inexperienced Closed Loop variant starts with an empty rule-base, while the experienced one relies on the feedback and Layer 2 rule generations received during three consecutive simulation runs. All results are averaged values of 5 simulation runs.

The first part of the evaluation analyses the general impact of the additional ONC control in this scenario. ONC's goal contains two aspects: assure the delivery of broadcasts and decrease the overhead needed to achieve this delivery. Since deliveries of broadcasts can only be considered at network-level, all messages during the whole simulation are analysed. Figure 5 illustrates the *delivery ratio* of broadcast messages, which is defined as the number of received distinct broadcasts divided by the number of sent broadcasts and the number of agent that have to receive this broadcast. The figure illustrates the desired effect. The OC-ready solution – which is the protocol's standard configuration for all six agents – reported a delivery ratio of 98.73% on average, which is clearly improved by the ONC-control. The *Open O/C Loop* variant resulted in a delivery ratio of 99.05%, the *Closed O/C Loop without experience* variant in a delivery ratio of 99.15%, and the *Closed O/C Loop with experience* variant in a delivery ratio of 99.41% (all values are averages over the complete simulation time). The second aspect in this context is the latency needed to achieve this delivery ratio. The latency values have been determined as the averaged time to deliver a unique broadcast to a receiver. All four values (*OC-ready*, *Open O/C Loop*, *Closed O/C Loop*, and *Closed O/C Loop with experience*) are within a similar range – the maximum deviation between two values is 0.98%. Thus, the impact of ONC on the latencies can be neglected.

The second part of the objective function aims at minimising the overhead needed to deliver the broadcasts successfully. In this context, *overhead* is defined as all messages that do not belong to the distinct broadcast message delivered to each of the other agents. In particular, this includes non-broadcast messages (e.g. *NACK* messages or "hello"-messages) and broadcast duplicates or re-transmissions. Figure 6 depicts the results for all four simulations. The *OC-ready* variant resulted in an average number of 45,785 overhead messages. This value has been decreased in case of an activated ONC for one agent. The *Open O/C Loop* variant resulted in 45,142 overhead messages (decrease of 1.41%), the *Closed O/C Loop without experience* variant resulted in 44,625 over-

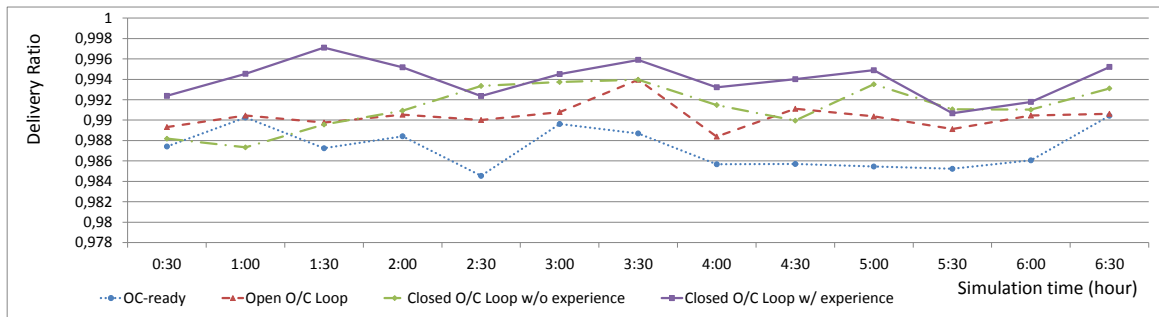


Figure 5: Delivery ratio of broadcast messages (higher values are better) for the four compared variants.

head messages (decrease of 2.60%), and the *Closed O/C Loop with experience* variant resulted in 43,406 overhead messages (decrease of 5.19%). This means, that the capacity of the network has been increased between 1.41 and up to 5.19% due to ONC control – which is a significant improvement. Considering the figure, all three ONC-based solutions are better than the OC-ready variant (i.e. the reference solution) – besides the overhead between 30 and 60 simulated minutes. In this case, the standard configuration of the protocol leads to better results than the *Open O/C Loop* and *Closed O/C Loop without experience* versions. But this effect decreases with increasing capabilities and therefore a longer learning duration.

The third part of the evaluation covers the aspect of analysing the required *effort* caused to achieve the particular behaviour. This aspect is concerned with the rule-generation and the SuOC-adaptation behaviour of ONC. Therefore, the following part investigates: a) the development of the rule base over time, b) the closely-connected number of Layer 2 optimisations during the simulation period, and c) the adaptation-demand realised by ONC. Due to the setup, the population is kept static for the Open O/C Loop variant – here, Layer 2 is not active. The Closed Loop variants without and with experience differ mainly due to the need of new rules. The former variant has to make heavy use of its Layer 2 component to discover novel behaviour, leading to 2,170 rule-generations during the simulated period. In contrast, the experienced variant got along with 139 rule generations. Hence, the rule-base is constantly increasing for the inexperienced variant, while being nearly static for the experienced one.

Finally, the effort can be estimated analytically. The sampling interval has been chosen as 1 s, which means that the ONC-controlled agent had the opportunity to adapt its SuOC 3,600 times per simulated hour. Analysis of the data showed that the Layer 1 component constantly took advantage of this possibility to about 90% each hour in all three variants. Thus, every 10th chance to adapt the SuOC has not

been used. In general, the number of adaptations corresponds to the speed at which the neighbourhood changes (and consequently to the nodes' movement speeds). Choosing lower movement speeds would decrease the number of adaptations significantly. But as choosing an appropriate rule depends on a linear search of the rule base and the size of the rule base converges to about 3,000 rules, the effort is manageable. Summarisingly, ONC control of a MANet-based broadcast algorithm has been successful. The overhead caused by retransmissions and “hello”-messages has been significantly decreased, while the delivery of broadcasts has been improved.

5 CONCLUSIONS

Standard design processes are not applicable to novel demands like moving parts of the design time effort to runtime and into the responsibility of organic systems, which leads to a novel meta-design process for the incremental development of adaptive systems as presented in this paper. This novel design process provides a modularised concept for building different stages of Organic Computing (OC) systems ranging from an OC-ready variant to an open Observer/Controller loop to a closed Observer/Controller variant. These three stages are characterised by an increasing degree of autonomy. In order to demonstrate the increasing capabilities of the OC system resulting from the novel process, an application scenario from the data communication domain has been chosen. Here, the increased performance of the control mechanism defined by the Observer/Controller component has been shown in terms of domain-specific metrics like Delivery Ratio and Overhead (in terms of messages).

Current and future work will focus on both aspects covered in this paper: the meta-design process and the application scenario. For the design process, approaches to generalise the collaboration part as out-

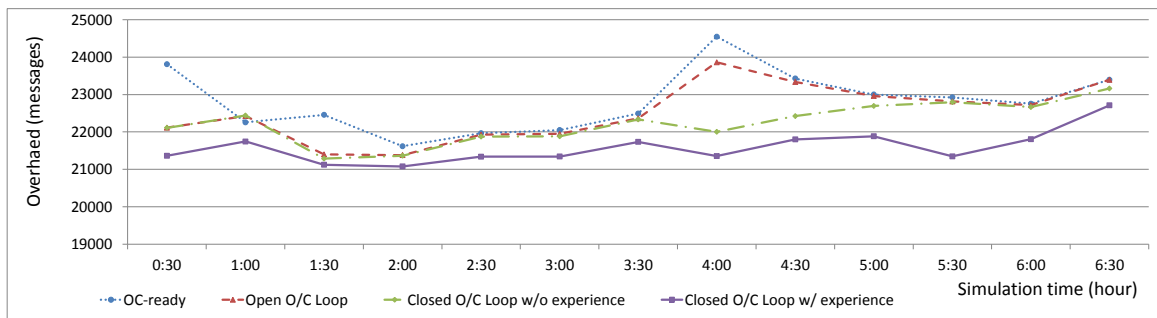


Figure 6: Overhead in messages (lower values are better) for the four compared variants.

lined by the sixth capability are needed – here, especially an automated detection of dependencies between control and observation parameters of neighbouring systems would be useful in order to further improve the quality of the adaptation process. In contrast, the network control example will be extended by covering multiple protocols or physical resources (e.g. switches using the Open Flow standard) instead of one single protocol.

REFERENCES

- Amoui, M., Derakhshanmanesh, M., Ebert, J., and Tahvil-dari, L. (2012). Achieving dynamic adaptation via management and interpretation of runtime models. *J. of Systems and Software*, 85(12):2720 – 2737.
- Boehm, B. (1986). A Spiral Model of Software Development and Enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4):14 – 24.
- Ecker, W. and Hofmeister, M. (1992). The design cube-a model for vhdl designflow representation. In *Design Automation Conference*, pages 752 – 757.
- Erickson, J., Lyytinen, K., and Siau, K. (2005). Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research. *Journal of Database Management*, 16(4):88 – 100.
- Fall, K. (1999). Network Emulation in the Vint/NS Simulator. In *Proc. of 4th IEEE Symp. on Computers and Communications (ISCC'99)*, page 244. IEEE.
- Forsberg, K. and Mooz, H. (1991). The Relationship of System Engineering to the Project Cycle. In *Proc. Symp. of Nat. Council on System Eng.*, pages 57 – 65.
- Forsberg, K. and Mooz, H. (1995). Application of the Vee to Incremental and Evolutionary Development. In *Proc. of Nat. Council for Sys. Eng.*, pages 801 – 808.
- Gajski, D., Peng, J., Gerstlauer, A., Yu, H., and Shin, D. (2003). System Design Methodology and Tools. Technical Report CECS-03-02, Center for Embedded Computer Systems University of California, Irvine.
- Good, D. I. (1982). The Proof of a Distributed System in GYPSY. Technical Report 30, Institute for Computing Science, The University of Texas at Austin.
- Kunz, T. (2003). *Reliable Multicasting in MANETs*. PhD thesis, Carleton University.
- Larman, C. and Basili, V. (2003). Iterative and incremental development: A brief history. *Computer*, 36:47–56.
- Lawler, G. F. and Limic, V. (2010). *Random walk : a modern introduction*. Cambridge Studies in Advanced Mathematics. Cambridge University Press.
- Luke, S., Cioffi-Revilla, C., Panait, L., and Sullivan, K. (2004). MASON: A New Multi-Agent Simulation Toolkit. In *Proc. of the 2004 Swarmfest Workshop*.
- Müller-Schloer, C. (2004). Organic Computing: On the feasibility of controlled emergence. In *Proc. of CODES and ISSS*, pages 2–5. ACM.
- Nafz, F., Seebach, H., Steghöfer, J.-P., Anders, G., and Reif, W. (2011). Constraining Self-organisation Through Corridors of Correct Behaviour: The Restore Invariant Approach. In *Organic Computing – A Paradigm Shift for Complex Systems*, pages 79 – 93. Birkhäuser.
- Pahlavan, K. and Krishnamurthy, P. (2001). *Principles of Wireless Networks: A Unified Approach*. Prentice Hall PTR, Upper Saddle River, NJ, USA.
- Pressman, R. (2012). *Software Engineering: A Practitioner's Approach*. McGraw Hill, Boston, US.
- Raccoon, L. B. S. (1995). The chaos model and the chaos cycle. *SIGSOFT Softw. Eng. Notes*, 20(1):55–66.
- Royce, W. W. (1988). The development of large software systems. *Software Engineering Project Management*, pages 1 – 9.
- Tomforde, S. (2012). *Runtime adaptation of technical systems: An architectural framework for self-configuration and self-improvement at runtime*. Südwestdeutscher Verlag für Hochschulschriften. ISBN: 978-3838131337.
- Tomforde, S., Hurling, B., and Hähner, J. (2011). Distributed Network Protocol Parameter Adaptation in Mobile Ad-Hoc Networks. In *Informatics in Control, Automation and Robotics*, volume 89 of *LNEE*, pages 91 – 104. Springer.
- Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265(3):66–75.