Self-Adaptation Techniques for Complex Service-oriented Systems

Schahram Dustdar, Karl M. Goeschka, Hong-Linh Truong, and Uwe Zdun
Distributed Systems Group
Vienna University of Technology
Vienna, Austria
{*dustdar,goeschka,truong,zdun*}@*infosys.tuwien.ac.at*

***Abstract*—Complex service-oriented systems comprise humans and software services spanning multiple organizations. These systems are highly dynamic, because services, processes, and teams are not only diverse but they also constantly evolve. Therefore, these systems introduce a variety of challenges on how to adapt services, processes, and teams to changing situations. We contribute with our approach to address these challenges, comprising model-driven compliance support, run-time interaction mining, run-time management of requirements, and an explicit control-loop architecture. We conclude with remaining issues the software and service engineering research community should address.**

***Keywords*-self-adaptation; service-oriented architectures; interaction-mining; model-driven compliance;**

I. Introduction

By utilizing service computing principles, heterogeneous applications and services provided by diverse organizations can be composed into loosely coupled service-oriented systems. In many such systems, humans are involved, either acting as participants of the systems or utilizing the components of the systems. Such systems are highly dynamic and they do evolve. Services, processes, humans, and teams of humans often have to adapt to constantly changing requirements and environments. Therefore, such systems have high demands with regard to designing and adapting services and processes to changing situations.

In this paper, we consider complex service-oriented systems comprising humans and software services and spanning multiple organizations. Examples are networked enterprise collaboration, e-science, and disaster management systems. Humans not only use services but can also provide services (so-called human-provided services, HPS). Software services and humans consequently establish resources which can be accessed through well-defined interfaces. Various types of processes can be built by utilizing such resources while processes are constructed to fulfill requirements from humans and teams of humans. Many research questions arise, such as (i) how to ensure compliance to changing requirements, (ii) how to manage changing requirements for achieving system assurance, and (iii) how to adapt resources suitably to the evolution of processes and teams.

Clearly, we cannot solely rely on modeling or runtime techniques. Rather, our approach aims at combining both, modeling and runtime management and adaptation, to build such evolvable systems. First, model-driven development (MDD) techniques are adopted and adapted to support the modeling and design of compliant Web services and processes at design time. Next, a Web service information model is developed to provide a holistic view of requirements associated with services. Service and process models and requirements from the MDD process are captured and stored in the service information model. This information model also includes runtime service information which is captured using runtime monitoring of services. All design-time and runtime requirements are managed by a Web Service Evolution Management Framework (SEMF). Then, based on service requirements managed by SEMF, explicit feedback-control techniques are used to perform adaptation strategies.

This paper analyzes research challenges for complex service-oriented systems and describes our solution approach from various research and industry projects. We discuss our experiences and achievements so far as well as open challenges that need to be addressed. The rest of this paper is organized as follows: Section II discusses the related work. Section III presents our generic application scenario. The most important research challenges are presented in Section IV. We describe our our techniques for self-adaptive, complex, and service-oriented systems in Section V. Section VI summarizes the paper and outlines future work.

II. Related Work

Self-adaptation techniques in the context of service-oriented systems have been addressed in many research efforts. While we focus only on a few representative approaches in this section, further related work is mentioned in the following sections, where appropriate.

Autonomic service-oriented systems are typically realized using monitoring information to adapt service composition and execution. For example, Reich et al. [1] present an autonomic peer-to-peer system of stateful service containers to self-adapt the service execution. Their adaptation techniques rely on performance information, SLA management, and service migration. This approach is related to our work in using control loops and Web services information. However, they typically rely only on performance measures and Web service interface information. They do not support the full life-cycle of developing complex, service-oriented systems.

MDD can be applied to model services and to associate information required for service adaptation and configuration with the services model. For example, Gronmo et al. present a model-driven methodology for specifying QoS requirements together with service models [2]. They utilize UML activity models and a UML profile for QoS modeling, and, based on QoS requirements, QoS-optimized service compositions can be built. Whereas our work on MDD focuses on the whole SOA stack and lifecycle, this work focuses only on one specific SOA requirement, the QoS specification.

Self-adaptation techniques can also be supported by combining design-time and runtime approaches to address the full life-cycle of developing and executing adaptive, service-oriented systems. Only a few frameworks support adaptive, service-oriented systems at both design-time and runtime. For example, the PAWS [3] addresses the adaptation of BPEL processes. Runtime adaptation relies on information specified at the design time and using QoS metrics. PAWS, however, does not apply MDD techniques and focuses only on BPEL processes.

Furthermore, the above-mentioned approaches do not aim at supporting complex service-oriented systems in which humans, services, and teams constitute the system. Their either consider only software services or processes and workflows. Furthermore, compared to our framework, they lack a rich support of capturing and managing design-time and runtime information associated with services in an integrated way.

Part of our research on service management requirements is presented in [4]. While existing approaches such as such as WSOL [5], SLA [6], [7], [8], and licensing information [9], address just a small fraction of the requirements for complex service-oriented systems, we have developed a model that integrates various requirements from different perspectives. Our work is based on the assumption that there are diverse types of information that originate from various sources and different perspectives on Web services [10].

III. Application Scenarios and Model

We consider complex service-oriented systems comprising humans and software services, and spanning multiple organizations. Such systems are required for many purposes, such as enterprise collaborations within networked enterprises, e-science research between different research labs, and crisis management. Those systems are complex and dynamic, thus services and processes need to be prepared for change and evolution.

Figure 1 depicts a sketch of our model of a complex SOA-based system which includes resources as services, dynamic processes, and teams. Teams conduct their collaborative works by using services and processes. Processes, which can

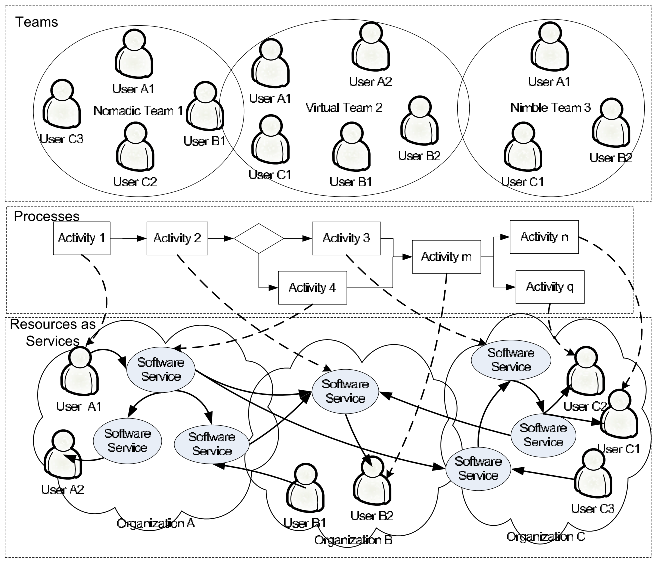

Figure 1. The complex service-oriented system including resources as services, dynamic processes, and teams

be well-defined or established on demand, include several activities which are mapped to resources. Resources can be humans or software services. They can be monitored and accessed via well-defined interfaces. In case of software services, a resource is a Web service, while a human as a resource is integrated into the system by different means, such as human-provided services (HPS) [11] or BPEL4People. Resources, typically, do not belong to a single organization, rather they span various organizations.

In such systems, services, processes, and teams change and evolve continuously. A number of research challenges that arose from analyzing such systems are described in the following section.

IV. Research Challenges

This section presents the major research challenges addressed in our approach.

1) Challenge 1: Compliance in self-adaptive SOAs.: Compliance to specific design rules is an omnipresent concern for software design in general, even more so for self-adaptive systems, as it is important to check that all design rules still hold, once an adaptation has been performed. In many cases, the compliance can be checked statically, but some technical concerns, such as performance, dependability, and security concerns, can only be reasonably checked at runtime.

The traditional way to enforce compliance by informally documenting the design rules in requirements specifications and hard-coding these requirements in the source code has a number of serious disadvantages: (i) Some of the design rules are not even encoded at all and are just documented as practices that must be obeyed either during software development or the use of the software. (ii) The encoded design rules are scattered throughout the source code and there is no traceability between requirements, models, and code. Clearly, this impairs changeability and maintainability, as well as usability: Users cannot simply specify changes to design rules, but have to resort to programmers.

Even more important for adaptive systems is that there is no integrated support for specifying or validating design rules which affect *both* the design time and runtime of a system. Consequently, we propose to use the model-driven software development paradigm to enable developers to rapidly develop and then stably evolve and maintain a customized self-adaptive framework that complies to the design rules that were specified for it. This infrastructure is described in detail in Section V-A.

2) Challenge 2: Integrated run-time management of diverse and heterogeneous requirements.: As discussed in [12], services are bound to diverse types of requirements, and it is a challenge to identify and monitor requirements and their changes for self-adaptive systems in an integrated and evolvable way. As indicated in [10], service-related requirements can originate from different perspectives, such as service providers, service developers, service customers, and hosting environments. These requirements include information for the design, such as pre- and post-condition and service interface, but also for runtime (QoS), user feedback, and licensing.

In order to support explicit runtime management of requirements for complex service-oriented systems, we need to capture and manage the above-mentioned information. In SOA-based environments, there is no standard or well-agreed way yet to gather Web services-related information which reflects the requirements. Although there are different ways of capturing Web services-related information such as WSOL [5], SLA [6], [7], [8], and licensing information [9], they support only a fraction of the possible requirements.

The two main challenges for the management of requirements are (i) how to integrate diverse types of requirements (modeled by different models), and (ii) how to manage these requirements in an evolvable fashion during the whole lifetime of services and processes. We discuss our approach's contribution to address this challenge in Section V-B.

3) Challenge 3: Runtime SOA Assurance.: Assurance for complex service-oriented systems is far from trivial [12] as it includes monitoring the service status, checking it against evolving requirements and adapting the service accordingly. While existing research has been focused on defining and detecting metrics and patterns for services and processes [13], [14], [15], the runtime monitoring of the interaction between people and software services has not yet been well addressed. We show our contribution to address this challenge in Section V-C

4) Challenge 4: Making the Control Loop Explicit.: To address changes of context, system, or users' needs, as well as imprecise or contradictory requirements or emerging behavior, many researchers have suggested to borrow theories from control engineering and apply them to software-intensive systems. These so-called autonomous or self-adaptive systems have in common the need to address change and uncertainty during run-time, so they generally move into run-time what previously had to be performed during design-time. Four key activities are described in [16]: Collect, analyze, decide, and act. In this paper the focus is on the collection (which requires monitoring) and on the action (which requires a way to re-compose, re-order, or re-configure running software).

In particular we support the argument that the control loop, although implicitly contained in many existing software systems, has to be made explicit and become a first-class citizen of architecture, design, and infrastructure support [17]. Moreover, we argue that one explicit loop is not enough: Real systems need different nested loops in order to address different ways and paces of change. Section V-D details our solution approach to address this challenge.

V. Techniques for Self-adaptive systems

A. Model-driven Approach to Support Compliant Self-adaptive SOAs

Our framework uses the model-driven approach [18] to compose business processes and services – as a foundational layer, and has the main task to express and validate all design rules related to these processes and services. Design rules can be based on the process specifications, the service specifications, collaboration specifications, human task specifications, information/data specifications, etc., and all these specifications can use multiple specification types (such as BPEL process definitions or UML activity diagrams for the process specifications). We express each concern in its own model. This, however, imposes the challenge how to integrate the various models.

In our current prototype implemented using the openArchitectureWare (oAW) generator [19], we have solved this problem using a view-based approach (this is explained in detail in [20]). In our work, a view is a representation of a process from the perspective of related concerns. The view is specified using an adequate view model. Each view model is a (semi)-formalized representation of a particular business process concern. Therefore, the view model specifies entities and their relationships that can appear in the corresponding view.

In particular, there is a Core View from which each view model is derived. The main task of the Core View is to provide integration points for the various view models defined as part of our view-based modeling framework (VbMF), as well as extension points for enabling the extension with view for other concerns or more specific view models, such as those for specific technologies. Example foundational view models that we have derived from the Core specify the following Views: Collaboration, Information, Control-Flow, etc. In addition, to these central concerns, many other concerns can be defined. We focus in these additional views for instances on models expressing compliance concerns, such as compliance to business rules and regulations. That is, we define view models for compliance concerns such as service composition policies, service deployment policies, service sequencing or ordering policies, information sharing/exchange policies, security policies, QoS policies, business policies, jurisdictional policies, preference rules, intellectual property and licenses. Each of these view models is either extending the core model or one of the other view models.

In addition, we define also a second level of extension models from these foundational models: For specific technologies, such as BPEL and WSDL we provide extension models, which add details to the general models that are required to depict the specifics of these technologies. Hence, we can use the distinction of Core View, generic views, and extension views to depict different abstraction levels, such as business-level concerns and technical concerns.

The separation of view abstraction levels helps in enhancing the adaptability of the process-driven SOA models. For instance, the domain experts analyze and modify the abstract views to meet the requirements of adaptations. Then, these modifications can be transformed into code in executable languages. The technical experts work with platform-specific views to define necessary configurations such that the generated code can be deployed into the corresponding runtime (i.e., process engines and Web service frameworks). An overview of these concepts can be found in Figure 2.

From the view models, the SOA runtime is generated. Model validator components are used to validate the compliance concerns that can be checked statically. Support for eventing and validating compliance to design rules at runtime can be generated into the SOA runtime, too. To make this work, it necessarily needs to be supported by a runtime infrastructure components: Online and/or offline monitoring components must be introduced to monitor the events and trigger the validation of compliance to design rules upon certain events happening in the SOA. Also, it makes sense to introduce tools such as a dashboard to allow the human users to observe the system and react on problems and critical situations.

B. *Explicit Runtime Management of Requirements*

To manage explicit requirements during runtime, we have developed an integrated information model that is capable of capturing requirements described in heterogeneous representations. We have further developed a management framework called SEMF (Service Evolution Management Framework) [4] that monitors the captured requirements and provides them to any party interested.

Figure 3 describes our Web Service Information Model that links to various other models, each of which describes a different type of requirements. Currently, the model can associate with SLA, QoS, Pre-conditions, License, Interaction Patterns, Post-condition, Interface, Taxonomy, Folksonomy, and Documentation.

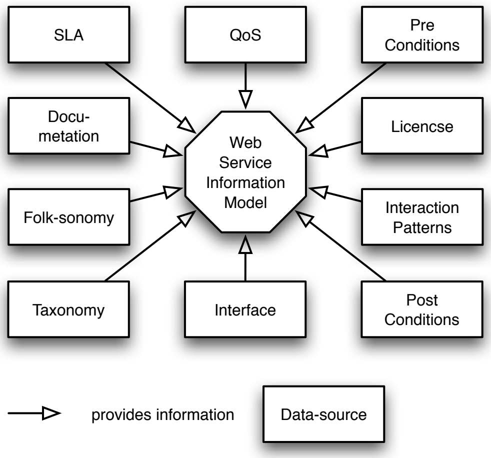

Figure 3. Data sources of the Web service information model[4]

The key feature of our model is that it can link to any type of requirements, described in any format, without any enforcement on data representation, e.g., in XML or an ontology. A type of requirements can further be described as metadata or as an external source of information. By utilizing this rich source of requirements information at runtime, services can publish their monitoring requirements as well as check whether their requirements are fulfilled. Services can also perform adaptation based on runtime manageable requirements.

C. *Runtime Interaction Mining*

To support the assurance for complex service-oriented systems, we have developed an online interaction mining framework which is capable of analyzing various metrics and interaction patterns [21]. We studied and categorized three kinds of interactions: service-to-service, human-to-service and human-to-human. For each kind of interaction, various metrics and interaction patterns are defined, providing a foundation for runtime analysis of the interactions. The metrics and interaction patterns are defined at three levels: *individual* (for individual human or service), *group* (a team or a set of services), and *collaboration* (all available services and humans within a collaboration).

In our prototype, several pattern specifications and templates are defined to specify the metrics and interaction patterns. Using event monitoring, we provided support for online monitoring of the metrics and interaction patterns. The metrics and interaction patterns detected can be queried

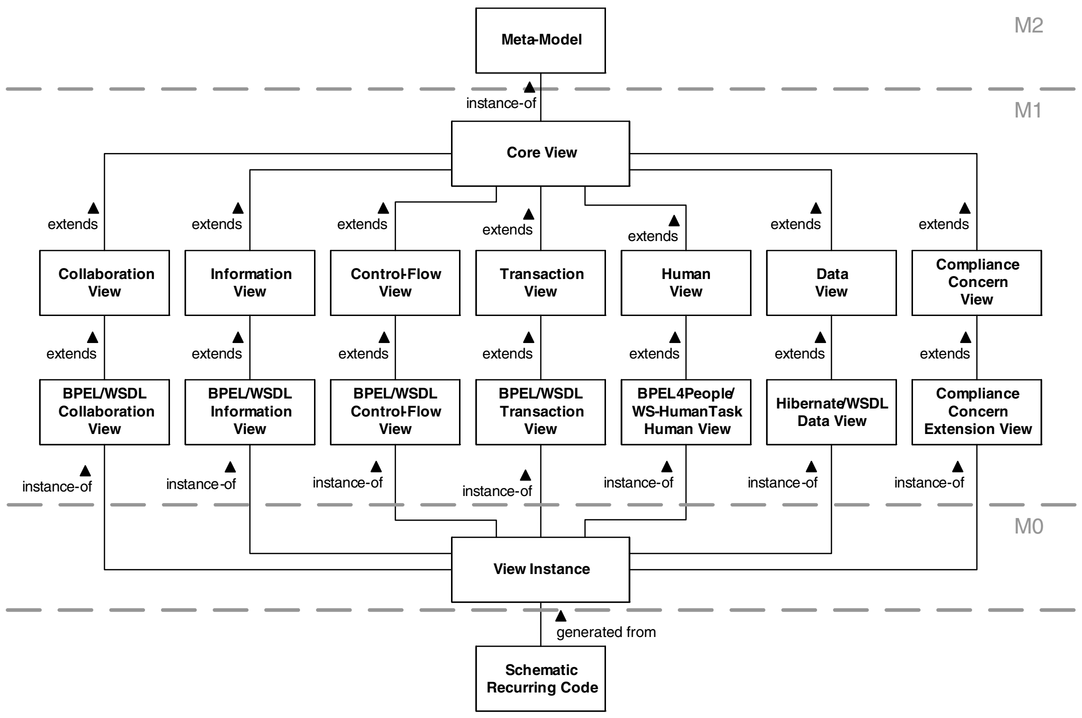

Figure 2. View-based modeling framework

at runtime. Hence, we provide valuable input for runtime compliance checking and system assurances. We also support the customized analysis of the monitored system events.

D. Nested control loop approach

In this section, we describe two nested loops, as an example to address the challenges of supporting explicit and configurable control loops.

1) Adaptive coupling: First, large and dynamic systems can benefit from short-term adaptivity to react to observed, or act upon expected (temporary) changes of the context/environment (e.g., resource variability or failure scenarios) or users' needs (e.g., day/night setting). As this kind of adaptivity should be provided without explicit user intervention, it is also termed autonomous behavior or self-properties, and often involves monitoring, diagnosis (analysis, interpretation), and reconfiguration (repair) [22].

One of the main reasons, why many approaches fell short in the past, lies in the major focus on the system's components (e.g., by focusing on recompilation, reconfiguration, and redeployment of components), while complexity theory [23] on the other hand clearly shows that the overall properties of large and complex software system are largely determined by the internal structure and interaction of its parts and less by the function of its individual components.

SOA already address this perception by putting a strong focus on structure completely separated from the implementation of individual constituents. Even more so, the overall system properties are determined not only by the structure but also by the strength of coupling of its relationships. Thus the inner control loop has to adaptively control the structure and strength of coupling between the system's constituents as the most promising approach to influence/control its overall properties and behavior.

To provide the desirable degree of adaptivity, competing properties of the overall system, such as dependability and security, have to be explicitly balanced according to the respective situation (context, failure scenarios, current user needs). This balancing should flexibly be performed via the interaction of infrastructure and application (or even the end user). It should be supported by an explicit control of adaptive coupling mechanisms between software services, for instance, through run-time selection and reconfiguration of dependability protocols, such as consistency of replication protocols.

2) Run-time software engineering: As not all possible evolution requirements can be foreseen for long-running software, long-term evolution has to be supported to regulate the emerging behavior of large and dynamic systems, again, with respect to the evolution of requirements and user expectations, but also in response to long-term changes in the context.

This will be performed by changing the system's design during run-time, which in turn requires run-time processable requirements and design-views in the form of constraints [24], models, or (partial) architectural configurations.

The idea here is to move aspects of the system into the run-time that previously have been modified only at design time.

These run-time accessible and processable requirements have to be managed as shown in Section V-B. The vision here is a convergence of software development tools with middleware (including traditional dependability, fault tolerance, and adaptivity concepts), to provide for run-time software development tools to support the envisaged adaptivity. Yet, in turn, this introduces new challenges for engineering, requiring methods for run-time verification and testing, for example.

Figure 4 shows the outer control loop: The properties of the system are measured ("software sensors") and compared to the users' current needs. During a negotiation process, it is decided which properties are traded against each other according to the current system state, context, and users' needs. Finally, the system is changed via adjustment of run-time managed requirements, in order to achieve the properties as intended. Short-term adaptivity and long-term evolution can accordingly be differentiated as a combination of two nested control loops, where the inner loop represents adaptivity and the outer loop (as depicted in Figure 4) represents software evolution.

Regardless of the pace of change, both approaches address the imprecise, emerging, and ever-changing nature of large and long-running software systems and introduce iterative steps of adaptation and evolution during run-time. Both approaches are needed in practice and will need different solutions, but have in common the need for reconfiguration of structure and coupling, for service monitoring and run-time interaction mining, and for the explicit run-time management of requirements.

VI. Conclusion

This paper presented research challenges in software and service engineering with regard to building self-adaptation techniques for complex service-oriented systems. The presented contributions are based on the assumption that increasingly complex service-oriented systems are built utilizing humans as well as software services as an ensemble. The trend to build self-adaptation capabilities into software requires a model-driven approach to support compliance, explicit run-time management of requirements, run-time interaction mining, and an explicit and potentially nested control loop as architectural model.

Certainly, the presented contributions do not solve all problems at hand but can be viewed as one next step to address the manifold challenges the software and service engineering community faces in the area of self-adaptive systems.

Acknowledgements

This research is partially supported by the EU-funded projects inContext, DeDiSys, COMPAS, COIN, and S-CUBE. We thank Christoph Dorn, Lorenz Froihofer, Ta'id Holmes, Daniel Schall, Huy Tran, and Martin Treiber for their contribution to the development of the techniques mentioned in this paper.

References

[1] C. Reich, K. Bubendorfer, and R. Buyya, "An autonomic peer-to-peer architecture for hosting stateful web services," in *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 250–257.

[2] R. Grønmo and M. C. Jaeger, "Model-driven methodology for building qos-optimised web service compositions," in *DAIS*, ser. Lecture Notes in Computer Science, L. Kutvonen and N. Alonistioti, Eds., vol. 3543. Springer, 2005, pp. 68–82.

[3] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani, "Paws: A framework for executing adaptive web-service processes," *IEEE Software*, vol. 24, no. 6, pp. 39–46, 2007.

[4] M. Treiber, H.-L. Truong, and S. Dustdar, "SEMF - Service Evolution Management Framework," in *34th EUROMICRO Conference on Software Engineering andAdvanced Applications (SEAA), Special session on Quality and Service-Oriented Applications*. IEEE Computer Society, 2008.

[5] V. Tosic, K. Patel, and B. Pagurek, "Wsol - web service offerings language," in *CAiSE '02/ WES '02: Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*. London, UK: Springer-Verlag, 2002, pp. 57–67.

[6] A. Keller and H. Ludwig, "The wsla framework: Specifying and monitoring service level agreements for web services," *J. Network Syst. Manage.*, vol. 11, no. 1, 2003.

[7] V. Tosic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma, "Management applications of the web service offerings language (wsol)," *Advanced Information Systems Engineering*, pp. 1029–1029, 2003. [Online]. Available: http://www.springerlink.com/content/8bjl2jgucqb0dprr

[8] D. D. Lamanna, J. Skene, and W. Emmerich, "Slang: A language for defining service level agreements," in *FTDCS '03: Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*. Washington, DC, USA: IEEE Computer Society, 2003, p. 100.

[9] G. R. Gangadharan, M. Weiss, V. D'Andrea, and R. Iannella, "Service license composition and compatibility analysis," in *ICSOC*, ser. Lecture Notes in Computer Science, B. J. Krämer, K.-J. Lin, and P. Narasimhan, Eds., vol. 4749. Springer, 2007, pp. 257–269.

[10] G. Canfora and M. D. Penta, "Testing services and service-centric systems: Challenges and opportunities," *IT Professional*, vol. 8, no. 2, pp. 10–17, 2006.

[11] D. Schall, H.-L. Truong, and S. Dustdar, "Unifying human and software services in web-scale collaborations," *IEEE Internet Computing*, vol. 12, no. 3, pp. 62–68, 2008.

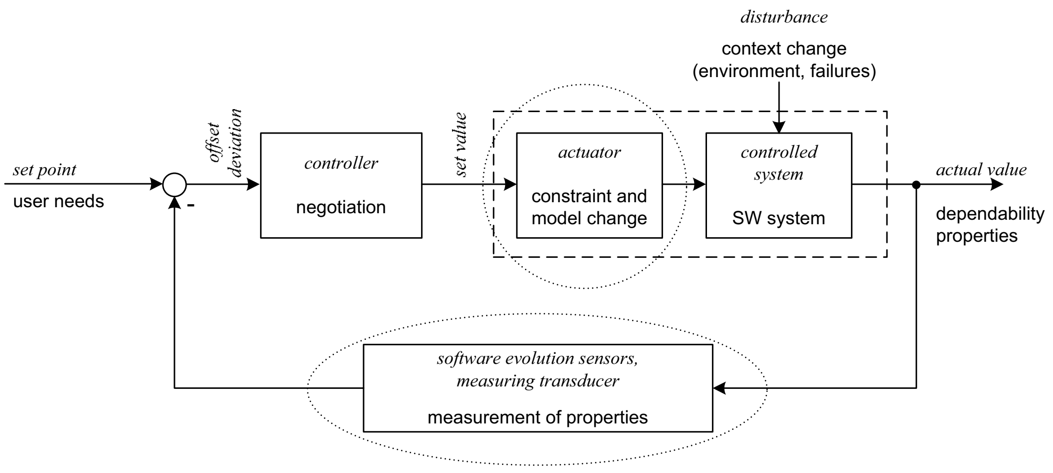

Figure 4. Control loop of dependable evolution through run-time model-/constraint-change.

[12] B. H. C. Cheng, H. Giese, P. Inverardi, J. Magee, and R. de Lemos, "08031 – software engineering for self-adaptive systems: A research road map," in *Software Engineering for Self-Adaptive Systems*, ser. Dagstuhl Seminar Proceedings, B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., vol. 08031. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.

[13] A. P. Barros, M. Dumas, and A. H. M. ter Hofstede, "Service interaction patterns," in *Business Process Management*, W. M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, Eds., vol. 3649, 2005, pp. 302–318.

[14] W. M. P. van der Aalst, B. F. van Dongen, C. W. Günther, R. S. Mans, A. K. A. de Medeiros, A. Rozinat, V. Rubin, M. Song, H. M. W. E. Verbeek, and A. J. M. M. Weijters, "Prom 4.0: Comprehensive support for *eal* process analysis," in *ICATPN*, ser. Lecture Notes in Computer Science, J. Kleijn and A. Yakovlev, Eds., vol. 4546. Springer, 2007, pp. 484–494.

[15] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[16] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Trans. Auton. Adapt. Syst.*, vol. 1, no. 2, pp. 223–259, 2006.

[17] H. Müller, M. Pezzè, and M. Shaw, "Visibility of control in adaptive systems," in *ULSSIS '08: Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems*. New York, NY, USA: ACM, 2008, pp. 23–26.

[18] T. Stahl and M. Voelter, *Model-Driven Software Development*. J. Wiley and Sons Ltd., 2006.

[19] oAW, "openArchitectureWare," http://www.openarchitectureware.org/, 2008.

[20] H. Tran, U. Zdun, and S. Dustdar, "View-based and model-driven approach for reducing the development complexity in process-driven SOA," in *BPSC*, ser. LNI, W. Abramowicz and L. A. Maciaszek, Eds., vol. 116. GI, 2007, pp. 105–124.

[21] H. L. Truong and S. Dustdar, "Online interaction analysis framework for ad-hoc collaborative processes in soa-based environments," *T. Petri Nets and Other Models of Concurrency*, vol. 2, pp. 260–277, 2009.

[22] D. Garlan and B. Schmerl, "Model-based adaptation for self-healing systems," in *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM Press, 2002, pp. 27–32.

[23] S. M. Manson, "Simplifying complexity: a review of complexity theory," *Geoforum*, vol. 32, no. 3, pp. 405–414, 2001.

[24] L. Froihofer, K. M. Goeschka, and J. Osrael, "Middleware support for adaptive dependability," in *Middleware 2007—Proc. of the ACM/IFIP/USENIX 8th International Middleware Conference*. Springer, 2007, pp. 308–327.