Databases and Information Systems VIII
H.-M. Haav et al. (Eds.)

doi:10.3233/978-1-61499-458-9-27

Requirements Engineering for Service-Oriented Enterprise Systems: Quality Requirements Negotiation

Audrone LUPEIKIENE[1] and Albertas CAPLINSKAS
Institute of Mathematics and Informatics, Vilnius University, Lithuania

Abstract. Service-oriented systems engineering (SoSE) counts its two-decade history. Nowadays it is a way of developing and deploying applications as well as the whole enterprise systems. Service-oriented requirements engineering (SoRE) as an integral part of SoSE and as a new requirements engineering subdiscipline faces a number of different kinds of challenges. The early SoRE approaches were derived from the initial phases of traditional software development methodologies, and the later ones are original, taking into account the specific characteristics of service-oriented systems. This paper discusses the specifics of service-oriented systems, describes the paradigm related SoRE issues and provides an overview of the characteristics of service-oriented enterprise systems. All these specifics and issues entail different methodological approaches to SoRE. The special attention is given to requirements negotiation activity. The paper presents a view-based approach to derive the balanced service quality requirements from an initial set of stakeholders' needs.

Keywords. Service-oriented requirements engineering, service-oriented software development, service-oriented enterprise systems, service quality requirements, requirements negotiation

Introduction

Service-oriented software engineering emerged in the last decade of previous century [1, 2, 3, 4, 5, 6, 7], as a response to the challenges of integration of heterogeneous applications, including legacy ones, to the cross-platform interoperability, bridging the gap between business models and software architectures and implementing the on-demand solutions. It has its roots in object-oriented software engineering, component-based software engineering, open distributed processing and business modelling techniques. SoSE is still growing research and development area.

The efforts towards service orientation had their impact on the whole enterprise system. So called service-oriented architecture (SOA) entailed the methodological changes in enterprise architecture (EA) domain. The synergy resulted in service-oriented enterprise architecture (SoEA) – an architectural style where enterprise system consists of service users and service providers, and any system's activity is treated as a service [8]. Therefore, there are at least two kinds of service-oriented systems – internet-wide and enterprise-centered.

[1] Corresponding Author: Audrone Lupeikiene, VU Institute of Mathematics and Informatics, Akademijos 4, LT-08663 Vilnius, Lithuania; E-mail: audrone.lupeikiene@vu.mii.lt.

Service-oriented requirements engineering (SoRE) as an integral part of SoSE emerged at the very beginning of the 21st century. The first publications discussed the nature of this discipline, the differences between SoRE and traditional requirements engineering (RE), the structure of service-oriented requirements lifecycle, and possible approaches to address the identification and handling of functional and non-functional requirements for software systems of service-oriented architectural (SOA) style [9, 10, 11]. The discipline has quite short history and many challenging issues still remain open. B. Verlaine, I. Jureta and S. Faulkner assert that even the question "Does service-orientation need new RE methodologies, or would the specialisation of existing ones be enough?" is still not answered [12].

The number of open SoRE problems including reduced utilization of service performance metrics, unclear, incomplete or static service specifications, and significant but yet unexplored socio-technical issues in negotiating conflicting requirements has been reported by many authors, including [10, 13, 14]. SoRe is steadily growing research and development area.

The aim of this paper is to discuss the specific characteristics of service-oriented systems, describe the paradigm related SoRE issues and provide an overview of the characteristics of service-oriented enterprise systems. All these specifics entail different methodological approaches to SoRE and pose new challengers. The special attention is given to requirements negotiation activity. A view-based approach, which enables to derive the agreed service quality requirements from an initial set of stakeholders' requirements, taking into account different viewpoints and perspectives on service quality, is presented.

The remainder of the paper is organised as follows. Section 1 discusses the innovations of service-oriented paradigm. Section 2 gives a brief list of service-oriented paradigm related requirements engineering issues focusing on SoRE process, requirements types, and service variability specification activities. Section 3 considers the differences between two kinds of service-oriented systems. Section 4 presents a view-based approach to take into consideration different attitudes and agree on the service quality requirements, and, finally, Section 5 concludes the paper.

1. Specifics of Service-Oriented Paradigm

From business point of view a service is a **unit of work** carried out by a service owner for a service consumer. In an enterprise context they can reside at all enterprise levels – the units of business functionality, business specific applications, platform and hardware functionality can be offered as services.

From technological point of view a service is **a mechanism** to enable access to one or more capabilities [15]. According to [16] such the definition emphasises "...capability as the notional or existing business functionality that would address a well-defined need. Service is therefore the implementation of such business functionality such that it is accessible through a well-defined interface". The other emphasis here can be done on the mechanism as an enabler to access to one or more capabilities. The term which is commonly used to designate such the mechanism is an *infrastructure*. By definition, infrastructure consists of essential common resources shared by any set of otherwise independent users.

Two most important innovations of service-oriented paradigm are associated with a specialisation of system engineering principles **separation of concerns** and

abstraction when developing software system. The innovative separation of concerns for service-oriented paradigm entails the following:

- different viewpoints are required to develop the whole service-oriented system;
- specification is split into those that concern the consumers' domain and those that characterise the required behaviour of the solution system;
- redistribution of functionality between system constituents is required.

The innovative specialisation of general abstraction principle entails the following:

- the paradigm related concept system includes a business service concept and a business process concept, and
- decisions can be postponed, systems can be composed at runtime.

Lets discuss these innovations in more detail.

1.1. Viewpoints and Perspectives

The use of viewpoints in the requirements engineering activities is well known [17, 18]. This approach recognises that all requirements cannot be discovered by considering system from a single point of view; it is essential to collect requirements from different classes of system end-users and other stakeholders, to collect information of different types – about the application domain, system environment and system development. There are a number of different models of viewpoints. We consider a viewpoint to be a role performed by a stakeholder when examining a system.

A perspective defines some aspect of service system on which any viewpoint in principle may be focused. For example, it can be matched with some component of service system. It does not mean that all perspectives cross-cut all the viewpoints. As a rule, any perspective is related with a set of characteristics that are observed from some viewpoint.

The viewpoint based approach to requirements engineering has been approved in, let us call it a traditional software systems development context. The service-oriented systems are analogous to a federation; i. e., a system consists of several constituents, which remain independent in internal and sometimes even in external matters. In other words, the constituents are under control of different stakeholders, which can have quite different goals. There are at least three paradigm related roles: service consumer, service infrastructure owner, and service owner. The goals of service consumer are aligned with the business goals. The goals of infrastructure owners are related to the development and maintenance of a service system infrastructure [9]. The goals of service owner concern the realisation of capabilities.

1.2. System and its Effect

By definition, a service is a unit of work carried out by a service owner to its consumer. The result of this work is a real world effect which is produced by service-oriented system[2]. M. Jackson points out that developers focuses their "attention on the artefact, not on the problem it is intended to solve" [19]. So, software development should be thought of as building a machine to solve a problem in a real world, i. e., to produce an effect. The application of Jackson's problem frame theory in our context entails the following:

[2] It should be remembered here that intangible service differs from tangible product.

- Methods and tools for examining, analysing and describing real world problems are required. “The main message here is that you need to get outside of the system boundary to identify services” [20].
- Clear differentiation between application domain description and service system description should be done.
- Business process modelling is essential; in addition, the specific techniques for business process modelling should be compatible with the techniques for interaction with service system modelling. So, use-case description is a second step and on a lower level [11].

1.3. Redistribution of Functionality

Various forms of infrastructure like networks, communication protocols, database management systems, etc. are well known. These are designed to provide the basic foundations for various systems within different domains. The separation between such underlying support and system using it enables to reduce complexity, to open up a field for future extensions. Service-oriented paradigm causes the situation to become more intense – it entails the move of functionality from applications to infrastructure [9], supports sharing and integrating at a global and an enterprise system level. This notable add-on fills the gap between needs and capabilities (Figure 1). The infrastructure underlying a service-oriented system plays mediator’s, integrator’s and some other roles. As pointed out in [21] its functionality covers discovery, composition, and invocation of services. This functionality will be broaden in coming years to include additions, for example, service monitoring, service evaluation at runtime.

This trend was foreseen 30 years ago. T. Capers Jones estimated that less than 15% of the code written in 1983 was “unique, novel and specific to individual applications. The remaining 85% appears to be common, generic, and concerned with putting applications into computers” [22]. His prediction, made in 1984 [22], was the following:

- 1990 – 50% of all code will be reused, 50% will be unique among leading-edge enterprises;
- 1990 – 25% of the applications in the industry will be new (i. e., will cover the business fields which have not previously been subject to automation);
- 2000 – 10-15% of the applications in the industry will be new.

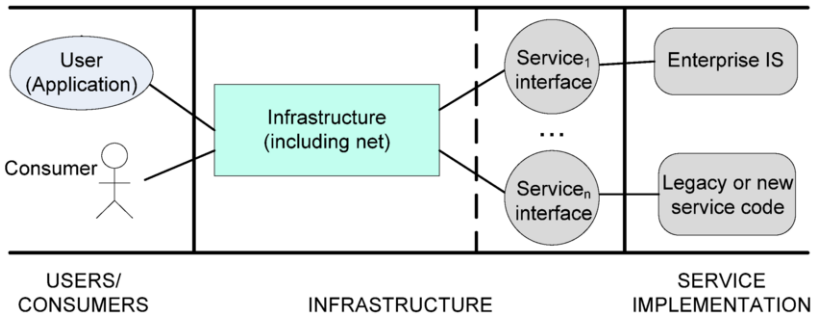

Figure 1. The main high-level types of constituents of service-oriented system

1.4. Abstractions

Many service-oriented systems are developed on an object-oriented infrastructure, which is concerned with object abstractions. Despite the fact that objects can correspond to real-world entities, this abstraction is low-level one in an enterprise context.

Service-oriented paradigm provides the native support for enterprise conceptualisation – it introduces two high-level abstractions, namely a service and a business process. These concepts can be used not just in SoSE, but also at all other levels of the enterprise system. Processes are the abstractions of the overall functioning. They are considered as orchestrations of services. As a result, service-oriented system is thought of as a set of interacting services, which are coordinated by business process. From service consumer's point of view a business process specification is an input for and can be executed by virtual machine.

1.5. Informational Content of Products

Instances of service-oriented system are usually produced at runtime. It means that in service-oriented systems development there is a trend towards increasing so called informational content (requirements, specification documents, design specification documents, etc.) of the "products". Since such the descriptions are easier to change than physical systems (i. e., software code in use), developers try to keep system under development in these abstraction forms as long as possible. So, development is divided into two phases: first is a phase, which creates different descriptions and models; and second is a phase, which produces the concrete implemented instances and which can be extremely short.

In summary, all the mentioned issues cause a number of new problems in the service-oriented systems engineering. Therefore, a new requirements engineering subdiscipline – service-oriented requirements engineering (SoRE) – comes into view. While for the previous paradigms we have well-known and stable requirements engineering processes and methods, in SoRE such processes, methods and techniques are still open research area.

2. Service-Oriented Paradigm Related Requirements Engineering Issues

The SoRE approaches proposed in recent years can be partitioned into two groups: the first group refer to slightly revised traditional approaches (e. g., [23]), and the second one includes innovative approaches [24, 25, 26, 27, 28, 29, 30]. In any case, because of the specifics of service-oriented systems, SoRE differs from traditional requirements engineering. Let us shortly discuss its characteristics and challenging issues focusing on a SoRE process, the categories of requirements, and on the continually updating a requirements specification to ensure its adequacy to service's variations.

2.1. SoRE Process Models

Requirements engineering covers requirements elicitation, analysis, specification, verification, and management. SoRE process defines how to specialise these classical RE activities in a service-oriented context.

The difference between traditional and SoRE activities is essentially in performing the service and workflow discovery [31], identifying the service requirements specified in service level agreements (SLA), identifying and specifying the dynamic requirements that may vary at runtime.

The requirements engineering process dovetails with the software design process. To be more precise, it is rather process initiated at the beginning and continuing throughout the whole life cycle. Service-oriented paradigm causes the more difficult decoupling of requirements process and the other ones in different life cycle stages, including runtime. Specification and use of policies and contracts can be given as the argument and example. Policies and contracts need to be specified separately, however, need to be evaluated together with the functional components. Policies are realised as computational constraints and enforced at runtime. A SLA is part of the contract between the service consumer and service owner. To enable the dynamic provisioning of services, all the phases of SLA life cycle should occur at runtime [32]. On the one hand, the activities include, among others, identification of service consumer needs and service characteristics, negotiation, evaluation. On the other hand, high level SLAs need to be translated to software level rules and measurable properties.

In summary, a number of end-to-end development process models described in the literature follow the classical waterfall model with some modifications that are particular to service-oriented computing [33]. Besides, all the methodologies consider SoRE issues only in passing, so SoRE process model(s) is the challenging issue in service-oriented systems theory.

2.2. Types of Requirements

Requirements can be categorised in several ways on a number of dimensions. SoRE needs the relevant requirements taxonomy, which is still under development.

Early and late requirements engineering has been adopted for service-oriented systems and is presented in [24, 25, 28, 34, 35, 36]. Early requirements are expressed in terms of high level concepts and correspond to the stakeholders' goals and needs [37]. Late requirements describe service or system within its operational environment, along with functionality and qualities. Late requirements define coarse grained business processes, which are refined to obtain workflows at the workflow level for Web services development [28].

SoRE covers requirements which are static and dynamic, i. e., of different variability over time [38], including variations at runtime. So, the questions "How extensive the initial specification should be?", "What type of requirements and to what extent can vary throughout the whole life cycle?" should be answered. As pointed out in [38], service requirements and services coordination requirements vary less frequently, while quality characteristics and priorities over qualities vary more frequently. Early requirements are the most stable in a specification document.

New types of requirements follow from "native" functionality of service oriented systems. Adaptation requirements include the circumstances, in which software should be adapted, priorities of these circumstances and the objectives of the adaptation

activity [37]. It should be noted, that specification of adaptation requirements needs an answer to the question "What situations and to what extent should be anticipated and specified in advance?". Monitoring requirements concern the situations that may trigger the adaptation of software or the situations when reaction to detected problems, including errors, is required. Monitoring requirements in some cases can be derived from the adaptation requirements [31]. This list of functional requirements is not complete, as infrastructure of service-oriented systems is still evolving.

2.3. Requirements Management

In almost all cases, a significant proportion of the requirements changes as development proceeds and requirements are revised in the late life cycle stages. This is due to "traditional" reasons, such as errors, continuously evolving understanding of requirements and changes in the business environment. In addition, a requirements specification of services and service-oriented systems includes requirements that are of different variability over time. In other words, new services can appear at runtime and the degree to which they satisfy the initial requirements may vary at runtime. Therefore, the initial requirements specification should be continually updated to reflect the variations [38]. This leads to the permanent revisions of requirements after deployment.

Changes have to be managed by ensuring an appropriate review and approval process. The methods and techniques to update the requirements at runtime to reflect all the changes are required as well.

3. SoRE vs - Service-Oriented Enterprise Systems RE

Service-oriented systems are characterised as crossing organisational boundaries, i. e., services and their control may be distributed in multiple independent units. Distributed ownership, despite the service contracts, makes a system and value delivered by a service to its consumers highly uncertain. The uncertainty can be minimised within an enterprise, where system may represent a more constrained and predictable operational environment and service contracts as a rule are long lasting. The conceptual, technological and technical differences between two kinds of service-oriented systems are presented in Table 1.

Table 1. Service-oriented systems vs service-oriented enterprise systems[3].

Service-Oriented Systems	Service-Oriented Enterprise Systems
Internet-wide open system. Developed in a bottom-up manner.	Relatively closed enterprise-wide system controlled on an enterprise-wide level. Developed in a top-down manner.
Not purported to support a particular business strategy and to implement predefined business processes.	Business-driven, i. e., support enterprise's business strategy and objectives. Enterprise business process coordinates a set of interacting enterprise business services (EBS).
Services published in the internet-wide registers.	Enterprise service inventory. Process logic is separated from business logic [41].

[3] A service-oriented enterprise system is also called an enterprise service-oriented architecture [39, 40].

Any business services. No ability to normalise[4] business services.	Normalised enterprise business services aligned with the enterprise business functions.
No ability to define global data types and use naming conventions.	Use of global data types and standard naming conventions [41] enabling simplified data exchange between services.
Some services are situation-aware but only in rare cases are context-aware because the context as a rule is ill-defined.	All services are context-aware because they run in the well-defined enterprise context.
Direct pear-to-pear communication between consumer and provider. UDDI for service registration and discovery.	Enterprise service bus as a mediator between consumers and providers.
Service level agreement is negotiated between provider and consumer at the run time.	Service level agreement is mandated (mostly) at the enterprise-wide system at the design time.
Neither service provides, nor consumers can control the SOA infrastructure including communication networks.	The whole infrastructure, including intranet, enterprise service bus, servers and other elements, are under control by the enterprise.
No guide on a set of services, on how they are built and deployed. No control over changes in services.	EBSs are developed and deployed in compliance with the enterprise-wide standards. All changes are under control.
The structure of messages is standardized (e.g. by SOAP) but not unified. EBS interfaces are standardized (by WSDL), but not clearly defined, not stable. No ability to use global data types in the interfaces.	The structure of messages is unified. EBS interfaces are clearly defined, stable, and make use of global data types [40].
Recommended security and safety standards.	Mandatory security and safety standards.

The differences presented in Table 1 play a role of supporting evidences and we can conclude that requirements engineering, its processes, methods and techniques should be different for both cases. These differences offer the following analogy – SoRE is similar to market-driven RE and service-oriented enterprise systems RE is similar to customer-specific RE. It should be noted that in the present paper we stay in enterprise system context.

4. Service Requirements: the Agreed Quality of Service

Stakeholders have different goals and interests, even when developing an enterprise-wide service-oriented system, which is managed by one organisation. They play not only the different roles, but also have the different attitudes (e.g., stakeholder wants an excellent service or a service at an acceptable price). So, stakeholders should negotiate to agree on a common set of requirements. A service quality specification that satisfies the consumers' quality requirements should be matched with the owner's/designer's service quality requirements. It means that evaluation of service/service composition requirements in service and workflow discovery activity is the prerequisite to avoid system development problems. The formal model of view-based enterprise business

[4] Normalisation means that each EBS should be developed with the intent to avoid similar or duplicate bodies of business logic.

service quality evaluation framework [42, 43] can be adopted here to derive service quality characteristics and to get the preliminary evaluation of its whole quality.

A system's quality requirements express a degree to which software system possesses a desired combination of quality attributes [44]. All these attributes, independently of their level, will be called quality characteristics, i. e. qualities. An example of the set of relevant quality characteristics, to which should adhere reference architecture necessary to facilitate the development of any ecosystem with means of service-oriented cloud computing, is described in [30].

The qualities by which the system's effect will be evaluated should be many-valued, because the quality characteristics as a rule are vague. It is required for, at least, two reasons: 1) service consumer should be allowed to describe the extent, to which the values of quality characteristics could be tolerated, 2) service adaptation is required, or in other words, the variations of service should be specified.

The quality characteristics are vague concepts, so, we consider them as linguistic variables with a common set of linguistic terms:

L_{tr} = (*unsatisfied, ... , satisfied*).

For example, L_{tr} consists of the following terms: *below low quality* (synonym to unsatisfied), *low quality, average quality, high quality, perfect quality* (synonym to *satisfied*).

Linguistic terms are to be assigned to the bottom level qualities. The fuzzy set approach to the set of linguistic terms provides a much better representation of these vague terms. So, for each pair <quality characteristic, linguistic term> its own membership function should be defined, which maps the linguistic term-related sub-domain of this characteristic to appropriate fuzzy set. The domains of quality characteristics can be discrete as well as continuous. The membership functions should be defined in such a way that for the quality characteristic under consideration the subdomain of its values, which is related to this same linguistic term, should be mapped to the same fuzzy set or, in other words, the interpretation of linguistic terms for any quality characteristic should not depend on a particular view. It means that the membership function should unify the understandings of linguistic terms.

We should specify service quality requirements at least from service provider's and designer's viewpoints. Any viewpoint $\omega_k \in \Omega$, where Ω is a set of weighted linguistic variables referred to as viewpoints to service quality:

$$\Omega = \{(\omega_k, \mu_\Omega(\omega_k)) \mid \mu_\Omega{:}\Omega \rightarrow L_{tr},\ 1 \leq k \leq s\}. \tag{1}$$

A perspective $\pi_i \in \Pi$ (e.g., net infrastructure, web service, application, etc.) defines a subset of quality characteristics, which are observed from viewpoint ω_i, and

$$\Pi = \{(\pi_i, \mu_\Pi(\pi_i)) \mid \mu_\Pi{:}\Pi \rightarrow L_{tr},\ 1 \leq i \leq m\}. \tag{2}$$

As pointed out in [55], "this set of quality attributes does not characterize only the service but any entity used in the path between the service and its client".

Let any viewpoint is associated with a matrix:

$$\Psi_k = \begin{pmatrix} \psi^k_{1,1} & \cdots & \psi^k_{1,m} \\ \vdots & \ddots & \vdots \\ \psi^k_{n,1} & \cdots & \psi^k_{n,m} \end{pmatrix}, \tag{3}$$

where

$\psi_{i,j}^{k} = (\pi_i, \gamma_j)$, $\pi_i \in \Pi$, $\gamma_j \in \Gamma$, $1 \le k \le s$, $1 \le i \le n$, $1 \le j \le m$, $k \le N_\Omega$, $n \le N_\Pi$, $m \le N_\Gamma$, (e. g., "the network should be highly reliable", "the web service should be highly available", "the executable application should be perfectly recoverable").

A finite set of linguistic variables

$$X=\{(\chi_i, \mu_X(\chi_i))|\ \mu X: X \rightarrow L_{tr},\ 1 \le i \le N_X,\ NX \in \aleph\} \tag{4}$$

is called the service quality characteristics. We can define a labelled equilibrium relation on that set

$$\rho_{eqlb}^{X} = \begin{matrix} \{((\chi_1,\chi_2), \mu_{\rho_{eqlb}^{X}}(\chi_1,\chi_2), label) | \mu_{\rho_{eqlb}^{X}} : X \rightarrow [0,1], (\chi_1,\chi_2) \in X \times X, \\ \mu_\Gamma(\chi_1) + \mu_\Gamma(\chi_2) \le C_{eqlb}^{(\chi_1,\chi_2)} \le 1, label \in \{\ll, \gg, \sim, \ll \sim, \sim \gg\}\} \end{matrix}. \tag{5}$$

$C_{eqlb}^{(\chi_1,\chi_2)}$ is an equilibrium constant, which means that a sum of the lengths of subintervals $\mu_\Gamma(\chi_1)$ and $\mu_\Gamma(\chi_2)$ cannot exceed the length defined by this constant, which, in turn, cannot exceed the length of the interval [0,1]. The label of this relation tells us how, if it is necessary, the lengths of subintervals $\mu_\Gamma(\chi_1)$ and $\mu_\Gamma(\chi_2)$ should be changed in order to preserve the equilibrium defined by C_{eqlb}.

Analogously, total equilibrium relations ρ^{Ω}_{eqlb} on the set Ω and $\rho^{\Psi_k}_{eqlb}$ on the each set Ψ_k can be defined.

Let $X'=<X, <_X, F^{\ll}>$ be a ranked set on X, where $<_X$ is a partial order relation, and $F^{\ll}$ is a ranking function. For each element $\psi_{i,j}^{k}$ of the matrix Ψ_k we define a fuzzy AND tree of the service quality characteristics:

$$T_{and}{}^{(\psi_{i,j}^{k})} = <\ X',\ \psi_{i,j}^{k}\ ,\ r^{(\psi_{i,j})}\ >, \tag{6}$$

where $1 \le k \le s$, $1 \le i \le n$, $1 \le j \le m$). The value of a linguistic variable $\psi_{i,j}^{k}$ describes the γ_j quality characteristic of service from the perspective π_i of the viewpoint ω_k (Figure 2 gives an example). Using the fuzzy implication relation $r^{(\psi_{i,j})}$ between source nodes t_1, t_2, …, t_n and target node t the satisfiability (or deniability) of the property can be propagated across the whole tree (Figure 3, step 2).

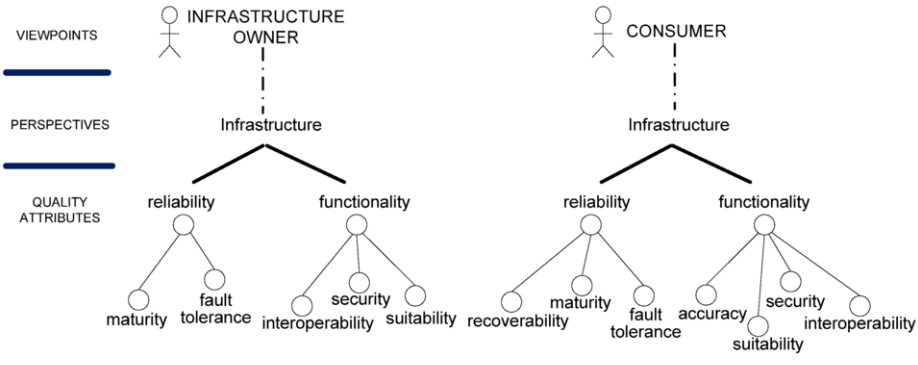

Figure 2. Two viewpoints, one perspective and its quality attributes

The columns of the matrix are vectors, which for each quality characteristic give the evaluations from each perspective defined for the viewpoint ω_k. Using the total equilibrium relations on each set Ψ_k these quality characteristics can be balanced (i. e., the conflicts can be resolved) and, using relations $r^{(\Psi_{i,j})}$, the obtained values can be propagated in a backward manner to the leaf nodes in each tree (Figure 3, step 4). The unions of these balanced trees are the final service quality characteristics observed from the viewpoints $\omega_1, \ldots, \omega_s$.

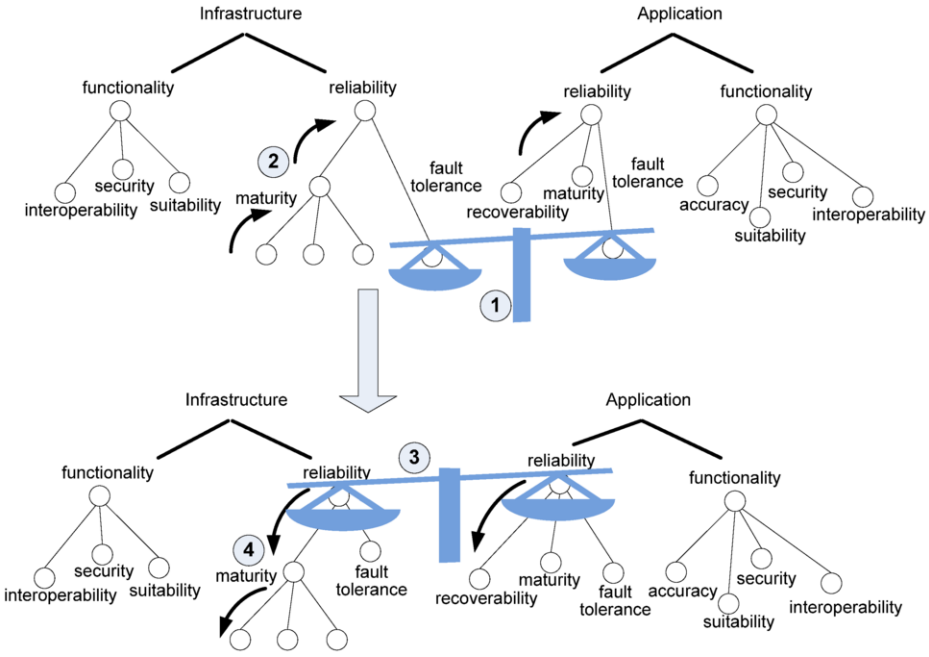

Figure 3. Conflict resolution and propagation of values in the fuzzy AND trees of service quality characteristics

In order to evaluate the whole service quality, the values of $\omega_1, \ldots, \omega_s$ are calculated considering the unions of trees as child nodes of the nodes $\omega_1, \ldots, \omega_s$. Using the relation ρ^{Ω}_{eqlb} the conflicts between the values of the set Ω elements can be resolved. Finally, the variable ω' with the greatest weight is to be chosen and the missing sub-trees (i. e., missing quality characteristics) from the trees associated with the other viewpoints are to be appended to the tree associated with this variable ω'. The whole service quality characteristic can be propagated in the backward manner to leaf nodes, if the values of service quality characteristics are required. A linguistic approximation of fuzzy values of all these variables can be done.

This approach is not adjusted to some particular service-oriented software development methodology. It can be included to all SoRE processes. The underlying philosophy was taken from the goal-oriented modelling and i* methodology [45, 46, 47]. As i* techniques cannot be directly applied to evaluate the quality goals, we modified these techniques by using fuzzy set theory and fuzzy logic, and by adding the equilibrium relations to enable conflict resolution between stakeholders' quality requirements.

5. Conclusions

Service-oriented systems engineering in-the-small can be considered as an integral part software engineering. Therefore, the specifics of service-oriented systems can play a role of supporting evidences that software engineering as a discipline continuously matures, i. e., nowadays it includes more features which have already been approved in systems of the other engineering branches (such as civil engineering, mechanical engineering), which have much longer history and experience.

Service-oriented requirements engineering inherits concepts and principles from earlier paradigms but differs from these paradigms in the manner how the separation of concerns in software system is done. In addition, it provides two new high-level abstractions, namely, enterprise business services and business processes. SoRE is more complex than traditional software requirements engineering, having deal with new specific problems. Moreover, service-oriented systems are highly uncertain and this uncertainty can be minimised within an enterprise. However, different service-oriented system development methodologies consider SoRE issues only in passing, no one proposes SoRE process model. The early SoRE approaches, methods and techniques have been derived from the initial phases of traditional software development methodologies, and the later ones are original, but as a rule not adjusted to particular SoRE process.

In recent years SoRE gives more attention to such areas as specification of adaptable services, identification of requirements, which should be specified in SLAs, and service quality requirements. The view integration and reconciliation methodology, which is widely accepted in the software and enterprise systems requirement engineering, can be applied in requirements negotiation activity to agree on acceptable to all stakeholders' requirements, as well as to preliminary evaluation of the service quality characteristics.

Acknowledgments

This work has been supported by the project „Theoretical and Engineering Aspects of e-Service Technology Development and Application in High-Performance Computing Platforms“ (No. VP1-3.1-ŠMM-08-K-01-010) funded by the European Social Fund.

References

[1] Rands T. Information technology as a service operation. *Journal of Information Technology,* 1992, 7(4), 189-201.

[2] Leyland P. F., Watson R., Kavan C. B. Service quality: a measure of information systems effectiveness. *Management Information Systems Quarterly*, 1995, 19(2), 173-187.

[3] Watson R. T., Leyland F. P., Kavan C. Measuring information systems service quality: lessons from two longitudinal case studies. *Management Information Systems Quarterly*, 1998, 22(1), 61-79.

[4] Niessink F., Vliet H. Towards mature IT services. *Software Process – Improvement and Practice*, 1998, 4(2), 55-71.

[5] Arsanjani A. Service provider: a meta-domain pattern and its business framework implementation. In: *Proceedings of the Pattern Languages of Programms Conference [PLoP]*, *15-18 August, Monticello, Illinois, USA*. The Hillside Group, 1999. 1-24.

[6] Nuseibeh B. A., Easterbrook S. M. Requirements engineering: a roadmap. In: Finkelstein A. C. W. (ed.) *Proceedings of the 22nd International Conference on Software Engineering, Future of Software Engineering Track [ICSE]*, 4-11 June, Limerick, Ireland. ACM, 2000. 35-46.

[7] Bennett K. Layzell P., Budgen D., Brereton P., Macaulay L., Munro M. Service-based software: the future for flexible software. In: *Proceedings of the 7th Asia-Pacific Software Engineering Conference [APSEC]*, 5-8 December, Singapore. IEEE Computer Society, 2000. 214-221.

[8] Bianco P., Kotermanski R., Merson P. Evaluating a Service-Oriented Architecture. Technical Report CMU/SEI-2007-TR-015, Software Engineering Institute, 2007. 79 p.

[9] van Eck P., Wieringa R. Requirements engineering for service-oriented computing. In: Gordijn J., Janssen M. (eds.) *Proceedings of the First International e-Services Workshop [ICEC]*, Pittsburg, USA, 2003. 23-28.

[10] Trienekens J., Bouman J. J., van der Zwan M. Specification of service level agreements: problems, principles and practices. *Software Quality Journal*, 2004, 12(1), 43-57.

[11] Zimmermann O., Krogdahl P., Gee C. Elements of Service-Oriented Analysis and Design. An Interdisciplinary Modeling Approach for SOA Projects. IBM developerWorks, 2004. 18p. [WWW] https://www.ibm.com/developerworks/webservices/library/ws-soad1/ (accessed 23.04.2014).

[12] Verlaine B., Jureta I., Faulkner S. Towards conceptual foundations of requirements engineering for services. In: *Proceedings of the 5th IEEE International Conference on Research Challenges in Information Science [RCIS]*, 19-21 May, Gosier, Guadeloupe, France. IEEE Computer Society, 2011. 1-11.

[13] Flores F., Mora M., Alvarez F., Garza L., Duran H. Towards a systematic service-oriented requirements engineering process (S-SoRE). In: Quintela Varajão J. E., Cruz-Cunha M. M., Putnik G. D., Trigo A. (eds.) *ENTERprise Information Systems - International Conference [CENTERIS 2010]*, Part I, CCIS 109, Berlin: Springer-Verlag, 2010. 111-120.

[14] van Lamsweerde A. Requirements engineering in the year 00: a research perspective. In: Ghezzi C., Jazayeri M., Wolf A. L. (eds.) *Proceedings of the 22nd International Conference on Software Engineering [ICSE]*, 4-11 June, Limerick, Ireland. ACM, 2000. 5-19.

[15] Reference Model for Service Oriented Architecture 1.0 OASIS Standard, 12 October 2006. OASIS Open [WWW] https://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf (accessed 23.04.2014).

[16] Reference Architecture Foundation for Service Oriented Architecture, Version 01. OASIS Committee Specification 01, 04 December 2012. OASIS Open [WWW] http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.pdf (accessed 28.07.2014).

[17] Leite J. C. P. Viewpoints analysis: a case study. *ACM SIGSOFT Software Engineering Notes*, 14 (3), 1989, 111-119.

[18] Sommerville I., Sawyer P. Viewpoints: principles, problems and a practical approach to requirements engineering. *Annals of Software Engineering*, 3, 1997, 101-130.

[19] Jackson M. Problem architectures. In: Garlan D. (ed.) *Proceedings of First International Workshop on the Architecture of Software Systems*. Technical Report CMU-CS-TR-95-151, Software Engineering Institute, 1995.

[20] Graham I. *Requirements modelling and specification for service oriented architecture*. John Wiley&Sons, 2008. 320 p.

[21] Bianco P., Lewis G. A., Merson P., Simanta S. Architecting Service-Oriented Systems. Technical Note CMU/SEI-2011-TN-008, Software Engineering Institute, 2011. 36 p.

[22] Capers Jones T. Reusability in programming: a survey of the state of the art. *IEEE Transactions on Software Engineering*, 1984, 10(5), 488-494.

[23] Papazoglou M. P., van den Heuvel W.-J. Service-oriented design and development methodology. *International Journal of Web Engineering and Technology*, 2(4), 2006, 412-442.

[24] Aiello M., Giorgini P. Applying the Tropos Methodology for Analysing Web Services Requirements and Reasoning about Qualities of Services. Technical Report DIT-04-034, University of Triento, 2004. 17 p.

[25] Lau D., Mylopoulos J. Designing web services with Tropos. In: *Proceedings of the IEEE International Conference on Web Services [ICWS'04]*, 6-9 June, San Diego, California, USA, IEEE Computer Society, 2004. 306-315.

[26] Penserini L., Perini A., Sus A., Mylopoulos J. From stakeholder needs to service requirements. In: *Proceedings of the Service-Oriented Computing: Consequences for Engineering Requirements [SOCCER'06]*, 12 September, Minneapolis, Minnesota, USA. IEEE Computer Society, 2006. 1-10.

[27] Lapouchnian A., Yu Y., Mylopoulos J. Requirements-driven design and configuration management of business processes. In: Alonso G., Dadam P., Rosemann M. (eds.) *Proceedings of the 5th International Conference On Business Process Management [BPM'07]*, LNCS 4714, Berlin: Springer-Verlag, 2007. 246-261.

[28] Frankova G. *Engineering business processes with service level agreements: from early requirements towards business processes*. LAP Lambert Academic Publishing, 2010. 176 p.
[29] Rolland C., Kirsch-Pinheiro M., Souveyet C. An intentional approach to service engineering. *Journal IEEE Transactions on Services Computing*, 3(4), 2010, 292-305.
[30] Norta A., Grefen P., Narendra N. C.. A reference architecture for managing dynamic inter-organizational business processes. *Data & Knowledge Engineering*, 2014, 91, 52–89.
[31] Andrikopoulos V. (ed.) Separate Design Knowledge Models for Software Engineering and Service Based Computing. S-Cube Technical Report CD-JRA-1.1.2, 2009. 66p. [WWW] http://www.s-cube-network.eu/results/deliverables/wp-jra-1.1/CD-JRA-1.1.2_Separate_design_knowledge_models_for_software_engineering_and_service_based_computing.pdf (accessed 28.07.2014).
[32] Bianco P., Lewis G. A., Merson P. Service Level Agreements in Service-Oriented Architecture Environments. Technical Note CMU/SEI-2008-TN-021, Software Engineering Institute, 2008, 40 p.
[33] Lane S., Richardson I. Process models for service based applications: a systematic literature review. *Journal Information & Software Technology*, 53(5), 2011, 424-439.
[34] Frankova G., Malfatti D., Aiello M. Semantics and extensions of WS-agreement. *Journal of Software*, 1(1), 2006, 23-31.
[35] Frankova G., Yautsiukhin A., Seguran M. From Early Requirements to Business Processes with Service Level Agreements. Technical Report DIT-07-037, University of Triento, 2007. 24 p.
[36] Frankova G., Seguran M., Gilcher F., Trabelsi S., Dorflinger J., Aiello M. Deriving business processes with service level agreements from early requirements. *The Journal of Systems and Software*, 84, 2011, 1351-1363.
[37] Pistore M., Kazhamiakin R., Bucchiarone A. (eds.) Integration Framework Baseline. S-Cube Technical Report CD-IA-3.1.1, 2009. 43p. [WWW] http://www.s-cube-network.eu/results/deliverables/wp-ia-3.1/CD-IA-3.1.1_Integration%20Framework%20Baseline.pdf (accessed 28.07.2014).
[38] Jureta I., Faulkner S., Thiran P. Dynamic requirements specification for adaptable and open service-oriented systems. In: Kramer B. J., Lin K.-J., Narasimhan P. (eds.) *Service-Oriented Computing – ICSOC 2007*, LNCS 4749, Berlin: Springer-Verlag, 2007. 270-282.
[39] Enterprise Service Oriented Architecture Using the OMG SoaML Standard. A Model Driven Solutions, Inc., 2009. 21 p. [WWW] http://www.omg.org/news/whitepapers/EnterpriseSoaML.pdf pdf (accessed 28.07.2014).
[40] Enterprise SOA Development Handbook 1.1. End-to-end Guide for Enterprise SOA Development. SAP AG, 2008, 85 p. [WWW] http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/40db4735-02f9-2a10-b198-a888a056bb67?QuickLink=index&overridelayout=true&32220844681654 (accessed 28.07.2014).
[41] Heidasch R. Get ready for the next generation of SAP business applications based on the Enterprise Service-Oriented Architecture (Enterprise SOA). *SAP Professional Journal*, July 2007, 103-128.
[42] Lupeikiene A., Miliauskaite J., Caplinskas A. A model of view-based enterprise business service quality evaluation framework. *Informatica*, 24(4), 2013, 543-560.
[43] Lupeikiene A., Miliauskaite J., Caplinskas A. A view-based approach to quality of service modelling in service-oriented enterprise systems. In: Kirikova M., Grabis J. (eds.) *Proceedings of the 2nd International Business and Systems Conference [BSC 2013]*, 5 November, Riga, Latvia. Riga Technical University, 2013. 7-19.
[44] IEEE Std 1061-1998 – IEEE Standard for a Software Quality Metrics Methodology. Institute of Electrical and Electronics Engineers, 2005.
[45] Fuxman A., Liu L., Mylopoulos J., Roveri M., Traverso P. Specifying and analyzing early requirements in Tropos. *Requirements Engineering Journal*, 9(2), 2004. 132-150
[46] Giorgini P., Mylopoulous J., Sebastiani R. Goal-oriented requirements analysis and reasoning in the Tropos methodology. *Engineering Applications of Artificial Intelligence*, 18(2), 2005, 159-171.
[47] Castro J., Kolp M., Mylopoulos J. Towards requirements-driven software development methodology: the Tropos project. *Information Systems*, 2002, 27(6), 365-389.