An Organic Computing Perspective on Self-Improving System Interweaving at Runtime

Sven Tomforde*, Stefan Rudolph†, Kirstie Bellman‡ and Rolf P. Würtz§
*University of Kassel, Intelligent Embedded Systems Group, Kassel, Germany
Email: *stomforde@uni-kassel.de*
†University of Augsburg, Organic Computing Group, Augsburg, Germany
Email: *stefan.rudolph@informatik.uni-augsburg.de*
‡Topcy House Consulting, Thousand Oaks CA, USA
Email: *bellmanhome@yahoo.com*
§Ruhr Universität Bochum, Institute for Neural Computation
Email: *rolf.wuertz@ini.ruhr-uni-bochum.de*

***Abstract*—The complexity of today's technical systems evolves over long periods of time. For example, systems may join or split, while subsystems may change their behaviour. We often observe that subsystems not intended to interact get coupled and influence each other. Apart from ICT-based coupling, as via the Internet, we also observe both direct and indirect interaction via the physical and social world as systems explicitly react to human behaviour and environmental conditions - and influence both through their actions. Thus, any two (sub-)systems coming into contact with the same part of the environment or group of users can influence each other in ways that can hardly be anticipated at design-time. Undesired effects may be the consequence of unintended, implicit interaction of elements that were not fully predictable or relevant beforehand. The existence of such effects is not a design decision, they are already present in the richness of our complex systems and components and their number is increasing. In previous work, we discussed some of the problems in continually integrating these complex systems, and in this paper, we dig deeper into both the challenges entailed in 'interwoven systems' and discuss some of the necessary aspects to developing architectures that will address these challenges.**

I. Introduction

The vision of *Ubiquitous Computing* as formulated by Marc Weiser in 1991 [1] is becoming increasingly realistic. Technology has become a fundamental part of life and is embedded in the environments that support us in our daily lives. Information and communication technology (ICT) pervades every aspect of our daily activities, e.g., in household appliances [2], autonomous and self-driving cars [3], smart metering technology [4], [5], or intelligent spaces [6], [7]. Another driving force of this development is the ubiquitous usage of smart cell phones – nearly everybody in the industrialised countries is carrying a mobile Internet-compliant device that can constantly be connected to local services[1]. This inclusion has changed our communities and human interaction in a significant manner. Based on the motivation to bring more comfort into our daily routine or to add capabilities to our work, this increasing interconnectedness fundamentally modifies our way to control and manage technical systems – it also entails a dramatic change in the way we design and integrate these utilised systems. As a result, we can observe tremendous complexity that evolves over long periods of time [8], [9].

In all phases of a system's lifetime (i.e. design, integration, operation, maintenance, disintegration), we face novel problems that are mainly a result of the massive interconnectedness and the corresponding dependencies that evolve between distributed subsystems. Because of these interdependencies and side-effects, we face a variety of outages and failures of individual subsystems and cascading failures among subsystems; the underlying causes are often difficult to identify. Most of today's systems evolve continuously over long periods of time [10]. During that time, they have to cooperate with changing and even completely novel devices and environments. This system of systems integration is further complicated by the lack of knowledge or the authority to control other cooperating systems [11]. Even carefully designed and tested solutions are not able to react properly to all the kinds of problems that might be entailed by such a continuously changing integration status. This is especially difficult due to the dependence of operating conditions on human, social or environmental factors — we will refer to systems with these properties as *'Interwoven Systems'* (IwS) [10].

Section II of this paper discusses a set of prominent examples where the aforementioned issues have already led to outages and failures. Afterwards, we summarise the common characteristics of those examples and highlight the existence of the class of systems that we want to treat in Interwoven Systems: IwS (Section III). We propose a self-improving runtime integration process as a solution to the challenges arising with IwS and present an architectural concept following the ideas of Organic Computing [12] in Section IV. Finally, we summarise the paper and give an outlook to future work (Section V).

[1]The International Data Group says that since 2010 the amount of mobile devices has increased more than 15% per quarter, see "Market Analysis Perspective: Worldwide Mobile Phone Market", available online at http://www.idc.com/getdoc.jsp?containerId=245240 (last access: July 12th, 2015).

978-1-5090-1654-9/16 $31.00 © 2016 IEEE
DOI 10.1109/ICAC.2016.15

II. Motivating Examples

Over the last two decades, computer scientists and many others have seen the tremendous increase in complexity due to the increasing interconnection and interaction of distributed systems, components, and services. There are many expressions of this complexity, such as seen in the law of Glass (An increase of 25% functionality in software doubles its complexity [13]) and the law of Moore (The complexity of integrated circuits is doubled with minimal component costs every one to two years [14]). There have also been many notorious failures in our technical systems, where one can see some of the risks and vulnerabilities of how we currently integrate our very large, distributed systems. We will briefly introduce some of these examples in the following paragraphs.

1) Skype: The Peer-to-Peer-based network *Skype* ran into an outage of significant impact in 2007. Initially, the network – mainly used for telephone calls and instant messaging – became unstable before experiencing a critical disruption[2]. Skype's engineers found that this disruption was caused by massive parallel rebooting of users' machines all across the world. The reason for this synchronised down-time of user machines was a routing patch issued by Microsoft for their current Windows installation. Due to this high-priority update, a massive number of computers went down at the same time, which affected Skype's network resources and routing algorithms. Machine relaunching after re-booting in combination with other Skype users that had become temporarily separated from the network led to a rapid peak in flooding of log-in requests. Simultaneously, the network itself was running with noticeably limited resources. A chain reaction was triggered that affected the whole network and resulted in disturbing the automated recovery routines built into Skype to correct malfunctions. Objective testimony of this disturbance was that Skype's services were not available for approximately two days. This example demonstrates an *indirect coupling of two supposedly independent systems*: the Skype communication infrastructure on the one hand and the Windows operating systems on the other.

2) Google Mail: A second example was reported roughly one and a half years later[3]. On February 24th 2009, the prominent mail service *'Google Mail'* was not available for about 150 minutes, meaning that a large number of people world-wide had no access to their emails. What happened was a routine maintenance event that had been performed in one of Google's European data centres. Under undisturbed conditions, email accounts are automatically served by other data-centres – and users do not experience any disruption. In this particular case, the service developed a cascading outage due to unexpected side-effects. The take-over routine had been configured to keep data geographically close to its owner – turning one data-centre after the other down due to overload reasons. After about 150 minutes, the service was re-initiated and all data-centres had been brought back to operation. What we can observe here is an *emergent* effect [15], [16]. Hence, there is a coupling between (local) data-centres and their escalation schemes on the one hand and user behaviour in combination with location-specific business strategies on the other hand. From a more global perspective, business had been affected world-wide, since GMail provides email services to professional customers. This means that those customers' services had not been available. Obviously, today's communication services depend largely and critically on the sustained and uninterrupted existence of services such as GMail.

3) US Blackout: The third example shifts the focus from pure ICT cases to those where ICT is used as an enabler for more hardware-oriented solutions. Northeastern America experienced one of the largest energy blackouts in history on August 14th, 2003[4]. Following the shutdown of a single nuclear power station in Eastlake, Ohio, cascading outages of voltage transmission lines shut down a large part of the network. This slow, but effective and viral failure resulted in more than 55 million households in the US and Canada missing electricity for roughly 18 hours. The final reports investigating the incident named several reasons, the most important being the initial shutdown of the nuclear plant in combination with the network utilisation at the time. Another major finding of the report was that "[...] internal and external links from Supervisory Control and Data Acquisition (SCADA) networks to other systems introduced vulnerabilities [...]"[5]. Hence, there are dependencies between ICT solutions that have a different and presumedly isolated scope and – by intuition – no clear relation to each other (in this case: grid control strategy and data acquisition).

These three specific examples demonstrate the challenges issued by the ongoing interweaving of systems. That is, users and developers often build pathways and interfaces from one system to another for many legitimate benefits and conveniences, but unwittingly open the door to new types of risks and interaction points that can lead, under the right operational conditions to new failure modes. In addition to providing new points of failures, the current trends in ICT development fuel the raise of complexity, interconnectedness, and hidden mutual influences. In order to understand the new characteristics introduced into interwoven systems (and that we therefore will need to address in any architectural and technical solutions), we examine several examples.

Consider for instance the *Smart Grid* as an example of new hidden mutual influences. We used to have strictly centralised and pre-planned systems a decade ago. The rise of renewable energies (such as biogas plants, solar plants, and wind farms),

[2]See e.g. Skype's own report at http://heartbeat.skype.com/2007/08/what_happened_on_august_16.html (accessed on December 22nd, 2014).

[3]See e.g. GMail's own report at http://gmailblog.blogspot.de/2009/02/update-on-todays-gmail-outage.html (accessed on December 24th, 2014).

[4]See e.g. report by history.com at http://www.history.com/this-day-in-history/blackout-hits-northeast-united-states (accessed on December 25th, 2014).

[5]See the final report by the US energy department (i.e. recommendation 34, page 165) at http://energy.gov/sites/prod/files/oeprod/DocumentsandMedia/BlackoutFinal-Web.pdf (accessed on December 25th, 2014).

the introduction of electrical vehicles, the possibility to control demands (e.g., due to smart meters), and similar developments have triggered a dramatic change in the overall energy systems [17]. Where formerly such systems were governed by the principle of "separation of concerns", they now exhibit an interwoven system structure. A dramatic growth in the number of independently operating power plants belonging to a variety of operational authorities is accompanied by a direct (i.e., large-scale European) and indirect coupling via the previously unconsidered communication network [18]. Specific challenges are raised by re-charging electrical vehicles (simultaneous charging) or the potential effects of price-based incentives to change consumption policies. Besides these apparent impacts, the coupling of formerly independent systems becomes even more obvious when considering the ongoing integration process of different energy carriers: One prominent prediction is that the next severe power grid outage will be caused by the gas network, see e.g. [19]). This example shows the types of challenges we face with an uncontrolled (or poorly controlled) integration across systems with behaviour that is largely unpredictable.

Another case where a formerly isolated system attains a more coupled and complex structure is the railway system. Until about two decades ago, the German rail system – the same holds for most of the other national railway systems in Europe as well – was run as a public institution operating all trains and the rail network itself. This changed due to the European Union's demand for deregulation. As a first step, the formerly integrated national railway operator, the *'incumbent'*, was split into an infrastructure manager (IM) and railway undertakings (RU). Several new RUs emerged that are responsible for freight services, long-distance passenger services, and local passenger transport. In contrast, the IM is in charge of network operation and maintaining the network. Cooperation works on the basis that RUs are operating trains, for which they rent routes or slots from the IM. In general, deregulation and a market driven by heterogeneous participants is not necessarily problematic. However, in the context of the German railway system, there have been new difficulties in correctly integrating these newly formed subsystems. For instance, the underlying dependencies and coupling effects that arise when regional operation of local passenger connections are offered to RUs through public bid invitations. Here, the RU winning the auction is normally founding a new subsidiary which is then becoming a new *independent element* that has to be integrated into the rail system – with a duration of typically 10 to 15 years. Such a new element has its own operations department, which again has to be integrated in the overall operations system. The dynamics of this change in structure lead to control and communication problems for the shared resource railway track, with non-conforming interfaces and processes worsening the impact.

Similar challenges are found in various application domains, where the impact of mutual influences and dependencies among distributed entities is continuously increasing. For instance, current highly sophisticated traffic guidance in road networks for vehicles (i.e., cars or lorries) provide up-to-date information obtained by, e.g., the infrastructure. Either the driver or an automated system acting on her behalf adapts the route selection behaviour and consequently impacts the traffic situation. This behaviour in turn influences the control strategy of the infrastructure (e.g., traffic light control or road management as a reaction to the traffic conditions) [20]. These mutual influences between infrastructure and driving behaviour had not been that severe when announcements or the radio's *Traffic Message Channel* were the only publicly available instance of traffic status information.

Another domain to consider is the smart household. Here, previously independent systems (such as TVs, fridges, or shutters) are combined into an integrated system that adjusts the behaviour of its elements according to the (estimated) needs of the users. The resulting solution consists of varying devices of different manufacturers – those that were explicitly made for smart environments are combined with legacy devices or other systems of limited functionality. In addition, devices typically rely on a variety of communication interfaces [2].

New challenges are brought to the fore when we extend the 'system' boundaries by taking into account human-computer interaction; collaboration tools utilised in software development processes further contribute to mutual influence effects. These tools help to coordinate the work of a potentially large group of developers in a highly distributed manner. Such a tool is used to allow for interaction and cooperation between software developers. We can observe that interaction through such a tool also influences the social layer, i.e., cooperation and concurrency in the technical world may entail similar social relationships. The reverse influence has been shown to be simultaneously present, while happening on a different timescale (i.e., from the social to the technical layer): A result of this influence is an increasing adoption of systems that are socially aware in the sense that they monitor structure and dynamics of social organisations and adapt their behaviour accordingly in real-time. Given the existence of these effects, we face hidden influences through existing social relations that depress (or support) the success of a project although the management cycle may not even be aware of these influences.

What we can observe from the above examples is: Systems increasingly consist of *heterogeneous, often geographically distributed subsystems* with a high degree of *independence and autonomy*. More precisely, this means that *subsystems exist and are operated independently from others* although, in real operation, they *mutually influence each other*. These subsystems *may share some goals*, but they *may also have opposing goals*. Interaction of subsystems takes place on different scales and in multiple direct and indirect ways, which may be purely technical, but may also involve humans, their decisions, and communication. Decisions of the subsystems, which often must be taken under *more or less strict real-time constraints*, are based on *uncertain information* (e.g., incomplete or imprecise information) and thus, the behaviour of the overall system can often not be fully anticipated. In this sense its *behaviour is emergent*. Moreover, *structure,*

organisation, and functionality of subsystems and the overall system evolve continuously making the guarantee of global functionality a moving target.

III. Interwoven Systems

Within the previous section, we identified a set of formidable challenges in the control and operation of technical systems. In this section, we develop a definition of the corresponding system class, to which we will refer to as IwS in the following.

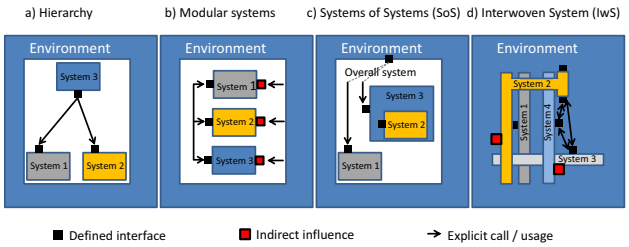

Fig. 1. From isolated systems towards Interwoven Systems.

The definition of an Interwoven System (IwS) [10] is driven by the fact that not all interfaces between system elements[6] are defined explicitly at design-time, because they do not only interact through conventional ICT interfaces, but also through the physical world including humans (cf. Figure 1 d). Therefore, the term *interaction* in this definition refers to these various kinds of interdependencies. We emphasise that system interweaving is already a fact and propose to guide this process explicitly at runtime by providing interweaving capabilities that allow for a self-improving self-integration throughout all phases of the design and maintenance cycle.

Definition: IwS are systems where as many elements (subsystems) as possible are equipped with interweaving capabilities[7] In general, the overall system has the following properties: (1) *mutual influences* (i.e., there exist a number of interactions not defined at design-time that shall be made explicit at runtime), (2) *heterogeneity of system elements* (i.e. system elements may belong to different application domains or authorities), (3) *uncertainty*[8] (i.e., the predictability of the entire system and its parts is usually incomplete and inaccurate), and (4) soft or hard real-time aspects (i.e., elements must operate complying to deadlines[9]). The interweaving capabilities of an element are based on techniques for (i) online dependency detection and modelling to reveal the various kinds of interactions, (ii) online goal adaptation (e.g., goal negotiation or re-weighting of subgoals), (iii) continuous redesign (e.g., parameters or internal structure of an element), and (iv) long-term self-improvement.

Note that IwS are not necessarily complex by themselves. Instead, simple instances might exist that can be handled appropriately with current methods – due to their understandable and reasonable size and structure. However, combining IwS elements of unknown structure and complexity quickly results in barely manageable systems. Therefore, the challenge we are facing in the context of IwS is to provide methods for improving the manageability of complex IwS.

Introducing the term IwS is not meant to define a novel class of systems; it is used to identify a dramatically increasing set of systems that have the properties that we have defined above and to focus the research community's attention on the development of strategies for meeting the challenges we have noted with respect to their controllability and manageability. Additionally, although IwS have much in common with the challenges of managing and maintaining relatively open Systems-of-Systems (SoS) [22] (see Figure 1c)), they also differ in several important ways: first, in many SoS, there is a well-specified purpose or set of functions for the integration of the separate systems, whereas IwS can be the result of temporary conditions (an emergency) or occur almost accidentally over time, as the result of individual interfacing of components across the systems; second, unlike most SoSs, there may be no system components that are knowledgeable in depth about other systems' components and there may be no decision-making authority over all of the systems in the SoS (cf. [23]).

We propose to guide this process by allowing for active interweaving, controlled and negotiated by participating systems at runtime. We outline an architectural concept following the ideas of Organic Computing in the next section.

IV. An Organic Computing Perspective on Runtime System Interweaving

The Organic Computing (OC) initiative [12] (and others such as Autonomic Computing [24]) postulate that the increasing complexity in technical systems has to be mastered by means of self-organisation. The basic idea is to equip technical systems with autonomous and self-organising or *'life-like'* properties, enabling adaptive and self-improving behaviour that results in desired characteristics such as robustness, flexibility, and resilience.

From a technical point of view, this means moving traditional design-time decisions to runtime and to the responsibility of the systems themselves. One early OC approach for doing this was to introduce the Observer/Controller (O/C) design pattern [25] that establishes a runtime feedback loop on top of a productive system (the *System under Observation and Control* – SuOC). This O/C unit monitors the conditions and performance of the SuOC and intervenes if necessary to maintain a goal-compliant and self-optimising behaviour (the

[6]Instead of subsystem we use the term system element, because there is not necessarily a strict hierarchy in the system.

[7]As we cannot expect that all elements are accessible to equip them with interweaving capabilities, the goal is to reduce failures in operation or even the chance of catastrophic outages as far as possible and to reduce costs in terms of system development and operation.

[8]The term *uncertainty* is used according to [21]. There, "uncertain" is a generic term for other terms such as "likely", "doubtful", "plausible", "reliable", "imprecise", "inconsistent", or "vague".

[9]These deadlines are not always in the order of, for example, milliseconds. They may also be in larger magnitudes, such as minutes, hours, days, or even years (as in maintainability and evolvability, which can pose different problems to large IwS.

concept is similar to the Monitor-Analyse-Plan-Execute cycle – MAPE – known from Autonomic Computing [24]). In the following paragraphs, we extend and augment this basic design pattern with components necessary for an interweaving process active at runtime.

The goal of the control loop presented here is to equip the SuOC with interweaving capabilities, meaning to analyse and self-improve its integration status. From a technical perspective, integration is considered from both an engineering and an ICT perspective. In engineering, the notion of integration describes a process in which several component subsystems are brought together and merged towards one unified system. Thereby, the focus is set on achieving a correctly working unit, meaning that the subsystems work together and function correctly, see [26]. In ICT, the focus is shifted towards different computation and processing units. Here, system integration is defined as a process of linking a potentially large set of heterogeneous computing systems and software applications. The linking itself is done physically or functionally. The process is finished as soon as all contained elements (i.e., software and hardware) act together as a coordinated whole, see e.g. [27]. Parts of this linking rely on the availability and correct functioning of a communication medium, typically realised on basis of network connections such as the Internet. In both of these perspectives, integration is understood as a clearly defined process with fixed starting and end points. In "integration science" [28], which grew from work on space systems and focuses on the integration in very large, complex heterogeneous systems, integration is considered to be a set of diverse and continual processes across many levels of the complex system or enterprise; this led to integration approaches that emphasised continual re-evaluation, self-monitoring, and self-testing as the system adds new capabilities and new operational environments.

Finally, the system is autonomous, meaning the O/C unit belonging to the system itself is responsible for maintaining the integration status. We refine this control loop with its functionalities in the following ways.

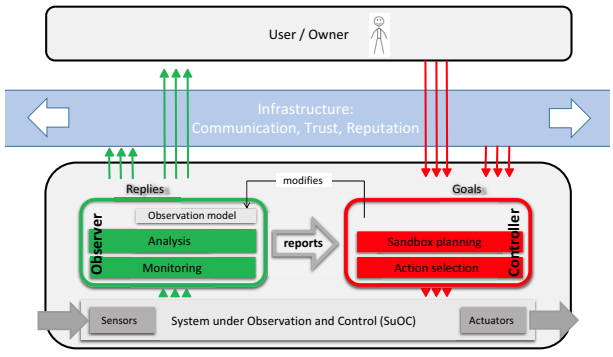

Fig. 2. Framework for Self-integration: View on a single element.

Figure 2 illustrates the architectural concept for an individual entity with interweaving capabilities. Following the O/C pattern, we distinguish between three components: the SuOC, an observer, and a controller. In addition, we introduce a distributed middleware solution that provides basic services including communication mechanisms, neighbour discovery, security, and trust.

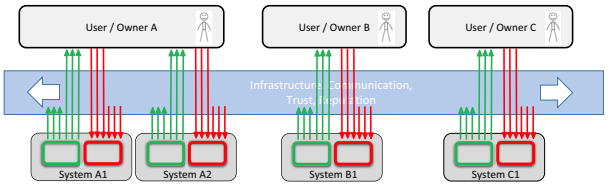

Fig. 3. Framework for Self-integration: View on a set of elements connected via the middleware.

In general, we assume that a potentially large set of heterogeneous, autonomous entities participate in an open system structure. For instance, this means that various entities interact with each other, although they might belong to different authorities or users (cf. Figure 3). In the following paragraphs, we provide more details on the components middleware, observer, and controller. However, we neglect the SuOC since we do not interfere with the internal logic of any individual system/component and only assume the existence of interfaces to observe and manipulate it (i.e. configuration parameters).

A. Observer Component

Figure 4 illustrates the schematic architecture of the observer component. It highlights the contained components and shows the connections between them. In the following, these components and their tasks are briefly introduced.

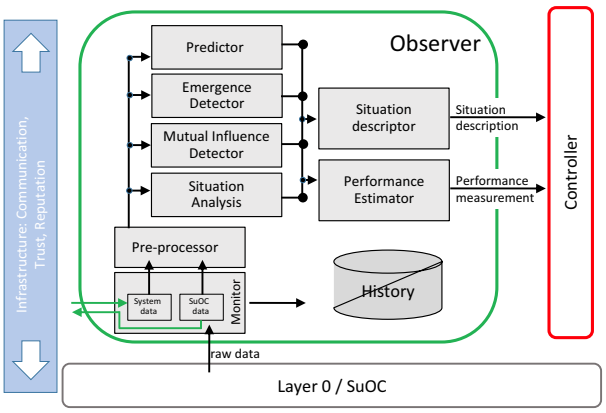

Fig. 4. Detailed view on the observer component.

Monitoring: The internal status of the system and the external conditions (i.e., the environment, the status of cooperation partners, and further influencing factors) are monitored and aggregated towards an appropriate situation description (which serves as the basis for the controller's decision process). Therefore, access to the local status variables of the SuOC is needed (in case of a SoS structure of the SuOC, this holds

for all contained SuOCs). In addition, the current local sensor readings must be available. This is accompanied by access to the information basis provided by the middleware, i.e., neighbour discovery or trust management. The corresponding values have to be monitored frequently and in response to the decision interval of the controller [29], [25].

Observation model: Based on the ability to monitor status variables, the observer has to decide which of them are needed when building a situation description. This is accompanied by a required specification of the values' resolution (i.e. the accuracy of the particular values). Therefore, the observer possesses an observation model that is configured by the controller according to the current needs. A dynamic adaptation may continue work presented in [30].

Preprocessing: Most of the values from the monitoring process are based on sensor data (although there may also be data archives for longer term reasoning and modelling). Thus, they may be noisy, incomplete or subject to disturbances (i.e., malfunctions of sensors), for instance. As a result, the perceived data is characterised by different degrees of unreliability. Some of these effects can be countered by preprocessing the data, e.g. filtering outlier values, estimating missing values, calculating sliding averages. Hence, the raw data is processed towards more consistent values – which can be done using standard techniques such as Exponential Smoothing or Kalman Filters (see e.g. [31] for an overview).

Prediction: Besides smoothing and preprocessing, a further data-oriented task is to provide a sophisticated estimation of how the values will develop in the near future. More precisely, forecasts and predictions may be derived for time-series data (or in some cases, model-driven simulation data). Here, a variety of standard techniques is available. Recently, ensemble concepts that combine several approaches at runtime and learn the most promising combination strategy have been proposed and can be used for this task [32].

Detection of mutual influences: Already in the examples discussed in Section II, the existence of other systems with direct and indirect influence on the system's status and performance has been identified as core challenge for mastering IwS. Based on knowledge about available neighbours (i.e. those systems that might have a certain influence or impact on the system of interest), techniques to find correlations in behaviour and mechanisms to detect hidden effects among these systems and their behaviours are utilised to identify such influences. First attempts to develop techniques that are able to fulfil this task have been presented in [33] and can serve as basis for more sophisticated solutions.

Emergence detection: As outlined in [15], systems that are based on self-organised behaviour may entail emergent effects. Since self-integration also controls the structural composition of the overall system organisation, self-organisation appears and emergence has to be considered. For this purpose, existing work from the OC domain can be utilised, see [15], [16]. These techniques are applicable to detect emergent behaviour in self-organised systems – based on such a detection, adaptive reactions need to be brought into operation to suppress negative and support positive emergent effects.

Logging and data analysis: In order to allow users to comprehend and reconstruct the system's behaviour, a history of past situations and the corresponding actions is needed. In line with the other methods discussed here, learning algorithms and more sophisticated data mining methods could be used here to identify new unexpected patterns (such as with manifold discovery methods) or to generate new rules (such as with grammatical inference methods) for reasoning and planning programs to utilise in determining next actions and adaptations by the system.

B. Controller Component

Figure 5 illustrates the schematic architecture of the controller component. Technically, the control process is triggered by the observer when providing the two input values: the situation description and the performance measurement. The former is typically realised as a data structure encapsulating a set of values (i.e., a vector) – this is passed to the three major control components: the sandbox rule generation, the behaviour control, and the structure control units. Thereby, the situation description serves as a basis for the corresponding decision process as well as necessary modifications of the utilised models (i.e., rule-base for SuOC adaptation, environment model, and entity model) according to correctness and completeness. The latter input (the performance measure) is utilised for adapting the models with respect to accuracy – it is taken into account when evaluating the past action(s) or delayed.

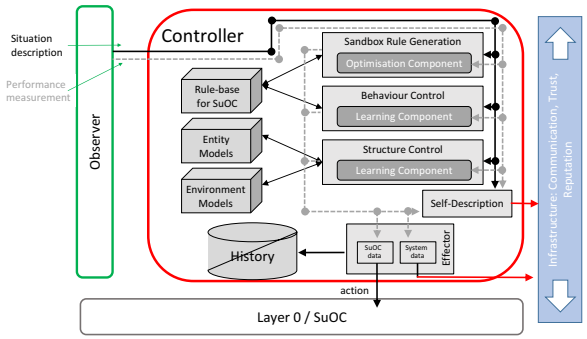

Fig. 5. Detailed view on the controller component. The notation of relations between components (i.e. highlighting the consecutive processing of inputs) distinguishes between grey and black as well as solid and dashed lines. This has been chosen to allow for better readability of the figure and the intention to code a certain semantic meaning.

The first model base, the *rule-base*, provides a mapping of situations to possible actions used to influence the SuOC. Additionally, it contains evaluation criteria that estimate the quality of this mapping. The second model base, the entity models, contains knowledge about possible cooperation partners. Most importantly, this incorporates predictions about the expected behaviour of the entity. Such a prediction is necessary to (a) select the most appropriate cooperation partner, and (b) consider others' behaviour for the own strategy. The third

model base, the environment model, contains all knowledge that is neither related to its own status (including control mechanism and SuOC), nor known to other entities. In addition to these control elements and models, the controller maintains a model of itself that allows for understanding the behaviour and determine the capabilities – this is found in the self-description component. The information provided by this component is used by the other entities when discovering and selecting cooperation partners, and by the user to track the system's behaviour. In the following, components contained in Figure 5 and their tasks are briefly introduced.

Model building and maintenance: As a fundamental part of the controller component, models are needed that encapsulate the estimated knowledge about successful strategies, expected outcome for certain actions, and predicted behaviour of others. In order to establish such models in the first place, design-time models can be used as a basis (see, e.g., the domain of Models@Runtime [34]), since a variety of design and development processes makes use of model-driven engineering in terms of behaviour modelling [35]. In addition, these models have to be updated according to observed conditions – with the goal to have as appropriate descriptions as possible that can be used to derive highly accurate predictions. Therefore, machine learning techniques can be utilised as well as standard statistics. These can be augmented by trust and reliability measures provided by the observer.

Behaviour control: Based on the models that reflect their own behaviour and its success, the first control loop of the controller decides about necessary adaptations of the SuOC's behaviour, i.e., the productive part of the system. Therefore, the control interfaces access parameter settings and choose techniques (algorithms) to be applied. Such a decision can be realised using pre-defined logic (if the situation description conforms to a given class, a certain action is applied) or enhanced with learning capabilities. In OC systems, typically rule-based reinforcement learning techniques are applied to cover this task [25], [36]. This concept can be reused here.

Structure control: The relations to other systems depend on varying influence factors, e.g., the degree to which a specific input is needed (which might be only available from a limited number of other entities), the reliability of an entity based on own experiences, the reputation of an entity based on others' ratings, its location, its availability, and so on. Furthermore, the detection of influences of another element's actions on the system itself turns an indirect relation into a direct one and consequently might allow for changing the impact. The intended modification of relationships and the selection of cooperation partners based on the aforementioned attributes allows for a continuous control process. Depending on the currently active goal function, the structural integration status is evaluated continuously. Therefore, metrics such as robustness [37], degree of self-organisation [38], or trustworthiness [39] can be utilised.

Runtime learning: In order to improve the decisions over time (i.e. behaviour and structure control), a feedback learning mechanism is utilised. The current performance as quantified by the observer in relation to the current goal is taken into account and used as reward in reinforcement learning [40]. To allow for user-understandable behaviour (i.e., traceable and human-repeatable decisions), rule-based systems that make use of a condition-to-action mapping are utilised [25]. This can be combined with sandbox learning concepts similar to the 'anytime learning' approach presented by Grefenstette and Ramsey [41].

Observation model adaptation: The observation model within the observer component defines which attributes are currently monitored, which data is gathered, and in what resolution. This model is dynamically adapted at runtime by the controller part. The idea here is that in some cases a dynamic adaptation may reduce effort since it allows for switching between abstract (or high-level; reduced number of observations, reduced frequency, reduced resolution, and consequently reduced storage and computation demands) to more precise low-level descriptions. Therefore, concepts such as the OC-based ideas presented in [30] can be used.

Self-description: As a basis for the system discovery service as provided by the middleware (and possibly the model building taking place in each entity), each system has to provide a self-description of capabilities and actual status. This is closely related to how the middleware solution provides access to the information and which protocol is used. Reflection capabilities, as seen in [42] can be used to support this.

C. Distributed Middleware

The third fundamental part of the framework is a middleware solution that allows for several distributed tasks. From an architectural point of view, it encapsulates the communication infrastructure and provides well-designed interfaces to all participating entities. In the following paragraphs, the main tasks are outlined and possible approaches to implement these tasks are briefly highlighted.

Communication platform: As a fundamental basis for the cooperation of distributed entities, data communication mechanisms are required. The rise of mobile and stationary communication, especially using the Internet, made distributed and remote data ubiquitously available. In addition, current trends such as the Internet-of-Things [43] showed that a large part of formerly isolated systems are coupled together using standard communication technology. In this work, we assume availability of a communication medium, such as cable or WiFi, as well as the corresponding data communication protocols, such as the ubiquitous TCP/IP standards. We further assume that encryption technology is in place allowing for identification and authorisation of elements.

Trust Management: Trust and reliability information are perceived based on one's experiences. In addition, distributed entities need to exchange information to be able to deal with unknown elements or those with limited experiences. Therefore, computational trust [44] and reliability estimations need to be established in a distributed manner, potentially augmented with a reputation system. Several concepts are

available in literature that can be utilised to cover this task, e.g., [45], [46].

Neighbour Discovery: One major problem when considering indirect influences and dependencies among distributed elements is the missing awareness of others' influence. A prerequisite to allow for such an awareness is initial information about which elements might potentially impact a system's own behaviour; accordingly, the system has to become, in the first place, of its neighbours[10] Hence, the middleware has to provide a neighbour discovery mechanism that updates a (local) neighbour cache. In order to allow for such a mechanism, existing techniques can be applied: Either simply by using broadcast procedures or by more advanced neighbour discovery routines from the data networking domain (especially from mobile ad-hoc networks, see [47]). More speculatively, new data mining techniques may enable one to gauge the most important variables impacting a complex system and to use those most 'important variables' as the defining metric for determining nearest neighbour given that specific operational context.

Capability description: In order to allow for an appropriate selection of interaction partners, an up-to-date description has to be provided as a basis for this decision process. Technically, solutions such as ontologies [48] from the semantic web domain [49] can be utilised: the current capabilities and further properties of a system can be expressed in such a way that all other elements that are familiar with the same terminology understand the information in an automated (i.e., machine-processable) way.

V. Conclusion

This paper has motivated the need for novel solutions for mastering Interwoven Systems (IwS) by developing active interweaving capabilities. We briefly discussed several specific examples where the complexity of interconnected, indirectly coupled systems led to outages and failures. Afterwards, we argued that the underlying trend towards coupling and mutual influences is expected to increase dramatically in the near future.

As a reaction to these challenges, we initially defined common characteristics of the underlying problem class, to which we refer as IwS. Based on these characteristics, we have developed an architectural concept for self-interweaving entities that follows the ideas of the Organic Computing Observer/Controller approach. This architectural concept distinguishes between the productive part, control mechanism, and distributed middleware. We have specified the necessary components and outlined how the desired functionality can be achieved. In future work, we will develop corresponding solutions and demonstrate the benefit within appropriate application scenarios.

[10]The term "neighbour" is used as representative for all other systems that either directly interact with the particular IwS element or indirectly influence it. This does not necessarily require a spatial or functional neighbourhood.

References

[1] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 66–75, September 1991.

[2] F. Allerding, B. Becker, and H. Schmeck, "Integration intelligenter Steuerungskomponenten in reale smart-home-Umgebungen," in *Informatik 2010: Service Science – neue Perspektiven für die Informatik, GI Jahrestagung, held September 2010 in Leipzig, Germany*, vol. 1, Bonn, Germany, 2010, pp. 455–460.

[3] E. Guizzo. (2011, October) How google's self-driving car works. IEEE. [Online]. Available: http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works(accessed2015-12-12)

[4] E. McKenna, I. Richardson, and M. Thomson, "Smart meter data: Balancing consumer privacy concerns with legitimate applications," *Energy Policy*, vol. 41, no. 0, pp. 807 – 814, 2012.

[5] S. Tomforde, I. Zgeras, J. Hähner, and C. Müller-Schloer, "Adaptive control of Wireless Sensor Networks," in *Proceedings of the 7th International Conference on Autonomic and Trusted Computing (ATC'10), held in Xi'an, China (October 26-29, 2010)*, 2010, pp. 77 – 91.

[6] J. Augusto, V. Callaghan, D. Cook, A. Kameas, and I. Satoh, "Intelligent environments: a manifesto," *Springer Open Access: Human-centric Computing and Information Sciences*, vol. 3, no. 12, 2013.

[7] H. Elzabadani, A. Helal, B. Abdulrazak, and E. Jansen, "Self-sensing Spaces: Smart Plugs For Smart Environments," in *From Smart Home to Smart Care*, S. Giroux and H. Pigot, Eds. IOS Press, 2005, pp. 91–98.

[8] P. McKinley, S. Sadjadi, E. Kasten, and B. Cheng, "Composing Adaptive Software," *IEEE Computer*, vol. 37, no. 7, pp. 56–64, Jul. 2004.

[9] C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds., *Organic Computing – A Paradigm Shift for Complex Systems*, ser. Autonomic Systems. Basel, CH: Birkhäuser Verlag, 2011.

[10] S. Tomforde, J. Hähner, H. Seebach, W. Reif, B. Sick, A. Wacker, and I. Scholtes, "Engineering and Mastering Interwoven Systems," in *ARCS 2014 - 27th International Conference on Architecture of Computing Systems, Workshop Proceedings, February 25-28, 2014, Luebeck, Germany, University of Luebeck, Institute of Computer Engineering*, 2014, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6775093

[11] K. L. Bellman, S. Tomforde, and R. P. Würtz, "Interwoven Systems: Self-Improving Systems Integration," in *Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014, London, UK, Sept. 8-12, 2014*, 2014, pp. 123–127.

[12] C. Müller-Schloer, "Organic Computing: On the feasibility of controlled emergence," in *2nd IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Sept. 8-10, 2004, Stockholm, Sweden*. ACM Press., 2004, pp. 2–5.

[13] R. L. Glass, *Facts and Fallacies of Software Engineering*, ser. Agile Software Development. Boston, US: Addison Wesley, 2002.

[14] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, vol. 38, no. 8, pp. 114–117, 1965.

[15] M. Mnif and C. Müller-Schloer, "Quantitative emergence," in *Proc. of the 2006 IEEE Mountain Workshop on Adaptive and Learning Systems, held 24 Jul - 26 Jul 2006, Utah State University College of Engineering Logan, USA*. IEEE, 2006, pp. 78–84.

[16] D. Fisch, M. Jänicke, B. Sick, and C. Müller-Schloer, "Quantitative Emergence – A Refined Approach Based on Divergence Measures," in *Proceedings of IEEE Conference on Self-Adaptive and Self-Organising Systems (SASO10), held in Budapest, Hungary, September 27-October 1, 2010*, Budapest, HU, 2010, pp. 94–103.

[17] J.-P. Steghöfer, G. Anders, F. Siefert, and W. Reif, "A System of Systems Approach to the Evolutionary Transformation of Power Management Systems," in *Informatik 2013, 43. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Informatik angepasst an Mensch, Organisation und Umwelt, 16.-20. September 2013, Koblenz*, ser. LNI, vol. 220, 2013, pp. 1500–1515.

[18] H.-J. Appelrath, H. Kagermann, and C. Mayer, "Future energy grid – migration to the internet of energy (acatech STUDY)," acatech, Tech. Rep., 2012.

[19] P. Rodriguez, "The - compromised? - future of natural gas (L'avenir - compromis? - du gaz naturel)," *Gaz d'Aujourd'hui (3)*, vol. 40, no. 30, pp. 30–31, 2009.

[20] H. Prothmann, S. Tomforde, J. Branke, J. Hähner, C. Müller-Schloer, and H. Schmeck, "Organic Traffic Control," in *Organic Computing – A*

Paradigm Shift for Complex Systems, ser. Autonomic Systems. Basel, CH: Birkhäuser Verlag, 2011, pp. 431–446.

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

[22] M. W. Maier, "Architecting principles for systems-of-systems," *Systems Engineering*, vol. 1, no. 4, pp. 267–284, 1998.

[23] A. P. Sage and C. Cuppan, "On the Systems Engineering and Management of Systems of Systems and Federations of Systems," *Information, Knowledge, Systems Management*, vol. 2, no. 4, pp. 325–345, 2001.

[24] J. Kephart and D. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[25] S. Tomforde, H. Prothmann, J. Branke, J. Hähner, M. Mnif, C. Müller-Schloer, U. Richter, and H. Schmeck, "Observation and Control of Organic Systems," in *Organic Computing – A Paradigm Shift for Complex Systems*, ser. Autonomic Systems, C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds. Birkhäuser Verlag, 2011, pp. 325–338.

[26] H. T. Gilkey, "New Air Heating Methods," in *New methods of heating buildings: a research correlation conference conducted by the Building Research Institute, Division of Engineering and Industrial Research, as one of the programs of the BRI fall conferences, November 1959*. Washigton D.C., USA: National Research Council / Building Research Institute, 1960, pp. 47–56, oCLC 184031.

[27] W. Hasselbring, "Information System Integration," *Commun. ACM*, vol. 43, no. 6, pp. 32–38, Jun. 2000.

[28] K. L. Bellman and C. Landauer, "Integration science: more than putting pieces together," *Aerospace Conference Proceedings*, vol. 4, no. 1, p. 397 409, 2000.

[29] S. Tomforde, E. Cakar, and J. Hähner, "Dynamic Control of Network Protocols - A new vision for future self-organised networks," in *Proceedings of the 6th International Conference on Informatics in Control, Automation, and Robotics (ICINCO'09), held in Milan, Italy (2 - 5 July, 2009)*, J. Filipe, J. A. Cetto, and J.-L. Ferrier, Eds. Milan: INSTICC, July 2009, pp. 285 – 290.

[30] Y. Bernard, *Trust-aware Agents for Self-organising Computing Systems*. Herzogenrath, Germany: Shaker Verlag, 2014.

[31] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester, *A Modern Introduction to Probability and Statistics: Understanding Why and How*, ser. Springer Texts in Statistics. London, UK: Springer Verlag, 2007.

[32] S. Tomforde, M. Sommer, and J. Hähner, "Learning to Predict: Automated Management and Correction of Prediction Techniques for Traffic Flows within a Self-organised Traffic Control System," in *Proc. of 11th International Congress on Advances in Civil Engineering (ACE14), held 21–25 October, 2014 in Istanbul, Turkey*, 2014, pp. 1–6.

[33] S. Rudolph, S. Tomforde, B. Sick, and J. Hähner, "A Mutual Influence Detection Algorithm for Systems with Local Performance Measurement," in *Proceedings of the 9th IEEE International Conference on Self-adapting and Self-organising Systems (SASO15), held September 21st to September 25th in Boston, USA*, 2015, pp. 144–150.

[34] U. Assmann, N. Bencomo, B. Cheng, and R. France, "Models@run.time," *Dagstuhl Reports*, vol. 1, no. 11, pp. 91–123, 2011.

[35] H. J. Goldsby and B. Cheng, "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*, ser. LNCS 5301. Springer, 2008, pp. 568–583.

[36] S. Tomforde, A. Brameshuber, J. Hähner, and C. Müller-Schloer, "Restricted On-line Learning in Real-world Systems," in *Proc. of the IEEE Congress on Evolutionary Computation (CEC11), held 05 Jun - 08 Jun 2011 in New Orleans, USA*. IEEE, 2011, pp. 1628 – 1635.

[37] H. Schmeck, C. Müller-Schloer, E. Çakar, M. Mnif, and U. Richter, "Adaptivity and Self-organisation in Organic Computing Systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 5, no. 3, pp. 1–32, 2010.

[38] J. Kantert, S. Tomforde, and C. Müller-Schloer, "Measuring Self-Organisation in Distributed Systems by External Observation," in *Proc. of the 28th GI/ITG International Conference on Architecture of Computing Systems – ARCS Workshops, held 24 - 27 March 2015 in Porto, Portugal, Workshop on Self-Optimisation in Organic and Autonomic Computing Systems (SAOS15)*. VDE, 2015, pp. 1–8.

[39] R. Kiefhaber, R. Jahr, N. Msadek, and T. Ungerer, "Ranking of Direct Trust, Confidence, and Reputation in an Abstract System with Unreliable Components," in *Proc. of ATC13, held in Vietri sul Mare, Sorrento Peninsula, Italy, December 18-21, 2013*, 2013, pp. 388–395.

[40] T. M. Mitchell, *Machine Learning*, ser. Computer Science Series. Singapure: McGraw-Hill Companies, Inc., 1997.

[41] J. Grefenstette and C. L. Ramsey, "An Approach to Anytime Learning," in *Proc. of 9th Int. Works. on Machine Learning*, 1992, pp. 189–195.

[42] C. Landauer and K. L. Bellman, "Self-Modeling Systems," in *Self-Adaptive Software*, ser. LNCS. Srpringer, 2002, vol. 2614, pp. 238–256.

[43] S. Greengard, *The Internet of Things*, ser. MIT Press Essential Knowledge. MIT Press, 2015, iSBN-13: 978-0262527736.

[44] C. Castelfranchi and R. Falcone, *Trust Theory: A Socio-Cognitive and Computational Model*. John Wiley & Sons, 2010, vol. 18.

[45] J. Kantert, H. Scharf, S. Edenhofer, S. Tomforde, J. Hähner, and C. Müller-Schloer, "A Graph Analysis Approach to Detect Attacks in Trusted Desktop Grids at Runtime," in *Proceedings of SASO 2014 (IEEE International Conferences on Self-Adaptive and Self-Organizing Systems), 08. - 12. September 2014, London, U.K.*, 2014, pp. 80 – 89.

[46] J. Kantert, S. Edenhofer, S. Tomforde, and C. Müller-Schloer, "Representation of trust and reputation in self-managed computing systems," in *13th IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC 2015, Liverpool, UK, October 26-28, 2015*, 2015, pp. 1827–1834.

[47] J. Loo, J. L. Mauri, and J. H. Ortiz, *Mobile Ad Hoc Networks: Current Status and Future Trends*. CRC Press, 2011.

[48] T. Gruber, "A translation approach to portable ontologies," *Knowledge Aquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[49] D. Oberle, N. Guarino, and S. Staab, "What is an ontology?" in *Handbook on Ontologies*, ser. International Handbooks on Information Systems, S. Staab and R. Studer, Eds. Springer, 2009, pp. 1–17.