INTERNETINĖS TECHNOLOGIJOS

Quality of Services in the Context of Internet of Services

Jérémy Besson

Institute of Mathematics and Informatics
Akademijos g. 4, LT 69121 Vilnius, Lithuania
E-mail: contact.jeremy.besson@gmail.com

Albertas Čaplinskas

Institute of Mathematics and Informatics
Akademijos g. 4, LT 69121 Vilnius, Lithuania
E-mail: alcapl@ktl.mii.lt

In the recent decade the component technologies have been evolved from object-oriented to service-oriented ones. A lot of different component models, component architectures, and component description languages have been proposed. To systematize and conceptualize the proposed approaches is a hard and complicated task. The paper aims to contribute to the solution of this problem. It analyses the most important approaches for specifying the quality of services (QoS) delivered by the software components. The analysis is made from the point of view of the Internet of Services (IoS).

1. Introduction

Software Engineering (SE) community faces a period of revolutionary changes. Advanced Enterprise Architectures, Cloud computing, and Services Oriented Architectures (SOA) change many traditional SE methods, approaches and techniques. One of the main SE next year challenges is the Internet of Services (IoS). In the IoS "a service can be characterised by the fact that the service consumer does not own the service, and by the existence of a service level agreement (SLA), either explicit or implicit, between a provider and a consumer. The SLA provides the shared context between different parties to a relationship based on the service. The relationships between provider and consumer can range from long-lived associations to dynamic single-use scenarios." (NESSI, 2006). This model requires to provide and guarantee a certain Quality of Service (QoS). QoS becomes a part of any contract between service consumers and providers. It should be expressed in SLA's terms. So, the mechanisms to define, to negotiate, and to monitor the QoS are essential components of the IoS. A number of approaches were proposed on how to specify and to manage end-to-end QoS in different architectures and environments. However, it is far not obvious which of those approaches and in what extend are suitable in the IoS environment. This paper analyses the most important approaches of the proposed and aims to evaluate the relevance of these approaches for the QoS management in the IoS environment.

The remainder of this paper is organized as follows. Section 2 briefly discusses the concept of QoS. Section 3 analyses and evaluates the most important approaches to the QoS management. Section 4 concludes the paper.

2. Quality of Services

The concept of QoS was developed for the telephony and other network services. Later on, it has been extended for packet-switched net-

works, including computer networks, and for such network resources as, for example, Web servers. Approximately of the same time QoS was defined for software components. However, the understanding and usage of QoS in networking and in SE were significantly distinct. In networks the QoS is a dynamical phenomenon. It takes into account the resource reservation control mechanisms. A network or protocol that supports the QoS may agree on such a traffic contract that the application software can reserve the required capacity in the network nodes. In component-based SE the QoS is a static concept that addresses the extra-functional properties of software components. The QoS specifications are mainly used to make the design decisions during the system development phase. More than ten years ago the concept of QoS was extended to be applicable to open distributed processing (ODP) systems, such as Corba or dotNet based systems. In ODP environments the QoS combines static and dynamic features because component systems are dynamically composed of distributed components communicating with each other through the network. In this context, QoS takes into account network-related characteristics as well as quality characteristics of software components. Although the list of system components is not a static one and permanently changes, it is defined at any particular moment of time. It means that only the components registered as a system's components can communicate one another. Besides, all the system's components should be built using the same technology and should run inside the same run-time environment (e.g. Corba environment). Similarly, the concept of the QoS is also interpreted in the current SOA systems, for example, in a system composed of Web-services. The IoS takes different point of view, because in the IoS the whole Internet is looked upon as a repository of components. It means that components are mainly developed independently, not taking into consideration any particular system. They might be built using different technologies and run in different run time environments. A system is a temporal, maybe one-off, composition of services. A service consumer (person, device, component or application) and a service provider negotiate at the run-time and agree on a QoS-contract dynamically, the QoS-policies can be changed at run-time and the behaviour cannot be well defined a priori. The QoS must be monitored dynamically and corrective operations may be taken to fulfil the agreed contract. The end-to-end QoS must take into account all architectural levels, including hardware, software components, applications, run-time environments and heterogeneous networks.

3. Analysis and evaluation of the proposed approaches

3.1 METEOR-S

METEOR-S (Verma, 2005) is a Web Service Annotation Framework that involves creation and application of a broad variety ontologies related to data, function, non-functional/QoS and execution semantics to support the complete web process lifecycle. Meteor-S grows up from the WSDL-S (W3C, 2005) It extends the WSDL in order to add ontological concepts to it. It integrates and co-exists with current industry technologies, including Eclipse BPWS4J Editor and BPEL4WS Execution Engine. METEOR-S provides a mechanism to add QoS semantics to WSDL files. It also provides a QoS model that allows for the description of non-functional aspects of workflow components from a QoS perspective and a mathematical model that allows automatic computing of the overall QoS of a workflow. METEOR-S provides a constraint based process composition which constrains both the generic QoS parameters (time, reliability, etc.) and domain specific QoS parameters (e.g., supply time). Constraints are converted into linear equalities/linear inequalities over a set of discovered services, which are solved using integer linear programming methods. To sum up, METEOR-S is intended to be used in large scale distributed information systems. It concentrates on workflow systems and assumes that all components are web services that are designed and developed for a particular system.

So, this approach cannot be used directly in the IoS environment.

3.2 OWL-S

OWL-S (Martin, 2004) is an OWL (Grau et al., 2006) ontology to describe the Web service. It provides a core set of mark-up language constructs required to describe the services offered by service providers and the services needed by service consumers. OWL-S provides non-functional properties of services. It enables us to describe the semantics of web services and is intended to be used in the service-oriented environment, but it is devoted only to Web Services. OWL-S provides only few predefined non-functional properties, but an explicit extension mechanism may be used to define new properties. It provides an attribute used for rating a web service but does not provide any special mechanisms to describe the QoS. Just like METEOR-S, OWL-S assumes that all components are designed and developed for a particular system.

3.3 Quality of Services Modelling Language (QML)

QML (Frolund, Koisten, 1998) is a kind of interface definition language which was developed for defining multi-category QoS specifications of the components in distributed object systems. QML provides three abstraction mechanisms: contract types, contracts, and profiles. A contract type represents some non-functional aspect of the component. A contract is an instance of a particular contract type. A profile bounds the contract to a component interface, operation, and operation argument or operation result using the language element known as a profile. A client-server relationship could have two QoS specifications allowing negotiation of QoS dynamically between clients and servers in the distributed systems. QML enables us to specify the QoS and to negotiate the QoS dynamically. It is intended to be used in distributed object systems and is not applicable directly in the service-oriented environment.

3.4 The Quality Objects (QuO/QDL)

QuO/QDL (Pal et al., 2000) is a framework for providing QoS in network-centric distributed applications, including the embedded ones. It extends a distributed object computing framework that is implemented as a middleware for developing and adding adaptation and QoS awareness and control to those applications. QuO/QDL provides mechanisms to specify the desired QoS for distributed applications and to inform the applications about the obtained QoS and adaptation as QoS changes. QuO bridges the gap between the socket-level QoS and the distributed object level QoS and provides some ideas how to guarantee the required QoS under the failure circumstances (Rubel et all., 2006). It provides 3 aspect-oriented Quality Description Languages:

- *Contract Description Language*. CDL is used to describe the QoS contract between a client and a component, including the QoS required by the client, the QoS that the component expects to provide, regions of possible levels of QoS, the behaviour to invoke to adapt to or notify of changes in QoS, and interfaces that can be used to measure and control the QoS. The QoS properties are assumed to be the result of invoking instrumentation methods on remote objects. No formal constraints are placed on the implementation of these methods.
- *Structure Description Language*. SDL describes the internal structure of component implementation and the amount of resources they require. It allows specifying adaptation alternatives and strategies, based upon the QoS measured in the system, the behaviours to invoke for method calls and/or returns, and the QuO/QDL connections.
- *Resource Description Language*. RDL describes the available resources and those that will be used.

QuO/QDL enables us to specify and customize the QoS requirements for applications, for the system elements that must be monitored and

controlled to measure and provide QoS, and for the behaviour, which adapts to QoS variations that occur at run-time (Schantz et all., 2000). It is intended to be used in distributed object systems and is not applicable directly in the service-oriented environment.

3.5 Service Component Architecture (SCA Policy)

SCA Policy (SCA, 2007)provides a general approach to define components and to describe how they interact in the system, modelling their interactions as services, that is, separating their functionality and implementation technology (Chapel, 2007). SCA Policy provides a set of specifications that describe a model for building component systems, using a Service-Oriented Architecture (SOA). The most important of them are the assembly model specification, component implementation specifications, binding specifications, and the policy framework specification. The main elements of the policy framework are:

- *Intents*: Intent is a single abstract assertion about QoS. It can be qualified and satisfied by a variety of bindings and with many different ways of configuring those bindings. It is allowed to attach intents to any element used in the definition of components and composites. Intents allow us to start the design with abstract QoS requirements and to add deployment details later in the process. They are independent of any implementation technology or of any binding and can be attached to any element used in the definition of a component.
- *Profiles*: Profiles are aggregations of intent names or something like macro, which declares a single name for a collection of intents. They represent common sets of QoS requirements. A profile intent is satisfied if all the underlying intents are satisfied. Using high level policy profiles, SCA Policy simplifies the potential complexity of infrastructure policy for QoS (Marino, Rowley, 2009).
- *Policy sets*: At the deployment time intents are mapped to corresponding policy sets containing the specifics of technology. One or more policy sets can be attached to any SCA element used in the definition of components and composites. Each policy set contains one or more policies expressed in a particular policy description language. How the intents or policies are specified within an implementation depends on the implementation technology. Both intents and policy sets may be used to specify the QoS requirements.

The policy framework specification defines how to add configurable infrastructure services (security, reliable messaging, transactions, etc.) to the application systems. Bindings and policies allow separating business logic from the infrastructure. The policy assertion represents a requirement, capability, or an other property of behaviour. Some assertions are relevant to service selection and usage (e.g. QoS characteristics) (Marino, Rowley, 2009). To sum up, SCA Policy is a language and technology independent approach that consequently follows the principle separation of concerns and supports a uniform declarative binding abstraction. However, it is more about building the components that consume services than about building services. Up to date, the SCA is not adequate for mobile services. In addition, it provides only a limited support for the specification of the QoS and is more vendor-oriented than oriented to an SOA system architect. SCA is well-suited for authentication, confidentiality, integrity, message reliability, and transaction propagation. However it is not clear how adequate it will be for other QoS issues.

3.6 SLAng

SLAng (Lamanna et al., 2003) is an XML-based language for defining service level agreements (SLA). Any SLA is a part of contract between a service consumer and a service provider describing the required QoS. In this approach a SLA includes an end-point description of the contractors, contractual statements, and the

required QoS description and associated metrics. SLAng provides a format for QoS negotiation and contract specification and is designed to be appropriate as input to automated reasoning systems or QoS-aware adaptive middleware. It aims to facilitate different levels of QoS abstraction and defines 7 different types of SLA: between applications/web-services and components (application SLA), between a service provider and host (hosting SLA), between a host and storage service provider (persistence SLA), between application or host and Internet service providers (communication SLA), between component and web service providers (service SLA), between container providers (container SLA), and between network providers (networking SLA). SLAng identifies the need for different levels of expressiveness of horizontal and vertical SLAs. Horizontal SLAs are contracts that govern interaction between components. Vertical SLAs regulate the support that parties get from their underlying infrastructure. Each SLA defines the relationship of responsibility between a client and a server. Responsibilities are expressed in terms of end-point, contractual and SLS parameters, which are specific to the type of SLA (Lamanna et al., 2003). In order to represent services and SLAs, the SLAng uses the UML profile for QoS and Fault Tolerance, but redefines the QoS catalogue. UML and OCL are used to precisely define the meaning of SLAs. The semantics of the language is formally defined in terms of the behaviour of services and clients involved in service usage. SLAng is oriented to the e-business domain. It is not well-suited to define SLAs in the service-oriented environment, because the syntactic structure and semantics of SLAng are defined with reference to a model of the distributed system client-server architecture. However, it focuses not only on web services exclusively and defines a vocabulary to model a number of other Internet services, including Application Service Provision (ASP), Internet Service Provision (ISP), Storage Service Provision (SSP) and component hosting (Skene et al., 2004).

3.7 UML profile for QoS and Fault Tolerance

The UML profile (OMG, 2005) defines a number of UML 2 extensions to represent QoS and introduces extra-functional aspects in UML models. Later, these requirements should be allocated in the analysis model and implemented by the software architecture. Thus, the profile is intended to be used to specify the QoS of the component that is going to be developed. The QoS is defined as a set of perceivable characteristics expressed in a user-friendly language with quantifiable parameters that may be subjective or objective. The characteristics of quality and their parameters are based on user satisfaction and resource consumption. A quality characteristic includes a set of quality attributes that are the dimensions to express a satisfaction. The quantifiable level of satisfaction of a non-functional property is described by the appropriate quality level. Quality levels are used to describe contracts for the QoS provided by the component. To fulfil these contracts, the component requires some amount of system resources and appropriate quality levels of services delivered to this component by other components. These quality levels are expressed in the required quality contracts. Quality contracts are expressed in terms of the values associated to quality characteristics. The profile allows the corresponding QoS values to be attached to messages using a communication diagram. Some infrastructure should be provided for the management of QoS contracts in the run-time environment. Various approaches can be used for this aim. For example, the interface description language can be extended to support the description of quality contracts. QuO (Pal et al., 2000) is an example of such an approach. To sum up, the profile defines constructs to model the described concepts and define constraints on the QoS characteristics. It allows us to model QoS contracts in the client-server architecture that, on the one hand, describe the quality values, which can be supported by the server (provider- offered QoS), and the requirements that must achieve their clients (provider-required QoS), and, on the other hand, the quality required by the client (client-required

QoS) and the quality that the client ensures (client-offered QoS). If the provider does not support the required QoS required, the contract and the final quality must be negotiated. The profile introduces the QoS catalogue for storing specifications of general QoS characteristics and categories that can be reused in different projects and domains. This makes it possible to build the components using different technologies and to run them in different run time environments. However, all the components are assumed to be designed and developed for a particular system. Besides, the end-to-end QoS takes into account not all architectural levels in this approach.

3.8 UniFrame

The UniFrame approach (Raje, 2001) has been proposed for the open distributed object-computing environment. Later on this approach evolved and was adapted to the service-oriented environment (Olson et al., 2005). The approach is based on the so-called Unified Meta-Component Model (UMM) treating a system "as as a collection of large number of heterogeneous, interconnected, (possibly) mobile, and "smart" components" (Raje, 2001), which constantly discover one another in the network, offer and utilize services, and negotiate the cost and the QoS. The paradigm beyond the UniFrame approach combines "the principles of distributed, component-based computing, model-driven architecture, service and quality of service guarantees, and generative techniques" (Olson et al., 2005). It provides an iterative, incremental process for assembling a distributed computing system from services available on the network and emphasizes determining the QoS during this process. Components in the UMM have public multiple level interfaces describing not only their functional responsibilities, but also guaranteed QoS ratings. A component is described by six groups of attributes: inherent, functional, non-functional, cooperative, auxiliary, and deployment attributes. The inherent attributes contain the bookkeeping information about a component. The non-functional attributes represent the QoS parameters, along with their values that the component developer guarantees in a specific deployment environment. They may also indicate the effects of the deployment environment and usage patterns on the QoS values. The requirements specification of a component is written in a natural language and inter alia describes the required values of QoS parameters. Using two-level grammars (Briant, Lee, 2002), this specification is transformed into a formal specification allowing the generation of a component's interface. The QoS parameters as well as generation rules in the formal specification are expressed in the two-level grammars formalism. The generated interface also incorporates the QoS aspects of a component. Once all the required components are developed and deployed on the network, one must compose them into system. The system is described by a generative domain-specific model, which is used to generate the glue/wrapper interfaces between the required components. The static QoS parameters are processed generating the system by the two-level grammars. The dynamic parameters result in the instrumentation of generated code. The instrumentation is necessary for the run-time QoS metrics evaluation. (Raje et all., 2002). In order to automate the instrumentation of the generated code, a system behaviour model is required. In the UniFrame approach, the attributed event grammar formalism (Auguston, 1995) is used for modelling the system's behaviour. The system's execution is represented as a set of events connected by an event trace. Possible configurations of events within the event trace are described by a set of axioms and the dynamic QoS metrics are expressed as computations over the event trace. After the instrumentation of the generated code is completed, the components that can guarantee appropriate values of the dynamic QoS parameters are chosen from the set of available components, using a representative set of test cases. However, the proposed approach does not guarantee that these components will be able to provide the required QoS under failure circumstances. To sum up, UniFrame is intended to be applicable in an open distributed object-computing environment.

It allows specifying QoS at both the component and system levels. Like UML profile for QoS and Fault Tolerance, it provides the QoS catalogue for storing specifications of reusable QoS attributes including the metrics, the evaluation methodologies and the relationships with other QoS attributes. It also provides an infrastructure for the management of QoS contracts in the runtime environment. UniFrame SLA specifications rely on the specification of measurements in external ontologies that provide structured natural language descriptions of measurements. This approach fits to most of the IoS requirements. However, it does not guarantee that the components will be able to provide the required QoS under failure circumstances.

3.9 Web Service Level Agreement (WSLA)

WSLA (Ludwig et al., 2003) is an XML-based language to describe SLAs. It advocates the idea of individually negotiated customized SLA. A web service level agreement is an agreement between a service provider and a service customer and as such defines the obligations of the parties involved. The WSLA supports negotiations and deployment. Mechanisms can be specified to define how the QoS attributes are measured. It is allowed to create new metrics defined as functions over the existing metrics. It is supposed that all measurements must be provided by a web service encapsulating monitors. The monitors are external constructs to the language because no constraints are placed on their implementation. The involved parties are automatically informed when the service does not meet the QoS specified in the WSLA. WSLA supports negotiation and monitoring of QoS. However, it is meant only for web services.

3.10 Web Service Level Offering (WSLO) framework

WSLO (Tosic et al., 2002) allows the formal and unambiguous specification of prices, monetary penalties, management responsibilities and third parties, especially accounting parties. The main targets of the WSOL project are the creation of service offerings and the definition of QoS constraints. Another important design goal is a low run-time overhead achieved through defining classes of services instead of individually managed SLAs. WSOL also supports the reusability of specifications. This is realized by means of the concept of constraint groups and constraint group templates to include the formerly defined elements and import of elements defined in other WSOL files. Similarly as in the UniFrame, SLAs are specified using external ontologies providing natural language descriptions of measurements. Like QML, it provides a type system for SLAs. Similarly as the WSLA, this approach is dedicated only to Web Services.

3.11 Web Service Modeling Ontology (WSMO)

WSMO (Roman et al., 2006) is upper layer ontology for describing various aspects of Semantic Web Services. For complete item descriptions, each WSMO element is described by the properties that contain relevant, non-functional aspects. These are based on the Dublin Core Metadata Set and other service specific properties like versioning information, QoS information, as well the owner and financial information. WSMO discriminates between the component's QoS and the network related QoS. The latter represents the QoS mechanisms operating in the transport network which is independent of the service. It can be measured by network delay, delay variation and/or message loss. WSMO allows us to describe only predefined non-functional properties. It provides flexible extension but does not provide any explicit mechanisms for this aim. Besides, it is dedicated only to Web Services.

3.12 Web Services Policy Framework (WS-Policy)

The WS-Policy framework (Schlimmer, 2006) provides a simple grammar for combining various policies. It is a part of the core Web

services architecture specifications. It defines a XML-based model that is used to describe the policies associated with a Web service. In the WS-Policy, an assertion is the basic unit of policy. The meaning of each individual assertion is beyond the scope of the WS-Policy framework and should be part of another specification. QoS is one of the issues that potentially can be expressed using the WS-Policy. The WS-Policy proposes an interesting police-based approach to describe QoS. However, it is dedicated only to Web Services.

4. Conclusions

We still do not have well-established standards to define the QoS in the SOA and none of the analyzed approaches is appropriate to be used directly in the IoS environment. The analysis made demonstrates that separation of concerns and virtualization are the main trends in the evolution of component technologies. The bindings and policies completely separate the business logic from the infrastructure concerns. The SCA, QuO and some other approaches provide truly aspect-oriented solutions. The QuO, UniFrame, and other approaches suggest how to bridge socket-level, network-level, and system-level QoS. However, many problems remain unsolved as yet. The list includes service level agreements, measurement and evaluation of the dynamic QoS characteristics, legal issues of negotiations. Many approaches, including the SCA, remain at least partly object-oriented. Almost all the service-oriented approaches concentrate on the web services and pay no attention to other kinds of services.

Acknowledgments. The research is funded by the Lithuanian State Science and Studies Foundation under the contract No. S 09/2009

REFERENCES

AUGUSTON, M. (1995). Program behavior model based on event grammar and its application for debugging automation. In: Ducassé M. (ed.) *AADEBUG, 2nd International Workshop on Automated and Algorithmic Debugging, May 22–24, 1995, Saint Malo, France, Proceedings*. IRISA-CNRS, p. 277–291

BRIANT, B. R.; LEE B.-S. (2002). Two-level grammar as an object-oriented requirements specification language. In: *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, vol. 9, p. 280–289

CHAPEL, D. (2007). *Introducing SCA*. Chapell&Associates. URL: http://www.davidchappell.com/articles/Introducing_SCA.pdf

FROLUND, S.; KOISTEN J. (1998). *QML: A Language for Quality of Service Specification*. HP Labs Technical Reports.

GRAU, B. C.; HORROCKS, I.; MOTIK, B.; PATEL-SCHNEIDER, P. F. (2006). *OWL 1.1 Web Ontology Language*. Submission Request to W3C, submitted 19 December, 2006, W3C, URL: http://www.w3.org/Submission/2006/10/.

LAMANNA, D. D.; SKENE, J.; EMMERICH, W. (2003). SLAng: A Language for Service Level Agreements. In: *Proc. of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems*. IEEE Computer Society Press, p. 100–106.

LUDWIG, H.; KELLER, A.; DAN, A.; KING, R., FRANCK, R. (2003). *Web Service Level Agreement (WSLA) - Language Specification. IBM specification*, URL: http://www. research.ibm.com/wsla/WSLASpecV1-20030128.pdf

MARINO, J.; ROWLEY, M. (2009). *Understanding SCA*. Addison-Wesley Professional.

MARTIN, D. (ed.) (2004). *OWL-S: Semantic Markup for Web Services*. Member Submission 22 November 2004, W3C. URL: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

NESSI (2006). *NESSI Strategic Research Agenda*. Vol. 1 *Framing the future of the Service Oriented Economy*. Public Draft 1 -Version 2006-2-13 -Revision 3.1. URL: ttp://cordis.europa.eu/technology-platforms/pdf/nessi1.pdf

OLSON, A. M; RAJE, R. R.; BRYANT, B. R.; BURT, C. C.; AUGUSTON, M. (2005). UniFrame: a unified framework for developing service-oriented, component-based, distributed software systems. In: Stojanovic, Z., Dahanayake, A. (eds.) *Service-Oriented Software System Engineering: Challenges and Practices* (Chapter IV, p. 68–87). Hershey,PA: Idea Group Publishing.

OMG Standard (2005). *UML™ Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*. URL: http://www.omg.org/docs/ptc/05-05-02.pdf

PAL, P.; LOYALL, J.; SCHANTZ, R.; ZINKY, J.; SHAPIRO, R.; MEGQUIER, J. (2000). Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration. In: *Proc. of the 3rd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'2000), March, 2000*, p. 310–319, IEEE.

RAJE, R.R. (2001). UMM: Unified Meta-object Model for Open Distributed Systems. In: *Proc. of 4th IEEE International Conf. on Algorithms and Architecture for Parallel Processing, ICA3PP'2000*, Hongkong, p. 454–465.

RAJE, R. R.; BRYANT, B. R.; OLSON, A. M.; AUGUSTON, M.; BURT, C. (2002). A quality of service-based framework for creating distributed heterogeneous software components. *Concurrency and Computation: Practice and Experience*, vol. 14, p. 1009–1034.

ROMAN, D.; LAUSEN, H.; KELLER, U. (eds.) (2006). *Web Service Modeling Ontology (WSMO)*. WSMO Final Draft 21 October 2006. D2v1.3. URL: http://www.wsmo.org/TR/d2/v1.3/

RUBEL, P.; LOYALL, J.; SCHANTZ, R. E.; GILLEN, M. (2006). Fault Tolerance in a Multi-Layered DRE System: A Case Study. *Journal of Computers*, vol.1, issue 6, p. 43–52.

SCA (2007). *SCA Policy Framework*. Version 1.00, March 07, BEA Systems, Inc., Cape Clear Software, International Business Machines Corp, Interface21, IONA Technologies, Oracle, Primeton Technologies, Progress Software, Red Hat, Rogue Wave Software, SAP AG., Siemens AG., Software AG., Sun Microsystems, Inc., Sybase Inc., TIBCO Software Inc., URL: http://www.osoa.org/download/attachments/35/SCA_Policy_Framework_V100.pdf

SCHANTZ, R.; ZINKY, J.; LOYALL, J.; SHAPIRO, R.; MEGQUIER, J. (2000). Adaptable Binding for Quality of Service in Highly Networked Applications. In: *Proc. of SSGRR-2000, International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, July 31–August 6, 2000, L'Aquila, Italy*. URL: http://dist-systems.bbn.com/papers/2000/SSGRR/ssgrr.doc

SCHLIMMER, J. (ed.) (2004–2006). *Web Services Policy 1.2 – Framework (WS-Policy)*. W3C Member Submission 25 April 2006, BEA Systems Inc., International Business Machines Corporation, Microsoft Corporation, Inc., SAP AG, Sonic Software, VeriSign Inc. URL: http://www.w3.org/Submission/WS-Policy/

SKENE, J.; LAMANNA, D.; EMMERICH, W. (2004). Precise Service Level Agreements. In: *Proc. of the 26th International Conference on Software Engineering,* p. 179–188, IEEE Computer Society.

TOSIC, V.; PATEL, K.; PAGUREK, B. (2002). WSOL - Web Service Offerings Language. In: Bussler, Ch., Hull, R., McIlraith, Sh., Orlowska, M. E., Pernici, B., Yang, J. (eds.) *Web Services, E-Business, and the Semantic Web, LNCS*; Vol. 2512, London: Springer, p. 57–67

VERMA, K.; GOMADAM, K.; SHETH, A. P.; MILLER, J. A., WU, Z. (2005). *The METEOR-S Approach for Configuring and Executing Dynamic Web Processes*. Technical report No. 6-24-05, LSDIS, University of Georgia, Athens. URL: ttp://lsdis.cs.uga.edu/projects/meteor-s/techRep6-24-05.pdf

PASLAUGŲ KOKYBĖ PASLAUGŲ INTERNETO KONTEKSTE

Jérémy Besson, Albertas Čaplinskas

Santrauka

Per pastarąjį dešimtmetį komponentinės technologijos iš objektinių išsivystė į paslaugų technologijas. Buvo pasiūlyta daug įvairių komponento modelių, komponento architektūrų ir komponento aprašymo kalbų. Todėl sukurtų paradigmų ir metodų sisteminimas bei konceptualizavimas yra sudėtingas uždavinys. Straipsnio autoriai siekia prisidėti prie šio uždavinio sprendimo. Darbe analizuojami svarbiausieji programinių komponentų teikiamų paslaugų kokybės specifikavimo būdai. Analizė atliekama paslaugų interneto kontekste.