# A Multilevel Contract Model for Quality-Driven Service Component Architecture

Maryem Rhanoui
IMS Team, SIME Laboratory
ENSIAS
Rabat, Morocco
mrhanoui@gmail.com

Bouchra El Asri
IMS Team, SIME Laboratory
ENSIAS
Rabat, Morocco
elasri@ensias.ma

*Abstract*—**Service Component Architecture (SCA) is a recent approach and an industry standard for developing complex and distributed systems. Despite the growing research work it still lacks a formal basis for handling trust and reliability of quality-driven systems. In this paper, we present main techniques and models for assuring quality and trustworthiness of component-based systems in general, and then we present our contract-aware service component meta model. We propose a multilevel contract model that aims to address reliability and quality issues for service component oriented systems by expressing a set of its properties and constraints.**

***Keywords-Service Component; Service Component Architecture; Quality-Driven System; Contract; Aspects.***

## I. INTRODUCTION

Service Oriented Architecture (SOA) 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.

In the context of a growing interest in reuse of business components, the development of critical and complex systems is confronted with limitations and challenges as service assembly difficulties and the complexity related to numerous SOA standards, therefore SCA emerged as a unifying response.

Official SCA specification document includes SCA assembly model specification [1] and SCA policy framework [2]. 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 [3], our literature review shows that most research efforts have focused on technical aspects of Service Component Architecture, 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 [4] 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 [5], 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 present main techniques and models for handling quality and trustworthiness of component-based systems.

Among the presented approaches, we are interested by the contract-based approach [6], 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.

Our contribution is as follow: we propose a multilevel contract model for modeling both functional and non-functional / quality properties of service components, this model covers different levels of systems, that is the component, composite and final system. Furthermore, it will allow the verification and validation of the constraints outlined in the contract.

In this article, we propose a meta-model for multilevel contracts for service component architecture.

The remainder of this paper is organized as follows: Section II will be dedicated to the presentation of the concept of service component and a survey of main techniques and models for assuring trustworthiness and quality of component-based systems.

Section III will present and justify the choice of our proposed multilevel contract model. In Section IV we will present a meta-model for contract-aware service component architecture. Finally in Section V we illustrate our approach with a case study.

## II. QUALITY-DRIVEN SERVICE COMPONENT ARCHITECTURE

Service-oriented computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications [7]. Service Component Architecture (SCA) proposes a programming model for building components based applications following the SOA paradigm.

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.

In this section, we describe, at a glance, the SCA architecture, present the component model and survey the main quality approaches for component-based systems.

## A. Service Component Architecture

### 1) Architecture

An SCA application consists of one or more components that can be implemented in different languages.

A component is a software entity and the basic element of a business function that contains zero or more services and / or reference. A component may have properties and can be either an implementation itself, or a composite. Fig. 1 shows an example of SCA component.

### 2) Benefits

SCA had emerged as a new architecture for addressing complexity issues of developing SOA solutions. Its offers many advantages:

- Simplify the development of business component and assembly and deployment of business solutions built as networks of services;
- Increase agility and flexibility and protects business logic assets by shielding from low-level technology change and improves testability.

### 3) Component Model

Various component models of Service Component Architecture were proposed in literature.

For Ding [8] 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.

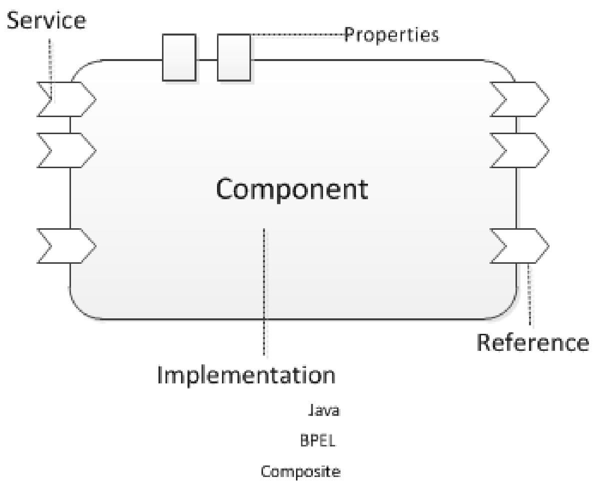

Figure 1 - SCA Component [10]

Moreover, Du et al [9] 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.

## B. Quality Approaches

There are a wide variety of works and techniques to ensure systems quality, we have identified the main techniques used for component-based systems during all phases of the system's life-cycle as shown in Fig. 2.

Hence, in design phase, functional and extra-functional requirements (as reliability, availability…) are defined and expressed. For this, Design by Contract [6] 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 pre-conditions, post conditions and invariants.

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

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.

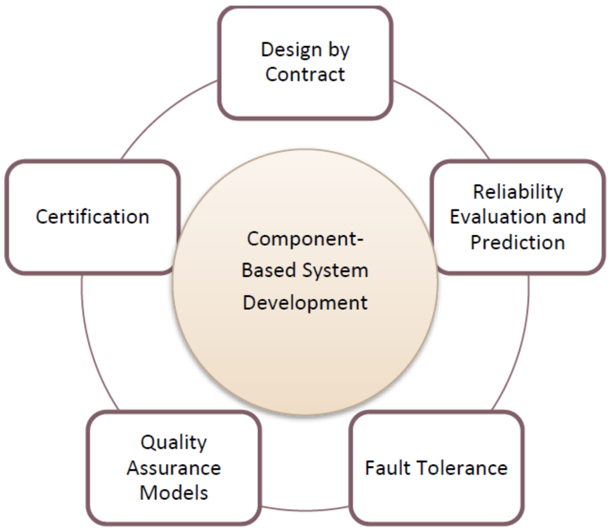
Design by Contract
Certification
Reliability Evaluation and Prediction
Component-Based System Development
Quality Assurance Models
Fault Tolerance

Figure 2 - Quality efforts in CBSs

In addition, the system should continue to operate even in the presence of a failure of one of its components; fault tolerance is the techniques and mechanisms that allow a system to be reliable, available and secure despite the presence of failures.

Furthermore, the development and build process should conform to quality standards; quality assurance is a planned and systematic pattern of all necessary actions to ensure that the item or project conforms to technical requirements [11].

Finally, the achieved quality and trustworthiness is certificated and asserted. Third-party certification is a method to ensure software components are conform to the defined standards; based on this certification, trusted assemblies of components can be constructed [12].

## III. MULTILEVEL CONTRACTS

### A. *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.

The requirements can be specified as preconditions, post conditions and invariants.

- **A precondition** is a constraint that must meet a client when calling a service.
- **A postcondition** is a constraint that must be met by the supplier after use of the service.
- **Invariants** are constraints that must meet all entities that fold to the contract.

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 pre-condition violation is a bug in the client side and a post condition violation a bug in the supplier side.

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.

### B. *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 [13]. Indeed, within the component and service paradigms, contracts have become an integral part of their definition [14]:

*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.

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.

### C. *Contracts Levels*

Beugnard [15] 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.

### D. *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) [16] 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 [17]. As such, a contract can be expressed in AOP terminology.

Lorenz [18] 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
- **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. The essential advantage of AOP is the externalization and isolation of crosscutting concerns so that requirement contracts will be expressed outside of the business code of the system.

As AspectJ [19] is one of the first and best known Aspect-Oriented Programming tools, we choose it to implement our approach.

### E. *Constraint Specification Language*

In addition, we formalize contracts in UML's Object Constraint Language (OCL) [20], which is a concrete specification language that will help improving the expressiveness of the contracts.

## IV. META MODEL

As part of the Model Driven Architecture [21], the Object Management Group (OMG) has defined a meta-metamodel called MOF (Meta Object Facility) [22]. MOF is a specification that defines the concepts to be used to define meta-models.

We propose a MOF compliant meta-model of quality-driven service component architecture; we introduce contract concept and a support of quality requirements as shown in Fig 3.

The meta-model can be divided into two parts: the service component meta-model and its extension with multilevel contract.

### A. *Service Component*

**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*.

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

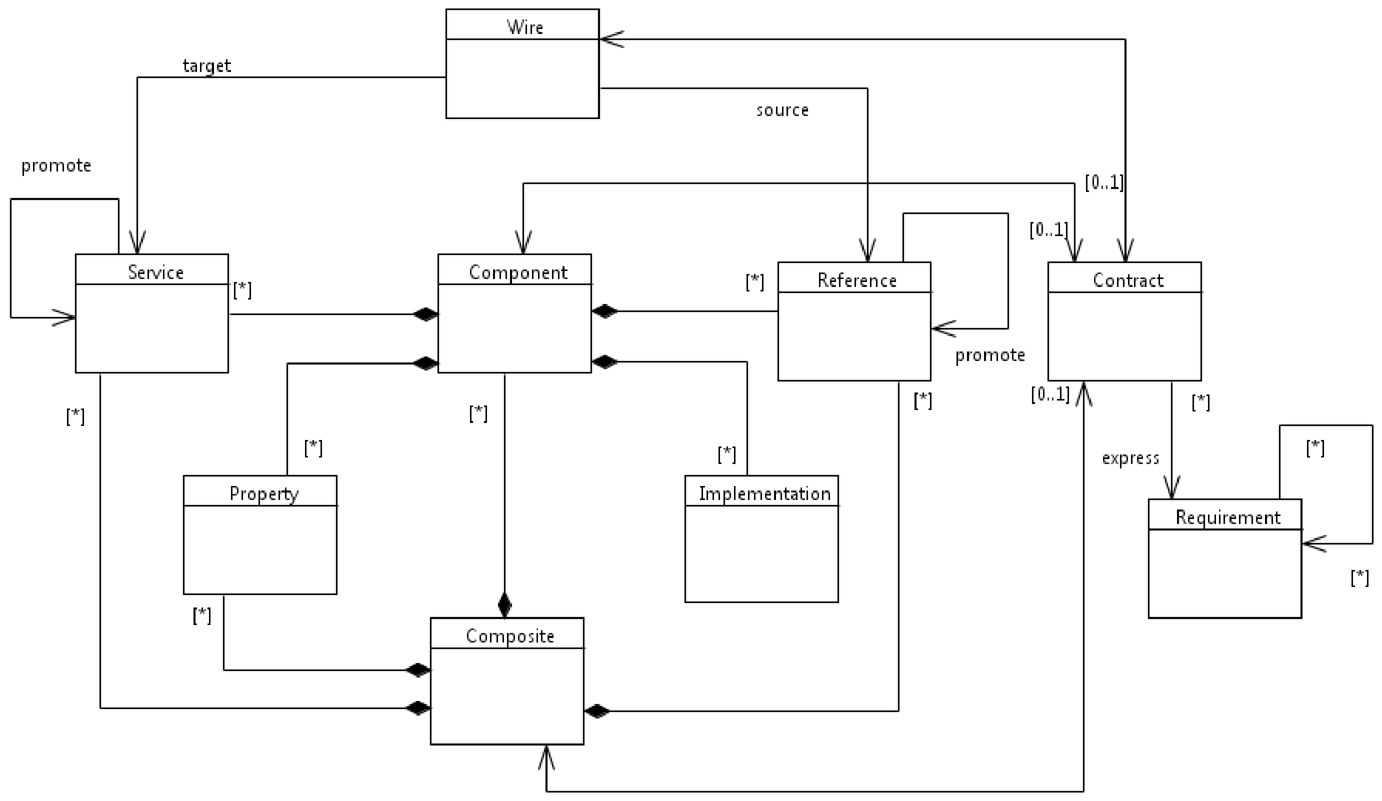

Figure 3 - Service Component Meta Model

**Reference:** Services required by the *component* from other components. A Component Reference can promote a Composite Service.

**Property:** External set values or attributes of a *component* or *composite*.

**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*.

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

## B. *Contract*

**Constraint:** The expressed constraints of the system, defines the obligations that must be verified by the software components.

**Contract:** A contract specifies the interfaces behavior of a *component* in terms of a configuration of pre-conditions, postconditions and invariants.

A contract is associated to:

- Component element, to ensure the good operations of the component and the respect of its quality requirements.
- Composite element, to ensure the good operations of the composite and the respect of its quality requirements.
- Wire element, to define a contract on the binding of two components.

A contract can express both functional and nonfunctional *requirements*.

**Requirement:** Functional and non-functional requirements expressed by the *contract*. The requirements are described by a structure of boolean expressions and can be constituted of a set of other requirements. A functional requirement is a property related to the functionality of a the service component. A non-functional requirement is the quality or characteristics of a service component that determines how and under which conditions the service will be delivered. These requirements are not directly related to the functionality provided by the component.

## V. CASE STUDY

In this section, we present a simplified case study to illustrate our multilevel contract approach and apply the enunciated concepts. We consider an Airport Management System. The airport has high reliability and dependability requirements. Our system is composed of five components as shown in Fig. 4.

**BoardingComponent** manage the boarding operations in the airport, it has one service which is promoted by the composite and has two references toward **CheckInComponent** and **SecurityInformationComponent.**

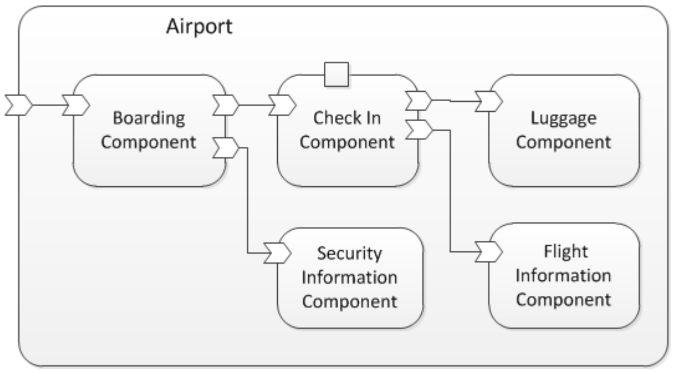

Figure 4 - Airport Management System

**CheckInComponent** manage the check in process of the passengers, it has one service wired with **BoardingComponent** and has a reference toward **LuggageComponent**.

**SecurityInformationComponent** manage the security information of the passengers. It has one service wired with **BoardingComponent.**

**LuggageComponent** manage the luggage check and control of the passengers.

**FlightInformationComponent** gives the necessary informations of the flights in the airport. It has one service wired with **CheckInComponent.**

To ensure the reliability of our system, we first identify its requirements and then we define the corresponding contracts.

To check in, the passengers must respect the check-in deadline, that is to say, the time beyond which they cannot not register or leave their luggage. Depending on the destination and departure airport, the check-in deadline varies from 15 to 90 minutes before departure time. Moreover, the check in service has to be available 7d/7 at any time of the day and respond within an acceptable period of time. This represents the functional and non-functional requirements of the check in component, then we define a component contract.

Furthermore, the component has two references towards luggage and flight information components; its good operation depends on the correct assembly of these components. Then we define an inter-component or assembly contract.

Finally the airport management system should be reliable and available, which correspond to a system quality contract.

The functional and quality contracts are defined in the design phase of the system development lifecycle, are implemented in the construction phase and are verified during the execution of the components, which allows us to monitor and confirm the compliance with the formerly defined requirements.

## VI. Related Works

Some research works related to implementing contracts by using aspects were proposed in the literature, as Contract4J [23] and ContractJava [24].

**Contract4J** [23] 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** [24] is a Java extension in which contracts are specified in interfaces. However, class invariants and "result" or "old" variables are not supported.

**Handshake** [25] 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** [26] 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.

## VII. Conclusion

This work presented our proposed meta-model of multilevel contract for service component architecture.

Based on a review of main techniques and models for modeling and verifying quality-driven systems; we concluded that contract-based approach is very suitable for component-based systems in general and consequently for service component based systems.

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

We propose a multilevel contract model for expressing and verifying functional and non-functional properties in all levels of service component based systems.

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] M. Beisiegel, "Service Component Architecture Specification." 2007.

[2] G. Barber, "SCA Policy Framework Specification", 2011, Available: http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1.html. [Sep. 15, 2012].

[3] B. Meyer, *Object-Oriented Software Construction*. 1997.

[4] U. Isaksen, J. P. Bowen, and N. Nissanke, "System and Software Safety in Critical Systems," 1996.

[5] J.-C. Laprie, *Guide de la sûreté de fonctionnement*. 1995.

[6] B. Meyer, "Applying 'Design by Contract'," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[7] M. P. Papazoglou and D. Georgakopoulos, "Introduction: Service-Oriented Computing," *Communcications of the ACM - Service-Oriented Computing*, vol. 46, no. 10, pp. 24–28, 2003.

[8] Z. Ding, Z. Chen, and J. Liu, "A Rigorous Model of Service Component Architecture," *Electronic Notes in Theoretical Computer Science*, vol. 207, pp. 33–48, 2008.

[9] D. Du, J. Liu, and H. Cao, "A Rigorous Model of Contract-based Service Component Architecture," *IEEE Computer Society*, vol. 2, pp. 409–412, 2008.

[10] SCA Consortium, "Service Component Architecture - Building Systems using a Service Oriented Architecture." 2005.

[11] ANSI/IEEE Std. 730-1981, IEEE Standard for Software Quality Assurance Plans, 1981.

[12] B. Councill, "Third-Party Certification and Its Required Elements," in *Proceedings of the 4th Workshop on Component-Based Software Engineering*, 2001.

[13] M. Rhanoui and B. E. Asri, "Toward a Quality-Driven Service Component Architecture, Techniques and Models," in *Proceedings of the 14th International Conference on Enterprise Information System*, pp. 192–196, 2012.

[14] C. Szyperski, *Component Software : Beyond Object-Oriented Programming*. Addison-Wesley, 2002.

[15] A. Beugnard, J.-M. Jezequel, N. Plouzeau, and D. Watkins, "Making Components Contract Aware," *Computer*, vol. 32, pp. 38–45, 1999.

[16] R. E. Filman, T. Elrad, S. Clarke, and M. Aksit, *Aspect-Oriented Software Development*. Addison-Wesley, 2004.

[17] F. Diotalevi, *Contract Enforcement With AOP*. IBM, 2004.

[18] D. H. Lorenz and T. Skotiniotis, "Extending Design by Contract for Aspect-Oriented Programming," 2005.

[19] The AspectJ Team, "The AspectJ Programming Guide." 2003.

[20] Object Management Group, "Unified Modeling Language (UML) 2.0 OCL Convenience Document." 2005.

[21] Object Management Group, "Model Driven Architecture (MDA)," 2003.

[22] Object Management Group, "Meta Object Facility (MOF) V2.4.1." 2011.

[23] D. Wampler, "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*, 2006.

[24] R. B. Findler and M. Felleisen, "Contract Soudness for Object Oriented Languages," in *Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 2001.

[25] A. Duncan and U. Hoelzle, "Adding Contracts to Java with Handshake," 1998.

[26] T. Skotiniotis and D. H. Lorenz, "Conaj: Generating Contracts as Aspects," 2004.