Towards a Meta-Model for Quality-aware Self-Adaptive Systems Design

Esma Maatougui, Chafia Bouanaka, Nadia Zeghib
LIRE Laboratory
University of Constantine 2-Abdelhamid Mehri
Constantine, Algeria.
esma.maatougui@univ-constantine2.dz, chafia.bouanaka@univ-constantine2.dz, nadia.zeghib@univ-constantine2.dz

***Abstract*—Self-adaptation is a promising technique to manage software systems maintainability and evolution. A self-adaptive system is able to adapt its structure and behavior autonomously at run-time in response to changes in the context in which it is actually running to achieve particular quality goals. However, designing and verifying quality-aware self-adaptive systems remains a challenging task. In this paper, we propose a formal approach that combines the advantages of both component-based modeling (e. g., reduces model complexity), MDE (e. g., facilitates the development process) and Maude (a formal language) to define a development process for quality-aware self-adaptive software. We particularly focus on the specification of quality-aware adaptation strategies required to ensure continuous satisfaction of non-functional requirements (Quality of service).**

***Index Terms*— Self-adaptive systems; QoS; Component-Based Software Engineering; Model-Driven Engineering; Maude.**

I. INTRODUCTION

Nowadays, users extensively rely on software systems quality, especially in the presence of parametric and variable execution contexts. However, ensuring the required qualities of software systems that might operate in dynamic environments, poses severe engineering challenges, since they must become more versatile, flexible, resilient, dependable, energy-efficient, recoverable, customizable, configurable, and self-optimizing by adapting themselves to changes that may occur in their operational contexts, environments and system requirements. Self-adaptation [1] is generally considered as a promising solution to manage the complexity of such software systems since it enables the system to adapt itself to internal dynamics and changing conditions in the runtime environment to achieve particular quality goals automatically.

A key characteristic of self-adaptive systems engineering is to provide guarantees about the required runtime quality properties. Nevertheless, the central role of QoS requirements has to be considered at the early stages of design. Hence, the emergence of the software system architecture provides the right level of abstraction, sets the basis to achieve both functional and non-functional requirements, and needs to be supported by methodologies and tools to capture these two dimensions of the product at the same time which generally can deal with the challenges of self-adaptation [3]. The component-based approach can provide an appropriate level of abstraction to describe dynamic changes in a system structure and increase the reusability and portability of software pieces. However, a key issue to be faced concerns the assessment of self-adaptive systems effectiveness, in terms of their ability to meet the required QoS under different context conditions. In particular, this assessment should take into account the cost of the adaptation process itself. Since, adapting a system can require time and system resources to be carried out, and this cost could even outweigh the potential benefit [3]. In addition to component-based software engineering (CBSE) [4], Model-driven engineering (MDE) [5] is an emerging approach to address these and other challenges.

MDE advocates the use of models, not only for capturing high-level design ideas and documenting the final product, but as key artefacts throughout the development process. The goal is to reduce the development time and efforts, and to increase product quality by raising the level of abstraction and automating some time consuming and error prone activities, e.g., by generating code directly from detailed models instead of implementing it manually [6].

One major advantage of MDE is the opportunity to automatically transform design models into analytical ones, thus enabling formal verification of system properties; including non-functional ones. A largely adopted approach is the combination of MDE and formal methods to ensure and guarantee functional correctness of the adaptation logic. This provides a rigorous means for modeling, specifying and reasoning about self-adaptive systems' behavior, both at design time and at runtime.

A variety of research work has been realized and significant efforts invested to propose models for QoS-aware self-adaptive systems. However, existing techniques for non-functional properties analysis rely on very specific quality-related formalisms such as Petri Nets (PNs), or Markovian models, but software systems are rarely represented in these terms [3]. Besides, most of these approaches do not take into account the separation of concerns between user requirements in terms of QoS contract and system QoS parameters. Moreover, designers, who usually lack sufficient experience in requirements engineering, prefer design-oriented formalisms such as UML [7] which reflects more the modeling intent.

In this paper, we present a component-based contractual approach to define a model for designing, specifying and verifying self-adaptive systems with respect to QoS contracts. To address this problem, we define a model for QoS contracts as a natural and effective way for user requirements.

The remainder of the paper is organized as follows. Section 2 discuses some models for self-adaptive systems that are relevant to our work. Section 3 is dedicated to the presentation of our model and the generation of the corresponding formal specification. Section 4 illustrates our proposal via a case study to validate our model. Finally, Section 5 rounds up the paper.

II. RELATED WORK

A variety of models for Self-adaptive systems have been proposed and various modeling methodologies have been adopted, including MDE [7, 8], requirements engineering [9] and component-based development [12].

Vogel and Giese [8] propose a MDE-based model for Self-Adaptive Software with EUREMA approach that realizes self-adaptation using the so-called executable runtime mega-models. In [7], a UML-based modelling language called Adapt Case Modeling Language (ACML) is presented. The language allows a separate and explicit specification of self-adaptivity concerns using the concept of the MAPE-K loop. Based on formal semantics, they apply quality assurance techniques to the modeled self-adaptive system.

Brown and Cheng [9] adopt a Goal-Oriented Requirements Engineering to present the Awareness-Requirement and propose a way to elicit and formalize such requirements using the OCL language. A methodology for generating feedback from such requirements, as well as fragments of a prototype implementation founded on an existing requirements monitoring framework is proposed. Elkodary et al. present an approach, named FUSION [10], which uses feature diagrams as a system model where self-adaptation is realized by switching between different system configurations. The self-adaptation in FUSION is goal-driven, i.e., relying on predefined functional or non-functional goals. Each goal consists of a metric and a utility. While the metric is a measurable entity as response time, a utility is a feature which has influence on the metric, and is triggered when FUSION detects that a goal is violated. The violation of a goal is detected via defined monitoring functions.

DYNAMICO [11] is a reference model for engineering adaptive software aligned with the vision of self-adaptive systems, where dynamic adaptation is necessary to ensure the continuous satisfaction of their functional requirements while preserving the predefined conditions on Quality of Service levels. These QoS levels are usually represented in the form of Service Level Agreements (SLAs), and their enforcement mechanisms are based on contracts and policies. Castaneda Bueno designs a component-based reference architecture [12] with distribution and extensible capabilities for self-adaptive systems according to the reference model DYNAMICO.

In the present work, we propose a component-based contractual approach for quality-aware self-adaptive software systems specification that supports system and QoS contracts modeling together with the corresponding adaptation logic. The proposed approach defines QoS constraints in an independent way from system QoS parameters. This separation of concerns reduces system modeling complexity and increases model reusability and maintainability. Our model for quality-aware self-adaptive systems provides a clear satisfaction of QoS contacts by applying adaptation strategies in case of violation of QoS constraints.

III. A COMPONENT-BASED CONTRACTUAL APPROACH FOR SELF- ADAPTIVE SYSTEMS

We adopt a component-based contractual approach to define a model for designing and formally specifying self-adaptive systems with respect to QoS contracts CBSE can help in the development of self-adaptive software in two ways. First, it is easier to design and implement adaptable software relying on component models. Second, the adaptation engine needs to be modular and reusable. Additionally, CBSE can also be adopted in the development phase of the self-adaptive system. However, the Component-oriented paradigm still requires comprehensive and sound QoS contract-aware self-adaptation theories, models and mechanisms further trustworthy, extensible and reusable in order to realize its contract. Moreover, in the CBSE vision, contracts play a fundamental role, as they must capture the functional and extra-functional user requirements.

We define a QoS-aware component-based model for self-adaptive systems where context and functional entities are viewed as components that interact via adaptation strategies, and designed in an entirely independent manner and only relationships between them are specified, thereby simplifying the adaptation mechanisms. To achieve this goal, we model an adaptation strategy as a pair of elements: an action associated with the notification of events that violate their contracted QoS constraints. The adaptation strategy adapts system functionalities according to context changes in terms of variations on system structure and/or behavior.

The model is designed with a focus on the separation of concerns between the specification of QoS parameters; defining user quality requirements, and software components quality parameters (see Figure 1). The first ones are specified in the QoS contract while the second ones are directly defined of the component specification.

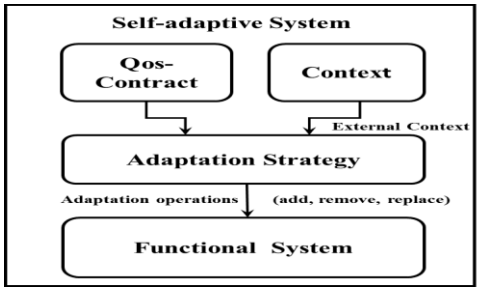

Figure. 1. An Overview of the proposed model.

QoS contracts comprise a number of quality of service constraints that might be satisfied and preserved by a managed system. These QoS constraints are specified for each of the different context conditions that the managed system is faced with while it is running. Thus, the continuous satisfaction of a

QoS contract (i.e., its preservation) implies satisfying each of the QoS constraints that the user expects, under each of the corresponding varying conditions of execution contexts. At runtime, once these conditions actually occur in the execution context of the managed application, the respective QoS constraints must be monitored, and their fulfillment enforced.

To be able to automatically ensure QoS contracts, a component-based self-adaptive system requires (i) to maintain a structural representation of itself (ii) to have a representation of the contracted QoS constraints under the different context conditions; (iii) to be self-monitoring, that is, to identify and notify events on the QoS constraints violations; and (iv) to apply the dynamic reconfiguration in response to events notifying imminent violation of QoS constraints, as specified in the QoS contracts.

Based on the previous considerations, we build our component-based QoS-aware model for self-adaptive systems. We first present our meta-model-based definitions for component-based self-adaptive software structure and QoS contracts respectively. Then, we define transformation rules to be applied to generate automatically a Maude formal specification of models instantiating the already defined meta-model.

3.1. Model-based self-adaptive systems design

Our model exploits the MDE techniques to provide a solution for self-adaptation via meta-models which describe concepts that can be used for constructing models that conform to its definition, and describes in an abstract way, the possible structure of the underlying models. The meta-model of Figure 2, specifies the various concepts that intervene to define the structure of quality-aware self-adaptive systems together with their pertinent relationships. It is structured in four parts:

A. The first part contains four meta-classes representing a quality of service contract. A **QosContract** is defined by its name and a set of QoS properties. A QoS property denotes a specific non-functional characteristic of the considered system such as its performance, reliability, and cost. A **QoSproperty** is defined by a name and a weight reflecting the relative importance of the QoSproperty with regards to the user preferences. To facilitate the specification of user preferences, three weight values are predefined in the Weight Enumeration (high, low, medium). Each QosProperty needs one or more metrics to be quantitatively measured. A **QosMetric**, defined by its idMetric, represents a non-functional property which belongs to a domain of values as response time. Finally, we associate a **QosConstraint** to the entire or a subset of QoS properties in different conditions of context. In general, a QosConstraint consists of a relational operator (e.g., <, >, =) and a value representing a threshold.

B. The second part contains two meta-classes representing context sensors used to model context sources and values. The **ContextSensor** meta-class is defined by its SensorID and sensor type. Three types of sensors are identified in [13] : Physical, Virtual and Logical sensors. Sensor types are represented via the SensorTypes Enumeration. The **Context** meta-class defines anything that interacts and affects the target or functional system. The Context is defined by its ContextID and the corresponding possible values.

C. The third part contains the **AdaptationStrategy** meta-class, which represents scenarios of adaptation that will be applied in the case of violation of the QoS Constraints. These scenarios are defined by a set of **adaptation rules** that can be of the following types: (i) add a component to the actual system configuration, (ii) remove a component, and (iii) replace one component by another.

D. The last part of the meta-model contains necessary concepts to define the **functional system** configuration, viewed as a set of components which require or provide services to each other through specific interfaces. These components are represented by the **Component** meta-class and defined by a name specified in the CName attribute. A component comprises a set of Quality attributes (quality attributes of the running service), and a set of provided interfaces (ProvidedInterface) and Required ones (RequiredInterface). Each interface exposes a set of services that are required or provided by the component. Connections in our model are dynamic and only established whenever one component is providing the service and the other one is requesting it.

3.2. Model Transformation for Generating Maude specifications

Albeit, MDE tries to facilitate software development and simplify the design process by specifying meta-models focusing on the structural and static semantics of software systems, it lacks necessary concepts to define the semantics or behavior of software systems and thus verification mechanisms that are among the major issues in specifying self-adaptive systems. A reasonable and desirable formal method to be adopted for this scope should be powerful enough to capture the principal models of computation and specification methods, and endowed with a meta-model-based definition conforming to the underlying meta-modeling framework. Additionally, the formal approach should allow working at different levels of abstraction, and be executable, in order to validate the meta-model semantics. Rewriting logic [14] via its implementation language Maude [15] is an adequate candidate for the definition of the semantics basis of our meta-model for many reasons. First, the versatility of rewrite theories can offer the appropriate level of abstraction for addressing the specification, modelling and analysis of self-adaptive systems and their environment within one single coherent framework. Second, since Maude is a rule-based language, the adaptation logic can be naturally expressed as a subset of the available rules, and the meta-programming capability of Maude can be exploited to enforce the execution of a given adaptation rule to maintain QoS parameters via Maude strategies. Third, the formal analysis toolset of Maude can support simulations and analysis over the self-adaptive system.

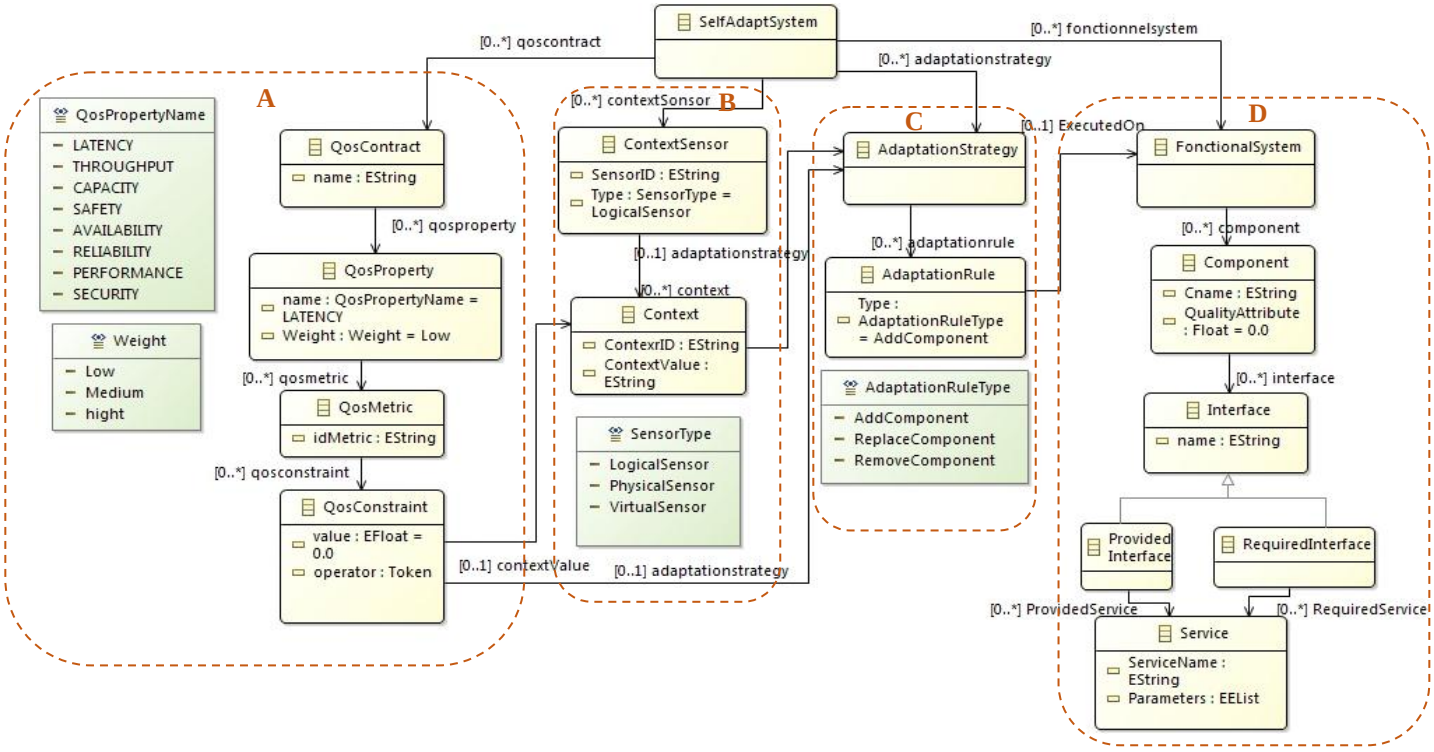

Figure 2.Self-adaptive system meta-model.

The bridge between MDE and formal methods is established via model transformation techniques, realized via a set of transformation rules. A model transformation consists in general of a computation that applies repeatedly a set of transformation rules to a model, where the model represents the structure of a sentence in a given formal language, defined by a meta-model. EMF (Eclipse Modeling Framework) [16] and specially Acceleo [17] are used in our case as a modeling framework and code generator implementation of the OMG's Model-to-text specification for building tools and applications based on models defined in the Ecore meta-model. This tool provides the capability to define advanced code generators for transforming models to a target code by defining transformation templates.

Table 1 illustrates some results of transformation rules defined between the self-adaptive meta-model and the formal semantics. The meta-model and the imposed constraints provide the capability to achieve a formal specification generation through template models. Our goal is to transform **EClass**, **EAttribute**, **EReference** and **EOperation** of the self-adaptive model to Maude constructs to facilitate self-adaptive systems specification.

Since Maude offers two possible representations, the algebraic and the object-oriented ones, we have adopted an object-oriented representation in order to reflect the hierarchical structure of self-adaptive systems and avoid the flat structure while adopting algebraic terms. In addition, all structural concepts are transformed to Maude classes while behavioral concepts as Adaptation Rules and Adaptation Strategies are transformed to rewriting rules and Maude Strategies respectively. The first mapping of Table 1 concerns structural concepts that can be defined as an Acceleo template as follows:

```
[template public generateElement(Package : EPackage)]
[comment @main/]
[file (Package.name.concat('.maude'), false, 'UTF-8')]
  (omod [Package.name.toUpperFirst()/] is
     for (c: EClass | Package.eAllContents(EClass))
separator('\t')]
    [if c.name.equalsIgnoreCase('AdaptationStrategy')=
false)]
     [if(c.name.equalsIgnoreCase('AdaptationRule')=
false)]
    class [c.name.toString()/] | [for (a: EAttribute
|c.eAttributes ) separator(',')] [a.name/] :
[if (a.eAttributeType.name='EString')]String [/if]
[if (a.eAttributeType.name<>'EString')]
[a.eAttributeType.name/] [/if] [/for]
[if (c.eReferences<>null)] ,
[c.eReferences->first().name/] [/if] : OidList.
     [/if]
     [/if]
     [/for]
endom)
[/file]
[/template]
```

The template for structural concepts generates a Maude file, using a tag [file] to specify the output file, that contains the various classes and their attributes as specified in Table 1. It begins by testing if the considered element is not a behavioral concept, i.e., neither an adaptation rule nor an adaptation strategy. Such verification is realized via the conditional statement [if]. Then, it generates a class from each EClass of the meta-model via the [for] bloc, together with the corresponding attributes.

TABLE 1. Transformation results.

Structural concepts	
Eclass	**Maude specification**
QosContract **QosProperty** **QosMetric** **QosConstraint**	**class** QosContract | name : String , QosProperties : OidListe . **class** QosProperty | name : QosPropertyName , Weight : Weight , Qosmetrics : OidListe . **class** QosMetric | idMetric : String , QosContraints : OidListe . **class** QosConstraint | value : Float , operator : String , contextValue : Oid.
FonctionnelSys **Component** **ProvidedInterface** **RequiredInterface** **Service**	**class** FonctionnelSystem | Components : OidListe . **class** Component | Cname : String , QualityAttribute : Oid , ProvidedInterfaces : OidListe , RequiredInterfaces : OidListe . **class** ProvidedInterface | ProvidedServices : OidListe . **class** RequiredInterface | RequiredServices : OidListe . **class** Service | Servicename : String ,QualityAttribute : Oid , isActive : Bool ,Parameters : OidListe. **class** QualityAttribute | name : String , value : Float .
Behavioral concepts	
AdaptationRule	**crl** [ReplaceComponent] : < F : FonctionnelSystem | Components : C CL > < C : Component | Cname : name , QualityAttribute: Q1 ,ProvidedInterfaces: I PIL ,RequiredInterfaces : RIL > < C' : Component |Cname: name2 ,QualityAttribute: Q2 ,ProvidedInterfaces: IL ,RequiredInterfaces: L > < Q1 : QualityAttribute | name : QN , value : V1 > < Q2 : QualityAttribute | name : QN , value : V2 > => < F : FonctionnelSystem | Components : (del(C, (add(C' , CL)))) > < Q2 : QualityAttribute | > < C' : Component |Cname: name2 ,QualityAttribute: Q2 ,ProvidedInterfaces: IL , RequiredInterfaces: L > **if** V2 < V1 .
AdaptationStrategy	**(fmod** SelfAdapt-STRA **is** **pr** REW-SEQ . **op** SelfAdaptStrat : -> List{Tuple{Qid, Substitution}} [**memo**] . **eq** SelfAdaptStrat = ('ReplaceComponent, 'F:Oid <- ''F.Qid ; 'C:Oid <- '' FireManComp.Qid ; 'C':Oid <- '' FireEngComp.Qid) .

IV. MOTIVATING ADAPTATION SCENARIO

The scenario of a firefighting system [18, 19] is used as an example. Fire fighters often work in dangerous and dynamic environments. Moreover, a fire accident is one of the most frequent incident types. The early detection and timely preventive measures are effective methods for limiting fire damage and reducing casualties. In this example, the firefighting system is a component-based software system designed to detect fire signals and make effective fire-management strategies. When fire danger occurs, these components dynamically restructure into a firefighting plan by choosing appropriate firefighting resources from the component library. These well-restructured components then drive the corresponding fire-extinguishing installations to perform the firefighting plan.

The Firefighting System automatically takes effective measures to prevent the fire disaster (Goal). This goal can be further decomposed into: (G1) detect fire signals in the early stage and (G2) assemble a set of fire-fighting devices in response to a real-time fire situation. To achieve these self-adaptation objectives, we should identify detectable contexts reflecting the software running state or physical environment, and then identify adaptive actions that can be performed at runtime to change the system behavior. In this example, the detectable fire signals (contexts) are various, such as CO, CO2, along with high temperature, and strong flame. Therefore, the context to be chosen depends on the occurring place and the fire disaster type.

Self-adaptive Firefighting System is used to monitor indoor fire disasters. It is composed of two essential parts, see Figure 3: context layer and functional one. We identify Temperature, Smoke Concentration, CO Concentration and Infrared Wavelengths as different contexts. The corresponding Maude specification of the available contexts is given by the following fragment of code:

```
< 'CTXS1 : ContextSonsor | SonsorID : "FireMonitor_TEM" , Type :
PhysicalSonsor , context : 'CTX1 >
< 'Temperature : Context | ContextID : "Temperature" ,ContextValue : "65" >
 < 'CTXS2 : ContextSonsor | SonsorID : "FireMonitor_CO" , Type :
PhysicalSonsor , context : 'CTX1 >
< 'CO-Con : Context | ContextID : "CO-Con" , ContextValue : "70%"  >
```

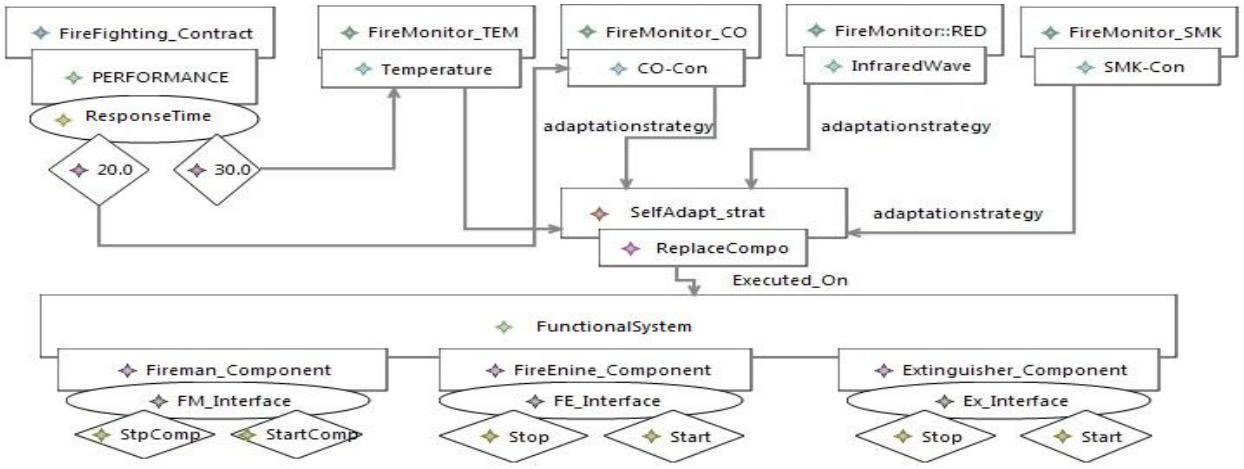

Figure 3. Self-adaptive Firefighting System model.

We also identify three types of components: Fireman, Fire Engine and Extinguisher. In the example, fire-prevention measures are made by dynamically restructuring the firefighting components. The corresponding Maude specification of these components is given by the following fragment of code:

```
< 'F : FonctionnelSystem | Components : 'FireManComp 'ExgComp >
< 'FireManComp : Component | Cname : "FireMan" , QualityAttribute : 'Q1 ,
ProvidedInterfaces : 'FM_Interface >
< 'FireEngComp : Component | Cname : "FireEngineComp" , QualityAttribute
: 'Q2 , ProvidedInterfaces : 'FE_Interface >
< 'ExgComp : Component | Cname : "Extinguisher_Comp" , QualityAttribute :
'Q3 , ProvidedInterfaces : 'EX_Interface >
< 'FM_Interface : ProvidedInterface | ProvidedServices : 'StartCompFM >
< 'FE_Interface : ProvidedInterface | ProvidedServices : 'StartCompFE >
< 'EX_Interface : ProvidedInterface | ProvidedServices : 'StartCompEX >
< 'StartCompFM : Service | Servicename : "StartCompFM" , QualityAttribute :
'Q1 , isActive : true , Parameters : 'PL >
< 'StartCompFE : Service | Servicename : "StartCompFE" , QualityAttribute :
Q2 , isActive : false , Parameters : 'PL >
< 'Q1 : QualityAttribute | name : "ResponseTime" , value : 50.0 >
< 'Q2 : QualityAttribute | name : "ResponseTime" , value : 20.0 >
```

The “FireManComp” component has “Q1” as a quality attribute which represents the response time of 50 sec and a Provided Interface “FM_Inerface” that proposes a unique running service “StartCompFM”.

In the firefighting system, we are concerned with the analysis of the performance quality parameters in terms of the response. For this reason, we identify the **Firefighting_Contract** which comprises the **Performance** as a QosProperty and **ResponseTime**, see Figure 3, as a metric that is used to evaluate the performance. We propose two QosConstraint in this example: The response time in the **Temperature** context must not exceed 30 sec. But, in the context of **CO-Concentration,** the response time might not exceed 20 sec. The corresponding Maude specification of this QosContract is given by the following fragment of code:

```
< 'QosContract : QosContract | name : "FireFighting" , QosProperties : 'P1 >
< 'P1 : QosProperty | name : "Performance" , Weight : hight , Qosmetrics :
'M >
< 'M : QosMetric | idMetric : "ResponseTime" , QosContraints : 'C1 'C2 >
< 'C1 : QosConstraint | value : 30.0 , operator : "<" , contextValue :
'Temperature >
< 'C2 : QosConstraint | value : 20.0 , operator : "<" ,contextValue : 'Co-Con >
```

As an example of adaptation strategies application, we consider the case of a violation of the response time in the **Temperature** context by the actually running component "FireManComp". In this case, the system detects a violation of QoS Constraint and applies the adaptation strategy that replaces the "FireManComp" by "FireEngineComp" component. Figure 4 shows the result of the adaptation strategy. "FireEngineCom” component that respects the QosConstraint “C1” (reponse time of FireEngineComp = 20ms), is added to the list of components in the functional system and its service “StartCompFE” becomes running (**isActive : true**). It replaces “FireManComp” which does not meet the quality requirements.

```
Maude> …
Introduced module SelfAdapt-STRA

result Configuration : …
< 'F : FonctionnelSystem |
    Components :('ExgComp 'FireEngComp)>' <
'FE_Interface : ProvidedInterface | ProvidedServices :
'StartCompFE >
< 'FireEngComp : Component | Cname :
    "FireEngineComp",ProvidedInterfaces :
'FE_Interface,QualityAttribute : 'Q2 >
< 'StartCompFE : Service | Parameters : 'PL,
QualityAttribute : 'Q2, Servicename : "StartCompFE",
isActive : true> < 'StartCompFM : Service | Parameters
: 'PL, QualityAttribute : 'Q1,Servicename :
“StartCompFM",isActive : false >
```

Figure. 4. A strategy application result.

V. Conclusion

In this paper, we have proposed a component-based contractual approach for designing and specifying self-adaptive systems with respects to Quality of Service contracts. The approach establishes a clear separation of concerns between the specification of user definable QoS quality parameters and quality parameters of the software components. To implement the proposed approach, we have combined the MDE techniques and a formal method in order to provide an intuitive modeling notation, supporting a graphical view, but still having a rigorous syntax and semantics. Such combination also facilitates the use of formal methods in many stages of the development process including the analysis phase that includes validation and verification techniques.

As future work, we intend to exploit main characteristics of formal methods to rigorously verify the behaviors of model-based self-adaptive systems, formal specifications are automatically generated. We will mainly adopt a stochastic model-checking technique to ensure quality properties of self-adaptive systems. Besides, we plan to develop a modeling tool that facilitates the creation and the implementation of quality-aware self-adaptive systems. We aim to integrate formal techniques within the MDE ones. The role of MDE is the definition of system graphical models and formal methods serve to validate and verify the self-adaptive system in order to guarantee that system model satisfies global properties and particularly quality ones. Furthermore, we aim to apply our approach on supplementary case studies in the goal of optimizing the existing quality properties modeling, the verification and implementation capabilities of the self-adaptive systems modeling framework.

ACKNOWLEDGMENTS

This work is published through funding provided under the CMEP/TASSILI project N°08MDU945.

REFERENCES

[1] R. Laddaga , P. Robertson , Self-adaptive software: a position paper, in: Proceed-ings of International Workshop onSelf-Star Properties in Complex Information Systems, Bertinoro, 2004.

[2] Cheng, B., et al. . Software Engineering for Self-Adaptive Systems: A Research Roadmap. Software Engineering for Self-Adaptive Systems, 2009.

[3] V.Grassi, R.Mirandola, E.Randazzo, Model-Driven Assessment of QoS-Aware Self-Adaptation, Software Engineering for Self-Adaptive Systems, 2009.

[4] George T. Heineman and William T. Councill, editors. Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley Longman, 2001.

[5] R. da Silva, Model-driven engineering: a survey supported by a unified conceptual model, Comput. Lang. Syst. Struct. 2015.

[6] J.Carlson & al. Deployment Modelling and Synthesis in a Component Model for Distributed Embedded Systems; 2010.

[7] M. Luckey and G. Engels, "High-quality specification of self-adaptive software systems," in Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems , ser. SEAMS '2013.

[8] T. Vogel and H. Giese. Model-driven engineering of self-adaptive software with EURUMA. ACM Trans. Auton Adapt. Syst., Jan. 2014.

[9] Brown, G., Cheng, B.H., Goldsby, H., Zhang, J.: Goal-oriented specification of adaptation requirements engineering in adaptive systems. In: ACM 2006 , Shanghai, China, 2006.

[10] A. Elkhodary, N. Esfahani, and S. Malek. FUSION: a framework for engineering self-tuning self-adaptive software systems. In Proceedings of the eighteenth ACM SIGSOFT international symposium on foundations of software engineering, FSE '10, pages 7-16, New York, NY, USA, 2010.

[11] N. M. Villegas, G. Tamura, H. A. Ḿuller, L. Duchien, and R. Casallas, DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems , ser. LNCS. Springer, 2013.

[12] L.Castaneda Bueno, A Reference Architecture for Component-Based Self-Adaptive Software Systems. Magister Graduation Project. Department of Information and communication Technologies Faculty of Engineering.Universidad ICESE. 2012

[13] J. Zhang and B. Cheng. Model-based development of dynamically adaptive software. In 28th International Conference on Software Engineering . ACM, 2006

[14] Meseguer, J., Conditional rewriting logic as a unified model of concurrency. Theor. Comput. Sci. 96 (1), 73–155. 1992.

[15] Clavel, M., Durn, F., Eker, S., Lincoln, P., Mart-oliet, N., Meseguer, J., Talcott, C.,. Maudemanual.version 2.6. 2011

[16] Steinberg D, Budinsky F, Paternostro M, Merks E, Eclipse EMF. Modeling framework. 2nd editionAddison-Wesley; 2009.

[17] http://www.eclipse.org/acceleo.

[18] D Han, Q Yang, J Xing, J Li, H Wang, FAME: A UML-based framework for modeling fuzzy self-adaptive software, Article in Information and Software Technology. April 2016.

[19] C An, Y Luo, A Timm-Giel.Adaptive Routing in Wireless Sensor Networks for Fire Fighting. - Information and Communication Technologies, Springer, 2012.