A Contractual Specification of Functional and Non-Functional Requirements of Domain-Specific Components

Maryem Rhanoui[1], Bouchra El Asri[2]

**[1] IMS Team, SIME Laboratory, ENSIAS, Mohammed V Souissi University
Rabat, Morocco**

**[2] IMS Team, SIME Laboratory, ENSIAS, Mohammed V Souissi University
Rabat, Morocco**

Abstract
Unlike traditional software engineering that aims to satisfy the requirements of a single system, domain-specific component-based engineering focuses on providing reusable solution for a family of systems. To be adopted in a safety-critical environment it must handle quality requirements and offer mechanisms to ensure the reliability level of the components and the system. For this purpose, the contract-based approach is a lightweight formal method for designing and specifying systems' requirements, it can be introduced in an early stage during the design phase.
In this paper, we present a multilevel contract model and a domain-specific modeling language that aims to address reliability and quality issues for component oriented systems by expressing and specifying a set of its properties and constraints.
Keywords: *Contract, Domain-Specific Components, Domain-Specific Language, Feature-Oriented Domain Analysis.*

1. Introduction

With the growing size and complexity of software systems, reuse has emerged as a promising methodology for system's construction; it consists in creating and in assembling systems with existing components. In order to promote large-scale reuse and reduce development costs, domain-specific components engineering aims at designing and implementing a family of systems so as to produce qualitative applications in a particular domain.

The software product line engineering is a novel approach which allows developing a multitude of products or software systems with a considerable gain in terms of cost, time and quality; it consists in developing a line of products rather than individual systems. Domain engineering [1] is the foundation of the construction of a system with reusable components and consists in defining the commonality and the variability of the product line and developing and building assets (reusable artefacts) which will be reused for the construction of products.

Among the methods to develop a software product lines, feature-oriented software development (FOSD) [2] is an approach for the development and customization of large-scale software systems, it implements features as first-class element to model, design, and implement software. The fundamental principle of FOSD is the modularization of the system features in order to optimize scalability and reusability.

In the context of a growing interest in reuse of business components, the development of critical and complex systems is confronted with dependability and reliability limitations and challenges. As a matter of fact, dependability [3], 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. To design a dependable and reliable system, it is necessary to handle functional requirements and behavioral relationships between components, and also take into account the quality of service properties.

As an expanding approach, the development of feature-oriented components still needs more formal models and frameworks for modeling and verifying systems. We are interested by the contract-based approach initiated by Meyer [4], which is a lightweight 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 present work is intended to contribute to the specification and verification of systems' requirements. Our contribution is as follows: we propose a formal contract model and a textual domain-specific language for modeling and specifying functional and non-functional / quality properties of domain-specific components, the model covers different levels of the system, which is the feature, component, and composition levels. Furthermore, it allows the verification and validation of the constraints outlined in the contract.

The remainder of this paper is organized as follows: section 2 provides a general overview of the knowledge base and motivation of our work, section 3 is dedicated to the presentation of the multilevel contract model then section 4 describes the contract specification language. Section 5 explains the motivating example and highlights our work problematic. Finally, section 6 positions our approach with related works, while section 7 concludes the paper.

2. Background and Motivations

Component-based software development is a major asset for the construction of complex and large systems. Components are reused for different products in the software product line. To develop a product line, common and variable components are identified and developed to meet the requirements of a specific application in a particular domain, the process of the development of these components is called Domain Engineering. It consists in developing and in building reusable artefacts that will be reused for the construction of products. In this section we present the main concepts of our work and the motivations and interest of our approach.

2.1. Software Product Line

The software product lines is a recent approach that favors systematic reuse throughout the software development process and enables the development of a set of software products with a considerable gain in terms of cost, time and quality. A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [5].

Software product line engineering is based on two fundamental principles, domain engineering and application engineering as well as the management of variability and commonality. The domain engineering is the development of assets which will be used in the product line, whereas the application engineering concerns the construction of final products with specific requirements.

2.2. Feature Oriented Domain Analysis

Feature-Oriented Software Development (FOSD) is a methodology for the design and construction of software product lines based on the separation of concern [6], each concern is modularized in a separate component called feature. Features are used as first-class entities to analyze, design, implement, customize, debug or develop a software system [2].

The commonalities and variabilities among products in the same domain can be expressed in terms of features. A feature is a prominent or distinctive and user visible aspect, quality, or characteristic of a software system or systems [7]. A feature is either [8]:

- **Mandatory**, it exists in all products,
- **Optional**, it is not present in all products
- **Alternative** (One Of), it specializes more general feature; only one option can be chosen from a set of features.
- **Or** : One or more features may be included in the product

FODA is a domain analysis method that focuses on providing a complete description of the features of the domain, giving less attention to the phases of design and implementation [7]. It combines the advantages of the component-based approach and domain engineering by providing generic components that improves components reuse and optimization.

2.3. A Contract Specification Language

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.

To satisfy quality requirements, we propose a contract specification language to support the model-based development of domain-specific component-based systems. Indeed, within the component and service paradigms, contracts have become an integral part of their definition [9] "*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*." The contract approach allows considering the qualitative aspects in all stages of the system development cycle, it is used from early requirements capture, to system design and implementation.

The contract language provides means for the specification and verification of functional and non-functional properties of component-based systems without requiring the full formality of proof-directed and mathematical development.

A domain-specific language DSL [10] is a programming language restricted to a precise domain to solve problems determined in a particular context. The use of a domain specific language provides several benefits compared to general languages such as improving productivity, data quality and longevity, it provides mechanisms for validation and verification and involves the intervention of the domain experts [11]. Unlike General Purpose Language, a DSL is more expressive in a particular domain and provides the possibility of using notations and constructions adapted to the field, which is a substantial advantage in terms of expressiveness and ease of use.

3. Multilevel Contract Artifact

The development of a domain-specific modeling language involves the following steps: domain analysis, design and implementation [12]:

- **Domain analysis**: The analysis identifies the domain and gathers domain knowledge. It defines the scope and terminology of the domain, describes the key concepts and provides a feature model describing the commonalities and variabilities of the domain.
- **Design**: The language design is essentially to define its abstract and concrete syntax and semantics.
- **Implementation**: The implementation of the language is to develop a library that implements the semantic concepts of the domain and a compiler.

3.1. Contract Model

The first step in defining a specific language is the domain analysis, the concept was introduced by Neighbors [13] to describe the study of the problem domain of a family of applications. It consists in the identification, acquisition and development of reusable information about a problem domain to be reused for the specification and construction of software.

The domain analysis identifies the main language concepts and the relationships between them. Feature Oriented Domain Analysis (FODA) is a methodology for domain analysis [12] where the concepts of the language are represented as features in a feature model. It is particularly adapted to the construction of reusable elements.

A contract is a mechanism that explicitly specifies behavior, requirements and interaction between components in order to improve the quality and dependability of the system and facilitate its understanding. The principle of design by contract dates back to the Hoare logic [14], Hoare triples provides a mathematical notation to express correction formulas. A correction formula is an expression of the form:

$$\{ \varphi \} \mathbf{A} \{ \psi \}$$

This means that "any execution of **A**, starting in a state where φ is true, will end in a state where ψ holds". In the field of software engineering, **A** denotes an operation or software while φ and ψ define assertions, respectively precondition and postcondition.

The contract models the relationship between an entity and its clients as a formal agreement, expressing and precisely defining the rights and obligations of each party in order to achieve a high degree of confidence in large software systems [15].

3.2. Multilevel Contract

The multilevel contract model specifies both functional and non-functional requirements for component. We distinguish several types of contracts; a contract can express the **syntactical** specifications as well as the **functional** and **nonfunctional** requirements of components. A requirement is a condition or a necessary capacity to solve a problem or reach a goal.

- **Syntactic contract**: basic contract that expresses principally syntactic specifications and potentially information about the operations provided by the component input ports and output.
- **Functional contract**: specifies the behavior of a component in terms of preconditions, postconditions and invariants. Behavioral contract guarantees that the component behaves according to its specification but does not guarantee its accuracy.
- **Nonfunctional contract**: specifies quality of service requirements of the component.

A contract consists of a set of constraints expressed in the form of assertions; a constraint is alternately a **precondition**, **postcondition** or **invariant**. A precondition expresses requirements that the element must meet to function properly, a postcondition expresses properties that are provided in return for the execution of the element. An invariant is a property that applies to all instances of the element beyond the specific routines.

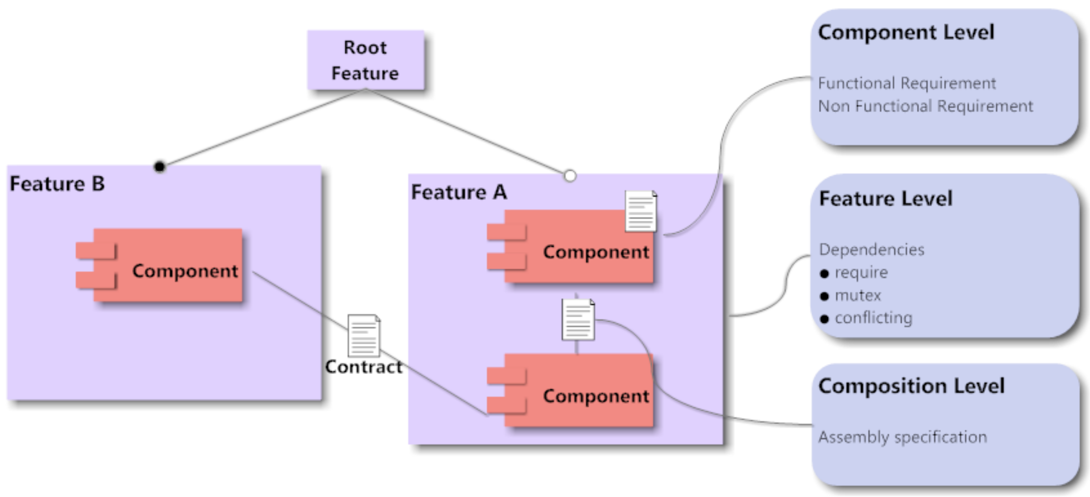


Figure 1 - Contract Levels

The multilevel contract model specifies both functional and non-functional requirements for feature, component and composite levels. We have defined three levels of component contracts as shown in Fig. 1:

Feature Contract: verifies the compliance and the absence of errors in the feature. The main errors are dead and false variables features caused by contradictions in the relations between features [16] [17]. An example of a contradiction: a feature ***A*** requires a feature ***B***, and ***B*** excludes feature ***A*** feature, this leads to an error in the resulting feature model.
Textual constraints are used to represent certain dependencies between features. The most common constraints are:

- **require**: to express the presence of feature requires the presence of another
- **mutex** (mutually exclusive with) to indicate that the two features cannot be present simultaneously
- **conflicting**: is a lightweight version of mutual exclusion and refers primarily to quality attributes. If the conflict cannot be dissolved, this relationship conflict can also be treated as mutually exclusive.

Component Contract: for the proper activity of the component and the respect of its functional and quality requirements. Component contract checks the conformity of the component and whether the implementation complies with all specifications and requirements defined for this component. The contract verifies the internal functional and non-functional compliance of the component, it provides behavioral specifications on different ports (or interfaces) of the component.

The contract is expressed in terms of precondition, postcondition and invariants as shown in figure 2.

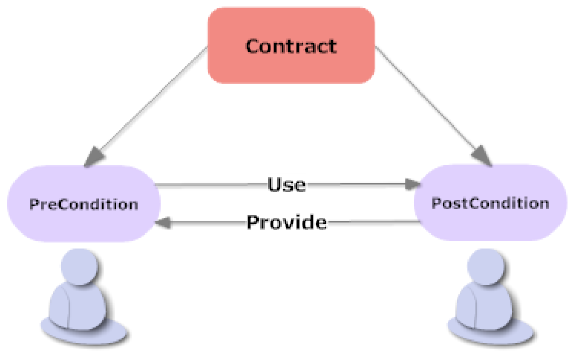


Figure 2 - Contract structure

- **Precondition**: A precondition expresses the requirements to be met by the component to run properly. An exception occurs when an operation is called with an unverified precondition.
- **Postcondition**: A postcondition expresses properties that are provided in return for the execution of the component. An exception occurs when an operation is called with a satisfied precondition and it returns with an unverified postcondition.
- **Invariant**: An invariant is an assertion that is always true during the execution of the component.

Configuration Contract: ensure safe composition and assembly of trustworthy components. The verification of composition of a component-based system ***S*** consists in proving that the composite system satisfies its specification if all its components satisfy their specifications [18]. So to prove that ***S*** guarantees a property ***M*** in an environment of hypothesis ***E***, supposing that every component meets a property ***Mi*** according to the environment of hypothesis ***Ei***.

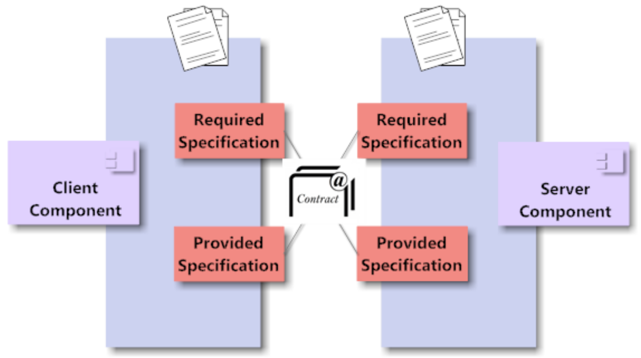


Figure 3 - Composition contract

To our knowledge, there is still no research work for the specification and validation of contractual requirement in the feature, component and composite level.

In the next section, we present our proposition of a domain-specific modeling language for contract specification, its abstract and concrete syntax as well as its implementation for requirements specification.

4. CCSL Language

The Component Contract Specification Language - CCSL is a declarative and descriptive domain-specific language, its objective is to allow the specification of different levels of contracts and the description of its elements.

Figure 4 presents the development roadmap of the main components of our language. In the following chapter, we detail the domain analysis phase and the definition of the abstract and concrete syntax. We define CCSL as a metamodel enriched with notations and corresponding tools to support it. The metamodel is the concept model of DSL; it defines the elements of a language allowing to express the models. The metamodel describes the concept, nature, association between language elements, the model hierarchy and rules of correctness of the model.

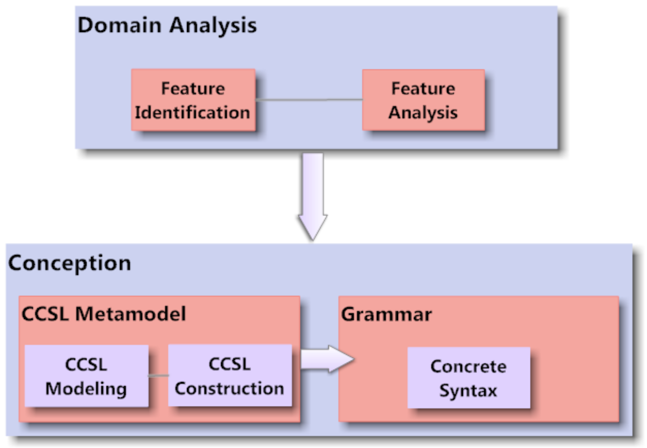


Figure 4 - CCSL Developement Steps

The contract feature model produced from the domain analysis presented in the previous section is drawn in figure 5 following the FODA method.

4.1. Abstract Syntax: CCSL Metamodel

A domain-specific language has the following properties [19]:

- **Abstract syntax**: An abstract syntax is a set of rules that defines a set of structures without prescription of a specific outside world. The abstract syntax specifies the basics of the language and their relationship through a metamodel.

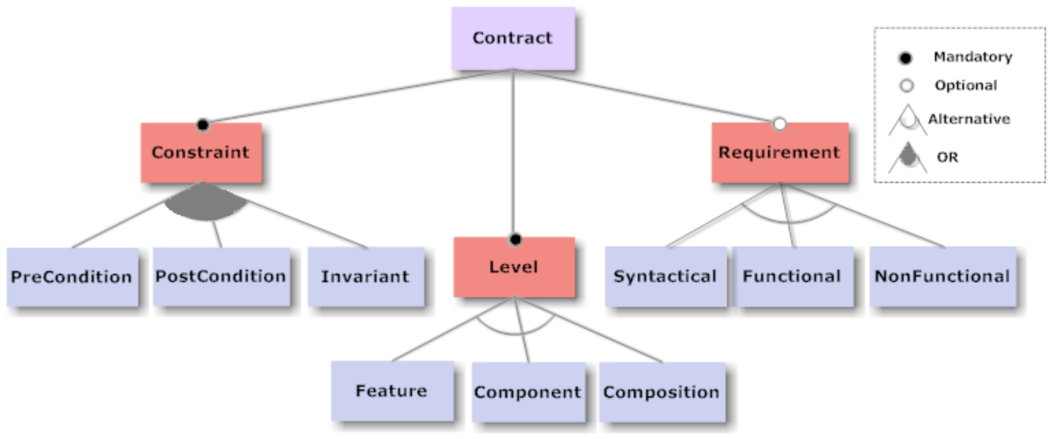


Figure 5 - Contract Feature Model

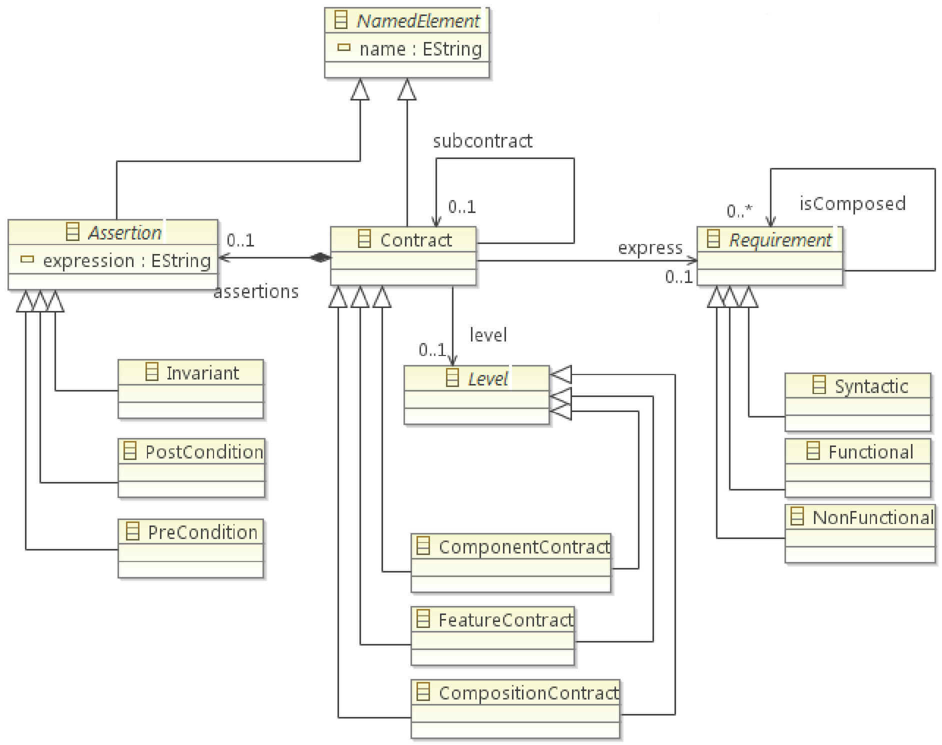


Figure 6 - CCSL Abstract Syntax Metamodel

- **Concrete syntax**: a concrete syntax specifies structures of real and representable computing data. The concrete syntax specifies the real aspect of the language by attributing a visual symbol or a textual description to the elements of languages according to the structure defined by the abstract syntax.
- **Semantic**: A semantic implicitly or explicitly describes the meaning of the language constructs.

The abstract syntax of the CCSL language is defined via a metamodel. The construction of the metamodel of a domain-specific can be based on two main approaches: the creation from scratch that despite a great design effort allows a better flexibility and scalability, or the extension of an existing metamodel to take profits from the existing and minimize the development effort.

The metamodel of the CCSL (Fig. 6) language is modeled with the Ecore metamodel of the Eclipse Modeling Framework (EMF) [20]. EMF is an Eclipse modeling tool based on engineering model driven approach which provides mechanisms for persistence, editing and processing models and definitions of abstract syntax.
The Ecore metametamodel follows the eMOF standard (Meta Object Facility) defined by the OMG [21]. The metametamodel defines named classes (*EClass*). A class has zero or more attributes (*EAttribute*) named and typed (*EDataType*) and zero or more references (*EReferences*).

The contract is the central element of language. A contract (*Contract metaclass*) is based on a pair of assertions (*Assertion metaclass*) and expresses a set of syntactic, functional and non-functional requirements (*Requirement metaclass*). An assertion consists of an assumption and a guarantee. There are three types of assertions: preconditions (*Precondition metaclass*), postconditions (*Postcondition metaclass*) and invariant (*Invariant metaclass).*

The requirements are described by Boolean expressions structure and may consist of a set of other requirements. These requirements are defined at different levels (*Level metaclass*) of the system:

- **Syntactic**: A syntactic requirement (*Syntactic metaclass*) provides the syntactic description of the signature of the component.
- **Functional**: A functional requirement (*Functional metaclass*) is a property related to the functionality of the component.
- **NonFunctional**: A non-functional requirement (*NonFunctional metaclass*) is the quality or characteristics of the component that determines how and under what conditions the service will be

delivered. These requirements are not directly related to the functionality provided by the component.

The contract is defined at different levels of the system:

- **Component**: The component level for the efficient operation of the component and compliance with its requirements.
- **Composition**: the composition level ensures the reliability of the composition of the components
- **Feature**: the feature level specifies constraints compliance features of the model.

4.2. Concrete Syntax

While abstract syntax specifies the concepts that are presented in the language and the relationships between these concepts, the concrete syntax defines a mapping between the meta-elements and their textual or graphical representations.

The CCSL concrete syntax allows domain-specific component-based systems designers to specify textually and precisely the contracts of individual system components. EMFText defines the CS language (Concrete Syntax Specification Language) for specifying concrete syntaxes. The generator automatically creates a CS syntax conforms to the HUTN specification standard (Human Usable Textual Notation) metamodel of language. HUTN is an OMG standard [22] to store the templates in a humanly understandable format.

Figure 7 shows a contract specification with the CCSL language.

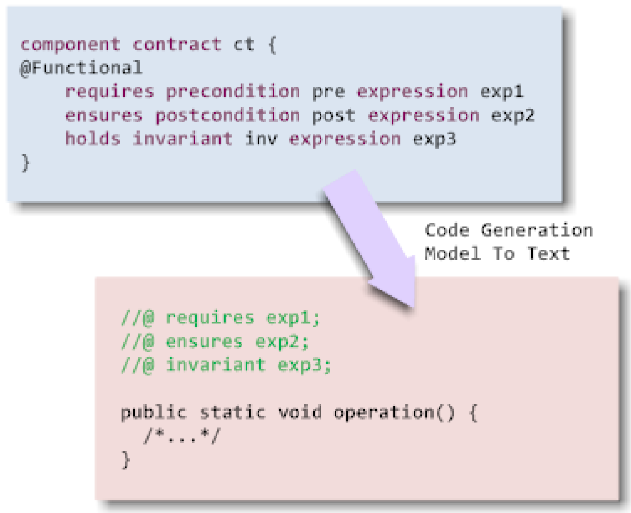
```
component contract ct {
   @Functional
   requires precondition pre expression exp1
   ensures postcondition post expression exp2
   holds invariant inv expression exp3

   @NonFunctional
   ….
   }
```

Figure 7 - A Component Contract Specification with CCSL

4.3. Code Generation

Domain-specific languages create models that express the structure and behavior of the system in an efficient, rigorous and domain-specific way. These models are afterward transformed into a code by a suite of model transformation. Figure 8 shows how the contract specification written in the DSL is transformed into a Java Modeling Language (JML) code.



```
component contract ct {
@Functional
    requires precondition pre expression exp1
    ensures postcondition post expression exp2
    holds invariant inv expression exp3
}
```

```
//@ requires exp1;
//@ ensures exp2;
//@ invariant exp3;

public static void operation() {
  /*...*/
}
```

Figure 8 - CCSL to JML

JML [23] is a formal specification language of the behavioral interfaces. It allows to specify the syntactic interface of the Java source code and as well as its behavior. The specifications are added to the code comments surrounded of **/*@** and **@*/**or after **//@**. It means that the final program is compiled with the standard Java compiler.

5. Case Study

In this section we present a motivating example showing the problem of requirement's specification and verification of a safety-critical component-based system through an e-health system. Health is a major issue for a country's economic and social development that requires both reducing costs and ensuring reliability and quality of service. The unprecedented development of mobile technologies - offering higher data transmission speeds and intelligent terminals - has improved the way services and health information can be accessed, delivered and managed. This has led to the expansion of e-health systems.

The DiabetesSM (Diabetes Self-Management) application is an e-health application that allows patients and clinicians to track and monitor the status and evolution of diabetes patients across different indicators. Alerts are sent when these indicators reach a critical level that may affect the patient's health. The application is organized into several main features:

Medical Record: The medical record containing the main information of the patient and the complete history of health problems and diseases and treatments related to medication received for these problems.

Health Record: A record where the patient records the results of the various daily checks, the observations of the condition of the eyes and feet, and the activities performed during the day or week in terms of diet and physical activity.
Patient Preferences: the preferences of the patient where he indicates his objectives in terms of diet, physical activity, control and blood test (glycaemia, cholesterol, blood pressure) and the possible obstacles. Based on these informations the system proposes a set of suitable strategies organized in an action plan.
Patient Coaching: assistance supplied by the system in the form of on-line or email reminders reminding the patient to take a medicine, to do sport and to inform the results of his daily controls.

5.1. DiabetesSM Feature Model

We represent the DiabetesSM system with the following Feature Model (Fig. 10). Feature Model is a way to represent the information of all possible configurations for a specific product that can be built. The features are organized hierarchically in a form of a tree diagram, the diagram contains a root element, and any feature can have sub-features as well as constraints, generally of inclusion or exclusion.

The Feature Model is generated using FeatureIDE tool [24]. Functional requirements of the DiabetesSM system are expressed in terms of features that are required, optional, alternative or exclusive.

5.2. Requirements

A non-functional requirement [25] is a requirement that characterizes a desired quality property of the system as performance, robustness, usability, maintainability, etc. For instance, we implement two non-functional requirements:

NFR1 Response Time: It is important to have real-time access to patient medical data, especially in emergency situations. Therefore, the response time of the feature profile Patient must not exceed 2s.

NFR2 Reliability: The attribute daily amount of carbohydrates should be accurate and reliable for this, the feature Health records shall, before its execution, calculate the rate several times, if they have the same value, then it can run normally.

```
component contract medicalRecord {
   @NonFunctional
   holds invariant inv1 expression
   response_time < 2s
   }
```

Figure 9 - Non Functional Contract

A functional requirement is a requirement defining a function of the system to be developed. What the system has to make.

FR1 Diet: the diet must be adapted to the profile (sex, weight cuts and age) of the patient. In the patient's objectives, the contract should calculate the minimal and maximal value of daily calories to be consumed according to the profile of the patient. The patient can then specify his desired diet. The contract verifies afterward that the specified value is conform to the calculated margin.

The e-health application has high reliability requirement, moreover, it should be adapted to the different specificities of the patients' profiles and healthcare needs. To meet these requirements, we use domain-specific components engineering enhanced with quality contracts.

We implement the DiabetesSM system's requirement with our CCSL language.

Also, it is important, especially for patients identified with insulin resistance, pre-diabetes and diabetes to monitor the glycemic load. Glycemic Load is a measure that uses the Glycemic Index and combines it with the amount of the eaten food.

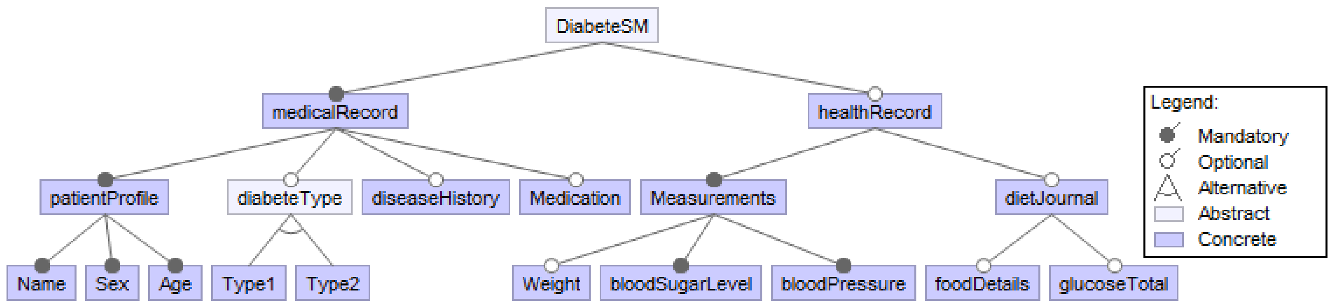


Figure 10 - DiabetesSM Feature Model

The formula for Glycemic Load is:

$$\text{Glycemic Load} = \frac{\text{Glycemic Index x Available Carbs (grams)}}{100}$$

It is recommended to keep the total Glycemic Load under 100 per day.

```
component contract dietJournal {
   @Functional
   ensures postcondition ps expression
   glycemic_index < 100
   }
```

Figure 11 - Functional Contract

6. Related Works

We propose a contract specification language to handle functional and non-functional properties of feature-oriented component-based systems. In this section we discuss proposed works related to the main concepts of our approach which are: design by contract, functional and non-functional specification and reliable feature-oriented component-based systems.

Several research works have focused on the design by contract, they carried on static verification and operate contracts for statically predict program errors [26]. Then Findler [27] introduced the notion of higher order contract values, concept which has been taken and expanded thereafter [28]. Polikarpova [29] proposed a model-based formal contractual specification for the Eiffel language.

As Feature-Oriented Programming is an expanding new approach, research in the area of quality and dependability are still in a preliminary phase. Among the approaches to ensure the security of systems, design by contract is a formal and efficient method systems' of design and construction to improve reliability and attest the absence of errors. In this context, Thum [30] presented five ways to integrate Design by Contract in the Feature-Oriented Programming.

As for non-functional properties specification, number of QoS specification language were proposed to specify quality aspects, **QML** [31] is a quality constraints specification language that separates the QoS specification from the specification of the functional aspects. Component Quality Modeling Language (**CQML**) [32] is a generic language for defining non-functional properties of component-based systems; it is platform-independent and annotates both the interface and the component. However CQML does not allow the specification of all types of quality of services (e.g. security aspects cannot be specified) nor the specification of resources.

It is interesting to notice that the main aspects discussed separately in the previous proposals are considered in our approach.

7. Conclusions

Contract-based design is a rigorous and effective approach for modeling and verifying quality-driven systems, it is particularly suitable for component-based systems. In this work we propose a multilevel contract model and a domain-specific language for expressing and verifying functional and non-functional properties in all levels of component based systems.

The main advantages of our approach are as drawn:

- **Large Coverage**: Quality is considered at all stages of the development cycle of the system. Indeed, we can define the functional and non-functional requirements, implement, and then verify them in the feature, component, and composition levels.
- **Reduced Complexity**: Most existing approaches have a heavy and complex formality, to be understood and adopted in industrial projects, our approach provide evidence of non-functional properties without requiring the complete formality of mathematical development.
- **Tooling Environment**: A framework and an environment that take profits of the MDA advantage enables better use and support the approach.

One limitation of our approach is the wide variety of non-functional requirements [25] that are affected by a large number of subjective factors. To overcome this limitation, we intend to extend the model-driven engineering benefits to requirements management by integrating and using the various proposed requirements models.

References

[1] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*: ACM Press/Addison-Wesley Publishing Co., 2000.
[2] S. Apel and C. Kästner, "An Overview of Feature-Oriented Software Development," *Journal of Object Technology (JOT)*, vol. 8, pp. 49-84, 2009.
[3] J.-C. Laprie, *Guide de la Sûreté de Fonctionnement*: Cépaduès, 1995.

[4] B. Meyer, "Applying Design By Contract," *Computer*, vol. 25, pp. 40-51, 1992.
[5] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*: Addison-Wesley Longman Publishing Co., Inc., 2001.
[6] D. L. Parnas, "On The Criteria To be Used In Decomposing Systems Into Modules," in *Software pioneers*, B. Manfred and D. Ernst, Eds., ed: Springer-Verlag New York, Inc., 2002, pp. 411-427.
[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," DTIC Document1990.
[8] J. V. Gurp, J. Bosch, and M. Svahnberg, "On the Notion of Variability in Software Product Lines," in *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, 2001, p. 45.
[9] C. Szyperski, D. Gruntz, and S. Murer, *Component Software: Beyond Object-Oriented Programming*: Addison-Wesley, 2002.
[10] A. van Deursen, P. Klint, and J. Visser, "Domain-Specific Languages: An Annotated Bibliography," *ACM SIGPLAN Notices*, vol. 35, pp. 26-36, 2000.
[11] M. Voelter, *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*: CreateSpace Independent Publishing Platform, 2013.
[12] M. Mernik, J. Heering, and A. M. Sloane, "When and How to Develop Domain-Specific Languages," *ACM Computing Surveys*, vol. 37, pp. 316-344, 2005.
[13] J. M. Neighbors, "Software Construction Using Components," University of California, Irvine, 1980.
[14] C. A. R. Hoare, "An Axiomatic Basis for Computer Programming," *Communications of the ACM*, vol. 12, pp. 576-580, 1969.
[15] B. Meyer, *Object-Oriented Software Construction (2nd ed.)*: Prentice-Hall, Inc., 1997.
[16] T. von der Maßen and H. Lichter, "Deficiencies in Feature Models," in *Workshop on Software Variability Management for Product Derivation-Towards Tool Support*, 2004.
[17] A. Hemakumar, "Finding Contradictions in Feature Models," in *Workshop on the Analysis of Software Product Lines*, 2008, pp. 183-190.
[18] M. Abadi and L. Lamport, "Composing Specifications," *ACM Transactions on Programming Languages and Systems*, vol. 15, pp. 73-132, 1993.
[19] A. Kleppe, *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*: Addison-Wesley Professional, 2008.
[20] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework*: Pearson Education, 2008.
[21] OMG, "Meta Object Facility (MOF) 2.0 Core Specification", 2011.
[22] OMG, "Human-Usable Textual Notation v1.0", 2004.
[23] G. T. Leavens and Y. Cheon, "Design by Contract with JML," 2006.
[24] C. Kastner, T. Thum, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, *et al.*, "FeatureIDE: A Tool Framework for Feature-Oriented Software Development," in *Proceedings of the 31st International Conference on Software Engineering*, 2009, pp. 611-614.
[25] L. Chung and J. C. P. Leite, "On Non-Functional Requirements in Software Engineering," in *Conceptual Modeling: Foundations and Applications*, T. B. Alexander, K. C. Vinay, G. Paolo, and S. Y. Eric, Eds., ed: Springer-Verlag, 2009, pp. 363-379.
[26] D. L. Detlefs, K. Rustan, M. Leino, G. Nelson, and J. B. Saxe, "Extended Static Checking," in *SRC Research Report 159, Compaq Systems Research Center*, 1998.
[27] R. B. Findler and M. Felleisen, "Contracts for Higher-Order Functions," *SIGPLAN Not.*, vol. 37, pp. 48-59, 2002.
[28] C. Dimoulas and M. Felleisen, "On Contract Satisfaction in a Higher-Order World," *ACM Transactions on Programming Languages and Systems*, vol. 33, pp. 1-29, 2011.
[29] N. Polikarpova, C. A. Furia, Y. Pei, Y. Wei, and B. Meyer, "What Good Are Strong Specifications?," in *Proceedings of the 2013 International Conference on Software Engineering*, San Francisco, CA, USA, 2013, pp. 262-271.
[30] T. Thum, I. Schaefer, M. Kuhlemann, S. Apel, and G. Saake, "Applying Design by Contract to Feature-Oriented Programming," in *Proceedings of the 15th international conference on Fundamental Approaches to Software Engineering*, Tallinn, Estonia, 2012, pp. 255-269.
[31] S. Frølund and J. Koistinen, "Quality-of-Service Specification in Distributed Object Systems," in *Proceedings of the 4th conference on USENIX Conference on Object-Oriented Technologies and Systems - Volume 4*, Santa Fe, New Mexico, 1998, pp. 1-1.
[32] J. Ø. Aagedal, "Quality of Service Support in Development of Distributed Systems," University of Oslo, 2001.

Maryem Rhanoui received the Engineer of state degree in Software Engineering from National High School of Computer Science and Systems Analysis (ENSIAS) in 2008. She is currently a PhD student in the IMS (Models and Systems Engineering) Team of SIME Laboratory at ENSIAS. Her research interests are Domain-Specific Engineering, Component-Based Systems, and Model-Driven Engineering.

Bouchra El Asri is a Professor in the Software engineering Department and a member of the IMS (Models and Systems Engineering) Team of SIME Laboratory at National Higher School for Computer Science and Systems Analysis (ENSIAS), Rabat. Her research interests are Service-Oriented Computing, Component Based Engineering, and Software Product Line Engineering