ROCL: New Extensions to OCL for Useful Verification of Flexible Software Systems

Hanen Grichi\textsuperscript{1,2}, Olfa Mosbahi\textsuperscript{3} and Mohamed Khalgui\textsuperscript{3}

\textsuperscript{1}Tunisia Polytechnic School, Ariana, Tunisia
\textsuperscript{2}Institut Supérieur d’Informatique, University of Tunis el Manar, Tunis, Tunisia
\textsuperscript{3}National Institute of Applied Science and Technology, University of Carthage, Tunis, Tunisia

Keywords: Flexible Real-time System, Reconfiguration, Object Constraint Language, Metamodelling and Validation, Wireless Sensor Network.

Abstract: The paper deals with the verification of reconfigurable real-time systems to be validated by using the Object Constraint Language (abbrev, OCL). A reconfiguration scenario is assumed to be any adaptation of the execution to the system environment according to user requirements. Nevertheless, since several behaviors can be redundant from an execution to another, the use of OCL is insufficient to specify the constraints to be satisfied by this kind of systems. We propose an extension of OCL, named Reconfigurable OCL, in order to optimize the specification and validation of constraints related to different execution scenarios of a flexible system. A metamodel of the new ROCL is proposed with formal syntax and semantics. This solution gains in term of the validation time and the quick expression of constraints. The paper’s contribution is applied to a case study that we propose to show the originality of this new language.

1 INTRODUCTION

Nowadays, the embedded systems migrate to an auto-programming technology which is based on intelligent architecture (J.Bellis et al., 2005). The system can change its behavior at run-time; it is what we call an adaptive system or reconfigurable one. The researchers in (M. Bocca and Eriksson, 2009), (Harish Ramamurthy, 2005), (Handziski et al., 2005) and (Kindratenko and Pointer, 2005) define the reconfigurable system as an adaptive embedded architecture. In a recent work, (H.Grichi et al., 2014), we define a reconfiguration of a distributed system as any addition/removal/update of one/more software-hardware elements. The reconfiguration touches first the material (allowing the activation/deactivation of elements), second the software (allowing the reconfiguration of tasks) and third the communication protocols (allowing the adaptation of routing protocols between elements). We proceed in this paper to validate a flexible system design. The validation phase is an interactive process to control the system behavior. Indeed, this is what ensures that the system will operate properly and will meet the expected design features, either in quality or durability. Recent researches (Baar, 2010), (Conrad and Turowski, 2001) tend to verify the temporal constraints in the model of real-time systems, using the OCL language. In (Sendall and Strohmeier, 2001), the authors propose a UML-based approach, for specifying concurrent behaviors and optimize timing constraints on UML state machines. In (Cengarle and Knapp, 2002) the authors propose an extension of the Object Constraint Language to model real-time and reactive systems by using the Unified Modeling Language, called OCL/RT.

The adaptive behaviors of any reconfigurable system can share redundant executions that should meet the same properties as described in user requirements. We assume that a flexible system (a multitude of system instances from one model), can be designed using a set of models such that each one generates a set of instances. Each instance can share a set of objects with others under required properties. In this case, the verification is complicated and the use of OCL is insufficient to specify and verify the different constraints in optimal times. Indeed, some properties can be verified several times when the corresponding instances are checked. To validate in practice UML models of flexible systems by using OCL, we should write the different constraints in mass (present duplication in the use of objects). We also note that OCL has no constraints on the properties of attributes,
which can be linked together in the same UML class (one object) or emergent properties on the attributes of multiple classes (multiple objects). Since we deal with flexible systems, all these limitations of OCL language, become a problem for the validation of flexible systems.

We propose a new language in order to control the verification complexity of reconfigurable systems. We propose an extension of OCL, named Reconfigurable OCL, in order to optimize the specification and validation of constraints related to different executions of dynamic systems. A metamodel of the new ROCL is proposed with formal syntax and semantics. Our solution gains in term of validation time and the quick expression of constraints because we can reduce the redundancy in the expression of constraints.

We apply the paper’s contribution to a case study of a flexible system to show the benefit and the originality of this new language. We deal in (H.Grichi et al., 2014) with Reconfigurable Wireless Sensor Networks (to be denoted RWSN). After that we verify this RWSN in (H.Grichi et al., 2015) by using a Timed Automaton and use the UPPAAL environment (G. Behrmann and Larsen, ) to apply a formal verification of our system. We are interested now in the current paper in the validation step of RWSN where we apply ROCL to gain in: (i) validation time of the flexible system, in particular the RWSN, and in term of (ii) the expression of constraints by elimination of the redundancy in the expression of constraints.

The paper is organized as follows: after introduction and background. Section 3 proposes a case study to be used in the totality of this paper. Section 4 presents the reconfigurable OCL language before concluding the paper in Section 5.

2 BACKGROUND

We briefly present some concepts and formalisms to be used in the following.

2.1 Flexible Systems

An embedded control system is a computer with a dedicated function within a larger mechanical or electrical platform, often with real-time computing constraints (Heath, 2003). Modern flexible embedded systems are often based on auto-programming technology. These systems are based on intelligent elements (J.Bellis et al., 2005). Since the flexible embedded control system, or reconfigurable one, is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. These types of systems are able to make substantial changes to the data-path in addition to the control flow at run-time. The architecture of a reconfigurable system combines flexible software and hardware components. The reconfiguration flexibility emerges in three parts: the software (operation), the hardware (architecture) and the communication between the elements of the system. Many projects, such as (Gharbi and Khalgui, 2014), (Chen et al., 2014a) and (Chen et al., 2014b) deal with flexible embedded control systems. We remark that, in these research, the definition of reconfiguration touches one or two reconfiguration forms (hardware, software or protocol) since they do not mix all of them. Our last research (H.Grichi et al., 2014) deals with the reconfigurable wireless sensor networks (RWSN) that we define like a flexible system combining hardware/software/communication reconfigurations together. We are interested in (H.Grichi et al., 2015) to the verification of RWSN system using a formal modeling and simulated with the environment UPPAAL. In this paper we try to validate RWSN by using an extension of a current constraint validation language to show if our system is ‘correct’ according to different behaviors.

2.2 OCL-based Validation in Related Works

This clause describes the Object Constraint Language (OCL) (OMG, 2010), as a textual language to describe constraints on any element of UML models (OMG, 2009). OCL is a modeling language in the first place. Before version 2.0, OCL uses natural language (English), no rules are laid on the expression of these conditions. With the arrival of OCL2.0, OCL constraints are now defined by a metamodel. We note that OCL language does not answer to the requirements of flexible system designers.

Recent researches tend to verify the temporal constraints modeled in the UML models of flexible systems. For that the authors in (Baar, 2010),(Conrad and Turowski, 2001) use the formal Object Constraint Language (OCL) for precisely defining the well-formedness rules of UML models on the meta-model level modeling the embedded control system. In (Sendall and Strohmeier, 2001) the authors propose a UML-based approach, for specifying concurrent behavioral and temporal constraints on UML state machines. This approach shows how the authors enriched operation schemas (pre/post condition) assertions of system operations written in OCL and to describe how they can use a new and existing con-
structs for UML state machines to specify temporal constraints on the system. The OCL language is a good means of validation and verification constraints in embedded control systems, but if we deal with the flexible and reconfigurable ones, this language presents some limits, cited in introduction, such that the validation time of object and the expression of constraints.

We propose in this paper an extension of OCL language in order to define well a set of constraints that respond to the reconfiguration in flexible systems. Due to characteristics of reconfigurable real-time systems and in order to analyze them better, we tend to modelize and verify well our system in particular its run-time reconfiguration. Generally, finding good models is a challenging task.

3 CASE STUDY

We start by exposing the case study that we will be assumed as a running example in the following. This case study is detailed in (H.Grichi et al., 2014) and deals with Reconfigurable Wireless Sensor Networks.

3.1 RWSN

3.1.1 Terminology

In a previous work (H.Grichi et al., 2014), we define a reconfiguration scenario as a structured sequence of reconfiguration operations. Each operation in this scenario is a transition from configuration to another which is triggered as a response to reconfiguration requests under particular conditions, in order to adapt the system to its environment and improve also its performance. We consider three kinds of reconfigurations: software, hardware and protocol reconfiguration. We denote in the following by RWSN reconfigurable WSN that automatically modifies its software, hardware and communication protocol. We propose a zone-based architecture to model the reconfiguration in a WSN. To handle all reconfiguration forms, we propose a multi-agent architecture for RWSN. This architecture is composed of a Controller Agent (CA) that controls the whole architecture, a Zone Agent (ZA) to be affected to each zone in order to control its nodes, and a Slave Agent (SA) that controls each node of any zone. All these agents handle different reconfiguration forms that we described above.

3.1.2 Metamodel

After definition of the RWSN architecture and verification with UPPAAL environment, we are interested to the modeling phase by using UML language to be verified with OCL in order to verify the temporal constraint of our flexible system. We present a part of RWSN metamodel which presents the design of the WSN controlled by the SmartAgent system to model all reconfiguration forms.

We propose in (Figure 1) our RWSN to be composed of a set of Zones and a Station. We use a design pattern composite to model this structure. We define a component: WSN class, which specifies the required behavior and composite objects (Station and Zone classes). We use this pattern because we have a composite (Zone class) that contains components, where each one could be a composite. Each zone is composed of a set of nodes defined in Node class, each one is composed of a set of sensor modeled by Sensor Element Class. Battery Element class stores the energy load in battery at run-time.

WSN class executes a strategy of reconfiguration. We define ControllerAgent, ZoneAgent and SlaveAgent classes that inherit from an abstract interface Strategy_Reconfig. The Command design pattern defines the behavior of agents. Each agent has a reconfiguration order to be executed on concrete objects (Zone and Node classes). We add the last design pattern singleton to model the Clock class. With this structure, we can calculate the execution time of agents and the cost of reconfiguration scenarios.

3.2 Application of RWSN

We propose a Reconfigurable Wireless Sensor Network named (Sys). It is composed of 2 zones (Z1, Z2) where each one is composed of two nodes. We suppose initially that all nodes are activated. We apply 3 forms of reconfigurations: (1) Software Reconfiguration: We define two tasks \( T1, T2 \): (i) \( T1 \): controls the temperature and detects signal when it is higher than 40°C. (ii) \( T2 \): reduces the threshold from 40°C to 20°C.

We define 2 software reconfigurations: \( \{ SR1, SR2 \} \). (a) \( SR1 \): reconfiguration that allows the addition of \( T1 \) to each node in a summer day; (b) \( SR2 \): is applied to each summer night to remove the task \( T1 \) and to add \( T2 \).

(2) Hardware Reconfiguration: In order to minimize the dissipated energy, we apply hardware reconfiguration \( \{ HR \} \) on one sensor node: (i) \( HR \): deactivates \( Nz1 \) from \( Z1 \). The hardware reconfiguration, in this case, can change the routing information between nodes.

(3) Protocol Reconfiguration: If we apply \( HR \), the routing information of \( (Nz1) \) will be changed: \( Nz2 \)
changes the neighbors node, it sends data directly to \( N_3 \).

We write some time constraints that we want to verify with \( OCL \). We propose three global constraints relative to the reconfiguration: (i) Constraint 1: If a reconfiguration scenario is activated, the relevant reconfiguration operations must be active in terms of their timing constraints (to deal with priorities). (a) If the constraint of synchronization between operations is simultaneous (\( AND \)):  

\[
\text{OCL1: context } Op := \text{ReconfigOrder inv:}
\]

\[
\text{self.SynchConst } \rightarrow \text{forall (Forms } \parallel \text{Forms.name} = \text{HardwareReconfig } \parallel \text{Forms.name} = \text{SoftwareReconfig } \parallel \text{Forms.name} = \text{ProtocolReconfig implies (Op.state = "activated" and self.state = "activated"))}
\]

(b) If the constraint synchronization between operations is optional (\( OR \)):

\[
\text{OCL2: context } Op := \text{ReconfigOrder inv:}
\]

\[
\text{self.SynchConst } \rightarrow \text{forall (Forms } \parallel \text{Forms.name} = \text{HardwareReconfig } \parallel \text{Forms.name} = \text{SoftwareReconfig } \parallel \text{Forms.name} = \text{ProtocolReconfig implies (Op.state = "activated" or self.state = "deactivated"))}
\]

(ii) Constraint 2: Each reconfiguration operation is relative to two constraints: start condition (START) and end condition (END) that respectively model the activation and deactivation of the reconfiguration operations.

\[
\text{OCL3: context } Op := \text{ReconfOrder}\::\text{Execut()} : \text{Boolean}
\]

\[
\text{pre START : WSN.allInstances } \rightarrow \text{exists (x } \rightarrow x\text{.state = 'activated') and WSN.allInstances } \rightarrow \text{size } = 1
\]

\[
\text{post END : (WSN.allInstances - WSN.allInstances@pre)}
\]

\[
\rightarrow \text{forAll(Op—Op.oclIsNew()) and (Op.allInstances } \rightarrow \text{size } = \text{NULL)}
\]
In each instance, we should verify all objects including those shared with other instances. This point presents a weakness of the OCL language as well as the number of properties to be written. The sum of the properties to write is: $\text{Nb}_\text{Prop} = 20 + 25 = 45$.

With OCL we can not reduce the writing properties, because this language does not allow writing parameterized expressions. By using OCL, we find some redundancies during validation of these objects. We find also similar objects during the validation of the first and second object diagram.

4 CONTRIBUTION: RECONFIGURABLE OBJECT CONSTRAINT LANGUAGE

We present in this section the Reconfigurable Object Constraint Language.

4.1 Motivation

By considering the weak points of OCL language for the validation of flexible systems, we propose an extension of OCL named Reconfigurable OCL. In order to define well the set of constraints that respond to the reconfiguration in flexible systems, we propose, in Figure 2, different services of ROCL by using UML language with a use case diagram.

With ROCL, we can minimize the validation time and optimize of properties to be verified, by minimizing its number. The optimization of properties include: (i) the configuration of them by the definition of a changed parameter, dependent to the implementations, (ii) the factorization of the properties by the definition of a global properties for all implementations and (iii) the composition of properties by using one or more objects from different instances in the same expression. We gain in terms of validation time and in terms of number of properties to be checked.

4.2 Specification of ROCL Language

In this section, we present the grammar of the ROCL presenting the syntax and semantic of this language.

4.2.1 Syntax

The code is represented by a block of instructions, which is itself composed of a set of operations or instructions. Since we’re going to add an extension to OCL, we use the grammar of OCL enriched by a set of operations.

//To specify explicitly in which package invariant, pre or post-condition Constraints belong, these constraints can be enclosed between 'package' and 'endpackage' statements forming an ROCL file (RoclFile)

⟨RoclFile⟩ ::= (package packageName
RoclExpressions
endpackage
)

⟨RoclExpressions⟩ ::= ( Rconstraint
| ⟨Rconstraint⟩ " ⟨Rconstraint⟩ " ///Composite properties
| ( Rconstraint (⟨Rglobalparameters⟩ ) ) /// Factorize properties
| ///An ROCL expression with stereotype ‘invariant’ the context of TypeName’ = ‘another string’
| ⟨Rconstraint⟩ ::= ‘context ’ TypeName ’inv’
| ( stereotype name? ':' RoclExpression)
| ( stereotype name? ':' RoclExpression
( Instance , RParameterList ) ) /// assign to each instance the set of parameters to be checked
)

/// We define an Instance as a snapshot of class diagram at an instant t started with a letter I succeeded by a number
(Instance) ::= I InstanceName
To see the gain of our contribution compared to the classic OCL language, we present the constraints, defined before, with ROCL.

(i) Constraint 1: If a reconfiguration scenario is activated, the relevant reconfiguration operations should be active in terms of their timing constraints (to deal with priorities).

(a) If the constraint synchronization between operations is simultaneous (AND):

ROCL1: context Op:= ReconfOrder inv:

(b) If the constraint synchronization between operations is optional (OR):

ROCL2: context Op:= ReconfOrder inv:
IF self. SynchConst : SetReconfig HardwareReconfig, SoftwareReconfig, ProtocolReconfig implies Op.state=activated or self.state=deactivated)

(ii) Constraint 2: Each reconfiguration operation is relative to two constraints: start condition (START) and end condition (END) that respectively model the activation and deactivation of the reconfiguration operations. We note that with ROCL, we gain in the expression of constraints. Coupling and factorization of constraints allows us to write a reduced expression compared to the constraints written with OCL.

4.2.2 Semantics

In this section, we give a logical sense to the ROCL service already presented. Define the semantics of a formal language is to give a mathematical meaning to allow the programmer to understand what the program does. The semantics should help language designers to define coherent, powerful and correct languages. In (Richters and Gogolla, 2002) the authors present a detailed description of the OCL semantics. We use the operational semantics (Subrahmanya, 1992), (Kayser, 2003) to describe the meaning of the ROCL language.

We start with the semantic of the first service of ROCL language:

Service 1: Minimize Validation Time. To minimize the validation time of constraints presented by a set of properties, we should:

//verify the objects constraints in a first instance
Iter1: VERIF ([Obj1,Obj2, .., Obj n], I_1) = Result1;
Calcut(VerifTime1); // save the verification time
SAVE (Result1, VerifTime1);
//check the objects added in the next instance (checks the
//constraints on the newly added objects)
Iter2: VERIF ((Obj n+1,Obj n+2, .., Obj m], I_next) = Result_next;
CALCUL (VerifTime1+VerifTime_next);
//update the data structure to store the new results
SAVE (Result_next, VerifTime_next);
//repeat the process to the last instance
INCREMENT (next); UNTIL I_next = 0

Service 2: Minimize Properties Number. To minimize the
number of properties to be verified, we should:

///validate, in a first instance, the constraints on a set of objects
after that we save the number of verified properties
Iter1: VERIF ([Obj1,Obj2, .., Obj n], I_1) = Result1;
//check the objects added in the next instance, we verify,
//after that, the constraints on the newly added objects
VERIF ([Obj m+1, .., Obj m], I_2) = Result2;
VERIF ([Obj m+1, .., Obj fin], I_fin) = Result_fin;
Iter2: CALCUL (VerifTime1, VerifTime Fin), // updates
the number of verified properties CALCUL (NbProp);

Service 3: Factorize Properties. To define a global
property for all implementations, we should:

///validate, in a first instance, the constraints on a set of objects
Iter1: VERIF ([Obj1,Obj2, .., Obj n], I_1) = Result1;
///check if we have a similar objects in the set of instances
Iter2: IF (SIMILAR (Obj i IN I_1)) == TRUE;
///write only one global property for all implementations
WRITE (Prop j, [I_1, .., I_j])
CALCUL (NbProp);

Service 4: Composite Properties. In order to write
two or more properties in the same time we add a set of operations: AND, OR and XOR.

///validate, in a first instance, the constraints on a set of objects
Iter1: VERIF ([Obj1,Obj2, .., Obj n], I_1) = Result1;
///write a property associated to the first instance
WRITE ( {Prop j}, I_1);
///look for similar objects in the next instances
Iter2: IF (SIMILAR (Obj k IN I_1)) == TRUE;
///write the constraints according to the used operator
WRITE (AND ( {Prop j}, {Prop m}), I_1)); or
WRITE (OR ( {Prop j}, {Prop m}), I_1)); or
WRITE (XOR ( {Prop j}, {Prop m}), I_1)); or
CALCUL (VerifTime j) xor CALCUL (VerifTime m);

Service 5: Configure Properties. In order to optimize the
expression of constraints, we write a setting of properties.
We write a parameterized expressions (write only one property
and change its parameter depending on the instance)

///validate, in a first instance, the constraints on a set of objects
Iter1: VERIF ([Obj1,Obj2, .., Obj n], I_1) = Result1;
//look for similar objects in the next instances
Iter2: IF (SIMILAR (Obj k IN I_1)) == TRUE
///assign to each instance a parameter
AddPARAM (Pj TO Prop j);
///write a property with the according parameters
WRITE ( Prop j, [I_1, Pj]); or
IF (Pj OR Pm ∈ OJECT ) == TRUE
///add another properties depending on the parameters type
///If parameters type=classes
WRITE ( Prop j, [I_1, {Pj, Pm}]);
ELSE IF parameters type=attribute
WRITE ( Prop j, [I_1, {Pj, Pm}]); END IF

4.3 ROCL: Benefits in RWSN

With ROCL, we write a lesser number of constraints compared to OCL language. For the verification time:
(i) the first instance (Inst1), we have 24 objects to verify, for that we have 24 time units, (ii) the second in-
stance (Inst2), we have 19 objects.

Table 1: Comparison of Validation times between Inst1 and Inst2.

<table>
<thead>
<tr>
<th>Instance</th>
<th>OCL Validation Time</th>
<th>ROCL Valid Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inst1</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Inst2</td>
<td>19</td>
<td>1</td>
</tr>
</tbody>
</table>

With ROCL, the total verification time is 43 units of
time, but with ROCL, the validation time is the sum
of the validation times of objects for the first instance
plus the validation times of new items added to the
next instances. We have no interest to check items
already checked before, just we should save the last result:
the total verification time is 1 time unit (for the
first instance), the second instance does not present
a new object to be checked. We observe a decrease
in the verification time: this is a benefit of the ROCL
language.

5 CONCLUSIONS AND PERSPECTIVES

The OCL is a language used to verify constraints in
embedded control systems, but if we deal with flex-
ible and reconfigurable ones, this language presents
some limits, such as the validation time of objects
(increases after each instance) and the expression of
constraints (can present duplications in the use of ob-
jets). This paper proposes an extension of OCL,
named Reconfigurable OCL, in order to optimize the
specification and validation of constraints related to
different execution scenarios of a flexible system.
We propose a formal syntax and semantics of the new
ROCL language. This solution gains in terms of validation time and the quick expression of constraints. To show the originality of this new language, we propose a metamodel of WSN, like a case study, to be verified with the ROCL. We plan in the future works to develop a tool that allows the validation of flexible systems by using the ROCL as a formal validation language.

REFERENCES


