EXPLORATION OF TECHNOLOGIES FOR AUTONOMIC DEPENDABLE SERVICE PLATFORMS

Eila Ovaska[1], Liliana Dobrica[2], Anu Purhonen[1] and Marko Jaakola[1]
[1]*VTT Technical Research Centre of Finland, Kaitoväylä 1, 90571, Oulu, Finland*
[2]*University Politehnica of Bucharest, Splaiul Independentei 313, 060042, Bucharest, Romania*

Keywords: Self-adaptive, Service architecture, Ontology, Dependability, Modeling.

Abstract: A city is smart if it can provide ambient services for citizens and especially for professionals who have to tackle emergency situations, including small and wide scale accidents and incidents. These ambient services embody intelligence of autonomic systems based on heterogeneous execution platforms enhanced with services that provide mechanisms for self-adaptation of dependable applications. This paper explores enabling technologies of autonomic dependable service platforms from three viewpoints: i) architectural options, ii) ontology models for services, context and dependability, and iii) modeling methods and practices for achieving high quality service platforms and intelligent applications. The main findings are summarized as a set of research items that need to be carried out for achieving an autonomic dependable service platform.

1 INTRODUCTION

Besides situation based information, smart cities also provide quality-critical services for their communities, i.e. high quality services for professionals such as firemen, medical service providers, police, etc. These workers are responsible for real-time reaction in emergency situations. An emergency situation – caused by an extensive accident - means that the environment immediately changes, and therefore, the type and amount of information required for decision making and actions are also changing. These changes affect on the use and availability of computing and communication resources and the applications and services they provide for problem solving.

Smart city environments heavily rely on a multitude of sensor networks, embedded systems and devices that produce a large amount of data to be analyzed and reacted on in the short run by the security and safety monitoring processes executed by critical information systems. In this paper, we focus on heterogeneous systems that embody large scale sensor networks, embedded systems, mobile devices and enterprise systems. On the one hand, sensors interact with other nodes in various ways and communication may be periodic or ad-hoc over wired and/or wireless networks. Moreover, computers differ in their architecture and computing resources, such as CPUs, operating systems, processing power, amount of memory, energy requirements, etc. For example, motes that are tiny devices powered by a battery and featuring low-power wireless communication capability bring challenges for application developers. On the other hand, information systems bring tight quality requirements, which are to be fulfilled in any case, preventing human and economical damages, by reacting with adaptive and autonomous behavior to changing situations and disappearing resources.

The development of smart cities benefits from the findings and experiences gained from worldwide sensor webs (Balazinka et al., 2007) and service oriented architectures, also applied to sensor webs (Chu and Buyya, 2007). However, smart city applications require more; the platform shall be able to adjust its behavior based on defined dependability requirements, users' goals, quality of services and quality of available resources. Thus, this adaptation requires intelligence that enables real-time identification, reasoning and proactive reaction on alerts.

The objective of this paper is to explore the existing software technologies that are applicable for developing a service platform that is able to make autonomously the needed corrective and preventive actions in abnormal situations, and thereby provide a dependable infrastructure upon which adaptive applications can easily be developed and deployed. In particular, we focus on how dependability of applications can be guaranteed in ad-hoc situations.

Ovaska E., Dobrica L., Purhonen A. and Jaakola M..
EXPLORATION OF TECHNOLOGIES FOR AUTONOMIC DEPENDABLE SERVICE PLATFORMS.
DOI: 10.5220/0003497301150124
In *Proceedings of the 6th International Conference on Software and Database Technologies* (ICSOFT-2011), pages 115-124
ISBN: 978-989-8425-76-8

The main contributions of this paper are 1) the options of adaptive service architectures for autonomic dependable service platform, 2) the inventory of potential ontologies that could be exploited in the development of a dependable service platform, and 3) the approaches applicable in the development of quality critical and situation based smart city applications. In summary, self-adaptation is based on context-awareness, realized as situation based behavior that takes into account the functional and quality properties of the environment and system itself, and the needs of system's users.

The structure of the paper is as follows. Section 2 explains the concept of an autonomic dependable service platform. Section 3 explores existing adaptive service architectures and platforms. Section 4 introduces a selected set of ontologies for representing services, context and dependability. Section 5 discusses modeling methods and techniques applicable for modeling context and dependability. Section 6 summarizes our findings, and conclusion closes the paper.

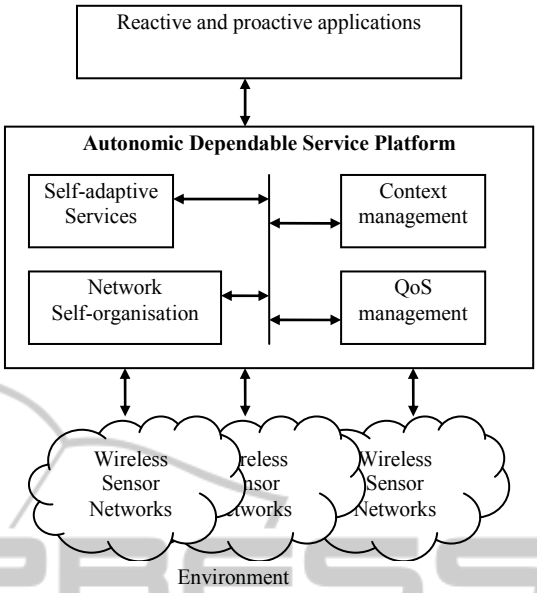

Figure 1: Overview of an autonomic dependable service platform.

2 THE CONCEPT

The increasing amount of cheap multi-purpose sensors enables enhancing applications with intelligence that adapts their behavior reactively and proactively (Figure 1). Sensor data gathered from the physical environment is clustered and merged with the system's internal state to define the application context. Thereafter, an application adapts its behavior based on the context, the quality of used services and available resources. Even though applications are founded on self-adaptive platform services, dependable applications require even more; the physical environment is also to be reorganized. Due to high cost of manual reorganization, sensor networks ought to be self-organizing wireless networks.

An autonomic dependable service platform is a key enabler of dependable smart city applications with situation based behavior and ability to exploit and tolerate the evolution of environment, technology and application fields.

Therefore, the following technologies embody a great potential for autonomic service platforms:

- Service oriented architectures are applied to information systems, embedded systems, mobile devices and sensor networks. Thus, self-adaptation based on service orientation would be a widely exploitable cross-domain solution.
- Ontologies are used for sharing domain knowledge among application and platform developers. Moreover, ontology orientation is exploited for developing reusable assets and managing the evolution of systems and their applications.
- Quality of service management is the mandatory requirement because of the purpose of the platform: to execute dependable applications on top of an autonomic service platform.
- Model driven development helps in sharing knowledge, information and artifacts in an unambiguous way, understood by both people and machines. Thus, models play an important role at design time and at run time.

3 SELF-ADAPTATION TECHNOLOGIES

3.1 Self-*ilities of Autonomic Systems

An autonomic system has six characteristics (Salehie and Tahvildari, 2009) also called self-*ilities:

- Reflexivity. The system must have knowledge of its capabilities, boundaries and interdependencies, and be aware of its possible configurations and their impact on particular quality requirements.
- Self-configuring. The system provides increased responsiveness by adapting to dynamic changes occurred internal of the system or in the external operating environment.

- Self-optimizing. The system provides operational efficiency by tuning resources and balancing workload.
- Self-healing. The system provides resiliency by discovering and preventing disruptions as well as recovering from malfunctions.
- Self-protecting. The system secures its assets by anticipating, detecting and protecting against attacks.
- Adapting. The core of the system is a control loop - sensing, decision making, and acting. The adaptive mechanisms are typically inspired by work on machine learning, multi-agent systems, and control theory.

Adaptive service platforms most often support self-configuring or self-optimization (Nahrstedt et al., 2001); (Zeng et al., 2004). However, recently self-healing has started to receive more attention (Baresi et al., 2008) (Cardellini et al., 2009c). As the system is autonomic in its 'normal' operation, it should be able to survive failures and to adjust system's characteristics to altering loads and resources autonomously (Botts et al., 2008).

3.2 Adaptive Middleware Architecture

The quality management in service-oriented systems requires additional features of the service brokering. QoS Broker utilizes a centralized approach where requests from clients are handled by a QoS-aware broker that evaluates each request using a performance model (Menasce and Dubey, 2007). In a quality assurance framework the brokers are created on-demand and all the service providers and consumers can have their own brokers (Robinson et al., 2008). AgFlow has a separate service composition manager that forms a composite service from proper sub-services, requested from the service broker (Zeng et al., 2004).

When quality management is added to the system it may cause that all the services are required to be changed to support the new middleware. VOLARE (Papakos et al., 2009) adds an adaptation middleware between Web services and the broker module that monitors the resources and context of a device, and adapts service requests accordingly. It also adapts the QoS levels advertised by service providers, to realistically reflect each provider's capabilities at any given moment. The adapted service descriptions and advertisements are syntactically identical to un-adapted versions, allowing interoperability with non-VOLARE nodes.

In the ubiquitous environments the time spent for quality management is critical. Consequently, the quality-aware service discovery can be divided into two levels (Cardellini et al., 2009a). First, the service provisioning level identifies the actual pool of concrete services that will be used to implement the component functionalities so that the user's end-to-end requirements are fulfilled and the service broker's utility function is maximized. At the second phase, the role of the service selection level is to determine, from the pool identified by the service provisioning, the actual concrete services which are bound to each incoming user request. The service provisioning can operate at a slower pace than the service selection.

MUSIC platform (Rouvoy et al., 2009) supports self-adaptation in ubiquitous and service-oriented environments. It provides an adaptation planning framework for managing the frequent and unexpected changes in the execution context of mobile applications. The purpose of the adaptation planning framework is to evaluate the utility of alternative configurations in response to context changes, to select a feasible one for the current context and to adapt the application accordingly.

The control loop of quality management can be roughly decomposed into monitoring, analyzing, planning and execution (Kephart and Chess, 2003). *Monitoring* means collecting the data needed for QoS adaptation from the system under interest. For example, in (Robinson et al., 2008) monitors are created at runtime to transparently intercept requests and responses between consumers and providers. *Analyzing* is the phase, where the collected data is combined to form proper QoS metrics, and possibly also predictions of future states are made. Predictions are used for finding out quality violations proactively (Robinson et al., 2008); (Papakos et al., 2009). *Planning* is the control phase, where the required action is selected. Planning can be an aggregate of local and global level reasoning, (Mokhtar et al., 2007). Local QoS requirements are filled by individual capabilities and global QoS requirements by the service composition. The optimization problem can be solved using, for example, models (Robinson et al., 2008) (Cardellini et al., 2009b) or fuzzy logic (Nahrstedt et al., 2001).

Execution of the decisions made may include adaptations at two different levels (Nahrstedt et al., 2001): 1) the resource management level performs application-neutral adaptation, and 2) the service management level is responsible of the application adaptation. The application adaptation can be about adjusting application components and configurations, or about selecting appropriate service providers for services.

Sometimes it is not possible to find any service providers that can fulfill the quality requirements set by the client. In that case negotiation is needed. In (Cardellini et al., 2009a) the service provisioning level takes care of the negotiation with the service provides so that the actual service selection is made faster. In (Robinson et al., 2008), the consumer broker negotiates with the provider brokers.

The existing adaptive middleware solutions seem to cover collectively all the main features required for a dependable services platform. For example, the MUSIC platform is able to take into account both context and quality issues and in addition, it is designed for ubiquitous and service-oriented environments. However, although it seems to support self-configuring and self-optimizing it is unclear how it would suit to an self-healing and self-protecting environment.

The performance and dependability of the existing platforms themselves are not yet clear. Some of them have been designed for ubiquitous environments, but they have been applied only in restricted contexts. Especially, more information is needed about how these systems would be able to support the management of large amounts of data in a short time frame required by the critical services in the smart city environment.

3.3 Enhanced Intelligence

An intelligent system is commonly thought to have a capability of learning. The goal of learning is to improve the performance of a system with respect to its environment. Machine learning paradigms can be divided into three major areas (Könönen, 2004):

- Supervised learning: A teacher, which knows the correct input-output pairs, provides these to the system, which is learning. The system tries to emulate the teacher's behavior and also generalize unseen data.
- Unsupervised learning: No teacher, and thereby no correct outputs exist in the learning process.
- Reinforcement learning: Neither here are correct outputs known, but the system learns those by interacting with its environment – the mechanism is called rewarding.

Supervised and unsupervised techniques have both training phases, although the unsupervised version has no labels – correct 'answers' of the training data – available (Jayaraj et al, 2008). A complete machine learning method includes steps of selecting a candidate model, and then estimating parameters for it. The estimation is done with a learning algorithm and available data. In practice, supervised learning utilizes often an error function, which should be naturally minimized. Unsupervised learning uses clustering; similarity of elements in the same cluster should be maximized, and similarity of elements in different clusters should be minimized.

In the formal model of the reinforcement learning, the system has a (discrete) state, which perceives either completely or partially, a group of actions possible in that state, and a reward which is received when a new state is entered. The system's behavior and knowledge of the environment are modeled with a function. Usually learning is not about maximization of direct reward belonging to the state transition, but long-term performance. Reinforcement learning can be applied to proactively adapting the service platform for stress peaks caused by users, overwhelming data or/and increased attacks.

Prediction provides four kinds of improvements to self-adaptation (Cheng et al., 2009): prevents unnecessary self-adaptation, reduces disruption from incremental adaptation, enables pre-adaptation to seasonal behaviour, and improves overall choice of adaptation. Smart city applications would benefit from proactive capabilities, for example, with using the sensor information for discovering activity patterns that might lead to emergency situations. In that way, it could be possible to act on the situations before they occur and possibly prevent them. The similar approach is common in smart-home systems. For example, CASAS (Rashidi and Cook, 2009) is an adaptive smart-home system that utilizes machine learning and data-mining techniques in order to detect activity patterns, generate automation policies for those patterns, and also adapt to the changes in those patterns.

Learning capabilities can be also used for self-protection. For example, an approach for wireless anomaly based intrusion detection and response system uses learning for detecting complex malicious attacks (Fayssal et al., 2008). Training sets are used by the system to generate rules for the behaviour to be considered normal. Those rules are used during runtime to detect complex wireless attacks and generate counter measures to protect one or more wireless resources and the privacy of their users. Fast recovery without human intervention requires proper policy management mechanisms and automated ways to learn and derive policies (Fuad, 2010). Unlike the current self-healing systems that most often diagnose and heal failures after they have occurred rather than anticipating failures, in consequence-oriented diagnosis and recovery the host predicts or diagnoses the possible consequences from the symptoms (Dai et al., 2009).

4 ONTOLOGIES

4.1 Service Ontologies

Ontologies are used to represent knowledge in a uniform way that machines are able to process. Ontologies provide knowledge for describing the required and provided capabilities of a service, ability and rights of achieving a service, and the quality guaranteed for a service. The eXtensible Markup Language (XML), Resource Definition Framework (RDF) (Hayes, 2004), and Web Ontology Language (OWL) (McGuinness and van Harmelen, 2004) schema provide a basis for service description languages and ontologies, such as Web Ontology Language for Services (OWL-S) (Barstow et al., 2004), Web Service Modeling Ontology (WSMO) (WSMO, 2004), and Internet Reasoning Service (IRS) (Motta et al., 2003), which in turn provide building blocks for service semantics. The above mentioned service ontologies describe functionality of services. Another set of ontologies focuses on service context and quality properties. However, existing service ontologies focus mainly on Web services, and none of them provides complete support for service descriptions as required in adaptive service platforms. After analysis of the existing ontologies (Kantorovitch and Niemelä, 2008), a conclusion is that the existing service ontologies have to be enhanced.

In the software architecture field, the use of viewpoints is a community-wide accepted approach to cluster stakeholder-related concerns into a single view. This principle can be lent to describing semantics of services. The use of multiple views is a necessity; the interests of stakeholders differ, application domains differ, and service functionality and quality differ according to the usage and execution contexts. Moreover, different application domains, for example, information systems and pervasive computing applications, require modeling languages that take into account the characteristics of a domain by providing a notation that can be enhanced and adapted by domain specific extensions. The approach used in software product line engineering, namely the separation of commonality and variability, would be a viable approach that solves the problems in separation of common and domain specific semantics, and the integrated use of the defined service ontologies.

4.2 Context Ontologies

(Dey and Newberger, 2009) define context: 'Context is any information that can be used to characterize the situation of an entity.' Understanding of context information is heavily improved during the last five years. Recently published articles, e.g. (Bettini et al., 2010) (Meier et al., 2009) (Kapitsaki et al., 2009) indicate that knowledge on specification, modeling and usage of context information might be mature enough for realizing context-aware smart space applications. Typically, context information has three dimensions; physical, computational and user context (Bettini et al., 2010). In order to assist for achieving interoperability on the levels that concern context data and change of context, the context shall (Preuveneers and Berbers, 2008) i) have a complete domain coverage and terminology; ii) be expressive and without semantic ambiguity; iii) be processed without complexity; and iv) be evolvable.

Three types of context modeling and reasoning approaches (Bettini et al., 2010) have been identified: an object-role based model, a spatial model and an ontology based model. The object-role based approach supports various stages of the software engineering process. Its weakness is a 'flat' information model, i.e. all context types are represented as atomic facts. The spatial context model is well suited for context-aware applications that are mainly location-based, like many mobile applications. The main consideration of the spatial context model is the choice of the underlying location model. Relational location models are easier to build up than geographic location models. SOUPA (Chen et al., 2005), as the only standardized context ontology, provides the most promising starting point for enhanced context ontology of smart cities. Therefore, an initial version of a context ontology introduced in (Pantsar-Syväniemi et al., 2010) is an enhancement of SOUPA. This context ontology defines three levels; i) the physical context deals with raw context data gathered from the environment by sensors; ii) the digital context exploits physical context information and merges it with the system's internal context information related to applications and information; and finally iii) the situation context clusters and abstracts the digital context information in a way that it matches to the application in hand and the preferences of its user. Thereafter, the situation context is used for adapting the application according to the view of the whole context information that relates to the application.

4.3 QoS and Dependability Ontologies

Quality of Service (QoS) has a traditional meaning as a property of communication technologies, including throughput, latency, jitter, error rate, availa-

bility, and network security. In service oriented architectures, QoS is defined as dependability, maintainability, usability and scalability (O'Brien et al., 2007). For end-users, QoS is the degree to which an executed service meets its quality requirements. Quality characteristics are often referred as non-functional requirements, although many of them (e.g. performance and dependability) are intertwined with functionality of software. Typically, existing quality ontologies have a specific focus. For example, a quality ontology may deal with one or few quality attribute(s) in defining, managing, or matching quality properties. However, to guarantee QoS requires comprehensive support for defining and managing all the relevant quality attributes of services, at design time and at run time.

There are several studies on QoS ontologies related to quality of Web Services. In (Anderson et al., 2007), an overview of resilience knowledge base (RKB) is described, in which dependability and security ontology is derived from the taxonomies of (Avizienis et al., 2004) and developed specifically for the RKB. The ontology is represented in OWL and incorporates 166 terms related to Dependability and Security, and 23 to Systems. Moreover, there are QoS attribute ontologies and QoS-aware discovery solutions based on service level agreements (Menasce, 2002). Some papers also discuss performance, dependability and service cost as well as mechanisms of their aggregation (Yang et al., 2006) (Lock and Dobson, 2009). Other dependability-related metadata included into the description is i) the development metadata, i.e. information about service developers and implementation technology, and ii) the deployment metadata, i.e. information related to the hosting organization, location, deployment environment, network connection capacity, etc.. Adding this meta-information will allow clients to decide how to use services by decreasing common mode failures.

Dynamic operational state parameters, such as current service load, CPU and memory usage, network loading, etc. might also be added to the extended description. Extending a service description with dependability metadata will bring us closer to a dependable semantic service platform.

5 MODELING APPROACHES

5.1 Service Modeling

Service modeling can apply ontology based service engineering, software engineering or/and domain engineering modeling techniques. Knowledge engineering applies ontologies for capturing and structuring topic knowledge shared across people, organizations, computers and software. Several methods for ontology development exist, e.g. METHONTOLOGY (Fernandez-Lopez et al., 1997), and a set of languages, such as XML, RDF, and OWL that can be applied to represent knowledge in a machine readable format. Moreover, OWL-S as a specific service description language can be used for describing service semantics.

Semantic Web, Ontology Engineering, Semantic Annotations, Semantic Search, Intelligent Services (Modeling, Discovery and Integration) are standards from W3C (www.w3c.org) and FIPA (www.fipa.org) for describing semantics models. The use of standards and open source tools (as W3C standards and OWL in Protégé 2000 environment) helps in sharing and using ontologies. Therefore, open standards and open source tools are the key enablers of semantics modeling. The advancement in open source tools has greatly improved the ability to test and build ontologies from scratch or/and to reuse existing ontologies.

Application programming interfaces for ontology languages provide programming language dependent means to load ontologies, manipulate the ontology classes and relations, perform reasoning, and provide persistent storage for the model. Jena and OWLS API are the most popular Java frameworks for building semantic Web applications. These tools provide an application developer with a programming language for working with ontologies. Reasoning tools, such as FaCT++, Pellet, and RacerPro, provide a standardized XML interface to description logics systems. These tools help in ontology testing and in the development of application level intelligence based on ontologies described in OWL. Domain ontology specific editors such as OWLS Editor and WSMO design studio help in creating error free semantic descriptions based on a specific ontology.

Domain specific modeling addresses the specifics of an application domain in the meta-models from which a domain specific language is derived (Kelly and Tolvanen, 2008). Although UML2 is a generic modeling language, it also provides constructs to extend the language with domain specific concepts. Thus, UML2 enhanced with domain specific ontologies that extends the language with service, context and dependability ontologies makes it applicable to the development of autonomic service platforms.

5.2 Context Modeling

Context-aware service engineering can be classified into two classes (Kapitsaki et al., 2009); language based approaches and model-driven approaches. Language based approaches such as context-oriented programming and aspect-oriented programming follow the separation of concerns; applications are kept context-free and context is handled as a first-class entity of the programming language while separate constructs are used to inject context-related behavior into the adaptable skeleton of an application. Context-aware aspects programming is one step further; the aspects are driven by context, i.e. a particular aspect may or may not be executed depending on the context of use.

When trying to solve the complexity of context-aware applications, the approaches for context modeling and reasoning, namely object, spatial and ontology based have the following strengths and weaknesses;

- The object-role based approach supports various stages of the software engineering process but has an information model not suitable for modeling context information of smart cities.
- The spatial context modeling suits well for location based applications, like mobile applications. The drawback is the effort the special context model takes to gather and keep up to date the location data of the context information. Thus, this model is suitable for those smart city applications that do not have critical performance and dependability requirements as in emergency situations.
- Ontological context models provide clear advantage both in terms of support for heterogeneity and interoperability. User-friendly graphical tools make the design of ontological context models viable to developers that are not particularly familiar with description logics. However, there is very little support for modeling temporal aspects in ontologies. However, the main problem might be that reasoning with OWL poses serious performance issues.

Programming based on the spatial context models (Meier et al., 2009) uses a small set of predefined types for composing context information. Thus, it is a topographical approach for modeling a space, i.e. the context of actors is modeled as a geometric shape based on a sequence of coordinates. This enables actors to independently define and use potentially overlapping spatial context in a consistent manner, when relationships between spatial objects are defined implicitly, i.e. as the positions of the spatial objects shapes within the coordinate system. Thus, the spatial programming model enables efficient integration of heterogeneous systems into a global smart space. Although the programming model might be a too sophisticated and overestimated approach for developing smart cities, it is a feasible enabler for self-organized sensor networks.

As mentioned, all approaches have weaknesses that make their use such as unfeasible in autonomic dependable service platforms. However, the generic model for context monitoring and situation based adaptation of application logic (Dey and Newberger, 2009) is part of a viable solution, as described in (Pantsar-Syväniemi et al., 2010).

5.3 Dependability Modeling

Dependability is to be considered from three viewpoints; as a system property; as a service capability, and a failure free operation. Dependability of a computing system is its ability to deliver a service that can justifiably be trusted. Dependability of a service is its behavior as it is perceived by the service user(s). Based on the definition of failure, an alternate definition of dependability exists, which complements the other definitions in providing a criterion for adjudicating whether the delivered service can be trusted or not: the ability of a system to avoid failures that are more frequent or more severe, and outage durations that are longer than is acceptable to the user(s). The first two definitions relate to the system and software design and implementation. The third one relates to the space's ability to survive under failures. Thus, it relates to self-healing and self-protecting, the characteristics of autonomic systems.

We understand dependability as a general concept that manages four quality attributes; reliability, availability, security and safety. Safety is not common in smart cities but extremely important in safety-critical systems, e.g. in trains and airplanes. Thus, when using sensor information for making context-aware smart city applications, we focus on reliability, availability and security. Especially, our interest is on how to deal with these quality properties in a situation based manner and how to assure that quality requirements are met when ad-hoc situation based adaptations are made.

Survivability concerns autonomic systems and is a system's capability to fulfill its mission, in a timely manner, in the presence of attacks, failures or accidents. There are two aspects of survivability: protection and adaptation. Survival by protection refers to run-time security management. Survival by adaptation is an ability of a system to adapt its behavior to the changes that occur either in the system or externally in the operating environment and users' re-

quirements (Tarvainen, 2007). Thus being self-adaptive and self-protecting, the dependable service platform should support survivability.

Security mechanisms like access control and encryption attempt to ensure survivability by protecting applications from harmful, accidental or malicious changes in the environment. Applications can also survive by adapting themselves to the changing conditions. Survival by adaptation typically involves monitoring and changing the quality goals so that they can be achieved. In order to exploit architectural design knowledge for runtime adaptation, the following activities should be supported; a) identification of the internal and external contexts of the system, b) reasoning the change of context, c) reasoning the activities to be taken in order to achieve the quality goals defined, and finally, d) reconfiguring the system in a manageable way. In proactive adaptation, these activities have to be made before they occur. Thus, appropriate learning techniques are used for predicting the system's behavior and making it survivable by proactive actions.

Dependability modeling has four main modeling phases. First, the semantics of dependability is described at the design time by applying the quality-driven architecture design and quality analysis methodology (Ovaska et al., 2010). As a result, the sub-attributes of dependability are described as separate ontologies utilized for defining quality requirements. Second, quality requirements are mapped to the elements of software architecture models (Niemelä et al., 2008). Third, the designed architecture is evaluated in order to detect whether required quality is met or not. Fourth, quality of the implemented software is measured and compared to requirements. In practice, the above described approach with supporting techniques and tools can be exploited but the approach needs enhanced middleware services that able to use the design knowledge, represented in the service, context and dependability ontologies, in monitoring, reasoning and adapting dependability of smart city applications.

6 DISCUSSION

As presented there are several technologies and solutions for the development of autonomic dependable systems composed of heterogeneous subsystems such as sensors, networks, and storage systems. While these subsystems can boost dependability, research is required towards a holistic approach that will consider large complex networked systems as a whole. Therefore, the focus should be on an approach that hides the heterogeneity of sensor technologies, provides proactive strategies for QoS adaptation and easiness to use and understand the sensor web and its applications. Moreover, the key enabling technologies required for the future pervasive computing environments to be explored extensively in the ongoing and future research projects are:

- *Semantics modeling (cf. section 4.1)*. A novel semantics modeling technique is required. It shall consist of i) stakeholder-centric views, ii) support for the generic ontologies such as quality attribute ontology of dependability and performance, iii) a core ontology of the technology domain (e.g. sensor webs), iv) application specific domain ontologies, and v) integrated orchestration of the developed ontologies in service engineering.
- *Dependability metrics and measuring techniques (cf. section 4.3)*. Uniform quality metrics for execution qualities are required. Moreover, common measuring techniques are needed for dependability (i.e. reliability, availability and security) and performance. A reuse-oriented approach is to be established for exploiting the dependability ontology in modeling semantics of autonomic dependable service platforms and applications.
- *Proactive adaptation (cf. section 3.3, 4.2, 5.2 and 5.3)*. A novel approach is required for measuring, monitoring, adapting and predicting of system's behavior from quality point of view. QoS-driven proactive adaptation requires innovative solutions for 1) managing semantic descriptions at runtime; 2) deriving quality indicators from basic QoS measurements; and 3) enhanced adaptation and learning algorithms. Furthermore, the proactive QoS adaptation mechanism has to take into account the constraints of the used technology; how to deal with context awareness and resource constraints of computing and communication environment.
- *Dynamic semantic middleware (cf. section 3.1, 3.2, 5.1 and 5.2)*. There is a need for a dynamic semantic middleware that allows proactive service discovery, service composition and negotiation, and evolution management of cross-domain service platforms intended for heterogeneous networked systems, devices, actuators and appliances used for environmental monitoring. The middleware shall deal with interoperability of sensors, devices and services in heterogeneous multi-vendor environments.

7 CONCLUSIONS

In this paper we explored the existing technologies

applicable for use in the development of autonomic dependable service platforms that embody the technical challenges of pervasive computing environments, the business challenges of the multi-vendor product development and the quality of service challenge of trusted services that insist on a dependable and high efficient service platform.

We scoped our work by smart cities, the context where intelligence of services is benefited the most and where end-users should be supported with novel software and service engineering technologies. Moreover, we adopted an approach that exploits and enhances legacy systems because making running systems more intelligent and self-adaptive is a big enough challenge.

As a conclusion, we identified four research topics that need extensive research and developments, namely i) semantics modeling, ii) dependability metrics and measuring techniques, iii) proactive adaptation architectures, and iv) middleware support for handling dynamism of self-organizing (ad-hoc) sensor networks. Our future work will address these topics.

ACKNOWLEDGEMENTS

This work is supported by the SOFIA/Artemis project, co-funded by EU, Tekes, and VTT and the Smash project, funded by VTT. The work of Liliana Dobrica was supported by Romanian Scientific Council CNCSIS –UEFISCSU, project number PNII – IDEI 1238/2008.

REFERENCES

Anderson T., Andrews Z. H., Fitzgerald J.S., Randell B., Glaser H., Millard I.C., 2007. The ReSIST Resilience Knowledge Base. *37th Annual IEEE/IFIP Intl. Conf. on. Dependable Systems and Networks (DSN'2007)*.

Avizienis A., Laprie J-C., Randell B., Landwehr C., 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11-33.

Balazinka M., Deshpande A., Flanklin M. J., Gibbons P.B., Gray J., Nath S., Hansen M., Liebhold M., Szalay A. and Tao V., 2007. Data management in the worldwide sensor web. *Pervasive Computing*, June-April, 30-40.

Baresi L., Guinea S. and Pasquale L., 2008. Towards a unified framework for the monitoring and recovery of BPEL processes. In: *Testing, Analysis and Verification of Web Software, TAV-WEB,* pp. 15-19.

Barstow A., Hendler J. and Skall M., 2004. OWL Web Ontology Language for Services, W3C. http://xml. coverpages. org/ni2004-01-08-a.html

Bettini C., Brdiczka O., Henricksen K., Indulska J., Niclas D., Ranganathan A., and D. Riboni. 2010. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*.

Botts M., Percivall G., Reed C. and J. Davidson, 2008. OSG® Sensor Web Enablement: Overview and High Level Architecture. *LNCS 4540*, Springer-Verlag, pp. 175–190.

Cardellini V., Casalicchio E., Grassi V., Presti F. L. and. Mirandola R, 2009a. A scalable approach to QoS-aware self-adaption in service-oriented architectures. In: *QSHINE,* pp. 431-447.

Cardellini V., Casalicchio E., Grassi V., Presti F. Lo and Mirandola R.,2009b. QoS-driven runtime adaptation of service oriented architectures. In: *The 7th Joint European Soft. Eng. Conf. and ACM SIGSOFT Symp. on the Foundations of Soft. Eng.*, pp. 131-140.

Cardellini V., Casalicchio E., Grassi V., Presti F. Lo and R. Mirandola, 2009c. Towards self-adaptation for dependable service-oriented systems. In: *Architecting Dependable Systems VI,* pp. 24-48.

Chen H., Finin T. and Joshi A., 2005. The SOUPA Ontology for Pervasive Computing. *Whitestein Series in Software Agent Technologies*, Springer.

Cheng, S-W, Poladian, V., Garlan, D., Schmerl, B., 2009. Improving Architecture-Based Self-Adaptation through Resource Prediction, In: *Self-Adaptive Systems*, LNCS 5525, Springer-Verlag, pp.71-88.

Chu X. and Buyya R., 2007. Service oriented sensor web. *Sensor Networks and Configuration*, Springer-Verlag, 51-74.

Dai, Y., Xiang, Y., Zhang, G., 2009. Self-healing and Hybrid Diagnosis in Cloud Computing. In: M.G. Jaatun, G.Zhao, and C.Rong (Eds.): *CloudCom 2009*, LNCS 5931, Springer-Verlag, pp- 45-56.

Dey A. K. and Newberger A., 2009. Support for Context-Aware Intelligibility and Control. In: *CHI 2009, Programming Tools and Architectures*, USA.

Fayssal, S., Al-Nashif, Y., Uk Kim, B., Hariri, S., 2008. A Proactive Wireless Self-Protection System. In: *ICPS'08*, July 6-10, Sorrento Italy, ACM, pp.11-20.

Fernandez-Lopez M., Gomez-Perez A., Juristo N., 1997. Methontology: from ontological art towards ontological engineering. In: *Procs. Spring Symposium on ontological engineering of AAAI*.

Fuad, M.M., 2010. Issues and Challenges of an Inductive learning Algorithm for Self-healing Applications. *The 7th Intl. Conf. on Information Technology: New Generations (ITNG10)*, IEEE Press, pp. 264-269.

Hayes P., 2004. RDF Semantics, W3C, http://www.w3.org /TR/rdf-schema/

Jayaraj A., Venkatesh T. and Murthy C.S.R., 2008. Loss classification in optical burst switching networks using machine learning techniques: improving the performance of TCP. *IEEE Journal on Selected Areas in Communications*, vol.26, no.6, pp.45-54.

Kantorovitch J. and Niemelä E., 2008. Service Description Ontologies. *Encyclopedia of Information Science and Technology*, Vol. VII, pp. 3445-3451.

Kapitsaki G., Prezerakos G., Tselikas N., and I. Venieris. 2009. Context-aware service engineering: A survey. *The Journal of Systems and Software,* 83, 1885-1297.

Kelly S, and Tolvanen J., 2008. *Domain-Specific Modelling: Enabling Full Code Generation*. Wiley.

Kephart J. O. and Chess D. M., 2003. The vision of autonomic computing. *Computer* 36(1), pp. 41-50.

Könönen V., 2004. *Multiagent reinforcement learning in Markov games: asymmetric and symmetric approaches*, Doctoral thesis, Helsinki University of Technology, Report D8, Espoo, Finland.

Lock R. and Dobson G., 2009. Developing an ontology for QoS. *Dependability interdisciplinary research Collaboration (Internal Annual Project Conference)*, Nesc. (National e-Science centre).

McGuinness D. and van Harmelen F., 2004. OWL Web Ontology Language Overview, W3C., http://www.w3.org/TR/owl-features/

Meier R., Harrington A., Beckmann K., and Cahill, V., 2009. A framework for incremental construction of real global smart space applications. *Pervasive and Mobile Computing*, 5, 350-368.

Menasce D. A. and Dubey V., 2007. Utility-based QoS brokering in service oriented architectures. In: *IEEE Intl Conf on Web Services, ICWS 2007,* pp. 422-430.

Menasce D. A., 2002. QoS Issue in Web Services. *IEEE Internet Computing*, vol. 6, issue 6, pp. 49-68.

Mokhtar S. B., Georgantas N. and Issarny V., 2007. COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support, *Journal of Systems and Software* 80(12), pp. 1941-1955.

Motta E., Domingue J., Cabral L.and Gaspari M., 2003. IRS-II: A Framework and Infrastructure for Semantic Web Services. *ISWC 2003*, in Sanibel Island, USA.

Nahrstedt K., Xu D., Wichadakul D. and Li B., 2001. QoS-aware middleware for ubiquitous and heterogeneous environments. *IEEE Communications Magazine,* vol. 39, no. 11, pp. 140-148.

Niemelä E., Evesti A. and Savolainen P., 2008. Modeling Quality Attribute Variability. *3rd International Conference on Evaluation of Novel Approaches of Software Engineering, ENASE 2008*.

O'Brien L., Merson P. and Bass L., 2007. Quality Attributes for Service-Oriented Architectures. *International Workshop on Systems Development in SOA Environments* (SDSOA'07).

Ovaska E., Evesti A., Henttonen K., Palviainen M. and Aho P., 2010. Knowledge Based Quality Driven Architecture Design and Evaluation. *Information and Software Technologies*, 52 (6), 577-601.

Pantsar-Syväniemi S., Simula K., Ovaska E., 2010. Context-awareness in smart spaces. *1st International Workshop on Semantic Information for Smart Spaces*, SISS 2010, IEEE, pp. 1023-1028.

Papakos P., Rosenblum D. S., Mukhija A. and Capra L., 2009. VOLARE: Adaptive web service discovery middleware for mobile systems. *2nd Intl DisCoTec on Context-Aware Adaptation Mechanisms for Pervasive and Ubiquitous Services* (CAMPUS 2009).

Preuveneers D., and Berbers Y., 2008. Internet of Things: A Context-Awareness Perspective, *The Internet of Things: From RFID to the Next Generation Pervasive Networked Systems,* CRC Press, 287-307.

Rashidi P. and Cook D. J., 2009. Keeping the resident in the loop: adapting the smart home to the user. *IEEE Transactions on Systems, Man and Cybernetics, Part A (Systems and Humans),* vol. 39, no.5, pp. 949-959.

Robinson D., Kotonya G. and Department C., 2008. A self-managing brokerage model for quality assurance in service-oriented systems. In: *11th IEEE High Assurance Systems Eng. Symp., HASE,* pp. 424-433.

Rouvoy R., Barone P., Ding Y., Eliassen F., Hallsteinsen S., Lorenzo J., Mamelli A. and Scholz U., 2009. MUSIC: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In: *Softw. Eng. for Self-Adaptive Systems,* pp. 164-182.

Salehie M. and Tahvildari L., 2009. Self-Adaptive Software: Landscape and Research Challenges. *ACM Trans. on Autonomous and Adaptive Systems*, 4(2).

Tarvainen P., 2007. Adaptability evaluation of software architectures; a case study. *31st Annual IEEE Int. Computer Software and Applications Conf. (COMPSAC 2007)*. IEEE Computer Science.

WSMO, 2004. WSMO studio, http://www. wsmostudio.org/

Yang S., Lan B. and Chung J-Y., 2006. Analysis of QoS Aware Web Services. *Intl. Comp. Symp. on Web Technologies and Information Security* (ICS) .

Zeng L., Benatallah B., Ngu A.H., Dumas M., Kalagnanam J., Chang H., 2004. QoS-aware middleware for web services composition, *IEEE Trans. Software Eng.* 30(5), pp. 311-327.