Optimizing Run-Time SOA Governance through Context-driven SLAs and Dynamic Monitoring

Norha M. Villegas*[†], Hausi A. Müller* and Gabriel Tamura[†]
*Department of Computer Science, University of Victoria
Victoria, British Columbia, Canada
Email: {nvillega, hausi}@cs.uvic.ca
[†]Faculty of Engineering, Icesi University
Cali, Valle del Cauca, Colombia
Email: {gtamura}@icesi.edu.co

***Abstract*—End-users increasingly demand the provisioning of secure, scalable, reliable, flexible, resilient, and cost-efficient infrastructures, platforms, and software. However, the preservation of these properties, particularly in SOA and cloud environments, is extremely affected by distributed, heterogeneous, transient, and volatile context information. We envision the implementation of *governance feedback loops*, an innovative approach that equips service-oriented systems with run-time governance capabilities able to control the fulfillment of service level agreements (SLA) under changing execution environments. However, the effectiveness of our approach depends on the capability of governance infrastructures to guarantee the consistency between monitoring strategies, governance objectives, and context situations. To advance our vision, this paper proposes (i) *contextual RDF graphs*, a machine-readable specification of monitoring requirements that enable governance feedback loops with dynamic context monitoring capabilities; and (ii) *context-driven SLAs*, an extension of SLAs where context requirements are explicitly mapped to service level objectives (SLO) to optimize the run-time control of contracted obligations.**

I. Introduction

Cloud computing is a large-scale distributed computing paradigm that enables wide area networks to support interactions between information technologies service providers and consumers. It focuses on providing software functionality and virtualized computing resources in the form of *services* delivered on demand. These cloud services can be classified into three main groups: software-as-a-service (SaaS), platform-as-a-service (PaaS), and infrastructure-as-a-service (IaaS) [17], [28], [19]. In any case, these services must be provisioned under contracted quality of service requirements generally expressed in the form of service level agreements (SLAs) [4].

Cloud computing and service-oriented architecture (SOA) can be considered as complementary technologies. On the one hand, SOA is a way of designing, implementing, and maintaining software systems composed of coarse-grained services that represent reusable functionality [13]. On the other hand, cloud-based services can be realized by composing multiple SOA-based services added on top of cloud resources. Furthermore, a cloud environment could be built on top of an SOA infrastructure by implementing a virtualization and self-provisioning level of indirection [19]. Although the two concepts have overlapping concerns and considerations, they target different problems. SOA's emphasis is on making software component integration in systems of systems more efficient, whereas cloud computing focuses on leveraging network resources to offer software and hardware infrastructure as commodities. Hence, SOA can complement cloud computing to address integration concerns, but cloud computing does not substitute SOA as an integration technology [17]. Most importantly, both cloud computing and SOA share the goal of delivering services in a cost-effective way, while guaranteeing desired quality attributes and satisfying business objectives. Thus, to achieve their respective visions, SOA and cloud computing solutions must ensure the design, implementation, running, maintenance, and evolution of service-oriented systems.

SOA governance defines the policies and mechanisms for implementing, executing, and evolving service-oriented software. It includes a variety of concerns such as the management of data repositories, replacement of services, control of the SOA governance infrastructure, resource allocation, and management of SLAs [11]. In a recent investigation conducted by researchers from Software Engineering Institute (SEI), Harrison and Lewis identified security, control, performance, and reliability as some of the main barriers to cloud computing adoption [19]. It is undeniable that the preservation of quality attributes and management of SLAs are crucial concerns in SOA governance, particularly for service-oriented systems in cloud environments. Therefore, the success of these technologies is highly dependent on their capacity to govern service-oriented systems, and to apply enforcement mechanisms according to policies, procedures, and responsibilities [22].

Service-oriented systems in SOA and cloud environments are highly affected by and dependent on distributed, heterogeneous, transient, and volatile context information. In particular, the desired QoS and system properties may be constantly violated due to changing situations of context entities that can affect the computing infrastructure and system behavior. Furthermore, monitoring mechanisms supporting governance objectives may be no longer relevant as the environment and system states are changing over time. To address these complex dynamics involved in effectively supporting SOA governance, we proposed the implementation of *context-aware governance feedback loops* [14], [22], a promising approach that equips service-oriented systems with run-time governance capabilities based on control loops [15], and adaptive context monitoring [22], [24], to control the fulfillment of SLAs under changing execution environments. To ensure that services are executed according to policies and responsibilities, context-aware governance feedback loops include monitoring capabilities to keep track of environmental changes than can affect system execution. This monitoring instrumentation is implemented as a self-adaptive component to support the deployment of new context sensors and context reasoning components at run-time.

To advance the realization of context-aware governance feedback loops, this paper focuses on two main contributions. The first one, *contextual RDF graphs*, is a machine-readable specification of monitoring requirements that enables governance feedback loops with dynamic context monitoring capabilities to address changes in monitoring requirements at run-time. Variations in monitoring requirements can be motivated by changes in either the governed service-oriented system (e.g., dynamic reconfiguration of its software architecture), governance objectives (e.g., the addition of a new SLO in an existing SLA), or the situation of relevant context entities. Our context specification approach is built on top of the graph-based formalism defined in the resource description framework (RDF) [20], and two of our previous contributions in context management: (i) our extensible context taxonomy proposed to characterize context information in different application domains [22], and (ii) our context management meta-model intended to support the dynamic deployment of infrastructural elements required to manage the context information lifecycle [22]. The second contribution, *context-driven SLAs*, is an extension of the SLA definition where context requirements are explicitly mapped onto SLOs and their corresponding metrics, to optimize the control of contracted obligations at run-time.

With our approach, we aim to implement context-aware governance feedback loops, to realize run-time SOA governance infrastructures where the monitoring mechanisms not only keep track of the relevant context, but adapt themselves to preserve their relevance with SLAs even when governance objectives are continuously changing.

The remainder of this paper is organized as follows. Section II presents an overview of our vision of governance feedback loops intended to optimize run-time governance. Section III analyzes the mapping between quality attributes and selected drivers for cloud computing adoption as an initial step for the identification of context monitoring requirements. Section IV highlights our research challenges in realizing dynamic monitoring. Section V presents our related work that constitutes enabling elements for realizing dynamic monitoring as envisioned in this paper. Sections VI and VII explain in detail our proposal for the specification of dynamic monitoring strategies and context-driven SLAs. With an example we illustrate the applicability of our approach. Finally, Section VIII discusses related work and concludes the paper.

II. Revisiting Governance Feedback Loops

To optimize business functions, service-oriented systems are instrumented with governance capabilities intended to ensure contracted conditions such as performance, reliability and resource consumption. SOA governance processes monitor these properties to identify trends, improve policies and business processes, and manage SLAs consequently [11]. For this, SOA governance implements a general feedback loop where service-oriented systems are continuously monitored to control the effects and outputs of policies and business processes. Then, based on the information monitored from policies and processes, selected governance variables are fed back to decide whether it is necessary to adapt policies, processes and SOA systems [14]. Figure 1 introduces the general SOA governance feedback loop.

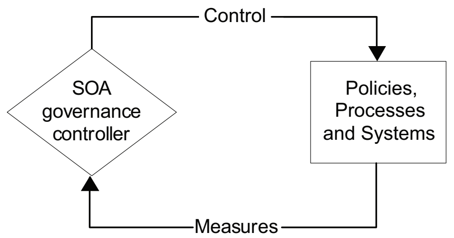

Fig. 1. General SOA governance feedback loop [14]

By implementing this general feedback loop, SOA governance processes are able to control, and monitor components of service-oriented infrastructures to optimize businesses and support system evolution. Thus, SOA governance requires instrumentation to keep track of service-oriented systems and environmental changes that affect governance objectives at run-time. Most importantly, as the environment, business objectives, and system situation are continuously changing, monitoring mechanisms must be self-adaptive to keep track of the right context entities even when monitoring requirements change as a consequence of changes in the execution environment [22].

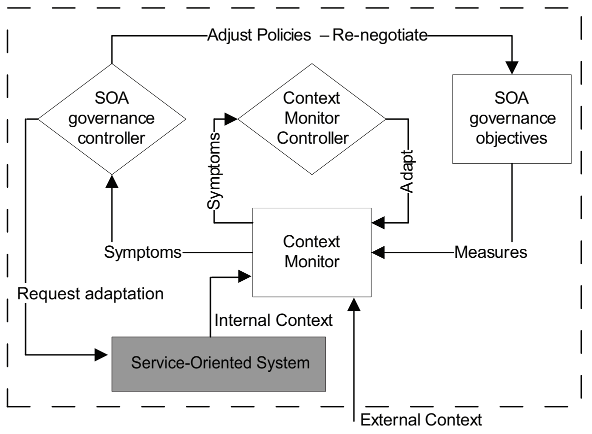

Fig. 2. General view of our proposed *context-aware governance feedback loops* [22]

Figure 2 presents an abstracted view of our proposed context-aware governance feedback loops [22]. The core of our approach is composed of explicit control loops inspired by control theory [9], [15] built into run-time governance infrastructures, as well as the management of context information as a first level entity from design to implementation with the goal of preserving monitoring relevance under changing requirements [23]. SOA governance infrastructures based on context-aware governance feedback loops not only are able to control the system situation in light of governance goals, but also the optimization and evolution of business goals and SLAs according to changes in relevant environmental entities. For this, *context monitors* gather relevant context from the environment and the service-oriented system to measure the system behavior in light of monitoring requirements inferred from SOA governance objectives (SLAs and SLOs). Then, *SOA governance controllers* use the symptoms identified by the context monitor to request the adaptation of the service-oriented system, and adjust or re-negotiate policies if necessary. Similarly, a *context monitor controller* manages the adaptation of the monitoring infrastructure according to changes in SOA governance objectives, and the situation of the governed service-oriented system and relevant external context.

III. Quality of Service in Cloud Environments

Quality of service (QoS) governance and the management of service-level agreements (SLAs) are crucial concerns not only to realize the vision of service-oriented systems in pure SOA environments [16], [12], but also to realize cloud computing promises [28], [17]. Ensuring contracted conditions in the usage of services to deliver software and hardware resources is clearly a mandatory requirement. In fact, most drivers for and barriers to cloud computing adoption can be directly mapped into quality attributes and corresponding quality factors [19]. Toward the specification of context monitoring requirements for supporting the governance of service-oriented systems in SOA and cloud environments, we propose a mapping between QoS attributes and context information types in the form of a context taxonomy to support the context monitoring requirements elicitation from quality attributes, and the design and implementation of context management strategies accordingly. For this, we analyzed the set of cloud computing drivers proposed by SEI [19] in light of well known quality attributes applicable to service-oriented computing such as performance and dependability [1], [2].

A. Mapping drivers to quality attributes

Our first step toward the mapping presented in Table I was the analysis of the drivers for cloud computing adoption characterized by Harrison and Lewis [19]. We concentrated on a selected set of drivers, those that can be traced at run-time. Drivers such as collaboration, the usage of cloud infrastructures to support human collaborative work, were not considered relevant for dynamic monitoring in supporting SOA governance.

Table I presents the selected drivers for cloud computing adoption (left column) mapped to quality attributes (middle column) and corresponding quality factors or quality concerns that must be used to define the corresponding metrics required to evaluate quality of service (right column).

TABLE I
Mapping cloud computing drivers to quality attributes

Cloud Computing Drivers [19]	Quality Attributes	Quality Factors
Availability	Performance	Latency
	Dependability	Availability Reliability Maintainability
Scalability	Performance	Throughput Capacity
Elasticity	Performance	Throughput Capacity
	Dependability	Safety Integrity Reliability
Virtualization	Performance	Latency Throughput Capacity Efficiency
Cost	Performance	Latency Throughput Capacity Efficiency

Our analysis focuses on *performance* and *dependability* quality attributes and their corresponding quality factors. For analyzing performance we started by generalizing the definition adopted by Barbacci et al. [2]. In SOA governance, *performance* can be defined as responsiveness in terms of either the time required for a service to complete specific tasks or the number of tasks processed in a given frame of time. Performance quality factors relevant for dynamic context monitoring in SOA and cloud environments are (i) *latency*—the time a service or a composition of services takes to respond to a specific event; (ii) *throughput*—the number of tasks that can be completed in a given time interval; (iii) *capacity*—a measure of the amount of work a service can perform, and (iv) *efficiency*—a measure of the relationship between resource utilization and time behavior in the accomplishment of a specific task [10]. We map performance to drivers for cloud computing adoption as follows:

- *Performance-Availability*: latency can compromise readiness for usage of software and hardware resources. The time a service or composition of services takes to perform particular tasks must be monitored to guarantee the availability of computational resources delivered to users.
- *Performance-Scalability*: the scalability of computational resources on demand is highly dependent on throughput, and capacity. One the one hand, important objectives of scalability strategies are improving throughput and system capacity. On the other hand, scalability is conditioned by the maximum overall system capacity.
- *Performance-Elasticity*: the elastic capabilities of cloud environments depend on two main factors. The first one is scalability from the perspective of resource constraints,

hence throughput an capacity are performance quality factors relevant to elasticity. The second important aspect, is the adaptive capability of cloud infrastructures to automatically scale at run-time, which raises the necessity of supporting run-time governance and dynamic monitoring capabilities.

- *Performance-Virtualization*: to maximize resource usage and leverage the advantages of virtualization, efficiency is the most important aspect to consider. Efficiency can be measured in terms of latency, throughput and capacity.
- *Performance-Cost*: as in virtualization, efficiency in terms of performance quality factors is crucial to guarantee contracted costs with users and maximize business goals.

In order to analyze dependability quality factors, we revisited the *dependability* definition we used in our reference model for evaluating quality-driven self-adaptive systems [27]: the level of reliance that can justifiably be placed on the services that the software system delivers. Relevant quality factors of dependability for service-oriented systems in SOA and cloud environments are [1]: (i) *availability*—readiness for usage of a particular service; (ii) *reliability*—continuity of correct service delivery; (iii) *maintainability*—self-healing and self-evolution capabilities; (iv) *safety*—absence of catastrophic consequences on the external environment; and (v) *integrity*—non-presence of undesirable system alterations. The mapping between dependability factors and drivers for cloud computing adoption is defined as follows:

- *Dependability-Availability*: availability is not only a driver for cloud computing adoption but also a dependability quality factor. Thus, the availability of computational resources demanded by users will depend on the availability (as quality factor) of services delivering cloud-based resources. Furthermore, dynamic services equipped with self-maintenance capabilities are crucial for guaranteeing the availability of reliable services in cloud environments.
- *Dependability-Elasticity*: the satisfaction of changing user requirements at run-time must be performed while ensuring dependability. Moreover, elasticity is generally enabled by any form of self-adaptation. The adaptive capabilities of cloud infrastructures must be governed at run-time to avoid severe consequences on the external environment or undesirable systems states as a result of self-adaptation. Moreover, the continuity of correct service delivery must be guaranteed before, during and after adapting the cloud infrastructure.

B. Metrics as starting points for monitoring strategies

Quality metrics are important cornerstones for the definition of SOA governance and monitoring strategies. Without explicit and observable metrics we cannot construct effective monitoring mechanisms to ensure the desired properties in a particular software system [27]. Moreover, these metrics must be not only observable at run-time but also directly mapped to monitoring objectives, for instance to the desired quality attributes of a service-oriented system. The explicit identification of relevant context entities from metrics is the first step toward the implementation of a dynamic monitoring strategy.

As an example, consider the *time behavior metric (TB)* proposed by Lee et al. as an efficiency measure for a particular task in a service [10]. They define time behavior as the ratio of task execution time over the total invocation time:

$$TB = \frac{task\ execution\ time}{total\ service\ invocation\ time}$$

The denominator is the time period from the moment a service request is sent to the moment the corresponding response is received. The numerator corresponds to the task execution time of a particular task in a service (a service can be composed of one or more interfaces, each interface may correspond to a task). The range is $0..1$, where higher values indicate better time efficiency.

This metric provides the means to characterize the situation of a relevant context entity—a particular service component. Service components, as any computational resource, are classified as *artificial* context entities. In this example the situation of this service is characterized in terms of *activity* and *time* context information. Activity context corresponds to particular tasks implemented by service interfaces. Time context characterizes both the execution time of the particular tasks and the total service invocation time. Both time observations correspond to the amount of *definite time* context in a time frame. These context categories are defined in the general taxonomy for context identification we proposed previously [23]. This paper discusses the extension of this taxonomy to support the identification of relevant context information and the definition of monitoring strategies for implementing run-time SOA governance using governance feedback loops.

IV. Challenges in Dynamic Context Management for SOA governance

Our investigation focuses on two main research challenges. The first challenge concerns the identification and representation of the context information that is relevant to the governance objectives of a particular service-oriented system. These models must be flexible enough to support context representation and reasoning about governance monitoring requirements in different problem domains without demanding manual changes in the monitoring instrumentation from one domain to another. For instance, a cloud computing provider would expect that the governance infrastructure (governance feedback loops and monitoring instrumentation) is able to accomplish its control objectives even when contracted QoS conditions change from one customer to another. The second challenge corresponds to the run-time management of the context information life cycle (i.e., context acquisition, processing, reasoning, provisioning, and disposal) [23]. To address these challenges, we investigated (i) the application of context-aware computing and semantic web technologies for representing and reasoning about context information as a first level entity in the design and implementation of SOA governance infrastructures [22]; and (ii) the application of feedback loops [8], [15]. We also considered as a basis for software automation instrumenting run-time governance with self-adaptive capabilities to address uncertain and changing requirements of context management to preserve QoS properties. Sections V-VII provide further details on our research on context identification and specification, and the definition of dynamic monitoring strategies. Aspects related to the implementation of self-adaptive monitoring infrastructures are beyond the scope of this paper, but can be found in our previous papers on this subject [22], [26], [7], [24].

V. Context Taxonomy and Meta-Model

A. The Context Taxonomy

In order to control and govern context information in dynamic environments, we proposed a characterization of context information as a foundation for context identification and representation [23]. Our taxonomy, organizes context information along five main categories (cf. squared nodes in Fig. 3). These categories can be extended to support more concrete classifications according to particular problem domains.

Figure 3 presents a partial view of *SmartContext*, our general context taxonomy extended with basic context information categories required for the implementation of dynamic context monitoring to support run-time governance in SOA and cloud environments.

Individual context includes anything that can be observed about an isolated subject (i.e., the state of the subject). These observations can be classified into four subcategories [23]. In particular, *artificial context*, describes the state of entities resulting from human actions or technical processes (e.g., a service component or a hardware resource in a cloud environment). *Location context*, physical or virtual, involves all the information about the place of settlement or activity of an object. Instances of a *physical location* are absolute coordinates of a user location, the position of an object with respect to another. An example of *virtual location* is the notion of Uniform Resource Identifier (URI) useful for instance to identify computational resources such as web services. *Time context* concerns timeliness of subjects. Time context can be either *definite* or indefinite. The former represents time frames with specific start and stop points (i.e., a definite duration). In contrast, the latter expresses a recurrent event which is happening while another situation is taking place. In other words, it is impossible to know its duration in advance. *Activity context* answers questions regarding future, current, and past goals as well as actions and tasks performed by an object. An example of activity context relevant to SOA governance is the execution of a specific method in an interface implementation.

Our taxonomy can be extended to define even more particular categories according to specific problem domains. For instance, the *operational faults* category (cf. the gray oval in Fig. 3) can be further extended according to domain specific taxonomies such as the one for dependability and security proposed by Avižienes et al. [1]. Our proposed context taxonomy defines the foundational vocabulary for the definition of machine-readable representations of context information in order to support the management of the context information lifecycle. We have tested the suitability of SmartContext by implementing an RDF-based [20] context representation and its corresponding reasoning rules using semantic web technologies [5], to support context-aware user-centric web interactions in *smart commerce* applications domains [25], [24].

B. The Context Management Meta-Model

For realizing dynamic monitoring needed for run-time governance, the representation of context information that can affect the behavior of services must include not only context entities and monitoring metrics, but also the relationships among the involved entities, as well as infrastructural context management aspects, such as context sources, provisioning mechanisms, and monitoring conditions. In light of this, we proposed a context management meta-model [22] intended to support the instantiation of context models that can be integrated into dynamic monitoring infrastructures supported by feedback loops. Figure 4 provides a simplified version of our context management meta-model. An appropriate model for supporting dynamic context management must enable the instantiation of all the entities (contextual and infrastructural) required to implement monitoring strategies at run-time. In general the *MonitoringCondition*, *Threshold* and *Observation* meta-classes provide a way of instantiating classes for representing SLA metrics; the *ContextEntity* meta-class supports the instantiation of any context entity that is relevant to monitor SLOs satisfaction; *AcquisitionMechanism* and *Source* support the instantiation of context sensors; and *ActionGuarantee* and *PostCondition* enable the implementation of enforcement mechanisms.

VI. Managing Context as a First-level Entity

The management of SLAs is one of the main activities of SOA governance [13]. However, current approaches for specifying machine-readable SLAs lack an explicit definition of contextual entities and context-monitoring objectives beyond the particular metrics for deciding on the accomplishment of service level objectives (SLOs) under static conditions [4]. If we intend to dynamically monitor context information for supporting run-time governance, context information that can affect the accomplishment of SOA governance objectives should be specified at negotiation time, in the same way as SLA parameters, metrics, and SLOs.

An SLA specifies the level of service contracted between the service consumer and the service provider [4]. The obligations of the agreement must define the service level objectives (SLOs), which can be specified as predicates over service level parameters, and action guarantees that define the actions to be executed whenever a service level objective is violated.

We propose the definition of machine-readable *context-driven SLAs*. Context-driven SLAs are service level agreements where context monitoring requirements (relevant context entities and context monitoring strategies) are explicitly

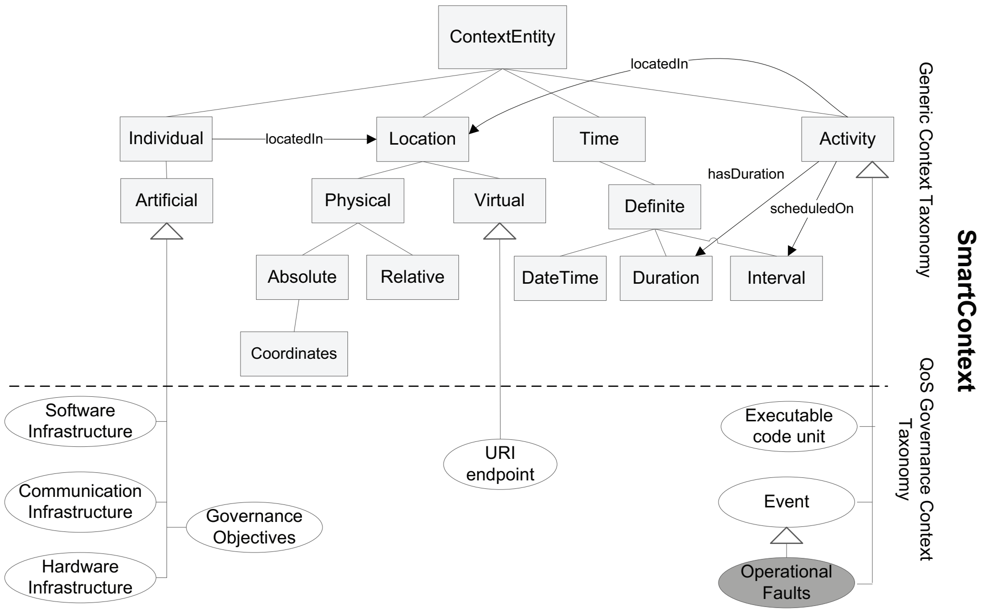

Fig. 3. SmartContext: Extended general context taxonomy for dynamic monitoring of QoS in run-time SOA governance

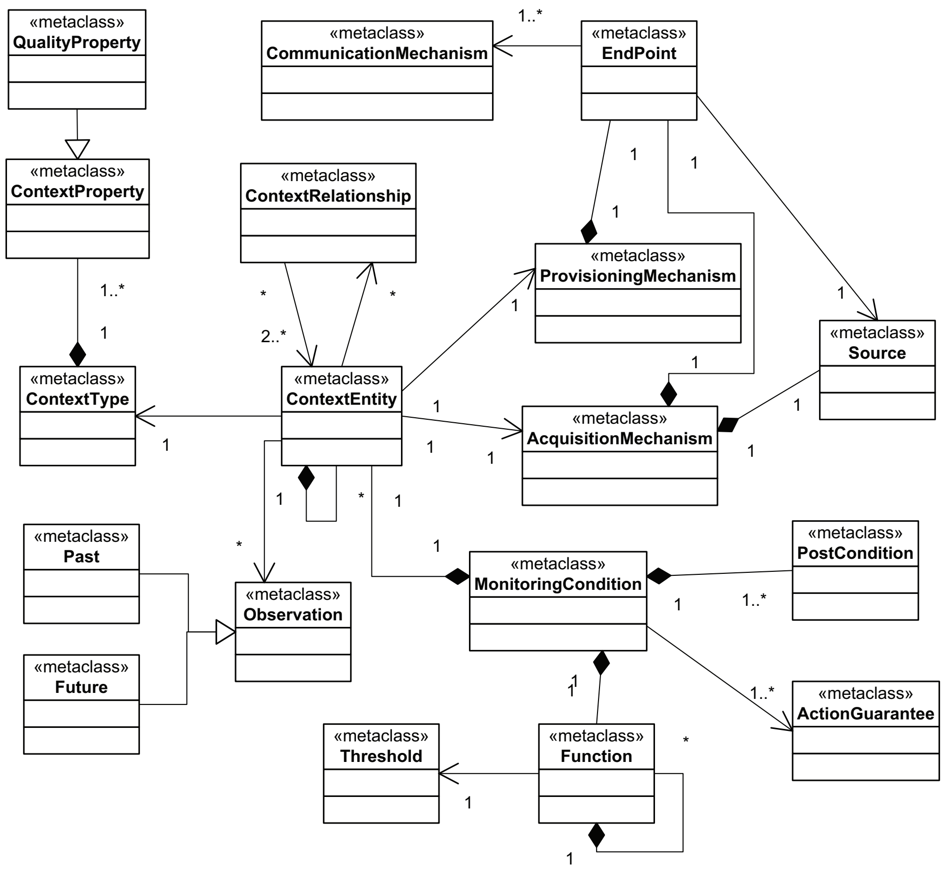
«metaclass» QualityProperty
«metaclass» CommunicationMechanism
«metaclass» EndPoint
«metaclass» ContextProperty
«metaclass» ContextRelationship
«metaclass» ProvisioningMechanism
«metaclass» Source
«metaclass» ContextType
«metaclass» ContextEntity
«metaclass» AcquisitionMechanism
«metaclass» Past
«metaclass» Observation
«metaclass» MonitoringCondition
«metaclass» PostCondition
«metaclass» Future
«metaclass» ActionGuarantee
«metaclass» Threshold
«metaclass» Function

Fig. 4. The context management meta-model (simplified version) [22]

specified as part of SLOs definitions. For this, we apply feedback loops [15] and our context-monitoring meta-model (cf. Fig. 4) to enable the adaptive behavior of monitoring infrastructures, and exploit semantic web technologies [3] together with our extensible context taxonomy (cf. Fig. 3) to provide the vocabulary required for context representation and reasoning. Machine-processable context monitoring requirements specified as part of SLAs are the basis for the implementation of dynamic monitoring strategies at run-time.

A. The Resource Description Framework (RDF)

Our approach exploits the *resource description framework (RDF)* [20] to support the representation of context monitoring requirements. Linked data uses RDF to make typed statements (predicates) that link arbitrary things in the world (objects and subjects) [3], [5]. In the same way, we propose an extension to RDF graphs to represent context entities and elements of the context management infrastructure required to implement dynamic monitoring strategies. In our approach (i) RDF URIs identify context entities and infrastructural elements for context management (e.g., services delivering context sensors and context providers), (ii) HTTP provides the vehicle to access these entities, and (iii) the RDF/XML syntax together with the available RDF processing tools (e.g., Jena [6]) and query languages (e.g., SPARQL [21]) provide the operational representation and applicable technologies required for supporting context acquisition, provisioning and reasoning at run-time. To support context reasoning in SOA governance and benefit from the semantic web principles, we are currently developing a definition of semantic rules built on top of *SmartContext*(our proposed context taxonomy).

B. Context-driven SLAs

In order to realize the management of context information as a first level entity for supporting run-time governance for SOA and cloud environments, we exploit our proposed context meta-model (cf. Fig. 4), and extensible context-taxonomy (cf. Fig. 3) to propose the definition of *context-driven SLAs*, and *contextual RDF graphs* to represent context monitoring strategies as follows:

Definition 1 (Context-driven SLA)**.** *A context-driven SLA* $SLA_{S_A QA_Y}$*, for a given service* S_A *and quality attribute* QA_Y*, is defined as a sequence of a finite arbitrary number of pairs,*

$$[(SLO_1, CMS_1), \ldots, (SLO_n, CMS_n)]$$

where the first element in each pair corresponds to a specific service level objective (SLO_i)*, and the second element represents the SLO corresponding context monitoring strategy* (CMS_i)*. An SLO is defined as a triple* (p, a, s)*, where p is an n-ary predicate used to evaluate a quality property on the variables specified as its parameters; a is an action that represents a guarantee to be executed in case the predicate evaluates to false; and s is a postcondition that must hold after executing action a. Conceptually, monitoring probes report events that may require updating either the monitoring infrastructure, the governed service-oriented system, or the governance objectives. Predicate p determines whether this update must be performed, action a represents an enforcement mechanism that must be executed to guarantee the SLO, and postcondition s provides the context facts that characterize the desired situation after the execution of the enforcement action.*

Definition 2 (Contextual RDF Triple)**.** *Given a set* U *of RDF URI references, where each RDF URI reference identifies an instance of any type derived from the context monitoring meta-model meta-classes as presented in Fig. 4; a subset* U_R *of RDF URI references* $(U_R \subset U)$*, where each of its elements identifies an instance derived from the meta-class* $ContextRelationship$ *defined in the meta-model as presented in Fig. 4; an set B of blank nodes as defined in RDF graphs [20]; and a set L of RDF literals instantiated from the types defined in any context taxonomy compliant with the SmartContext taxonomy (cf. Fig. 3), a* ***contextual RDF triple*** *is a triple* $(e_1, e_2, e_3) \in (U \cup B) \times U_R \times (U \cup B \cup L)$*, where* e_1 *is called the subject,* e_2 *the predicate, and* e_3 *is called the object.*

Definition 3 (Context Monitoring Strategy)**.** *A context monitoring strategy (CMS), also called a contextual RDF graph, is a set of contextual RDF triples. That is, we define a CMS or contextual RDF graph as a directed labeled RDF graph* $G = (N, A, M, source, target)$*, where* $N \equiv ((U - U_R) \cup B \cup L)$*,* $A \equiv U_R$*, M is a set of strings to name the elements of* U_R*, and* $source, target : A \times M \rightarrow N$*, the mappings to represent the corresponding source and target nodes of the arcs. The set of nodes* N *represents context entities and context monitoring infrastructural elements derived from the context-monitoring meta-model presented in Fig. 4, the set of arcs* A *represents context relationships between pairs of nodes, and the set of strings* M *represents the names of the context relationships. Therefore, a CMS is a regular labeled RDF graph such that its universe (the elements of* $U \cup B \cup L$ *that occur in the triples of G) is constrained by the types defined by the meta-model for representing context monitoring strategies presented in Fig. 4, and the context taxonomy in Fig. 3.*

The detailed definitions of the Resource Description Framework (RDF), its concepts, and abstract syntax are presented in the RDF W3C Recommendation [20].

VII. An Illustrative Example

Software-as-a-Service (SaaS) is one of the business models targeted by cloud computing environments. In SaaS models, service-oriented systems are the most widely applied mechanism to deliver software functionality as a service [28]. SaaS provides customers with several benefits such as maintenance and evolution supported by the cloud provider, high availability, pay-per-use, and low operation costs. To guarantee low operation costs and thus contracted offers, performance governance is an important concern. The mapping between cost and performance was discussed in Sect. III.

Suppose an SaaS cloud provider is interested in governing the efficiency of the service-oriented infrastructure with the goal of optimizing operational costs. For this, a performance SLA defines an SLO to guarantee a minimum efficiency of 0.9 for a particular *ServiceA*. The metric associated to the SLO is the time behavior metric presented in Sect. III-B $(TB = execution\ time/total\ service\ invocation\ time)$. The denominator, *total service invocation time*, represents the total it takes for the service to respond after the corresponding request. The numerator, *execution time*, indicates the time consumed for processing a given interface functionality. *ServiceA* is composed only of one task defined as one interface implementation, thus the numerator is the processing time required for executing that individual task (i.e.,$total\ service\ invocation\ time - waitingtime$). TB is in the range $0..1$, where higher values indicate a better measure of performance in terms of time efficiency. Finally, suppose that an action guarantee will trigger a self-optimizing feature that performs an on-line architectural reconfiguration to improve the system efficiency.

This illustrative example is used in the following subsections to explain how we can optimize run-time SOA governance with our approach by managing context information as a first level entity from negotiation to implementation, thus equipping governance feedback loops with dynamic monitoring capabilities. These subsections illustrate the usage of our proposed definitions for context-driven SLAs, contextual RDF triples, and context monitoring strategies (i.e., contextual RDF graphs).

A. The Performance Context-driven SLA

Using Definition 1, the performance SLA for *ServiceA* ($SLA_{S_A Performance}$) application example is defined as follows:

$$[(EfficiencySLO, CMS_{TimeBehavior})]$$

where the SLO is defined by the triple composed of the $minTimeEfficiency(TB, 0.9)$ predicate p, the $triggerSelfOptimization(Service_A, measuredEfficiency)$ guarantee action a, and the $(p = true)$ postcondition s. That is, the performance SLA defines an SLO to control the *ServiceA* time efficiency; the SLO must be evaluated by applying the metric time behavior TB, with a minimum expected efficiency of 0.9; in case the SLO is violated, a self-optimization action will be triggered to re-configure the system with the goal of taking the service efficiency to the desired levels. After the reconfiguration, the efficiency SLO must hold (i.e., the predicate must evaluate to true).

B. The contextual RDF triples

Using Definition 2,

$$(InvocationInterface, isInstanceOf, ExecutableCodeUnit)$$

represents an instance of a contextual RDF triple in the monitoring strategy for the efficiency SLO (cf. gray nodes in Fig. 5 and their corresponding arc). In contextual RDF triples, subjects (i.e., first element) and objects (i.e., third element) are either entities derived from the context monitoring meta-model meta-classes (except from the meta-class *ContextRelationship*), blank nodes, or literals compliant with the SmartContext taxonomy (cf. Definition 2). Predicates (i.e., second element) are relationships between subjects and objects. In this triple the context entity acting as the subject is the invocation interface of *ServiceA*. The predicate indicates that the invocation service interface is an instance of another context type, the object *ExecutableCodeUnit*.

For simplicity, the contextual RDF graph presented in Fig. 5 defines nodes as conceptual values instead of URIs. Table VII-B presents the mapping between each element in the graph and the elements of our context-monitoring meta-model and context taxonomy. Those elements with no mapping to context type (i.e, middle column) are elements required by the context monitoring infrastructure to realize the monitoring strategy (i.e. elements that are not classified as any of the context entity types defined by the taxonomy).

C. The Dynamic Context Monitoring Strategy

The contextual RDF graph representation of the monitoring strategy for the efficiency SLO of our illustrative example is presented in Fig. 5. We optimize run-time SOA governance with our approach by supporting dynamic deployment of monitoring strategies to address changes in monitoring requirements. Variations in monitoring requirements can be generated by changes in either the governed system, the governance objectives, or context entities.

Suppose that *ServiceA* is replaced by a chain of service compositions. With traditional static monitoring mechanisms, the governance of the performance SLA is compromised as the monitoring infrastructure was implemented to monitor the time efficiency of the original service (in this case the monitoring infrastructure was not designed to automatically monitor a different service task interface). By supporting dynamic monitoring, governance feedback loops are able to deal with changes in monitoring requirements at run-time. Our context taxonomy and context monitoring meta-model enable the design and implementation of monitoring infrastructures with automatic inference capabilities to configure and deploy new sensors and monitoring conditions at run-time. In our illustrative example, immediately upon changes in the composition of *ServiceA*, the infrastructure must be able to recognize that new *ExecutableCodeUnit* entities have arrived to the execution environment. That is, new task interfaces and possibly new services must now be monitored. Then, by applying RDF-based rules, the infrastructure will be able to infer, based on the current monitoring strategy (i.e., the contextual graph presented in Fig. 5), the meta-model and the context taxonomy, that (i) the new entities to be monitored, as the original monitored task interface, are also instances of *ExecutableCodeUnit* context; and (ii) as *ExecutableCodeUnit* is *subClassOf ActivityContext*, then the new entities have a *hasDuration* relationship with a *Time* context entity. As a result, the original monitoring strategy can be adapted to deploy a new time sensor to gather time context from the recently discovered task interfaces.

To realize the previous claims, we are working on the extension of our taxonomy and context monitoring meta-model with RDF-based context reasoning rules to support the automatic inference of sensor types and other kinds of infrastructure elements. These rules, extended from the basic semantic rules defined in RDF, are cornerstones for realizing context reasoning using our proposed contextual RDF graphs.

TABLE II
CONTEXT IDENTIFICATION AND CLASSIFICATION FOR THE APPLICATION EXAMPLE

Entity	Context Type	Meta-Class
Performance SLA	Individual/.../GovernanceObjectives	ContextEntity
Efficiency SLO	Individual/.../GovernanceObjectives	ContextEntity
SLO PostCondition		PostCondition
AdaptationRequester		ActionGuarantee
Monitoring Condition		MonitoringCondition
Monitoring Function		Function
ExecutionTimeSensor	Individual/.../Duration	AcquisitionMechanism
InvocationTimeSensor	Individual/.../Duration	AcquisitionMechanism
TaskInterface	Activity/.../ExecutableCodeUnit	Source
InvocationInterface	Activity/.../ExecutableCodeUnit	Source
ServiceA	Individual/.../SoftwareInfrastructure	ContextEntity
ExecutableCodeUnit	Activity	ContextType
Time	ContextEntity	ContextType
ActivityContext	ContextEntity	ContextType

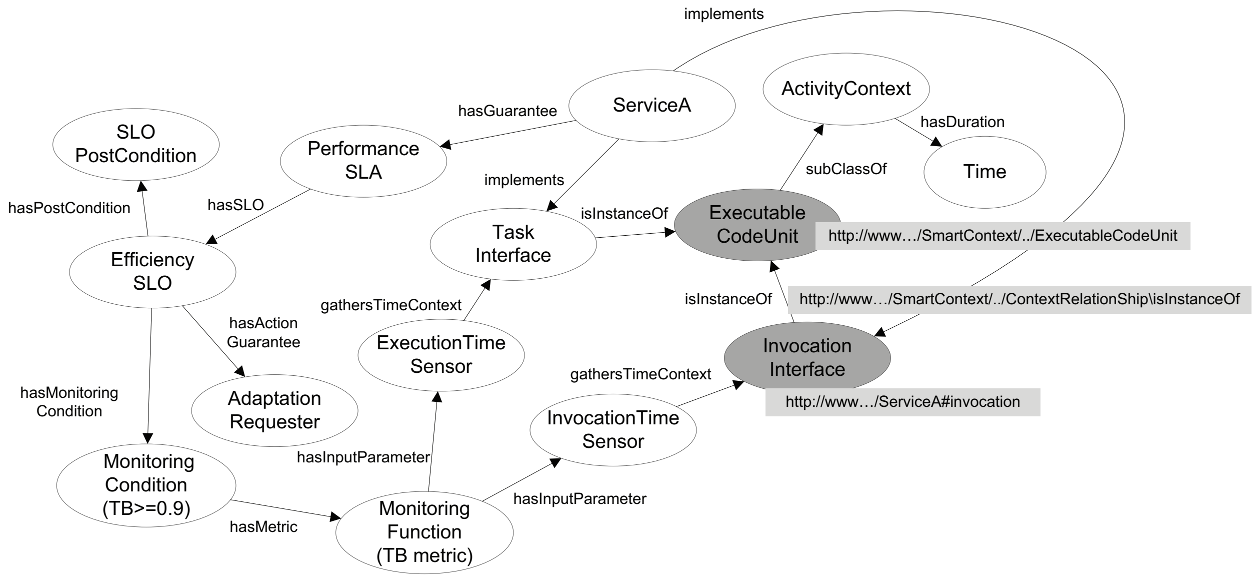

Fig. 5. The context monitoring strategy for the application example as a contextual RDF graph (simplified version)

VIII. RELATED WORK AND FINAL REMARKS

Advancing dynamic monitoring involves significant research challenges to optimize not only run-time SOA governance, but the adaptive behavior of any software system affected by or dependent on internal or external stimuli. To address many of their requirements, such systems must be instrumented with effective context management mechanisms. Particularly in SOA governance, the integration of dynamic monitoring strategies onto SLAs is a necessity to (i) guarantee the relevance between monitoring infrastructures and governance objectives, and (ii) control the consistency of governance objectives with execution environments. We propose a machine-readable specification of monitoring requirements for QoS preservation intended to enable the inference of new monitoring strategies at run-time according to changes in context entities. Our approach instruments run-time governance with control mechanisms to manage the consistency between governance objectives and monitoring instrumentation under dynamic execution conditions. Monitoring infrastructures based on our *contextual RDF graphs* and their explicit mapping with governance objectives are able to identify changes in monitoring requirements to reconfigure themselves accordingly at run-time.

Other approaches have been proposed to deal with SLA management at run-time. For instance, Zeng et al. proposed a high-performance QoS monitoring system for web services where the relevant context is observed from service operations defined as operational events [29]. Another example is the mapping between SLAs and QoS requirements of business processes proposed by Stantchev and Schröpferas, as part of SLA specification in grid and cloud environments [18]. At negotiation time, business process requirements are compared to infrastructure capabilities to define SLOs. Enforcement mechanisms are executed based on service replication. However, despite the general agreement about the important role of context information in preserving desired system properties, contextual conditions are generally assumed as static observations that not require dynamic monitoring instrumentation to be controlled. To our knowledge, the problem of monitoring the preservation of system properties while managing context as a first level entity throughout the system lifecycle, to guarantee the consistency between monitoring strategies, governance objectives, and context situations, is still an unsolved challenge.

Acknowledgments

This work was funded in part by the National Sciences and Engineering Research Council (NSERC) of Canada (CRDPJ 320529-04 and CRDPJ 356154-07), IBM Corporation via the CSER Consortium, Icesi University, Cali, Colombia, and University of Victoria, British Columbia, Canada.

References

[1] A. Avižienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 11–33, 2004.

[2] M. Barbacci, M. H. Klein, T. A. Longstaff, and C. B. Weinstock, "Quality attributes," CMU/SEI, Technical Report CMU/SEI-95-TR-021, 1995.

[3] T. Berners-Lee, W. Hall, J. A. Hendler, K. O'Hara, N. Shadbolt, and D. J. Weitzner, "A framework for web science," *Foundations and Trends in Web Science*, vol. 1, no. 1, pp. 1–130, 2006.

[4] P. Bianco, G. Lewis, and P. Merson, "Service level agreements in service-oriented architecture environments," Carnegie Mellon University Software Engineering Institute, Tech. Rep. CMU/SEI-2008-TN-021, 2008. [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/08tn021.cfm

[5] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data—the story so far," *International Journal on Semantic Web and Information Systems*, vol. 5, no. 3, pp. 1–22, 2009. [Online]. Available: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jswis.2009081901

[6] J. J. Carroll, I. Dickinson, C. Dollin, A. Seaborne, K. Wilkinson, and D. Reynolds, "Jena: Implementing the semantic web recommendations," in *Proceedings 13th International World Wide Web Conference (WWW 2004)*, 2004, pp. 74–83.

[7] M. E. Frincu, N. M. Villegas, D. Petcu, H. A. Müller, and R. Rouvoy, "Self-healing distributed scheduling platform," in *Proceedings IEEE International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011)*. USA: IEEE Computer Society, 2011, pp. 225–234.

[8] H. Giese, Y. Brun, J. D. M. Serugendo, C. Gacek, H. Kienle, H. Müller, M. Pezzè, and M. Shaw, "Engineering self-adaptive and self-managing systems," in *Software Engineering for Self-Adaptive Systems*, ser. LNCS, B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., vol. 5525. Springer, 2009, pp. 47–69.

[9] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[10] J. Y. Lee, J. W. Lee, D. W. Cheun, and S. D. Kim, "A quality model for evaluating software-as-a-service in cloud computing," in *Proceedings Seventh ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2009)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 261–266. [Online]. Available: http://dx.doi.org/10.1109/SERA.2009.43

[11] G. A. Lewis and D. B. Smith, "Service-oriented architecture and its implications for software maintenance and evolution," in *Proceedings Frontiers of Software Maintenance (FoSM 2008)*. IEEE Computer Society, 2008, pp. 1–10. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4659243

[12] G. A. Lewis, D. B. Smith, N. Chapin, and K. Kontogiannis, "MESOA 2009: Proceedings 3rd International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems," CMU/SEI, Tech. Rep. CMU/SEI-2010-SR-004, 2010. [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/10sr004.cfm

[13] G. A. Lewis, D. B. Smith, and K. Kontogiannis, "A research agenda for service-oriented architecture (SOA): Maintenance and evolution of service-oriented systems," CMU/SEI, Tech. Rep. CMU/SEI-2010-TN-003, 2010. [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/10tn003.cfm

[14] H. A. Müller, P. Gupta, R. Desmarais, A. Rudkovskiy, N. M. Villegas, Q. Zhu, and L. Nigul, "SOA governance optimizes the business and evolution of service-oriented systems," in *Proceedings 3rd International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2009)*, no. CMU/SEI-2010-SR-004. Carnegie Mellon University Software Engineering Institute, 2010, p. 67. [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/10sr004.cfm

[15] H. Müller, M. Pezzè, and M. Shaw, "Visibility of control in adaptive systems," in *Proceedings 2nd International Workshop on Ultra-Large-Scale Software-Intensive Systems (ULSSIS 2008)*. 30th IEEE/ACM International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, 2008, pp. 23–26.

[16] M. P. Papazoglou and W.-J. Heuvel, "Service oriented architectures: approaches, technologies and research issues," *The VLDB Journal*, vol. 16, pp. 389–415, July 2007. [Online]. Available: http://dx.doi.org/10.1007/s00778-007-0044-3

[17] G. Raines, "Cloud computing and SOA," in *Systems Engineering at MITRE: Service-oriented architecture (SOA) series*. The MITRE Corporation, 2009, no. MTR090026, pp. 1–12.

[18] V. Stantchev and C. Schröpfer, "Negotiating and enforcing qos and slas in grid and cloud computing," in *Proceedings 4th International Conference on Advances in Grid and Pervasive Computing (GPC 2009)*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 25–35.

[19] H. Strowd and G. Lewis, "T-Check in System-of-Systems Technologies: Cloud Computing," Carnegie Mellon University Software Engineering Institute, Tech. Rep. CMU/SEI-2010-TN-009, 2010. [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/10tn009.cfm

[20] The World Wide Web Consortium (W3C), "Resource Description Framework (RDF): Concepts and Abstract Syntax," http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, 2004.

[21] ——, "SPARQL Query Language for RDF," http://www.w3.org/TR/rdf-sparql-query/, 2008.

[22] N. M. Villegas and H. A. Müller, "Context-driven adaptive monitoring for supporting SOA governance," in *Proceedings 4th International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2010)*. Carnegie Mellon University Software Engineering Institute, 2010, to appear. [Online]. Available: http://www.sei.cmu.edu/workshops/mesoa/2010/main.htm

[23] ——, "Managing dynamic context to optimize smart interactions and services," in *The Smart Internet: Current Research and Future Applications*, ser. LNCS, M. Chignell, J. Cordy, J. Ng, and Y. Yesha, Eds., vol. 6400. Springer, 2010, pp. 289–318.

[24] N. M. Villegas, H. A. Müller, J. C. Munoz, A. Lau, J. Ng, and C. Brealey, "A dynamic context management infrastructure for supporting user-driven web integration in the personal web," in *Proceedings the 2011 Conference of the Center for Advanced Studies on Collaborative Research, Canada (CASCON 2010)*. ACM, 2011, in press.

[25] N. M. Villegas, H. A. Müller, J. Ng, and A. Lau, "Managing dynamic context to enable user-driven web integration in the personal web," in *Proceedings 1st Symposium on the Personal Web*, 2010. [Online]. Available: http://research.cs.queensu.ca/home/cordy/SPW/Proceedings/UvicPWS-ContextMgment.pdf

[26] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A control-engineered reference model to optimize context relevance in self-adaptation," 2011, in evaluation to be published in Software Engineering for Self-Adaptive Software Systems, 2nd Dagstuhl Seminar on Software Engineering for Self-Adaptive Systems.

[27] ——, "A framework for evaluating quality-driven self-adaptive software systems," in *Proceedings 6th international symposium on Software engineering for adaptive and self-managing systems (SEAMS 2011)*. New York, NY, USA: ACM, 2011, pp. 80–89. [Online]. Available: http://doi.acm.org/10.1145/1988008.1988020

[28] A. Weiss, "Computing in the clouds," *netWorker*, vol. 11, pp. 16–25, 2007. [Online]. Available: http://doi.acm.org/10.1145/1327512.1327513

[29] L. Zeng, H. Lei, and H. Chang, "Monitoring the QoS for Web Services," in *Proceedings 5th international conference on Service-Oriented Computing (ICSOC 2007)*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 132–144.