Addressing Challenges Beyond Classic Control with Organic Computing

Jan Kantert[1], Sven Tomforde[2] and Christian Müller-Schloer[1]
[1]*Institute of Systems Engineering, Leibniz University Hannover, Appelstr. 4, 30167 Hanover, Germany*
[2]*Lehrstuhl für Organic Computing, Augsburg University, Eichleitnerstr. 30, 86159 Augsburg, Germany*

Keywords: Adaptive Control Loop, Organic Computing, Intelligent Control.

Abstract: The increasing coupling of former isolated systems towards an interwoven complex structure poses questions about the controllability and maintainability of the corresponding systems. This paper discusses challenges resulting from the growing complexity of technical systems and derives solution perspectives by utilising concepts from the domain of self-organising and self-optimising systems, in particular from the Organic Computing and Autonomic Computing initiatives.

1 INTRODUCTION

Nowadays, the nightmare of *Ubiquitous Computing* as formulated by Marc Weiser in 1991 (Weiser, 1991) turns out to become increasingly realistic. Information and communication technology (ICT) has become a fundamental part of human lives and supports us embedded in the environments that we encounter on a daily basis – it already pervades every aspect of our lives. Besides the apparent benefits of this inclusion, there are several severe drawbacks that entail management and administration complexity. This raising complexity increasingly results in threatening outages and failures.

One particular observation fuelling this trend is the growingly interwoven character of formerly isolated systems: By utilising communication technology, distributed systems are tremendously coupled forming one single complex and interwoven structure (Tomforde et al., 2014). Especially at this point, classic control concepts reach their limits. Controlling this class of systems demands for novel solutions dealing with challenges such as coupling of a variety of systems, self-referential fitness landscapes, reliability and trustworthy issues, or coping with unpredictable behaviour.

In this paper, we discuss how classic control can be supported and enriched by insights from initiatives such as Organic Computing (OC) (Müller-Schloer, 2004) and Autonomic Computing (AC) (Kephart and Chess, 2003). We discuss challenges resulting from the trend towards interwoven systems and derive solution perspectives utilising concepts from the Organic Computing domains.

The remainder of this paper is organised as follows: Section 2 briefly introduces the field of Organic Computing. Afterwards, Section 3 discusses challenges in control theory that need to be addressed in current and upcoming system development. Finally, Section 4 describes a solution perspective for these control problems by utilising concepts from Organic Computing.

2 ORGANIC COMPUTING

Organic Computing aims to build adaptive and robust systems which interact in a dynamic environment. They should be robust against failures and disturbances, flexible in terms of user-originated change of utility functions and adaptive towards changing environmental and internal conditions. To achieve this, OC systems aim to achieve self-x properties, e.g. self-managing, self-configuring, self-learning, and self-optimising.

In general, OC postulates to transfer design decisions that are traditionally situated at design-time to runtime and consequently move them into the responsibility of systems themselves. As a result, we face systems that are increasingly autonomous with decision freedom at different abstraction layers, ranging from pure re-configuration of parameter setting to adapting the decision models and to finally revising higher-levelled design decisions.

Kantert J., Tomforde S. and Müller-Schloer C..
Addressing Challenges Beyond Classic Control with Organic Computing.
DOI: 10.5220/0005556102640269
In *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2015), pages 264-269
ISBN: 978-989-758-122-9

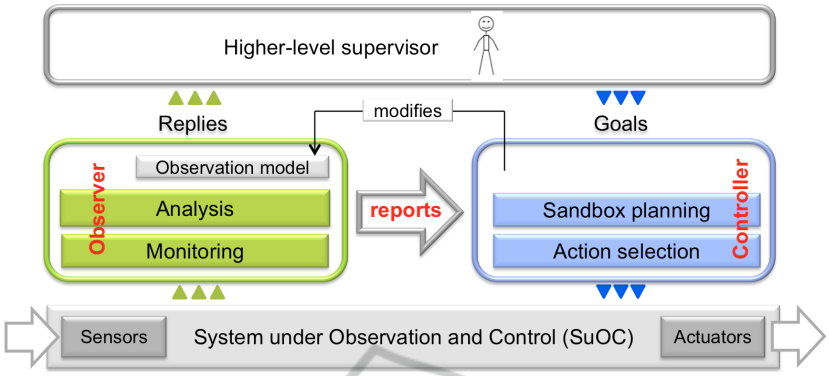

Figure 1: Observer/Controller architecture of an Organic Computing system.

2.1 Observer/Controller

OC systems need to adapt to their environment and, therefore, consist of an observer and a controller component controlling a System under Observation and Control (SuOC) (see Figure 1). The observer monitors the environment, performs analysis, and reports a situation description. Based on this knowledge, the controller runs sandbox planning, and selects actions to configure the SuOC. The actual work is performed by the SuOC which gets sensor inputs and performs simple control tasks. Additionally, the controller supervises the SuOC to match the user's goal and can change the observation model to adapt to the current system state (e.g. at system cold start).

2.1.1 System under Observation and Control (SuOC)

The System under Observation and Control (SuOC) is the actual working part. It is connected to external sensor inputs and controls all actuators hereby forming a classical real-time control loop.

2.1.2 Observer

In an OC system the observer receives the same sensor input as the SuOC. Additionally, it collects status and performance data from the SuOC itself and may have access to further high-level information. The observation process may be modified by the controller via the observation model. This mechanism can be used e.g. to focus attention. Based on this information, the observer updates its internal observation data, performs short and long-term predictions and reports these situation descriptions to the controller.

2.1.3 Controller

Based on the situation description, the controller compares the current system state with the goal provided by the user. However, the goal is not static and can change during runtime. If situation and goal do not match the controller can change the configuration of the SuOC and, thereby, influence the system to approach the correct state.

2.2 Differences to Classic Control

Schmeck et al. (2010) described a classic control loop as a specialised observer/controller. In Figure 2, we show a classic control loop in observer/controller terminology on the left side and as comparison an observer/controller on the right side. As example, we use a heating control with temperature sensors and a heater which should keep a room at a certain temperature. On the left side, we depict a PID controller as an observer, which smooths the sensor value, and a controller, which compares the value to the target value and controls the heater.

On the right side, we show an Organic Computing system controlling a System under Observation and Control (SuOC) which actually controls a heater. Temperature sensor values are monitored and processed by both observer and SuOC. On the upper level, the observer creates a situation description containing the temperature, the observer compares the situation with the target or goal, and influences the SuOC if there are any differences. The SuOC will act based on sensor values and based on influences by the controller. This structure allows a more robust system which can still work with degraded performance when observer and controller fail. Additionally, the controller performs short and long-term forecast to ensure stable control.

In general, Organic Computing systems reach a goal without an explicit analytic model. Responses to changes can be learned at runtime and, thereby, adaptation to a new environment at runtime is possible. However, there cannot be a specific control law and good behaviour cannot be proven. Fortunately, it can

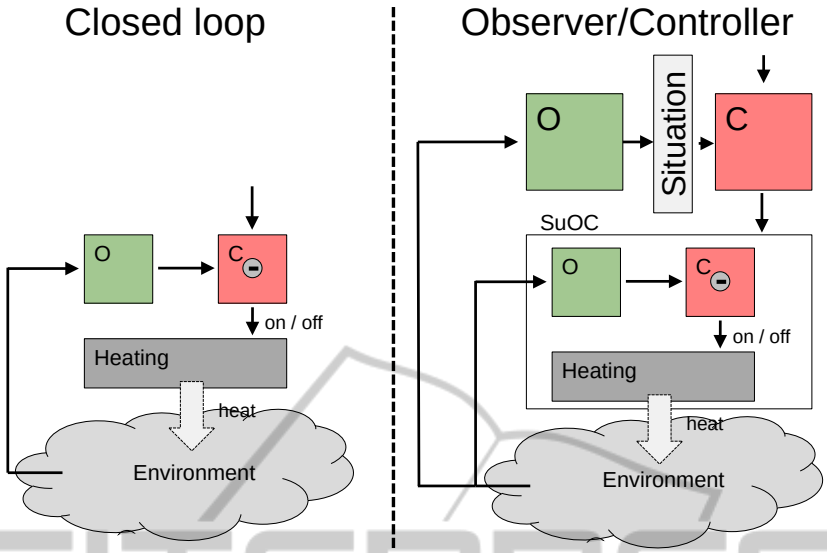

Figure 2: Comparison of closed loop and Observer/Controller architecture.

be shown that OC systems work well in dynamic environments.

3 CHALLENGES BEYOND CLASSIC CONTROL

Compared to classic control systems, novel developments in system engineering raise a set of challenges that are not addressable by standard control techniques any more. Within this section, we discuss the most severe issues.

3.1 Coupling

Intentionally or unintentionally, entities (in the sense of autonomous subsystems) are coupled to other systems and the environment. Influences such as the current status and behaviour of other entities impact the system's performance and the available actions. Thereby, *direct* and *indirect* influences are distinguished. For *direct* influences, the subsystems are aware of each other and their operation within the same domain. For instance, a shared communication medium with a neighbourhood cache provides the basis for such a mechanism. As a result, these systems can be explicitly modelled and taken into account within the control law. In contrast, indirect influences lack the possibility of considering them at runtime. In this context, *indirect* means that a system is either not aware of the other's existence or it cannot observe the actions performed by this entity. As a result, we face mutual influences and dependencies among entities.

As one example consider a smart household: Previously independent systems (such as TVs, fridges, or blinds) are combined forming an integrated system that adjusts the behaviour of its elements according to the (estimated) demands of the users. The resulting solution consists of various devices of different manufacturers – those that were explicitly made for smart environments are combined with other legacy devices or systems that have a limited functionality. In addition, devices will typically rely on varying communication interfaces (Allerding et al., 2010).

Consider for instance a smart household where the music and sound system adapts the loudspeaker configuration according to sensor observations that estimate the user's position. Introducing a novel class of sensors (e.g. infra-red in addition to movement detectors) might lead to values that are not recognisable by existing controllers, or these controllers misinterpret the readings (i.e. activate sound for pets). Furthermore, interdependencies can be observed – e.g. between shutters and lighting. The smart household system is hence a complex of interwoven subsystems that have to cooperate.

3.2 Unpredictable Behaviour

System engineering typically introduces a "system boundary" up to which the behaviour is considered in more detail than outside. Based on these boundaries, hierarchies of abstraction layers are formulated. In current interwoven systems, we face a blurring of these boundaries – no strict separation of concerns is feasible any more. This is accompanied by taking

decisions based on uncertain information[1]. This includes incomplete, noisy, and not reliable data that might originate from untrusted or unknown sources and might be exposed to hidden influences. Furthermore, not all data might be fully accessible and perceivable; for instance, other entities are considered as black-boxes with unknown status and transfer functions. Finally, delayed effects for executed actions lead to uncertainty regarding the actual effectiveness of actions which is accompanied by observations that are not (or not directly) classifiable to a potential origin (e.g. in terms of executed actions).

In addition, structure, organisation, and functionality of subsystems are subject to changes. Novel entities are joining the system throughout the lifecycle, others are removed or replaced. Thereby, legacy systems have to cooperate with those reflecting the up-to-date standard. In software engineering, such a process is already known for isolated entities and to some degree maintained by updates or patches. As a conclusion, we can state that we have to deal with unpredictable behaviour – at structure-level, at information-level, and at control-level.

3.3 Self-referential Fitness Landscapes

The internal control mechanism of a system (i.e. the "controller") is responsible for finding an optimal control strategy in terms of a given performance metric or utility function. Standard control systems work on direct feedback by comparing the target output with a reference and adapting the system's behaviour accordingly.

In contrast, OC systems (and consequently a large set of application systems under construction at the moment) have to deal with various unpredictable influences and vast situation spaces. Thereby, we can observe direct and indirect coupling effects among distributed systems operating within the same environment. In this context, *direct* coupling refers to accessible information (i.e. increase heating power leads to higher room temperature) and *indirect* coupling refers to effects that cannot be traced back to certain actions and might originate from other systems.

Standard systems are characterised by a static, continuous fitness landscape. Such a *fitness landscape* is an objective function defined as the mapping of configuration settings on certain performance values for all possible configurations – the controller's task is to find the optimal point in this landscape. Controllers can also react to slowly changing fitness landscapes by adapting the control law over time (e.g. as response to burning fuel in aeroplanes). For OC systems, the fitness landscape is even more complex. The own configuration impacts the objective function's shape – meaning that changing the configuration towards a current optimum influences the mapping of configuration to performance values. The resulting fitness landscapes of the underlying optimisation problem are called "self-referential" (Cakar et al., 2011). Depending on the time constants of the involved processes, such coupled systems can become unstable. The highly automated stock trading system is an example.

3.4 Vast Situation and Configuration Spaces

The "self-referential" properties of the fitness landscape are even worsened due to vast situation and configuration spaces and the resulting unpredictability of conditions at runtime. With the term *situation space* we refer to all potentially occurring conditions the system will be exposed to at runtime. In contrast, the *configuration space* comprises all available setting of the SuOC, i.e. parameter configurations that are typically altered by the controller at runtime as response to non-fulfilled target values. In classical systems engineering, development processes build and test systems at *run-time* against a set of pre-defined requirements and test data. Interconnectedness and interactions during operation, however, introduce dynamic conditions that cannot be anticipated at design-time. This means that an engineer is not able to search the complete situation and configuration spaces in advance.

For instance, consider the smart household scenario again. Devices might be replaced by novel solutions at runtime, or additional equipment might be installed. The effective control strategy of an individual device cannot be tested together with another solution that is not even invented at the same time. Furthermore, the law of combinatorics suggests that searching the configuration space for an optimal setting (neglecting the challenges of self-referential fitness landscape for a moment) is increasingly hard (and mostly impossible) if many devices have to cooperate. Hence, we have to accept the fact that systems cannot be brought to operation with a configuration that has been optimised once and remains constant.

[1]The term uncertainty is used according to Motro and Smets (1997). There, "uncertain" is a generic term for other terms such as "likely", "doubtful", "plausible", "reliable", "imprecise", "inconsistent", or "vague".

3.5 Open Systems and Trustworthiness

In general, faulty systems, actuators or sensors pose a challenge for reliable control. However, they can be detected in some control loops which is typically performed using fixed strategies, e.g. when a sensor value does not change any more at all or returns totally invalid values. Unfortunately, this changes when systems become open. Now, intruding components may behave intentionally maliciously. For example an attacker may exploit a heater by tampering some temperature sensors, let them return a very low temperature value and the room would get overheated. However, with a reasonable system model taking into account some context knowledge such as outside temperature and historical values this "attack" could be detected and mitigated.

OC systems typically do not assume any benevolence of components and have to cope with uncertainties according to sensor values or the effect of actions taken by the controller or SuOC. Therefore, they validate the trustworthiness of components and measurements and, thereby, can prevent such attacks or at least recover from them. Additionally, a learning OC system may memorise historical attacks to defend against them in the future (Steghöfer et al., 2010).

4 SOLUTION PERSPECTIVES

Based on the challenges introduced in the previous section, the remainder of this paper proposes a solution perspective. We highlight what insights from the OC domain can be utilised to allow for controlling interwoven systems.

4.1 Self-observation

Mastering complex behaviour (e.g. as result of coupling) requires an appropriate description of the current conditions in the first place. This includes local information (i.e. the status of the entity, its goals, and the decision options), available other resources to interact with in combination with their estimated status and reliability, and environmental conditions, both accessible values and estimated inaccessible values.

4.2 Self-modelling

Based on the perceived information, each entity has to develop a model of the current status; again including the view on itself, others, and the environment. These models have to reflect the estimated reliability of the underlying information sources, the trust towards other entities, and the entity's goal (or current chain of goals). Here, concepts such as *Models@Runtime* (Assmann et al., 2011) can be utilised that allow for a runtime usage of design models. Furthermore, trust and reliability techniques (Steghöfer et al., 2010) can be applied to estimate the potential benefits and drawbacks of interaction partners. The research field of *Models@Runtime* started with the motivation that model-driven engineering processes generate varying models at design-time which are not further used at the system's runtime. Hence, possibilities to extend the applicability of models and abstractions during operation of the system under development are investigated. The motivation of the conducted research is to find ways for providing effective technologies for mastering and managing the complexity of evolving software behaviour. Thereby, not just the design-time but also the runtime aspects can be taken into account making solutions more appropriate (Assmann et al., 2011). Since *Models@Runtime* is a very heterogeneous initiative, several concepts from different fields (such as model-driven development processes, requirements engineering, or validation and verification) can serve as input to investigate self-reflective systems. Especially work on the generation of behavioural models for dynamic adaptive systems (Goldsby and Cheng, 2008) provides a good starting point.

4.3 Self-adaptation of Behaviour

According to the derived models, each entity has to adapt its own behaviour. Thereby, we distinguish between behaviour (i.e. local processes performed by the entity itself that are influenced by configuration parameters) and structure from a systems engineering point of view (i.e. connections with other entities). For the behaviour aspect, safety-oriented learning techniques such as developed within Organic Computing (Tomforde et al., 2011) provide a basis.

4.4 Self-management of System Structure

Besides the own behaviour, the entity has to decide about its position within the overall structure. This means it has to establish and remove relations to other entities – which is again based on the observations and, consequently, on reliability and trust, on performance estimations, and on quality measurements with respect to further influencing factors such as redundancy, latency, or ownership of the possible interaction partner. Furthermore, the derived models might indicate *indirect* coupling effects

with other entities (where no *direct* relation exists) which might not be desired and therefore avoided. For explicit dependencies, the knowledge models of Organic Computing systems have been mostly adequate, since self-learning systems can find such relations based on e.g. reinforcement learning techniques (Sutton and Barto, 1998). For implicit dependencies, methods to model dependencies between randomised variables are needed. For instance, risk measures (McNeil et al., 2005) from Operational Research or copula models (Nelsen, 1998) can be used. Current research effort works towards self-integrating techniques that merge design-time and runtime processes (Bellman et al., 2014).

4.5 Self-optimisation

Finally, all the previous steps are performed on the basis of a currently active configuration that serves as general decision guideline. This is subject to a higher-level process that tries to self-optimise all the underlying self-x processes. Thereby, applied reinforcement techniques provide the first step towards self-optimisation – a further step is to integrate suitable search heuristics (Cakar et al., 2011).

In contrast to state-of-the-art model-based controllers that work on static and design-time-based models, we need capabilities to automatically derive and maintain models at runtime (Niemann et al., 2013).

5 CONCLUSION

Current and future technical systems show increasingly interwoven characteristics. This leads to novel challenges regarding their controllability. Based on this observation, this paper discussed challenges resulting from the interwoven character and outlines solution strategies based on Organic Computing techniques. We derive five key enablers to counter complexity issues and outlines how classic control concepts can be enriched to provide sufficient and stable feedback mechanism that can deal with interwoven systems.

REFERENCES

Allerding, F., Becker, B., and Schmeck, H. (2010). Integration intelligenter Steuerungskomponenten in reale smart-home-Umgebungen. In *Informatik 2010: Service Science - neue Perspektiven für die Informatik*, volume 1, pages 455 – 460, Bonn, Germany. Gesellschaft für Informatik. ISBN978-3-88579-269-7.

Assmann, U., Bencomo, N., Cheng, B., and France, R. (2011). Models@run.time. *Dagstuhl Reports*, 1(11):91 – 123.

Bellman, K. L., Tomforde, S., and Würtz, R. P. (2014). Interwoven systems: Self-improving systems integration. In *Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014, London, United Kingdom, September 8-12, 2014*, pages 123–127.

Cakar, E., Tomforde, S., and Müller-Schloer, C. (2011). A Role-based Imitation Algorithm for the Optimisation in Dynamic Fitness Landscapes. In *Swarm Intelligence (SIS), 2011 IEEE Symposium on*, pages 139 – 146. IEEE.

Goldsby, H. J. and Cheng, B. (2008). Automatically Generating Behavioural Models of Adaptive Systems to Address Uncertainty. In *Proc. of Model Driven Engineering Languages and Systems. 11th International Conference, MoDELS 2008, Toulouse, France, September 28 - October 3*, LNCS 5301, pages 568 – 583. Springer.

Kephart, J. and Chess, D. (2003). The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50.

McNeil, A. J., Frey, R., and Embrechts, P. (2005). *Quantitative risk management: concepts, techniques and tools*. Princeton University Press.

Motro, A. and Smets, P. (1997). *Uncertainty Management in Information Systems – From Needs to Solutions*. Springer Verlag.

Müller-Schloer, C. (2004). Organic Computing: On the Feasibility of Controlled Emergence. In *Proc. of CODES and ISSS'04*, pages 2–5. ACM Press.

Nelsen, R. (1998). *An Introduction to Copulas (Lecture Notes in Statistics)*. Springer, New York, NY, USA.

Niemann, S., Kotlarski, J., Ortmaier, T., and Müller-Schloer, C. (2013). Reducing the Optimization Problem for the Efficient Motion Planning of Kinematically Redundant Parallel Robots. In *Proceedings of the International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 618–624. IEEE/ASME.

Schmeck, H., Müller-Schloer, C., Çakar, E., Mnif, M., and Richter, U. (2010). Adaptivity and self-organization in organic computing systems. *ACM Trans. Auton. Adapt. Syst.*, 5:10:1–10:32.

Steghöfer, J.-P., Kiefhaber, R., Leichtenstern, K., Bernard, Y., Klejnowski, L., Reif, W., Ungerer, T., André, E., Hähner, J., and Müller-Schloer, C. (2010). Trustworthy Organic Computing Systems: Challenges and Perspectives. In *Proc. of ATC 2010*, pages 62–76, Boston, MA. Springer.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, US.

Tomforde, S., Hähner, J., and Sick, B. (2014). Interwoven Systems. *Informatik-Spektrum*, 37(5):483–487. Aktuelles Schlagwort.

Tomforde, S., Prothmann, H., Branke, J., Hähner, J., Mnif, M., Müller-Schloer, C., Richter, U., and Schmeck, H. (2011). Observation and Control of Organic Systems. In *Organic Computing - A Paradigm Shift for Complex Systems*, pages 325 – 338. Birkhäuser, Basel, CH.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265(3):66–75.