# Maintaining Trustworthiness of Socio-Technical Systems at Run-Time

Nazila Gol Mohammadi[1], Torsten Bandyszak[1], Micha Moffie[2], Xiaoyu Chen[3], Thorsten Weyer[1], Costas Kalogiros[4], Bassem Nasser[3], and Mike Surridge[3]

[1]paluno - The Ruhr Institute for Software Technology, University of Duisburg-Essen, Germany
[2]IBM Research, Haifa, Israel
[3]IT-Innovation Center, School of Electronics and Computer Science, University of Southampton, Southampton, United Kingdom
[4]Athens University of Economics and Business, Athens, Greece
{nazila.golmohammadi, torsten.bandyszak, thorsten.weyer}@paluno.uni-due.de, moffie@il.ibm.com, {wxc, bmn, ms}@it-innovation.soton.ac.uk, ckalog@aueb.gr

**Abstract.** Trustworthiness of dynamical and distributed socio-technical systems is a key factor for the success and wide adoption of these systems in digital businesses. Different trustworthiness attributes should be identified and accounted for when such systems are built, and in order to maintain their overall trustworthiness they should be monitored during run-time. Trustworthiness monitoring is a critical task which enables providers to significantly improve the systems' overall acceptance. However, trustworthiness characteristics are poorly monitored, diagnosed and assessed by existing methods and technologies. In this paper, we address this problem and provide support for semi-automatic trustworthiness maintenance. We propose a trustworthiness maintenance framework for monitoring and managing the system's trustworthiness properties in order to preserve the overall established trust during run time. The framework provides an ontology for run-time trustworthiness maintenance, and respective business processes for identifying threats and enacting control decisions to mitigate these threats. We also present use cases and an architecture for developing trustworthiness maintenance systems that support system providers.

**Keywords:** Socio-Technical Systems, Trustworthiness, Run-time Maintenance

## 1 Introduction

Humans, organizations, and their information systems are part of Socio-Technical Systems (STS) as social and technical components that interact and strongly influence each other [3]. These systems, nowadays, are distributed, connected, and communicating via the Internet in order to support and enable digital business processes, and thereby provide benefits for economy and society. For example, in the healthcare domain, STS enable patients to be medically supervised in their own home by care providers [18]. Trust underlies almost every social and economic relation. However, the end-users involved in online digital businesses generally have limited information about the STS supporting their transactions. Reports (e.g., [8]) indicate an increasing

number of cyber-crime victims, which leads to massive deterioration of trust in current STS (e.g., w.r.t. business-critical data). Thus, in the past years, growing interest in trustworthy computing has emerged in both research and practice.

Socio-technical systems can be considered worthy of stakeholders' trust if they permit confidence in satisfying a set of relevant requirements or expectations (cf. [2]). A holistic approach towards trustworthiness assurance should consider trustworthiness throughout all phases of the system life-cycle, which involves: 1) *trustworthiness-by-design*, i.e., applying engineering methodologies that regard trustworthiness to be built and evaluated in the development process; and 2) *run-time trustworthiness maintenance* when the system is in operation. Stakeholders expect a system to stay trustworthy during its execution, which might be compromised by e.g. security attacks or system failures. Furthermore, changes in the system context may affect the trustworthiness of an STS in a way that trustworthiness requirements are violated. Therefore it is crucial to monitor and assure trustworthiness at run-time, following defined processes that build upon a sound theoretical basis.

By studying existing approaches that address maintaining STS trustworthiness at run-time, we identified a lack of generally applicable and domain-independent concepts. In addition, existing frameworks and technologies do not appropriately address all facets of trustworthiness. There is also insufficient guidance for service providers to understand and conduct maintenance processes, and to build corresponding tools. We seek to go beyond the state-of-the-art of trustworthiness run-time maintenance by establishing a better understanding of key concepts for measuring and controlling trustworthiness at run-time, and by providing guidance to operate and maintain STS in a trustworthy manner, supported by tools.

The contribution of this paper consist of three parts: First, we introduce a domain-independent ontology that describes the key concepts of our approach. Second, we propose business processes for monitoring, measuring, and managing trustworthiness, as well as mitigating trustworthiness issues at run-time. Third, we present use cases and an architecture for trustworthiness maintenance systems that are able to facilitate the processes using fundamental concepts of autonomous systems.

The remainder of this paper is structured as follows: In Section 2 we describe the fundamentals w.r.t. trustworthiness of STS and the underlying runtime maintenance approach. Section 3 presents the different parts of our approach, i.e., an ontology for run-time trustworthiness of STS, respective business processes, as well as use cases and a an architecture for trustworthiness maintenance systems that support STS providers. In Section 4, we briefly discuss the related work. We conclude this paper with a summary and a brief discussion of our ongoing research activities in Section 5.

## 2 Fundamentals

In this section, we present the fundamental concepts that form the basis for our approach. First, we present our notion of trustworthiness related to STS. Then, we briefly introduce the concept of run-time maintenance in autonomic systems.

### 2.1 Trustworthiness of Socio-Technical Systems

The term "trustworthiness" is not consistently used in the literature, especially with respect to software. Some approaches merely focus on single trustworthiness characteristics. However, even if combined, these one-dimensional approaches are not sufficient to capture all kinds of trustworthiness concerns for a broad spectrum of different STS, since the conception of trustworthiness depends on a specific system's context and goals [1]. For example, in safety-critical domains, failure tolerance of a system might be prioritized higher than its usability. In case of STS, we additionally need to consider different types of system components, e.g. humans or software assets [3].

Trustworthiness in general can be defined as the assurance that the system will perform as expected, or meets certain requirements [2]. With a focus on software trustworthiness, we adapt the notion of trustworthiness from [1], which covers a comprehensive set of quality attributes (e.g., availability or reliability). This allows us to measure overall trustworthiness as the degrees to which relevant quality attributes (then referred to as trustworthiness attributes) are satisfied. To this end, metrics for objectively measuring these values can be defined, as shown in [19].

### 2.2 Run-time Maintenance in Autonomic Computing

Our approach for maintain trustworthiness at run-time is mainly based on the vision of Autonomic Computing [6]. The goal of Autonomic Computing is to design and develop distributed and service-oriented systems that can easily adapt to changes which affect the system administration and service delivery, while alleviating some of the complexities associated in managing these systems. Considering assets of STS as managed elements of an autonomic system allows us to apply the concepts of Autonomic Computing to trustworthiness maintenance. MAPE-K (Monitor, Analyze, Plan, Execute, and Knowledge) is a reference model for control loops used in Autonomic Computing with the objective of supporting the concepts of self-management, specifically: self-configuration, self-optimization, self-healing, and self-protection [5, 6]. Fig. 1 shows the elements of an autonomic system: the control loop activities, sensor and effector interfaces, and the system being managed.

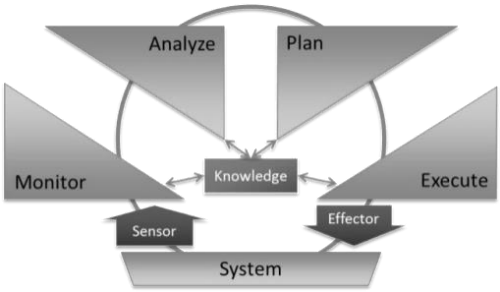


**Fig. 1.** Autonomic Computing and MAPE-K Loop

The *Monitor* provides mechanisms to collect events from the system. It is also able to filter and aggregate the data, and report details or metrics [5]. To this end, system-

specific *Sensors* provide interfaces for gathering required monitoring data, and can also raise events when the system configuration changes [5]. *Analyze* provides the means to correlate and model the reported details or measures. It is able to handle complex situations, learns the environment, and predicts future situations. *Plan* provides mechanisms to construct the set of actions required to achieve a certain goal or objective or respond to a certain event. *Execute* offers the mechanisms to realize the actions involved in a plan, i.e., to control the system by means of *Effectors* which modify the managed element [6]. A *System* is a managed element (e.g., software) that contains resources and provides services. Here, managed elements are assets of STS. Additionally, a common *Knowledge* base acts as the central part of the control loop, and is shared by the activities to store and access collected and analyzed data.

# 3 A Framework for Maintaining Trustworthiness of Socio-Technical Systems at Run-Time

This section presents our approach for maintaining STS trustworthiness at run-time. We describe a framework that consists of the following parts: 1) an ontology that provides general concepts for run-time trustworthiness maintenance, 2) processes for monitoring and managing trustworthiness, 3) functional use cases of a system for supporting the execution of these processes, and 4) a reference architecture that guides the development of such maintenance systems. Based on the ontology and processes, we provide guidance for developing supporting maintenance systems (i.e., use cases and reference architecture). The reference architecture is furthermore based on MAPE-K (see Section 2.2), which, in principle allows for realizing automated maintenance. However, our approach focuses on semi-automatic trustworthiness maintenance, which involves decisions taken by a human system operator. In the next subsections, we elaborate on the elements of the framework in detail.

## 3.1 Ontology for Run-Time Trustworthiness Maintenance

This section outlines the underlying ontology on which the development of run-time trustworthiness maintenance is based. Rather than focusing on a specific domain, our approach provides a meta-model that abstracts concrete system characteristics, in such a way that it can be interpreted by different stakeholders and applied across disciplines. Fig. 2 illustrates the key concepts of the ontology and their interrelations.

The definition of qualitative trustworthiness attributes forms the basis for identifying the concepts, since they allow for assessing the trustworthiness of a great variety of STS. However, trustworthiness attributes are not modelled directly; instead they are encoded implicitly using a set of quantitative concepts. The core elements abstract common concepts that are used to model trustworthiness of STS, while the run-time concepts are particularly required for our maintenance approach.

Trustworthiness attributes of *Assets*, i.e., anything of value in an STS, are concretized by *Trustworthiness Properties* that describe the system's quality at a lower abstraction level with measurable values of a certain data type, e.g., the response time

related to a specific input, or current availability of an asset. These properties are atomic in the sense that they refer to a particular system snapshot in time. The relation between trustworthiness attributes and properties is many to many; an attribute can potentially be concretized by means of multiple properties, whereas a property might be an indicator for various trustworthiness attributes. Values of trustworthiness properties can be read and processed by metrics in order to estimate the current levels of trustworthiness attributes. A *Metric* is a function that consumes a set of properties and produces a measure related to trustworthiness attributes. Based on metrics, statements about the behavior of an STS can be derived. It also allows for specifying reference threshold values captured in *Trustworthiness Service-Level Agreements* (TSLAs).

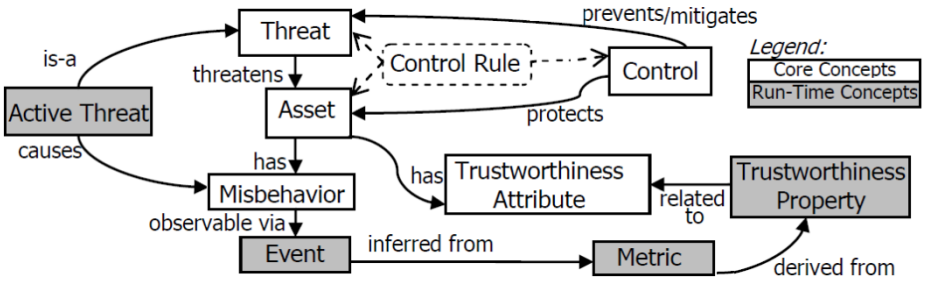


**Fig. 2.** Ontology for Run-Time Trustworthiness Maintenance

A system's behavior is observed by means of *Events*, i.e., induced asset behaviors perceivable from interacting with the system. Events can indicate either normal or abnormal behavior, e.g., underperformance or unaccountable accesses. *Misbehavior* observed from an event or a sequence of events may manifest in a *Threat* which undermines an asset's value and reduces the trustworthiness of the STS. This in turn leads to an output that is unacceptable for the system's stakeholders, reducing their level of trust in the system. Given these consequences, we denote a threat "active". Threats (e.g., loss of data) can be mitigated by either preventing them from becoming active, or counteracting their effects (e.g., corrupted outputs). Therefore, *Controls* (e.g., service substitution) are to be executed. *Control Rules* specify which controls can block or mitigate a given type of threat. Identifying and analyzing potential threats, their consequences, and adequate controls is a challenging task, which should be started in early requirements phases.

### 3.2 Processes for Run-Time Trustworthiness Maintenance

In order to provide STS providers with concrete guidance for realizing trustworthiness maintenance, we define two complementary reference processes, i.e., *Trustworthiness Monitoring* and *Management*. These processes illustrate the utilization of the ontology concepts. We denote them as "reference processes" since they provide a high-level view on the activities that need to be carried out in order to implement trustworthiness maintenance, without considering system-specific characteristics. Our approach is designed to be semi-automatic, i.e., we assume a human maintenance operator to be consulted for taking critical decisions.

**Trustworthiness Monitoring.** Monitoring is responsible for observing the behavior of STS in order to identify and report potential threats to the *Management*, which will then analyze the STS state and enact corrective actions, if necessary. In general, our monitoring and measuring approach is based on metrics which allow for quantifying the current value of relevant trustworthiness attributes. The reference process for trustworthiness monitoring is shown in the BPMN diagram depicted in Fig. 3.

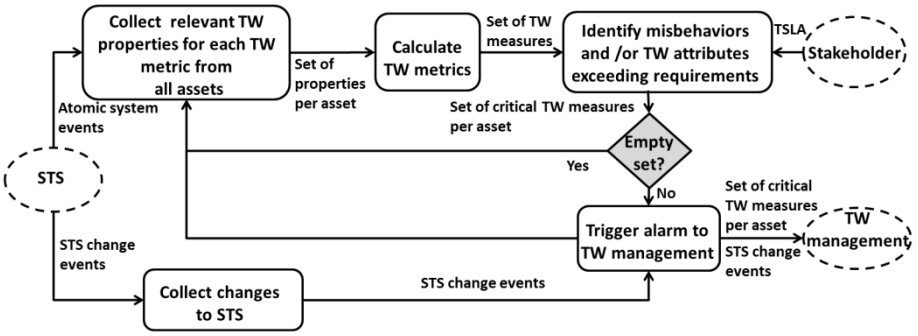


**Fig. 3.** Process of Trustworthiness Monitoring

According to our modelling ontology, each measure is based on collected data, called atomic properties. Thus, the first step involves collecting all relevant trustworthiness properties (e.g., indicating system usage). These can be either 1) system properties that are necessary to compute the metrics for the set of relevant trustworthiness attributes, or 2) system topology changes, such as the inclusion of a new asset. Atomic system events indicate changes of properties. For each system asset, trustworthiness metrics are computed. Having enough monitoring data, statistical analysis can be used for aggregating atomic measurements into composite ones, e.g., the mean response time of an asset. These measures and further processed in order to identify violations of trustworthiness requirements that are captured in user-specific TSLAs. For each trustworthiness metrics, it is observed whether the required threshold(s) are exceeded. If so, the critical assets are consequently reported to the management, so that potentially active threats can be identified and mitigation actions can be triggered.

Each STS has its individual characteristics and requirements for trustworthiness. At run-time, system characteristics may change, e.g., due to adaptations to the environment. Consequently, another important monitoring task is to accept change notifications from the STS, and forward them to the trustworthiness management.

**Trustworthiness Management.** The key objective of STS trustworthiness management (see Fig. 4) is to guarantee correct system and service behavior in real-time by continuously analyzing system behavior, identifying potential threats, recommending and executing possible mitigation actions.

The reference management process is triggered by incoming events (i.e., misbehaviors or system changes) reported by the trustworthiness monitoring. Misbehaviors identified in the form of deviations from required trustworthiness levels indicate an abnormal status of the target STS, e.g., underperformance due to insufficient resources, or malicious attacks. The management keeps tracks of the system status over

time, and analyzes the causes of misbehaviors. Once threats are classified, it is necessary to analyze their effect on the asset's behavior and understand the links between them in order to analyze complex observations and sequences of threats that may be active, and identify suitable controls. Statistical reasoning is necessary for estimating threat probabilities (for each trustworthiness attribute).

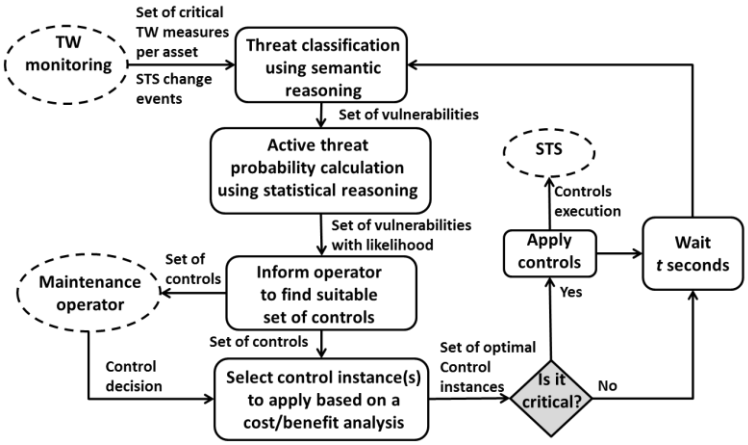


**Fig. 4.** Process of Trustworthiness Management

Regarding control selection and deployment, we focus on semi-automated control deployment, as illustrated in Fig. 4, which requires human intervention. The system operator is notified whenever new threats are identified. These threats may be active, indicating vulnerabilities due to lacking of necessary controls. Each threat is given a likelihood based on the observed system behaviors. It is then the system operator's responsibility to select appropriate controls that can be applied to the STS in order to realize mitigation. These controls involve, e.g., authentication or encryption. A number of control instances may be available for each control (e.g., different encryption technologies), having different benefits and costs. Based on cost-effective recommendations, the operator can select control instance to be deployed. As a consequence, previously identified active threats should be classified as blocked or mitigated. Note that we do not provide a separate mitigation process, since the actual mitigation execution is rather a technical issue that does not involve complex logic.

The system may be dynamic, i.e., components can be added or removed. Whenever a notification arrives that the topology of the system has changed, the management process is carried out again.

### 3.3 Use Cases of a Run-Time Trustworthiness Maintenance System

Based on the reference processes introduced in Section 3.2, we elicited functional requirements of a tool that supports STS providers in maintaining trustworthiness. Such a system is supposed to facilitate and realize the business processes in a semi-automatic manner. We distinguish three main areas of functionality, i.e., *Monitoring*,

*Management*, and *Mitigation*. The latter is included for a better separation of concerns, although we did not define a separate reference process for mitigation. We analyzed possible maintenance use cases, and actors that interact with the system. The results of this analysis are shown in the UML use case diagram in Fig. 5.

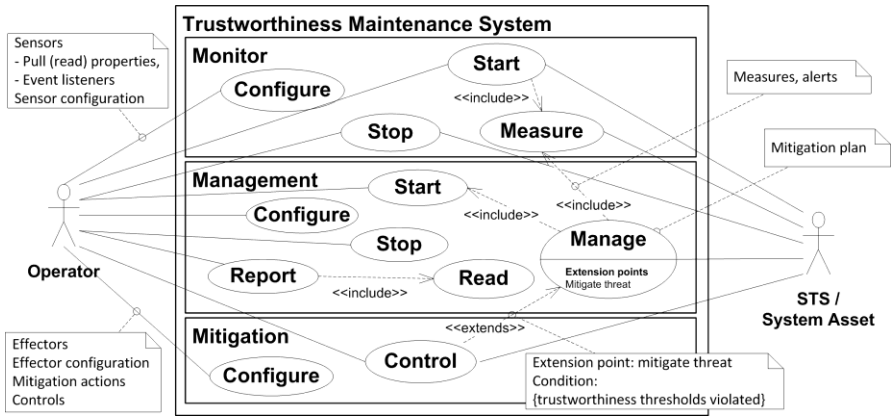


**Fig. 5.** Trustworthiness Maintenance Use Cases

The Monitoring functionality is responsible for collecting events and properties from the system (measuring the STS) and computing metrics. The inputs to the component are system properties and atomic events that are collected from the STS. The output, i.e., measures, is provided to the Management. The maintenance operator (e.g., the service provider) is able to start and stop the measurement, and to configure the monitor. Specifically, the operator can utilize the concept of trustworthiness requirements specified in TSLAs (cf. Section 3.1) to derive appropriate configuration.

The Management part provides the means to assess current trustworthiness attributes using the metrics provided from monitoring, choose an appropriate plan of action (if needed) and forward it to the mitigation. The operator is able to configure the Management component and provides a list of monitor(s) from which measures should be read, a list of metrics and trustworthiness attributes that are of interest, as well as management processes. Additionally, the operator is able to start/stop the management process, retrieve trustworthiness metric values, and to generate reports which contain summaries of trustworthiness evolution over time.

Lastly, the Mitigation part has one main purpose – to control the STS assets by realizing and enforcing mitigation actions, i.e., executing controls to adjust the trustworthiness level. The maintenance operator will configure the service with available mitigation actions and controls that are to be executed by means of effectors.

### 3.4 Architecture for Run-Time Trustworthiness Maintenance Systems

We view the trustworthiness maintenance system as an autonomic computing system (see Section 2.2). The autonomic system elements can be mapped to three maintenance components, similar to the distribution of functionality in the use case diagram

in Fig. 5. The Monitor and Mitigation components are each responsible for a single functionality - monitoring and executing controls. Analyze and plan functionalities are mapped to a single management package, since they are closely related, and in order to simplify the interfaces. Fig. 6 shows the reference architecture of a maintenance system as a UML component diagram, depicting the components that are structured in three main packages *Monitor*, *Management* and *Mitigation*.

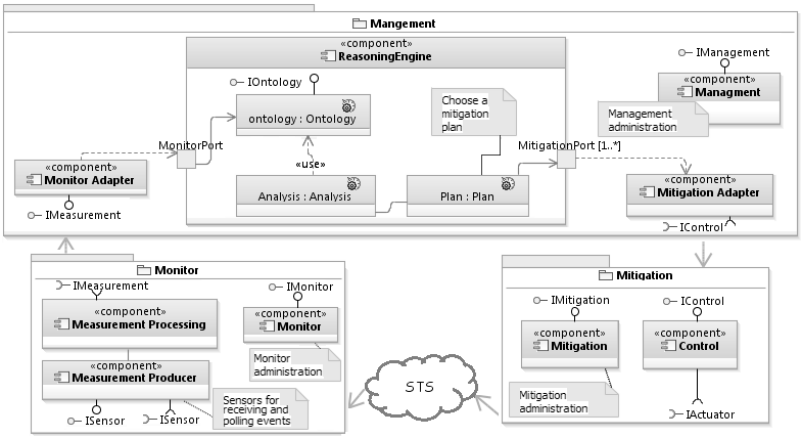


**Fig. 6.** Reference System Architecture for Run-Time Trustworthiness Maintenance

Trustworthiness maintenance systems are designed around one centralized management component and support distributed monitoring and mitigation. This modular architecture enables instantiating multiple monitors on different systems, each reporting to a single centralized management. Likewise, Mitigation can be distributed among multiple systems, too. This allows for greater scalability and flexibility.

**Monitor.** The Monitor package contains three components. The *Monitor* component provides an API to administer and configure the package, while the *Measurement Producer* is responsible for interfacing with the STS via sensors. The latter supports both passive sensors listening to events, as well as active sensors that actively measure the STS (e.g., to check if the system is available). Hence, the STS-specific event capturing implementation is decoupled from the more generic *Measurement Processing* component which gathers and processes all events. It is able to compute metrics and forward summarized information to the management. In addition, it may adjust the processes controlling the sensors (e.g., w.r.t. frequency of measurements).

One way to implement the *Monitor* component is using an event-based approach like Complex Event Processing (CEP) [4]. CEP handles events in a processing unit in order to perform monitor activities, and to identify unexpected and abnormal situations at run-time. This offers the ability of taking actions based on enclosed information in events about the current situation of an STS.

**Management.** The Management package is responsible for gathering all information from the different monitors, store it, analyze it, and find appropriate plans to execute mitigation controls. It contains *Monitor* and *Mitigation* adapters that allow multiple monitors or mitigation packages to interact with the management, and provide the reasoning engine with unified view of all input sources and a single view of

all mitigation packages. It also includes the *Management* administration component that is used to configure all connected Monitor and Mitigation packages, and exposes APIs for configuration, display and report generation. The central component, the *Reasoning Engine*, encapsulates all the logic for the analysis of the measurements and planning of actions. This allows us to define an API for the engine and then replace it with different engines. Internally, an instance of the *Reasoning Engine* contains *Analysis* and *Plan* components as expected from an autonomic computing system (cf. Section 2.2), as well as an *Ontology* component. The ontology component encapsulates all required system models, which define e.g. threats and attributes. This allows for performing semantic reasoning by executing rules against the provisional system status and, estimating the likelihood of threat activeness (e.g., vulnerabilities) based on the current monitoring state. Given active threats, the management can instruct the mitigation component what action(s) to perform in order to restore or maintain STS trustworthiness, based on decisions taken by the operator.

**Mitigation.** The Mitigation package contains a *Control* component that encapsulates all interaction with the STS, and a *Mitigation* administration component. This allows us to separate and abstract STS control details, mitigation configuration and expose a generic API. The Mitigation package is responsible for executing mitigation actions by means of appropriate STS-specific effectors. These actions may be complex such as deploying another instance of the service, or as simple as presenting a warning to the operator including information for him to act on.

## 4 Related Work

Related work can be found in several areas, since trustworthiness of STS comprises many disciplines, especially software development. For example, methodologies for designing and developing trustworthy systems, such as [2], focus on best practices, techniques, and tools that can be applied at design-time, including the trustworthiness evaluation of development artifacts and processes. However, these trustworthiness-by-design approaches do not consider the issues related to run-time trustworthiness assessment. Metrics as a means for quantifying software quality attributes can be found in several publications, e.g. related to security and dependability [9], personalization [10], or resource consumption [11].

The problem of trustworthiness evaluation that we address has many similarities with the monitoring and adaption of web services in Service-Oriented Architectures, responding to the violation of quality criteria. Users generally favor web services that can be expected to perform as described in Service Level Agreements. To this end, reputation mechanisms can be used (e.g., [12]). However, these are not appropriate for objectively measuring trustworthiness based on system characteristics. In contrast, using online monitoring approaches, analyses and conflict resolution can be carried out based on logging the service interactions. Online monitoring can be performed by the service provider, service consumer, or trusted third parties [13, 14]. The ANIKETOS TrustWorthinessModule [15] allows for monitoring the dependability of service-oriented systems, considering system composition as well as specific compo-

nent characteristics. Zhao et al. [7] also consider service composition related to availability, reliability, response time, reputation, and security. Service composition plays an important role in evaluation, as well as in management. For example, in [15] substitution of services is considered as the major means of restoring trustworthiness. Decisions to change the system composition should not only consider system qualities [17], but also related costs and profits [15, 11]. Lenzini et al. [16] propose a Trustworthiness Management Framework in the domain of component-based embedded systems, which aims at evaluating and controlling trustworthiness, e.g., w.r.t. dependability and security characteristics, such as CPU consumption, memory usage, or presence of encryption mechanisms. Conceptually, their framework is closely related to ours, since it provides a software system that allows for monitoring multiple quality attributes based on metrics and compliance to user-specific trustworthiness profiles.

To summarize, there are no comprehensive approaches towards trustworthiness maintenance, which consider a multitude of system qualities and different types of STS. There is also a lack of a common terminology of relevant run-time trustworthiness concepts. Furthermore, appropriate tool-support for enabling monitoring and management processes is rare. There is insufficient guidance for service providers to understand and establish maintenance processes, and to develop supporting systems.

## 5 Conclusion and Future Work

Maintaining trustworthiness of STS at run-time is a complex task for service providers. In this paper, we have addressed this problem by proposing a framework for maintaining trustworthiness. The framework is generic in the sense that it is based on a domain-specific ontology suitable for all kinds of STS. This ontology provides key concepts that are valuable for understanding and addressing run-time trustworthiness issues. Our framework defines reference processes for trustworthiness monitoring and management, which guide STS providers in realizing run-time maintenance. As the first step towards realizing trustworthiness maintenance processes in practice, we presented results of a use case analysis, in which high-level functional requirements of maintenance systems have been elicited, as well as a general architecture for such systems.

We are currently in the process of developing a prototype of a trustworthiness maintenance system that implements our general architecture. Therefore, we will define more concrete scenarios that will further detail the abstract functional requirements presented herein, and also serve as a reference for validating the system in order to show the applicability of our approach. We also aim at extending the framework and the maintenance system by providing capabilities to monitor and maintain the user's trust in the STS. The overall aim is to balance trust and trustworthiness, i.e., to prevent unjustified trust, and to foster trust in trustworthy systems. To some extent, trust monitoring and management may be based on monitoring trustworthiness as well, since some changes of the trustworthiness level are directly visible to the user. Though additional concepts and processes are needed, we designed our architecture in a way that allows for easily expanding the scope to include trust concerns.

## References


1. Gol Mohammadi, N., Paulus, S., Bishr, M., Metzger, A., Koennecke, H., Hartenstein, S., Pohl, K.: An Analysis of Software Quality Attributes and Their Contribution to Trustworthiness. In: 3rd Int. Conference on Cloud Computing and Service Science, Aachen (2013)
2. Amoroso, E., Taylor, C., Watson, J., Weiss, J.: A Process-Oriented Methodology for Assessing and Improving Software Trustworthiness. In: 2nd ACM Conference on Computer and Communications Security, pp. 39-50. ACM, New York (1994)
3. Sommerville, I.: Software Engineering, 9th edition. Pearson, Boston (2011)
4. Luckham, D.: The Power of Events – An Introduction to Complex Event Processing in Distributed Enterprise Systems". Addison-Wesley (2002)
5. IBM: An Architectural Blueprint for Autonomic Computing, Autonomic Computing, White paper, IBM (2003)
6. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. IEEE Computer, Vol. 36, Issue 1 , 41-50 (2003)
7. Zhao, S., Wu, G., Li, Y., Yu, K.: A Framework for Trustworthy Web Service Management. In: 2nd Int. Symp. on Electronic Commerce and Security, pp. 479-482. IEEE (2009)
8. Computer Security Institute: 15th Annual 2010/2011 Computer Crime and Security Survey. Technical Report, Computer Security Institute (2011)
9. Arlitt, M., Krishnamurthy, D., Rolia, J.: Characterizing the Scalability of a Large Web Based Shopping System, ACM Trans. on Internet Tech., Vol. 1, No. 1, 44–69 (2001)
10. Bassin, K., Biyani, S., Santhanam, P: Metrics to Evaluate Vendor-developed Software based on Test Case Execution Results, IBM Systems Journal, Software Testing and Verification, Volume 41, Number 1, 13-30 (2002)
11. Zivkovic, M., Bosman, J.W., van den Berg, J.L., van der Mei, R.D., Meeuwissen, H.B., Nunez-Queija, R.: Dynamic Profit Optimization of Composite Web Services with SLAs." Global Telecom. Conference (GLOBECOM), pp. 1-6. IEEE (2011)
12. O. Rana, M.E. Warnier, T.B. Quillinan and F.M.T Brazier. Monitoring and Reputation Mechanisms for Service Level Agreements. In: 5th Int. Workshop on Grid Economics and Business Models (GenCon), pp. 125-139. Springer, Berlin Heidelberg (2008)
13. Clark, K., Warnier, M., Quillinan, T., Brazier, F.: Secure Monitoring of Service Level Agreements. In: 5th Int. Conference on Availability, Reliability, Security, pp. 454-461, IEEE (2010)
14. Quillinan, T., Clark, K., Warnier, M, Rana, O., Brazier, F.: Negotiation and Monitoring of Service Level Agreements. In: Wieder, P., Yahyapour, R., Ziegler, W. (eds.) Grids and Service-Oriented Architectures for Service Level Agreements, pp. 167-176. Springer, Berlin Heidelberg (2010)
15. Elshaafi, H., McGibney, J., Botvich, D: Trustworthiness Monitoring and Prediction of Composite Services. In: IEEE Symp. on Computers and Comm., pp. 000580–000587. IEEE (2012)
16. Lenzini, G., Tokmakoff, A., Muskens, J.: Managing trustworthiness in component-based embedded systems. Electronic Notes in Theoretical Computer Science 179, 143–155. Elsevier (2007)
17. Yu, T., Zhang, Y., Lin, K.: Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints, ACM Trans. on the Web (TWEB), Vol. 1, Issue 1, 1-26 (2007)
18. OPTET Consortium: D8.1 – Description of Use Cases and Application Concepts. Technical Report, OPTET Project (2013)
19. OPTET Consortium: D6.2 – Business Process Enactment for Measurement and Management. Technical Report, OPTET Project (2013)