Context-awareness Meta-model for Reconfigurable Control Systems

Soumoud Fkaier[1,2,3], Mohamed Romdhani[2], Mohamed Khalgui[1,2] and Georg Frey[3]
[1]*Polytechnic School of Tunisia, Carthage University, Tunisia*
[2]*LISI Lab. of INSAT, Carthage University, Tunisia*
[3]*Automation and Energy Systems, Saarland University, Germany*

Keywords: Meta-model, Reconfiguration, Real-time System, Software Architecture, Adaptive System, Functional Constraint.

Abstract: With the increasing evolution of adaptive control systems, control applications are asked to satisfy more constraints. On one hand, these applications have to guarantee flexible adaptation processes. On the other hand, they must offer an efficient interaction with the environment. Satisfying such needs is still challenging due to real-time requirements. Developing an application covering these constraints needs a robust software support. Some properties like clear structuring, flexibility and reuse are becoming necessary to ease the development of the appropriate applications. In this paper, we propose a context-aware meta-model enabling the development of real-time reconfigurable control systems. This meta-model offers a development flexibility while meeting the functional as well as the real-time constraints. To prove the efficiency of this meta-model, we implemented a framework on the basis of the new concepts. Also, we simulated a case study using the new tool. As a performance test, we calculated the system response time and we compared it with other work.

1 INTRODUCTION

Adaptive control systems are systems that provide the possibility to adapt their behavior to the eventual changes (Lepuschitz et al., 2011). The main feature that distinguishes them is leading self-reconfiguration (Valentini et al., 2013). Reconfiguration scenarios are dedicated to allow adding, removing and/or updating some system tasks at the aim to keep its effectiveness (Idriss et al., 2015). Reconfigurations can have various origins: They can be caused by changes in the control targets, the state of the system and even the environment that the system behaves within (Calinescu et al., 2011). In this paper we focus on developing reconfigurations coming from the environment. We consider that this issue can be treated based on context-awareness paradigm. This concept has dealt at first with the field of mobile computing. It was introduced by (Schilit and Theimer, 1994) as a solution for mobile applications to enable the adaptation to the location, nearby users/objects, and modifications of those objects at run-time. After that, the notion has been boosted and it was developed in a wide range of fields: Internet of Things (Sukode et al., 2015), health care systems (Lo et al., 2011), transportation systems (Al-Sultan et al., 2013), and many other fields (Zhang et al., 2013) (Zhan et al., 2014)(Schuller et al., 2013). One important promising field of study is improving the control systems by providing them with the aptitude to adjust their execution according to the dynamic changes of the environment.

To be useful, a reconfigurable system has to meet some constraints. One common challenge of the most of the adaptive systems is respecting real-time restrictions. These systems are considered meaningful unless they do not exceed the deadlines of their tasks. Hence, satisfying real-time constraints is primarily needed. Furthermore, ensuring a certain level of awareness about what happens in the environment is of great importance. So, these systems are asked to be flexible enough in order to lead fluent reconfigurations. Also, they are inquired to pick-up as maximum as possible the occurred events in their proximity. Moreover, treating reconfigurations at runtime is not a trivial task since there are some events that can be contradictious or that may need some shared resources. The system services must be executed respecting the dependencies relations. As can be seen, there are various constraints to be fulfilled. Developing such systems will be really a hard or even a complex task. In that case, to avoid design complexity a high-level of abstraction becomes a necessity.

Fkaier, S., Romdhani, M., Khalgui, M. and Frey, G.
Context-awareness Meta-model for Reconfigurable Control Systems.
DOI: 10.5220/0006328502260234
In *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2017)*, pages 226-234
ISBN: 978-989-758-250-9

Our contribution in this paper, is to offer a new modeling concept of real-time context-aware reconfigurable systems. We propose a context-aware meta-model allowing the specification of such system in an easy way. The proposed meta-model aim to guarantee the interaction with the environment and the awareness about the context of the use. It offers the respect of real-time as well as the functional constraints of the system. Also, it offers a flexibility of the reconfiguration processes. In addition, we implemented a framework using C# programming language. We simulated a formal case study with the developed tool. We calculated the system response time to prove the efficiency of the new meta-model and we compared the results to another work.

The remainder of this paper is given as follows: Section 2 presents briefly the background of our proposition. Section 3 contains the definition and modeling of the new meta-model. Section 4 presents the application to the case study. Finally, Section 5 concludes this paper and highlights the perspectives.

2 BACKGROUND

Numerous works have provided specifications facilitating the development of adaptive control systems (Siddiqi and de Weck, 2008). In the following, we briefly show some works on modeling of real-time reconfigurable systems. (Capilla et al., 2011) have proposed an UML meta-model to be able to capture decisions. They focused on some important issues such as maintenance, evolution and run-time decisions. (Feiler et al., 2006) proposed the Architecture Analysis and Design Language (AADL) which is an architecture description language allowing the specification of both software and hardware of a system. It offers specifying reconfigurable systems using state machines in which the state represents a configuration and a transition represents an event. (Gumzej et al., 2006) have dealt with PEARL specification which is a hardware/software co-design methodology. They offered a guide for system reconfigurations. (Nafti et al., 2015) have presented a meta-model for reconfigurable embedded systems called Chameleon. Chameleon is an object oriented meta-model allowing reconfiguration flexibility and reduces the consumption of both the processor time and the memory. (Krichen et al., 2011) and (Hamid and Krichen, 2010) have proposed a meta-model aiming at specifying the distributed reconfigurable real-time embedded systems. Also, they implemented an UML profile called RCA4RTES as an application of their meta-model. (Dowling and Cahill, 2001) have proposed an architecture meta-model enabling the creation of dynamic software architectures. They provided an adaptation contract description language aiming at separating the adaptive behavior of systems from their functional behavior. (Lehmann et al., 2010) have presented a meta-modeling process for modeling self-adaptive applications in order to facilitate the dynamic adaptation at run-time.

All the previous researches have proposed several modeling and specification methods to describe reconfigurable systems. However managing the context information is practically not studied. Providing the models of context collection, classification, processing and dissemination is useful to minimize the complexity of the reconfiguration processes. Moreover, some of the existing models do not highlight well how real-time constraints can be treated. Adding to that, organizing the relation between the different reconfigurations and resolving conflicting scenarios are not mentioned as well. The functional constraints like the inclusion or exclusion rules, the dependencies relations, the sharing resources issues are not satisfied with a clear and easy way. On these basis, we propose a new meta-model overcoming all these constraints. We design a three levels meta-model where each level concerns a particular directive. The first level tackles the relation of the system with its environment. The second level is dedicated to manage the system behavior by respecting all the possible constraints. The third level contains the set of the system services. The next section presents a description of the proposed meta-model.

3 NEW CONTEXT-AWARENESS META-MODEL

In this section, we present the proposed concept by showing a detailed description of the different components of the meta-model.

3.1 Concept Description

Adaptive control systems have many common characteristics. There are some execution steps repeated in most of them regardless their field of application and their specificity of operation. For instance, the majority of them have an agent or an entity responsible for the digital control. This agent includes the controllers and algorithms, the logic entities, etc. Also, they contain another agent responsible for the mechanical part (management of the sensors and actuators). And finally, most of them include an entity responsible for the interaction with the users (handling the I/O and

the graphical display). In order to provide an easy modeling of these entities and to express the relations between them, a high level of abstraction is really needed. In this context, we propose a new reconfigurable context-aware real-time meta-model. We define this meta-model through giving three viewpoints: (i) Structural approach, (ii) Behavioral approach, (iii) Constraints perspective approach.

3.1.1 Structural Approach

The study of the adaptive systems allows us to deduct that they can be designed through three levels architecture (see Fig. 1).

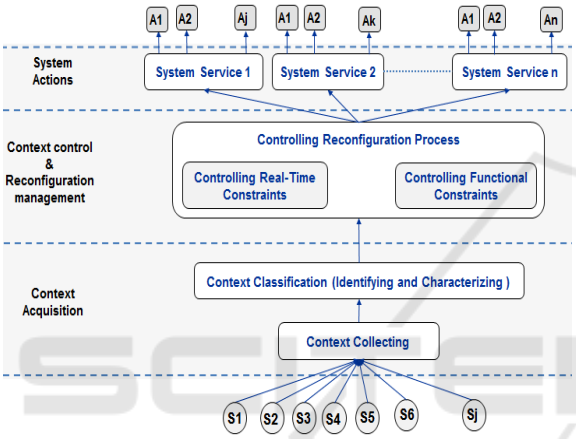

Figure 1: A generic structure of the context-aware real-time systems.

The first level can be called ***Context Acquisition***. It has to manage the communication with the events triggered in the environment by means of sensors.*The context acquisition* has to be achieved in two steps: (i) Context collecting, (ii) Context classification (it can be reached by identifying the type of the event and by attributing to it a set of properties). The second level can be called ***Context Control and Reconfiguration Management***. This level should contain the control logic as well as the constraints verification modules which are basically the real-time and functional restrictions. The third level can be called ***System Actions*** and it has to include the system services. Having a separation between the system services and the control aspects helps developers to benefit from the re-usability and the flexibility.

3.1.2 Behavioral Approach

The analysis of the behavior of adaptive control systems enables us to note many observations about their control attitudes. Most of these systems need to perform an adaptation to the context they are putted in. The adaptation logic has many shared steps, therefore we propose a generic process expressed via the FSM (Finite State Machine) of Fig. 2. Most of the self-adaptive control systems have to detect to the eventual changes. Then, classifying the reconfiguration requests and identifying them is a necessary step that helps to decide which control process to lead. After that, checking the constraints that can be faced has to be performed. Finally, sending the update commands to the appropriate actuators is needed.

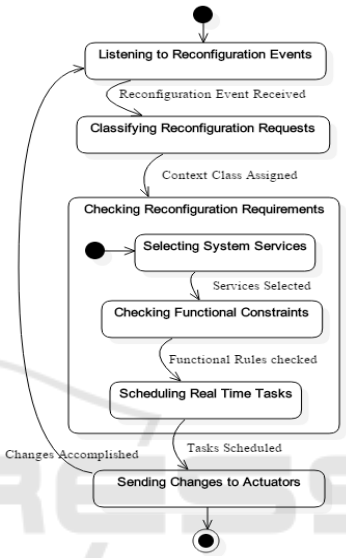

Figure 2: The FSM expressing a generic context adaptation.

3.1.3 Constraints Perspective Approach

A survey of the mostly faced constraints in adaptive systems drives us to assume that they have a set of shared constraints. Basically, all these systems have to perform their tasks under real-time constraints. Functional constraints also are always required. Thus, we propose to use generic OCL rules (the Object Constraint Language which is a formal language for the definition of constraints on UML models) with the aim of abstracting the commonly faced constraints.

3.2 Meta-Model Definition

The meta-model is composed of three main packages (see Fig. 3). ***RealTime ContextAware MetaModel*** is the container package that contains: (i) ***Context Acquisition***, (ii) ***Context Control and Reconfiguration Management***, and (iii) ***System Actions***. These three packages are linked through dependencies relations (in particular *import* that allows to get the services from another package and *use* which allows to utilize the content of a package).

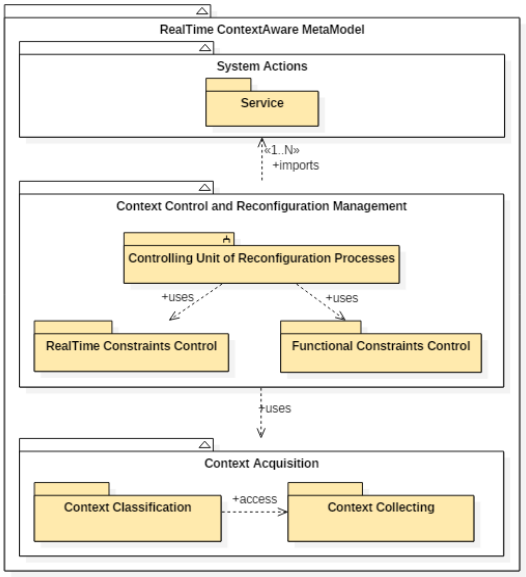

Figure 3: Model diagram of the proposed meta-model.

3.2.1 Context Acquisition

This package is composed of two packages: *Context Collecting* and *Context Classification* (see Fig. 4). *Context Collecting* contains the meta-class *Event Listening* which is responsible for picking-up the triggered events from the surrounding environment. For this, it uses the meta-class *Sensors Handling* which manages the different kind of the used sensors. *Sensors Handling* is composed of the meta-class *Sensor* that offers an abstraction of sensors (it has a set of attributes like the name of the sensor, the detected value...).

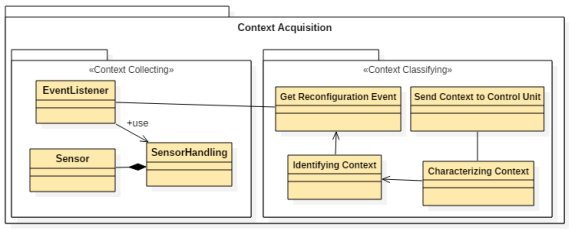

Figure 4: Context Acquisition Level.

Context Collecting package is accessed by the package *Context Classification* through to the meta-class *Get Reconfiguration Event*. Then, the reconfiguration events are imported by the meta-class *Identifying Context* in order to recognize their type. The meta-class *Characterizing Context* will use the identified events and then it performs an abstraction. Finally, it will send them to the control unit through the meta-class *Communicate Context to Control Unit*.

3.2.2 Context Control and Reconfiguration Management

This package uses the *Context Acquisition* in order to get the reconfiguration requests. It also imports the services to be offered by the system from the package *System Actions*. It is composed of three packages linked through dependencies relationships. The main package is called *Controlling Unit of Reconfiguration Processes* (see Fig. 5). This package uses two other packages which contain the system constraints. The specification of constraints is performed on the basis of OCL. We distinguish between two types of constraints: functional and real-time.

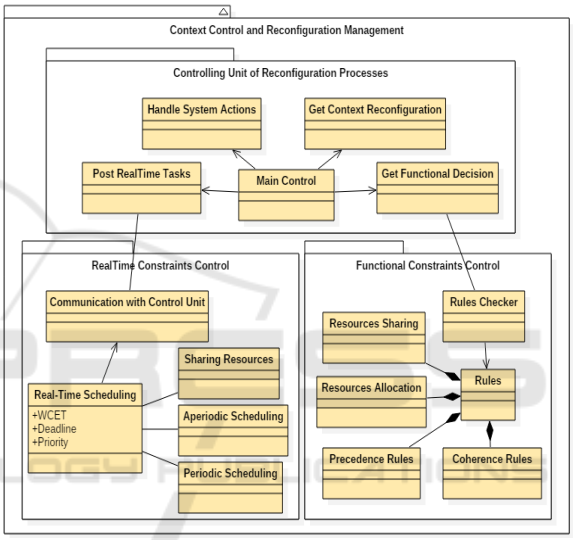

Figure 5: Context Control and Reconfiguration Management Level.

Controlling Unit of Reconfiguration Processes contains five meta-classes. The meta-class *Main Control* includes the main control algorithm. It imports the reconfiguration requests through the meta-class *Get Context Reconfiguration*. To accomplish its task, it uses the meta-class *Get Functional Decision* that will interact with the package *Functional Constraints Control* in order to pick-up the services to be executed after having been checked by this package. Services selection will be performed through the meta-class *Handle System Actions*. The chosen services will be sent to the meta-class *Post Real-Time Tasks* which is responsible for the interaction with the package *Real-Time Constraints* control.

The package *Real-Time Constraints Control* is responsible for scheduling the real-time tasks of the systems. It contains a meta-class called *Communication with Control Unit* responsible for the interaction with the package *Controlling Unit of Reconfiguration Pro-*

cesses. It contains also a meta-class called *Real-Time Scheduling* including the main scheduling algorithm which is responsible for keeping the feasibility of the system. This is a generic meta-class that can be instantiated to specify the type of scheduling. These instances are called as follows: *Periodic Scheduling*, *Aperiodic Scheduling* and *Sharing Resources*.

Functional Constraints Control is the third package. It contains a meta-class called *Rules Checker*. This meta-class is responsible for checking the functional rules of the smooth execution of the system services. It uses a meta-class called *Rules* which is composed of four other meta-classes specifying the type of rules to be checked. These meta-classes are: (i) Resources Allocation responsible for reserving the needed resources for the system execution. (ii) Precedence Rules responsible for organizing the tasks according the precedence logic. (iii) Resources Sharing responsible for indicating all the shared resources to be used in the system. (iv) Coherence Rules indicating the system tasks that have to work simultaneously and the tasks that have not to work together.

3.2.3 Systems Actions

This package is dedicated to be the container of the systems services. It can contain *N* service packages. Every service meta-class can contain also *M* task meta-classes. These meta-classes can be linked through many types of relationship such as import, use, access, etc. In Fig. 6 shows the abstraction of the system service meta-classes.

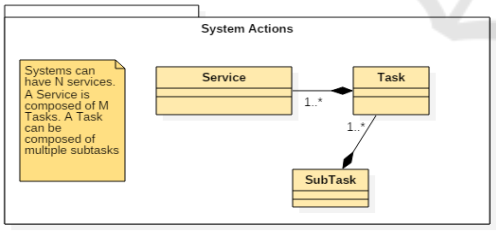

Figure 6: The Package System Actions.

4 CASE STUDY

In order to evaluate the suitability of the proposed meta-model, we present the modeling of a case study.

4.1 Case Study Presentation

The formal adaptive control system must adapt its behavior according to the dynamic changes of the environment (Landau et al., 1998). It is running under real-time as well as functional constraints. It has a set of services that contains the tasks to be performed. Let us assume that this system uses a set of different sensors (temperature, humidity, accelerometer, etc.). Let the ten services called as follows: S_i, i=1..10. These services are distinct units containing the main tasks of the different execution modes. A service is executed after having been elected by the control unit. This control unit organizes the services according to the needs of the user: It can be executed in a periodic way (fixed by the developers) or in aperiodic way (reconfiguration triggered by the events). Some of the services are urgent and need to be treated before exceeding their deadlines.

Services are linked between each other through various relationship types. Some of them are linked by precedence relationship, which means that a service S_i cannot begin the execution before the end of the service S_j. The precedence order for this case study is given by {(S1,6), (S2, 9), (S3, 5), (S4, 10), (S5, 1), (S6, 7), (S7, 2), (S8, 3), (S9, 8), (S10, 4)} (i.e S5 has to be executed at first, then S7, then S8, etc.)

Other services have a resources constraint, which means that they need some shared resources in their execution. The shared resources of this case study are { X, Y, Z, R, Q,} and they are used as follows: {(S1, X), (S2, R-Z), (S4, Y-Q), (S5, Y,R), (S7, X), (S10, R-Z)}. Not only that, but also there are services having a coherence relationship. It determines whether the services can be executed together or not. In this system, we assume that (S2 and S3), (S4 and S7) and (S8 and S5) are not coherent and the execution of the one implies the blocking of the other. Finally, to be executed correctly, the system services need to allocate some resources like the memory, energy, and processor time. Let us consider the resources amounts indicated in the resources table. (see Table 1).

Table 1: Resources table.

Service	1	2	3	4	5	6	7	8	9	10
Memory	10	8	15	20	5	12	10	8	15	30
Energy	20	16	30	40	10	24	20	16	30	60
Processor	8	6	13	18	3	10	8	6	13	28

4.2 Case Study Modeling

The proposed formal adaptive system uses the concepts proposed in the meta-model. A triggered event in the environment is picked-up by the acquisition level (see Fig. 7).

At this level, it is identified by attributing a type and characterized by attributing a set of properties like the Worst Case Execution Time (WCET), the priority, and the deadline. After being abstracted and presented by the first level, the reconfiguration request is sent to the *Context Control and Reconfiguration Man-*

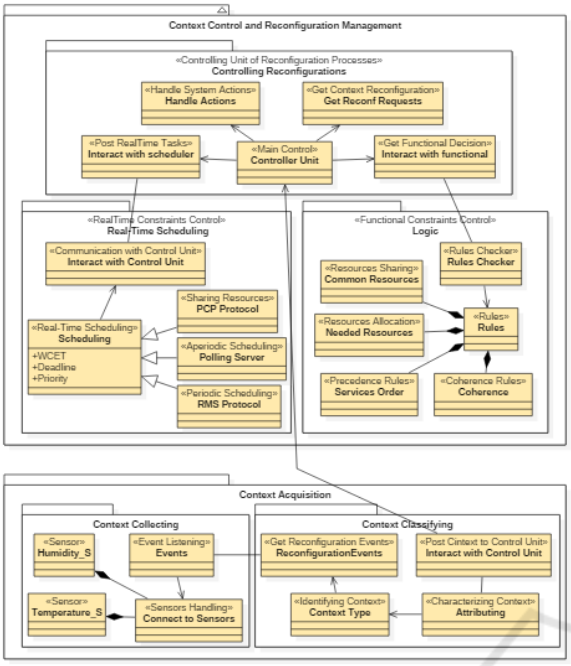

Figure 7: The model of the case study.

agement level in order to be verified. The control unit will guarantee the global control of the reconfiguration process. In fact, the package *Controlling Reconfigurations* will collaborate with the packages of the verification of the system constraints. It will interact with both *Real-Time Scheduling* and *Logic* packages.

The Rate Monotonic (Lehoczky et al., 1989) is implemented to handle periodic tasks, the Polling Server (Davis et al., 1993) for the aperiodic tasks and the Priority Ceiling Protocol (Sha et al., 1990) for the scheduling under sharing resources. The system feasibility is guaranteed when services do not exceed their deadline. Also when the rate of utilization of the processor is not at its maximum level. For this reasons, we have developed a set of real-time constraints by means of OCL (see Fig. 8). These queries have to be always satisfied in order to keep system correctness.

Scheduling constraints:

Context PCPProtocol inv: (SharedVariable.occupied=true) implies (NewTask.State= Blocked)

Context PCPProtocol inv: if (SharedVariable.free=true and Task.Priority> CurrentPrioriyCeiling) then (Task.State=AllocatingsharedVariable) then Task.State=Blocked

Context PollingServer inv: (ServerCapacity>0 and Time % ServerPeriod=0) implies (Task.State=executing)

Context PollingServer inv: ((TasksNumber+1)*(2 exp (1/TasksNumber+1)-1)>= ((ServerPeriod/ServerCapacity) +forall Tasks (Task.Capacity/Task.Period)) implies System.State=feasible

Context RMSProtocol inv: Task.Period==MinPeriod implies Task.Priority==MaxPriority

Context RMSProtocol inv: (TasksNumber*(2 exp (1/TasksNumber)-1) >= forall Tasks (Task.Capacity/Task.Period)) implies System.State=feasible

Figure 8: The feasibility OCL queries.

In the same manner, the package *Logic* is designed to hold the functional constraints of the system. The class *Rules* is composed of four other classes. *Needed Resources* will manage the used resources during the execution of the control application. It leads all allocation and de-allocation processes. *Common resources* manage the shared resources that can be used by the system. *Services Order* is responsible for controlling precedence constraints. *Coherence* mention the services that are able to be treated together. The OCL rules are implemented by the classes of the functional constraints package in order to avoid confusion (see Fig. 9).

Functional Constraints

Context Coherence inv: S1.State=executing implies S2.Executable=true and S4.Executable=true and S6.Executable and S8.Executable=true and S9.Executable=true;

Context Needed Resources inv: S1.Memory.Size(10) and S2.Memory.Size(8) and S3.Memory.Size(15) and S4.Memory.Size(20) and S5.Memory.Size(5) and S6.Memory.Size(12) and S7.Memory.Size(10) and S8.Memory.Size(8) and S9.Memory.Size(15) and S10.Memory.Size(30);

Context Services Order inv: S1.End< S7.Start and S7.end< (S2.start &S8.start) and (S2.end& S8.end) <S10.start and S10.end< S3.start and S3.end<S1.start and S1.end < (S4.start & S6.start) and (S4.end &S6.end) < S9.start.

Figure 9: The functional OCL rules.

4.3 Case Study Simulation

In this subsection, we present the simulation scenario of the modeled adaptive system.

4.3.1 Implementation

We have developed a framework following the new meta-model. This framework is holding the modules existing in the three levels described by the meta-model. We used C# programming language in the implementation phase. Fig. 10 shows the class responsible for detecting events from the environment.

```
namespace Event
{
    public class Listener
    {
        // Receive event
        public void ReceiveEvent(object sender, EventArgs e)
        {
            InfoFromSenderToRecieverArgs info = e as InfoFromSenderToRecieverArgs;
            string formatedInfo = String.Format("Event received on {0}. Name of Event: {1}
            info.ArrivalDate, info.Event.name, info.TypeEvent, info.Event.priority, info.Ev
            Console.WriteLine(formatedInfo);
        }
    }
    public delegate void InfoFromSenderToRecieverHandler(object sender, EventArgs e);
    public class InfoFromSenderToRecieverArgs : EventArgs
    {
        // public enum MessageType { Service, Object, Data };
        public string TypeEvent;
        public SendAperiodicEvent Event;
        public DateTime ArrivalDate;
        public InfoFromSenderToRecieverArgs(SendAperiodicEvent e, string msg)
        {
            Event = e;
            TypeEvent = msg;
            ArrivalDate = DateTime.Now;
        }
    }
}
```

Figure 10: Listener class of the context collecting package.

4.3.2 Simulation

We consider the formal adaptive system presented in the previous subsection. The services priorities are

given as follows: {(S1, 5), (S2, 6), (S3, 3), (S4, 7), (S5, 4), (S6, 5), (S7, 6), (S8, 6), (S9, 3), (S10, 1) } . S10 has the highest priority, S4 the next highest priority, and so on. The first step performed by the control unit is loading the services and checking the functional constraints. Then, it starts the resources allocation for each service (see Fig. 11)

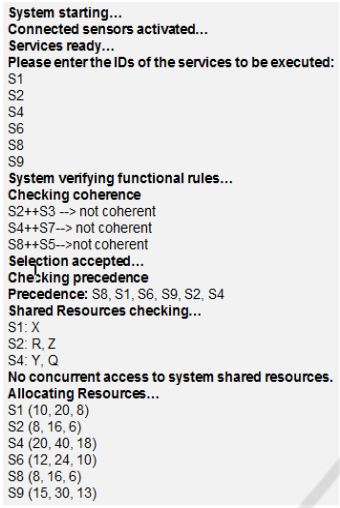

Figure 11: The execution trace of the functional checking.

After that, the control unit interacts with the scheduling package and set-up the scheduling table in the real-time constraints module. Table 2 indicates the capacity of each service as well as its period of activation.

Table 2: Services capacities and periods.

Service	1	2	3	4	5	6	7	8	9	10
Capacity	5	6	3	2	7	4	5	6	3	1
Period	22	28	8	6	45	16	22	35	11	4

The Rate Monotonic scheduling protocol is selected as the protocol to be used to handle periodic services. The execution of the services is given in the next Fig. 12. In each timer tick, the processor treats a unit of a service.

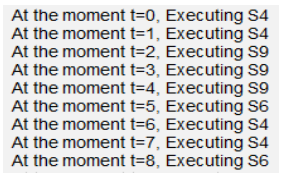

Figure 12: The execution of the services according RMS protocol.

The polling server has these properties: *capacity=4* and *period=10*. So, every 10 time units the polling server will be activated in order to check the list of aperiodic services (Fig. 13).

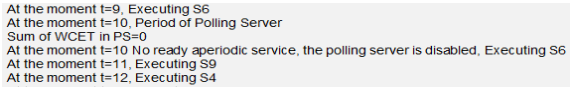

Figure 13: The activation of the Polling Server.

A reconfiguration scenario take place when a reconfiguration request is received by the control unit. A reconfiguration request can be a change in the environment, human intervention, components added or deleted from the system, etc. These reconfigurations are unpredictable and they are event driven.

We suppose that three reconfiguration requests are picked-up during the execution and that they have the following properties (see Table 3).

Table 3: Reconfiguration table.

id	arrival	wcet	deadline	priority	target
R1	t=14	20	22	2	S3
R2	t=25	30	32	1	S10
R3	t=40	35	36	3	S7

Every reconfiguration request is labeled by a deadline, WCET, and priority and it aims at loading a specific service. When they come in the system, the control unit checks the priority of the current executing service and decides whether to interrupt the execution or to re-schedule the ready tasks.

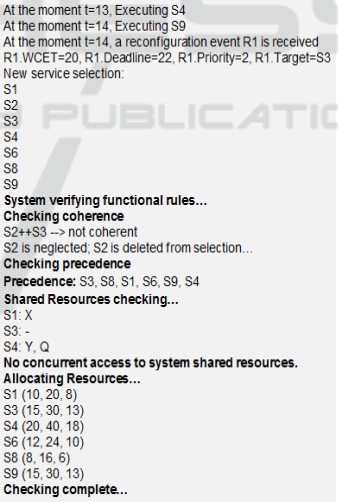

Figure 14: Checking the functional constraints of the reconfiguration request.

At *t=14*, a reconfiguration event is coming to load the service S3. This change is urgent and has a high priority (*priority=2*). At this moment, the control unit will check all the functional constraints and prepare the new configuration to be applied on the next activation of the Polling Server. The services order is necessarily updated, the service loaded by the reconfiguration request will take place in the schedule. Fig. 14 shows the execution trace of these updates. The

execution of the reconfiguration service starts at the activation period of the Polling Server (the moment *t=20*) as shown in Fig. 15.

```
At the moment t=15, Executing S9
At the moment t=16, Executing S6
At the moment t=17, Executing S6
At the moment t=18, Executing S4
At the moment t=19, Executing S4
At the moment t=20, Period of Polling Server,
At the moment t=20, Executing S3
At the moment t=21, Executing S3
At the moment t=22, Executing S3
At the moment t=23, Executing S3
```

Figure 15: The execution of the loaded service.

The simulation is performed in this way: every received request will induce a checking of the functional constraints then the aimed service will be scheduled according to the real-time protocols.

4.4 Comparison with other work

Comparing our proposed meta-model with other existing works will help us to highlight the originality as well as the efficiency of our contribution. That is why we compared our meta-model with the Chameleon (Nafti et al., 2015). We chose to compare it with Chameleon meta-model since it includes some similarities. In fact, both meta-models intend to model the reconfiguration of control systems. Reconfigurations are performed with the aim of satisfying the adaptation of the system to its environment as well as satisfying an optimal operation of the system.

In Chameleon, the classes composing the system may change their behavior automatically. They contain all the common methods for all the possible configurations and a set of specific methods for each configuration. Therefore, if developers choose to implement the control applications of an adaptive control system with Chameleon, they have to develop their system services through the Chameleon classes. And according to the definition of the Chameleon classes, some components have to be created in all of them such as: the method table, the scheduling table, the data table, the constant table, etc.

Our meta-model shows a higher level of consistency and robustness in fulfilling these goals. In fact, it presents a separation between the system services and the control processes. As seen before, all the control aspects are encapsulated into the Context Control and Reconfiguration Management level while the system services are encapsulated in the System Actions level. This distinction and separation of concepts helps to benefit from re-usability and make the development process more flexible. The most important thing is that our architecture allows a better resource consumption since no redundancy or extra processing is done.

As a proof, we present an approximate calculation of the response time of the system when using our meta-model compared to using Chameleon meta-model (see Fig. 16).

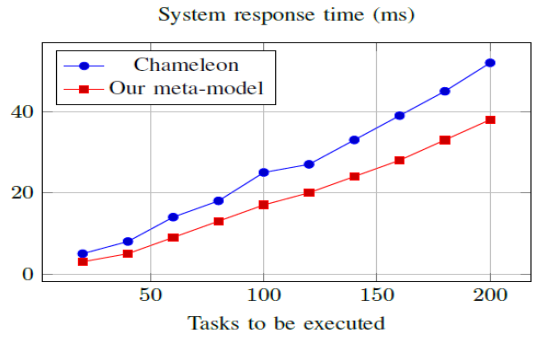

Figure 16: The system response time.

According to Figure 16, with the increasing number of the system services the system response time taken by Chameleon is higher than the response time taken by our meta-model. This can be explained by the definition of several tables in each Chameleon class. Surely these tables require much processing time comparing to our meta-model. Thus, our meta-model prove its robustness.

5 CONCLUSION

This paper proposed a new context-awareness meta-model for developing real-time reconfigurable adaptive control systems. This new meta-model offers the satisfaction of real-time as well as functional constraints that can be needed in adaptive control systems. Compared to other meta-models developed for adaptive control systems, the reconfiguration flexibility, the awareness about the context of use and the system response time are optimized. A framework is implemented using C# programming language and a formal case study is simulated as a running example. To test the performance of the proposed meta-model, we have performed measurement of the system response time and we have compared the results with Chameleon meta-model. We presented the analysis of the obtained results. In future works, we aim to expand the flexibility and robustness of the meta-model by covering other control aspects such as the quality of service issues and the compliance to multi-platforms.

REFERENCES

Al-Sultan, S., Al-Bayatti, A. H., and Zedan, H. (2013). Context-aware driver behavior detection system in intelligent transportation systems. *IEEE transactions on vehicular technology*, 62(9):4264–4275.

Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., and Tamburrelli, G. (2011). Dynamic qos management and optimization in service-based systems. *IEEE Transactions on Software Engineering*, 37(3):387–409.

Capilla, R., Zimmermann, O., Zdun, U., Avgeriou, P., and Küster, J. M. (2011). An enhanced architectural knowledge metamodel linking architectural design decisions to other artifacts in the software engineering lifecycle. In *European Conference on Software Architecture*, pages 303–318. Springer.

Davis, R. I., Tindell, K. W., and Burns, A. (1993). Scheduling slack time in fixed priority pre-emptive systems. In *Real-Time Systems Symposium, 1993., Proceedings.*, pages 222–231. IEEE.

Dowling, J. and Cahill, V. (2001). The k-component architecture meta-model for self-adaptive software. In *International Conference on Metalevel Architectures and Reflection*, pages 81–88. Springer.

Feiler, P. H., Lewis, B. A., and Vestal, S. (2006). The sae architecture analysis & design language (aadl) a standard for engineering performance critical systems. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pages 1206–1211. IEEE.

Gumzej, R., Colnaric, M., and Halang, W. A. (2006). Safe and timely scenario switching in uml real-time projects. In *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06)*, pages 8–pp. IEEE.

Hamid, B. and Krichen, F. (2010). Model-based engineering for dynamic reconfiguration in drtes. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, pages 269–276. ACM.

Idriss, R., Loukil, A., and Khalgui, M. (2015). New middleware for secured reconfigurable real-time systems. In *International Conference on Intelligent Software Methodologies, Tools, and Techniques*, pages 469–483. Springer.

Krichen, F., Hamid, B., Zalila, B., and Jmaiel, M. (2011). Towards a model-based approach for reconfigurable dre systems. In *European Conference on Software Architecture*, pages 295–302. Springer.

Landau, I. D., Lozano, R., M'Saad, M., and Karimi, A. (1998). *Adaptive control*, volume 51. Springer Berlin.

Lehmann, G., Blumendorf, M., Trollmann, F., and Albayrak, S. (2010). Meta-modeling runtime models. In *International Conference on Model Driven Engineering Languages and Systems*, pages 209–223. Springer.

Lehoczky, J., Sha, L., and Ding, Y. (1989). The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171. IEEE.

Lepuschitz, W., Zoitl, A., Vallée, M., and Merdan, M. (2011). Toward self-reconfiguration of manufacturing systems using automation agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(1):52–69.

Lo, C.-C., Chen, C.-H., Cheng, D.-Y., and Kung, H.-Y. (2011). Ubiquitous healthcare service system with context-awareness capability: Design and implementation. *Expert Systems with Applications*, 38(4):4416–4436.

Nafti, A., Romdhani, M., and Khalgui, M. (2015). Chameleon: New object oriented solution for adaptive control systems. In *Pervasive and Embedded Computing and Communication Systems (PECCS), 2015 International Conference on*, pages 1–8. SCITEPRESS.

Schilit, B. N. and Theimer, M. M. (1994). Disseminating active map information to mobile hosts. *IEEE network*, 8(5):22–32.

Schuller, B., Dunwell, I., Weninger, F., and Paletta, L. (2013). Pervasive serious gaming for behavior change–the state of play. *IEEE pervasive computing*, 3(12):48–55.

Sha, L., Rajkumar, R., and Lehoczky, J. P. (1990). Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on computers*, 39(9):1175–1185.

Siddiqi, A. and de Weck, O. L. (2008). Modeling methods and conceptual design principles for reconfigurable systems. *Journal of Mechanical Design*, 130(10):101102.

Sukode, S., Gite, S., and Agrawal, H. (2015). Context aware framework in iot: A survey. *International Journal*, 4(1).

Valentini, A., Gharbi, A., Khalgui, M., and Gharsellaoui, H. (2013). Safety reconfiguration of embedded control systems. *Embedded Computing Systems: Applications, Optimization, and Advanced Design: Applications, Optimization, and Advanced Design*, page 184.

Zhan, K., Faux, S., and Ramos, F. (2014). Multi-scale conditional random fields for first-person activity recognition. In *Pervasive Computing and Communications (PerCom), 2014 IEEE International Conference on*, pages 51–59. IEEE.

Zhang, D., Huang, H., Lai, C.-F., Liang, X., Zou, Q., and Guo, M. (2013). Survey on context-awareness in ubiquitous media. *Multimedia tools and applications*, 67(1):179–211.