An Analysis Study of Contract-Based Approach for Quality-Driven Service Component Architecture

Maryem Rhanoui[1, 2] , Bouchra El Asri[1]
[1]IMS Team, SIME Laboratory
ENSIAS, Rabat
Morocco
[2]ESI Rabat
Morocco
mrhanoui@gmail.com, elasri@ensias.ma

ABSTRACT: *Dependability is a major requirement of complex systems which consists of the system's ability to offer a trusted and reliable service. Service Component Architecture is a promising and recent approach and an industry standard for developing complex and distributed systems. To be adopted in a safety-critical environment it must handle dependability requirements and offer tools and frameworks to certify the reliability level of the components and the system.*

In this paper we present main techniques and models for assuring quality and trustworthiness of component-based systems in general, and then we present and justify the choice of the design by contract approach that we adopted for the following of our research about SCA-based systems. Design by Contract is an approach of software design that aims to address reliability and quality issues in software development by expressing a set of its properties and constraints.

Keywords: Service Component Architecture, Dependability, Quality-Driven System, Fault Tolerance, Contracts, Aspects

Received: 2 June 2012, Revised 29 July 2012, Accepted 4 August 2012

1. Introduction

Service Oriented Architecture is a promising paradigm for developing complex systems that utilizes services as fundamental elements for developing applications. In this perspective, Service Component Architecture (SCA) is a new concept that offers a component model for building SOA architecture.

Official SCA specification document includes SCA assembly model specification [12] and SCA policy framework [10]. However, as an expanding approach, it still needs more formal models and frameworks for modeling and verifying systems.

In spite that the main purpose of software engineering is to find ways of building quality software [38], our literature review shows that most research efforts have focused on technical aspects of SCA, leaving aside the treatment of quality issues and extra-functional properties of service component.

In this scope, our fields of research focus on the design and development of complex and safety-critical systems. Critical systems [29] are systems whose failure could cause loss of human lives, cause property damage, or damage to the environment, such as aviation, nuclear, medical applications, etc.

As a matter of fact, dependability [33], which is the property that allows placing a justified confidence in the quality of the delivered service, is becoming increasingly important in complex systems design.

In this paper we remind the definition of Service Component Architecture and the concept of dependability and its properties and threats, and then we present main techniques and models for handling quality and trustworthiness of component-based systems such as reliability evaluation and prediction [32] [16], fault tolerance [5] [40], quality assurance models [15] [39] [6] and certification [3] [44] [59].

Among the presented approaches, we are interested by the contract-based approach [37] which is a light-weight formal method for designing quality-driven systems by specifying its non-functional and quality properties. Despite the fact that the concept of component contracts was formerly proposed, it still not commonly used in software development. The remainder of this paper is organized as follows: Section II will be dedicated to the presentation of the concept of service component and the survey of models and frameworks for service component architecture. Then in section III, we will define the dependability and the various quality attributes expected of a safety critical system.

In Section IV we will present an analysis of main techniques and models for assuring trustworthiness and quality of component-based systems and finally section V will present and justify the choice of our proposed approach.

2. Service Component Architecture

Service-oriented computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications [43]. Service Component Architecture (SCA) proposes a programming model for building components based applications following the SOA paradigm.

SCA had emerged as a new architecture which offers many advantages: it simplifies development of business component and assembly and deployment of business solutions built as networks of services; increases agility and flexibility and protects business logic assets by shielding from low-level technology change and improves testability.

In this section we describe, at a glance, the SOA architecture, present the component model and depict a metamodel and frameworks for SCA.

2.1 Service Oriented Architecture

Service Oriented Architecture (SOA) is a logical architecture of software systems based on the notion of "*service*", which is the basic building block of this architecture. A service a set of modular, self-contained and self-descriptive applications that can be published, located and invoked from the web.

According to Bieber and Carpenter [14]:

A service is a contractually defined behavior that can be implemented and provided by any component for use by any component, based solely on the contract.

The paradigm of service orientation defines an interaction scheme (see Figure 1) between the service provider, the service requestor and the service registery which share the knowledge of a service interface

The **Service Provider** creates or offers the Web service. The service provider must describe the Web service in a standard format, which is XML and publishes this description in a service registery.

The **Service Registery** contains additional information on the service provider, such as address and technical details about the service.

The **Service Requestor** search of information "directory and uses the service description obtained to bind the service and to invoke the Web service.

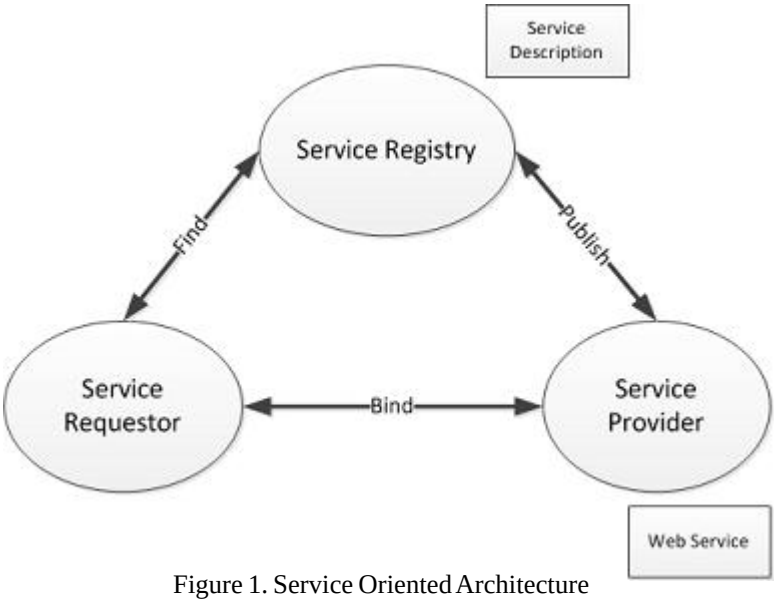

Figure 1. Service Oriented Architecture

2.2 Component Model

Various component models of Service Component Architecture were proposed in literature. For Ding [21] proposed component model, a service component provide and require services. A service can be described by operation activities as by well-defined business function. A component provides and consumes services via ports.

- A port p is a tuple (M, t, c), where M is a finite set of methods, t is the port type and c is the communication type.
- A component Com is a tuple (Pp, Pr, G, W), in which :

– Pp is a finite set of provided ports.

– Pr is a finite set of required ports.

– G is a finite sub component set.

– W is a wire set.

Moreover, Du et al [23] included contract concept in the Service Component meta-model.

A contract Ctr is a quadruple (P, Init, Spec, Prot) where

- P is a port;
- Spec maps each operation m of P to its specification (am, gm , pm) where:

– am contains the resource names of the port P and the input and output parameters of m.

– gm is the firing condition of operation m, specifying the environments under which m can be activated.

– pm is a reactive design, describing the behaviour of m.

- Init identifies the initial states.
- Prot is a set of operations or service calling events

2.3 Service Component Architecture

The purpose of SCA is to provide a model for creating service-oriented component independent of any specific programming language and to unify the methods of encapsulation and communication in service-oriented architectures by providing a component model.

The SCA assembly model defines the component model and the configuration of SCA systems. The basic concepts of the assembly model are presented in Figure 2 :

• **Component:** A component is the unit of construction of Service Component Architecture and an instance of an implementation; it is characterized by services executing operations, properties and references to other services. Components can be combined into a composite.

• **Composite:** A composite contains assemblies of service components and wire them together. Composites also contain services, references and properties.

• **Service:** Services provided by the component for other components. A Composite Service can promote a Component Service.

• **Implementation:** Implementation is a program code implementing business functions; a component can implement different implementation technologies such as Java, C++, BPEL, etc.

• **Wire:** Wiring that describes the connections between services (source) and references (target) of components.

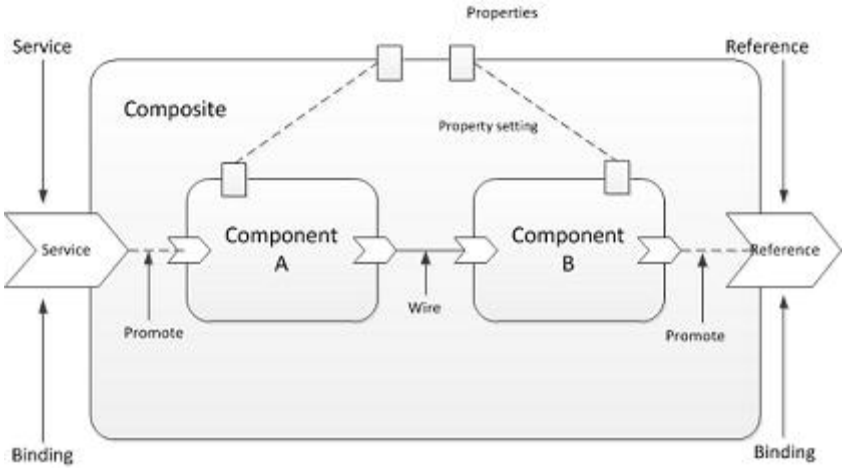

Figure 2. SCA Composite diagram [17]

2.4 Framework and Meta-Model for SCA

Zhang et al [61] realized a framework for modeling service-oriented systems. In this framework, the concept of "*service component* " is considered as a unit of first-class modeling to capture the functional unit linked to the abstract service.

A "*service component*" is autonomous, self-descriptive and can offer and deliver functionality to other "*component service*" through interfaces without displaying the implementation details.

The meta-model of the component service [60] is defined from the following aspects:

• **Identification:** Identifies a service component;

• **Specification:** Declares the service interfaces and service contracts;

• **Content:** Represents the implementation of an implicit component service and details of its realization;

• **Context:** Sets the environment in which the component service exists and where it can be adopted;

• **Choreography:** Specifies whether the service component is a composite service or not.

Du [23] proposed another meta-model that includes contracts to the service component.

Contracts can specify both the functional properties and the properties of QoS (Quality of Service) of the various system service components.

Then In 2009, Zhang et al. [60] presented a set of architectural building blocks for the Service Component layer in the Service-

Oriented Solution Stack reference architecture proposed by IBM.

Service Component Architecture is used to implement high-level services and complex systems; our works aims to ensure the dependability and reliability of service component oriented systems.

3. Quality Issues And Trustworthiness Of Service-Component Based Systems

As Service Component Architecture is a recent approach, there are very few research works for addressing quality issues and trustworthiness of service-component based systems.

To be considered secure, a system must exhibit trustworthiness and dependability properties. In this section we define the concept of dependability and main quality approaches and features.

3.1 Dependability

Dependability is a major requirement of the modern systems which consists of the system's ability to offer a trusted service. In this section we explain the concept of dependability and its main attributes.

Computer system dependability is the quality of the delivered service such that reliance can justifiably be placed on this service [33]. It can be divided into three parts: attributes, threats and means as shown in Figure 3.

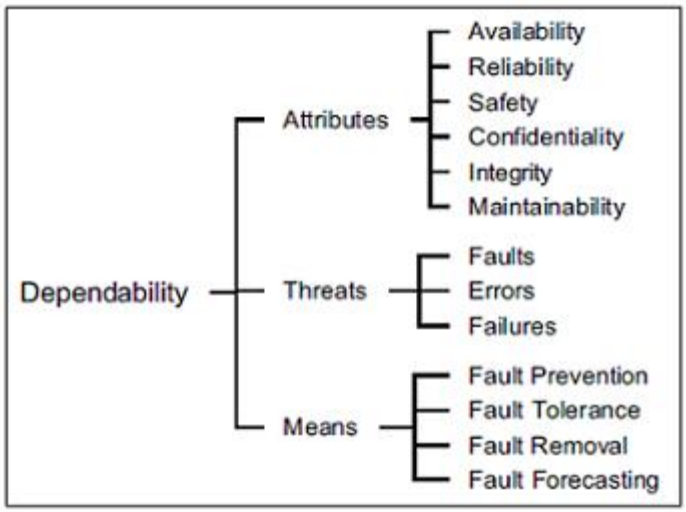

Figure 3. Dependability Tree [9]

According to Avizienis and Laprie [8] dependability is an integrating concept that encompasses the following basic attributes:

- **Availability:** Availability for the correct service,
- **Reliability:** Continuity of correct service,
- **Safety:** Absence of catastrophic consequences on the user and the environment,
- **Confidentiality:** The absence of unauthorized disclosure of information,
- **Integrity:** Absence of improper system state alterations,
- **Maintainability:** Ability to undergo repairs and modifications.

The three types of threats to dependability are faults, errors and failures [9]:

- **Failure:** When the system does not fulfill its intended function,
- **Fault:** A fault in the system that could lead to failure,

• **Error:** A manifestation of a fault,

Dependability can be achieved with the following means:

• **Fault Prediction:** Estimate of the presence, creation, and the consequences of mistakes in the assessment,

• **Fault Prevention:** Prevent the occurrence of errors, and decrease the likelihood of their occurrence,

• **Fault Tolerance:** Reduce the probability of failure despite the presence of faults,

• **Fault Removal:** Detect, identify and remove the faults of the system.

3.2 Main Quality Approaches

There are a wide variety of works to ensure systems quality, we have identified the main techniques used for componentbased systems during all phases of the system's life-cycle as shown in Figure 4.

Hence, in design phase, functional and extra-functional requirement are defined and expressed.

Figure 4. Quality efforts in CBSs

3.2.1 Design By Contract : Design by Contract [37] is an approach and method of software design. It is based on the legal definition of contracts which binds both parties and highlights the interest to precisely specify the interfaces behavior of a software component in terms of preconditions, post conditions and invariants.

Subsequently, the reliability of the components and the composite system is evaluated and predicted.

3.2.2 Reliability Evaluation and Prediction

The evaluation and prediction of reliability is to predict the failure rate of components and overall system reliability. They can be used in the operational phase and the early stages of system design software.

In addition, the system should continue to operate even in the presence of a failure of one of its components.

3.2.3 Fault Tolerance

Fault tolerance is the techniques and mechanisms that allow a system to be reliable, available and secure despite the presence of failures, it offers algorithmic and / or hardware solutions that allow applications to provide a quality service although being in an environment subject to failures.

Furthermore, the development and build process should conform to quality standards.

3.2.4 Quality Assurance Models

Quality Assurance is a planned and systematic pattern of all actions necessary to provide adequate confidence that the item or project conforms to established technical requirements [7]. Several standards and guidelines are proposed to control the quality activities of traditional software development process.

Finally, the achieved quality and trustworthiness is certificated and asserted.

3.2.5 Certification

There is no consensus on the definition of components certification in component-based systems engineering, Councill [19] proposed the following definition:

Third-party certification is a method to ensure software components conform to well-defined standards; based on this certification, trusted assemblies of components can be constructed.

4. Quality-Driven Models and Techniques for Component Based Systems

In this section we present a survey and an analysis of main techniques and models for quality assurance of componentbased systems in general.

4.1 Design by Contract

The contract based approach provides proofs of non-functional and quality properties without requiring the full formality of proof-directed and mathematical development. This approach is particularly appropriate in the component-based context. In fact, a pre-condition on the parameters of an operation or a service defines a contract that the required/given component agrees to respect.

Conversely, post-conditions on the return types of a required component define the customer's expectation from the service provider. Any violation of the contract is the manifestation of a software bug; a precondition violation is a bug in the client side and a post condition violation a bug in the supplier side.

Many contract extensions have been proposed for other programming and modeling languages like UML [58], [57] Java and C Sharp.

Programming languages like Java [27] and C Sharp [2] do not natively define the preconditions, post conditions and invariants. However, in both languages, there are research works for proposing extensions and tools to use design by contract and its components.

Java: Rieken [46] proposed a tool and a class library that allows better use of design by contract based on Jass (Java with assertions) [11].

Other extensions have been proposed for Java, as JContract [18] and jContractor [1] [30].

C Sharp: The C Sharp framework includes from version 4.0 a library that supports the notion of contract. Contracts are expressed using the static method calls to the inputs of the method. These methods are in the namespace System. Diagnostics.Contracts.

For component-based systems, Messabihi et al. [36] proposed multilevel contracts for components, however no complete contract feature has been proposed for service component until now.

4.2 Reliability Evaluation and Prediction

Reliability models of the system are classified into three types: state-based, path-based and propagation-based.

- **State-based:** uses control flow graph to represent the system architecture and software to predict the reliability analytically.
- **Path-based:** software reliability calculated by considering the possible execution paths of the program.
- **Propagation-based:** includes the dependence of component failures by considering the error propagation between system components.

Krishnamurthy [32] proposed a path-based method to estimate system reliability through the reliability of its components and sequence of components executed during system execution.

Cheung et al. [16] studied a new method for evaluating component reliability by constructing state models with an explicit error behavior. For this, they proposed a framework in three phases: Determination of the states, transitions determination and reliability calculation.

Singh [49] used another approach based on Bayesian networks to predict reliability of component-based systems. In other related works, we cite [48] and [28].

4.3 Fault Tolerance

Anderson and Lee [5] have identified four phases of fault tolerance: error detection, damage assessment, restoration of the state, and continuous service:

Error detection determines the presence of a fault by detecting an erroneous state in a system component.

Damage assessment determines the extent of damage to the system state caused by component failures and limit damage to the extent possible.

Restoration of the state realizes the recovery of the error by restoring the system to a defined state, error-free.

Continuation of service, despite the fault has been identified, means either that the fault must be repaired or the system must operate in a configuration where the effects of the fault no longer lead to an erroneous state.

Fault tolerance is a primary means of reliability of components-based systems. Mohamed and Zulkerine [40] classified the efforts of fault tolerance depending on the type of component into three main classes: programming based paradigm, environment based fault tolerance models based fault tolerance.

Fault tolerance is achieved by adapting traditional fault tolerance approaches to component-based systems, mainly by introducing software redundancy at the system architecture.

4.4 Quality Assurance Models

There are several standards for controlling and improving software and development process quality like ISO9001 and CMMI model [53], however, there is still no standard and effective process specific for component-based systems.

In the literature, several frameworks and quality assurance models have been proposed for the development of systems to bases of components like [15], [39] and [6].

A recent component quality framework is proposed Alvaro et al. [4], the framework consists of four modules:

- **Component Quality Model** that determines which quality characteristics should be considered and which subattributes are necessary,
- **Evaluation Techniques Framework** which defines a number of techniques to be applied to software components evaluation,
- **Evaluation Process** in charge of defining a set of techniques, metrics, models and tools to assess and certify software components, to establish an evaluation standard for components,
- **Metrics Framework** responsible for defining a set of measures to monitor the properties of components and control the evaluation process.

4.5 Certification

Research on the quality certification of component-based systems can be divided in time into two categories: pre-2001 research focused on mathematical proofs and tests, and then the research has focused on models and prediction techniques and on quality assessment [3].

In 1993, Poore [44] has developed an approach based the use of three mathematical models, using test cases to report failures

of a system. In 1994, Wohlin [59] presented the first method of certifying components using modeling techniques, other research has been done then as [47] [55] and [20].

For models and prediction techniques, Stafford et al. [51] proposed a model to test the functional and quality-related properties associated with a component. In 2003 McGregor et al. [35] proposed a method of component reliability (CoRe) that supports the empirical measure of the reliability of a software component.

Other models have been proposed as [31], [41] and [48].

4.6 Discussion

In this section we have presented a selection of techniques and models for assuring quality and trustworthiness of component-based systems.

We classify these approaches according to the following criteria as shown in table 1. :

- A priori or a posteriori approaches, the first one address quality issues in build and construction time and the second one is concerned by testing and correcting the final product.;
- Formality, some are based on formal mathematical proofs and other on informal methods and models;
- The level criteria identifies if quality is considered only in the component level or in the entire composite system;

	A Priori / Posteriori	Formality	Component Level	Target	FunctionalScope
Fault Tolerance	A priori	Formal	Component / Composite	Product	Non-Functional
Reliability Evaluation and Prediction	A priori	Formal	Component / Composite	Process	Non-Functional
Design By Contract	A priori / A posteriori	Formal	Component / Composite	Product	Functional / Non-Functional
Certification	A posteriori	Informal	Composite	Product	Functional / Non-Functional
Quality Assurance Models	A posteriori	Informal	Composite	Process	Non-Functional

Table 1. Classification of Quality Approaches

- The target criteria, the selected approaches focuses either on the process of construction and development or on the final product.
- The functional criteria determine if the quality approach handles functional and/or non-functional properties.

It is important that quality is considered during all stages of the development lifecycle of the software. In fact, the contract-based approach allows both defining the desired quality properties and verifying and validating their accuracy.

Furthermore, contract-based approach allows specifying and verifying functional and non-functional requirements for both component and component assemblies' level.

Based on this evaluation, we conclude that design by contract is appropriate for component-based systems.

5. Multilevel Contract Model

5.1 Why Contracts?

Dependability is a major requirement of modern systems which consists of the system's ability to offer a trusted service. It is important to be able to affirm the respect of quality assertions of these systems.

To meet these requirements, we choose a contractual approach [45]. Indeed, within the component and service paradigms, contracts have become an integral part of their definition [52]:

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

A contract defines the constraints between components that is to say, the rights and obligations between the service provider and the client. It has the advantage of expressing the conditions of use of a service by clarifying the obligations and benefits of stakeholders.

In Du's definition of contract in the service component model [15], the component requirements and obligations can be expressed in the Spec property.

We believe that design by contracts can address some of the quality problems of large and complex systems development by explicitly specifying functional and non-functional properties of its components.

Unlike mathematical evaluation and prediction techniques, the contract-based approach is a light-weight formal method for specifying and designing quality-driven systems, it can be introduced in an early stage during the design phase.

To our knowledge, there is still no research work for introducing the concept of contract in service component based systems in order to manage and handle quality requirements.

5.2 Contracts Categories

Beugnard [13] proposed a classification of contracts into four categories:

- **Basic contracts** that ensure the possibility of running the system properly;
- **Behavioral contracts** that improve trust in the system functionalities;
- **Synchronization contracts** that specify synchronization strategies and policies;
- **QoS contracts** which is the highest level and specify quality of service attributes.

This classification has a good coverage of functional and qualitative aspects of components, nevertheless, they don't handle trustworthiness of composition operations and composite components, yet we have defined three levels of component contracts:

- **Intra-component contracts** concerns the good operations of the component and the respect of its quality requirements;
- **Inter-component / Compositional contracts** ensure the safe composition and trusted assembly of components;
- **System contracts** concerns properties and requirement of the whole system.

5.3 Contracts by Aspects

Separation of concerns is the process of dividing software into distinct features that overlap in functionality as little as possible, Aspect-Oriented Programming (AOP) [25] aims at providing a means to identify and modularize crosscutting concerns, by encapsulating them in a new unit called aspect.

It was already stated that the design by contract methodology is an aspect of the software system [22]. As such, a contract can be expressed in AOP terminology.

Lorenz [34] classifies aspects for design-by-contract in three types: agnostic aspects that don't affect a method's assertions, obedient aspects where the input and output states remains unchanged and rebellious aspects which changes the behavior of existing methods.

Our proposed solution is based on the aspect oriented programming (AOP) for building contract-aware service component based systems.

As AspectJ [54] is one of the first and best known Aspect-Oriented Programming tools, we choose it to implement our approach. In addition, we formalize contracts in UML's Object Constraint Language (OCL) [42] which is a concrete specification language that will help improving the expressiveness of the contracts.

5.4 RelatedWorks

Some research works related to implementing contracts by using aspects were proposed in the literature, as Contract4J [56] and ContractJava [26].

Contract4J [56] is an open source tool that uses Java 5 annotations and AspectJ. Contract4J offers three annotation types: pre, post-conditions, and class-invariants. However, although it is still functional Contract4J is no longer maintained since 2007.

ContractJava [26] is a Java extension in which contracts are speciïfied in interfaces. However, class invariants and "*result* " or "*old* " variables are not supported.

Handshake [24] is a Java extension that can be enabled where contracts are specified in a separate file with special syntax. However it is not compatible with recent JVM releases.

CONA [50] is a tool that extends the Java and AspectJ syntax with support for Design by Contract and enforces their runtime validation. However its architecture is very complex.

Besides they are not suitable for component-based systems they are mostly limited and academic tools and do not offer a complete and available framework.

Furthermore, our approach is more generic; it handles both functional and non-functional properties of service component, and covers the single component and the composite levels.

6. Conclusion

The dependability is the system's property that allows users to place a justified confidence in the quality of the service it delivers. The purpose of the research efforts in this field is to specify, design, build and verify systems where the fault is natural, expected and tolerable.

This work presented an analysis of main techniques and models for modeling and verifying quality-driven systems; we have listed several approaches that cover different phases of systems construction and we concluded that contractbased approach is very suitable for component-based systems in general and consequently for service component based systems. It handles all stages of the system's development lifecycle and allows both defining the desired quality properties and verifying and validating their accuracy.

Contracts is a design approach for describing both functional and non-functional properties of complex and qualitydriven systems, it also involves synchronization and Quality of Service (QoS) aspects. We will implement it using aspect oriented programming.

As a continuation of this work, our objective is to propose a modeling framework with a tooling environment and adapt it to Service Component Architecture for safety-critical and quality sensitive systems.

References

[1] Parker Abercrombiea, Murat Karaormanb. (2002). jcontractor: Bytecode instrumentation techniques for implementing design by contract in java. *Electronic Notes in Theoretical Computer Science*, 70.

[2] Joseph Albahari, Ben Albahari. (2010). *C 4.0 in a Nutshell: The Definitive Reference*. O'Reilly Media.

[3] Alexandre Alvaro, Eduardo Santana de Almeida, Silvio Romero de Lemos Meira. (2005). Software component certification: A survey. *In*: *Proceedings of the* 31st *EUROMICRO Conference on Software Engineering and Advanced Applications*, 106 – 113.

[4] Alexandre Alvaro, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. A software component quality framework. *ACM SIGSOFT Software Engineering Notes*, 35 (1) 1–18, January.

[5] Anderson, T., Lee, P. A. (1981). *Fault tolerance: Principles and Practice*. Prentice-Hall.

[6] Andreas S. Andreou, Marios Tziakouris. (2007). A quality framework for developing and evaluating original software components. *Information and Software Technology*.

[7] ANSI/IEEE. (1981). *IEEE Standard for Software Quality Assurance Plans*.

[8] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell. (2001). Fundamental concepts of computer system dependability. *In*: *Proceedings of IARP/IEEERAS Workshop on Robot Dependability:Technological Challenge of Dependable, Robots in Human Environments*.

[9] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell. (2004). Dependability and its threats - a taxonomy. *IFIP Congress Topical Sessions*.

[10] Graham Barber. Sca policy framework specification.

[11] Detlef Bartetzko, Clemens Fischer, Michael Möller, Heike Wehrheim. (2001). Jass - java with assertions. Technical report.

[12] Michael Beisiegel. (2007). Service component architecture specification.

[13] Antoine Beugnard, Jean-Marc Jézéquel, Noël Plouzeau, Damien Watkins. (1999). Making components contract aware. *Computer*, 32, 38–45.

[14] Guy Bieber, Jeff Carpenter. (2002). Introduction to service-oriented programming.

[15] Xia Cai, Michael R. Lyu, Kam-Fai Wong, Roy Ko. (2000). Component-based software engineering: Technologies, development frameworks, and quality assurance schemes. *International Journal of Software Engineering and Knowledge Engineering*.

[16] Leslie Cheung, Roshanak Roshandel, Leana Golubchik Nenad Medvidovic. (2008). Early prediction of software component reliability. *In*: ICSE '08 *Proceedings of the* 30th *International Conference on Software Engineering*, p. 111 – 120. ACM.

[17] SCA Consortium. (2005). Building systems using a service oriented architecture. Technical report, SCA Consortium.

[18] Parasoft Corporation. (2005). Using design by contract to automate java software testing. Technical report.

[19] Bill Councill. (2001). Third-party certification and its required elements. *In*: *Proceedings of the* 4th *Workshop on Component-Based Software Engineering*.

[20] Bill Councill, George T. Heineman. (2000). Component-based software engineering and the issue of trust. *In*: *Proceedings of the* 22nd *International Conference on Software Engineering*.

[21] Zuohua Ding, Zhenbang Chen, Jing Liu. (2008). A rigorous model of service component architecture. *Electronic Notes in Theoretical Computer Science*, 207, 33–48.

[22] Filippo Diotalevi. (2004). *Contract Enforcement With AOP*. IBM.

[23] Dehui Du, Jing Liu, Honghua Cao. (2008). A rigorous model of contract-based service component architecture. *IEEE Computer Society*, 2, 409–412.

[24] Andrew Duncan, Urs Hoelzle. (1998). Adding contracts to java with handshake. Technical report, University of California.

[25] Robert E. Filman, Tzilla Elrad, Siobhán Clarke, Mehmet Aksit. (2004). *Aspect-Oriented Software Development*. Addison-Wesley.

[26] Robert Bruce Findler, Matthias Felleisen. (2001). Contract soudness for object oriented languages. *In*: Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications.

[27] James Gosling, Jr. Steele Guy L., Bill Joy. (2005). *The Java(TM) Language Specification*. Addison-Wesley.

[28] Latife Görkemli, Selda Kapan Ulusoy. (2010). Fuzzy bayesian reliability and availability analysis of production systems. *Computers and Industrial Engineering*, 59, 690–696.

[29] Ulla Isaksen, Jonathan P. Bowen, Nimal Nissanke. (1996). System and software safety in critical systems. Technical report, The University of Reading.

[30] Murat Karaorman, Parker Abercrombie. (2005). jcontractor: Introducing design-by-contract to java using refletive bytecode instrumentation. *Formal Methods in System Design*, 27, 275–312.

[31] Daniel Karlsson. (2006). *Verification of Component-based Embedded Systems Designs*. PhD thesis, Linkoping University.

[32] Krishnamurthy, S., Aditya P. Mathur. (1997). On the estimation of reliability of a software system using reliabilities of its components. *In*: Proceedings of the Eighth International Symposium on Software Reliability Engineering.

[33] Jean-Claude Laprie. (1995). *Guide de la sûreté de fonctionnement*.

[34] David H. Lorenz, Therapon Skotiniotis. (2005). Extending design by contract for aspect-oriented programming. Technical report.

[35] John D. McGregor, Judith A. Stafford, Il-Hyung Cho. (2003). Measuring component reliability. *In*: Proceedings of the 6th Workshop on Component-Based Software Engineering, p. 13–24.

[36] Mohamed Messabihi, Pascal André, Christian Attiogbé. (2010). Multilevel contracts for trusted components. *In*: *Proceedings of DCNET/OPTICS*, 2010.

[37] Bertrand Meyer. (1992). *Applying Design by Contract. Computer*.

[38] Bertrand Meyer. (1997). *Object-Oriented Software Construction*.

[39] Bertrand Meyer. (2003). The grand challenge of trusted components. *In*: Proceedings of the 25th International Conference on Software Engineering, p. 660-667.

[40] Atef Mohamed, Mohammad Zulkernine. (2009). A comparative study on the reliability efforts in component-based software systems. Technical report, Queen's University.

[41] Regina Moraes, João Durães, Eliane Martinset, Henrique Madeira. (2007). Component based software certification based on experimental risk assessment.

[42] OMG. (2005). *Unified Modeling Language* (*UML*) *2.0 OCL Convenience Document*.

[43] Mike P. Papazoglou, Dimitrios Georgakopoulos. (2003). Service-oriented computing. *Communcications of the ACM - Service-Oriented Computing*, 46.

[44] Jesse H. Poore, Harlan D. Mills, David Mutchler. (1993). Planning and certifying software system reliability. *IEEE Software*, 10.

[45] Maryem Rhanoui, Bouchra El Asri. (2012). Toward a quality-driven service component architecture, techniques and models. *In:* Proceedings of the 14th International Conference on Enterprise Information System.

[46] Johannes Rieken. (2007). *Design By Contract for Java ÝU Revised. PhD thesis*, Carl von Ossietzky Universität Oldenburg.

[47] Sharon L. Rohde, Karen A. Dyson, Pamela T. Geriner, Deborah A. Cerino. (1996). Certification of reusable software omponents: Summary of work in progress. *In*: Proceedings of the 2nd IEEE International Conference on Engineering of Complex Computer Systems.

[48] Roshanak Roshandel, Nenad Medvidovic, Leana Golubchik. (2007). A bayesian model for predicting reliability of software systems at the architectural level. *In*: Proceedings of the Quality of software architectures 3rd international conference on Software architectures, components, and applications.

[49] Harshinder Singh, Vittorio Cortellessa, Bojan Cukic, Erdogan Gunel, Bharadwaj, V. (2001). A bayesian approach to reliability prediction and assessment of component based systems. *In*: *Proceedings of the* 12*th International Symposium on Software Reliability Engineering*.

[50] Therapon Skotiniotis, David H. Lorenz. (2004). Conaj: Generating contracts as aspects. Technical report, College of Computer and Information Science, Northeastern University.

[51] Judith Stafford, Kurt Wallnau. (2001). Is third party certification necessary? *In*: Proceedings of the 4th Workshop on Component-Based Software Engineering.

[52] Clemens Szyperski. (2002). *Component Software : Beyond Object-Oriented Programming. Addison-Wesley.*

[53] CMMI Product Team. (2009). Capability maturity model integration. Technical report, Software Engineering Institute.

[54] The AspectJ Team. (2003). *The AspectJ Programming Guide*.

[55] Jeffrey M. Voas. (1998). Certifying off-the-shelf software components. 31, 53–59.

[56] Dean Wampler. (2006). Contract4j for design by contract in java: Design pattern-like protocols and aspect interfaces. *In*: Proceedings of the Fifth AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software.

[57] Jos Warmer, Anneke Kleppe. (2003). *The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley*.

[58] Torben Weis, Christian Becker, Kurt Geihs, Noël Plouzeau. (2001). A uml meta-model for contract aware components. *In*: Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools.

[59] ClaesWohlin, Per Runeson. (1994). Certification of software components. *IEEE Transactions on Software Engineering*, 20.

[60] Liang-Jie Zhang, Jia Zhang. (2009). Design of service component layer in soa reference architecture. *In*: Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference.

[61] Tao Zhang, Shi Ying, Sheng Cao, Xiangyang Jia. (2006). A modeling framework for service-oriented architecture. *In*: Proceedings of the Sixth International Conference on Quality Software, p. 219 – 226.