R2TCA: New Tool for Developing Reconfigurable Real-Time Context-Aware Framework

-Application to Baggage Handling Systems-

Soumoud Fkaier*†§, Mohamed Romdhani*, Mohamed Khalgui*‡,and Georg Frey§
*LISI Laboratory, INSAT, University of Carthage, Tunis, Tunisia
†Polytechnic School of Tunisia, University of Carthage, Tunis, Tunisia
‡Systems Control Lab, Xidian University, China
§Chair of Automation and Energy Systems, Saarland University, Saarbrucken,Germany
Email:{soumoud.fkaier, georg.frey}@aut.uni-saarland.de , { khalgui.mohamed, mromdhani7}@gmail.com

***Abstract*—Context-awareness was introduced in various domains of ubiquitous computing ranging from mobile computing to automated manufacturing. It gained this importance based on the fact that it provides the possibility to handle adaptive systems according to the environment changes. Therefore, a wide variety of frameworks was developed. However, some requirements are still not satisfied, especially those related to resolving functional constraints, such as inclusion, precedence, and shared resources constraints. Dealing with real-time issues also has not been satisfied. In this work, we propose a new tool for developing a context-aware framework able to overcome the mentioned problems. As proof of concept, we simulated a case study and performed results analysis.**

Keywords–Context-aware framework; Reconfigurable system; Real-time system; Functional constraint; Flexible software service.

I. INTRODUCTION

Context-aware systems are characterized by their ability to interact with the surrounding environment [1]. They sense changes in their environment and adapt their behavior accordingly [2]. These changes act as contexts that will induce system reconfiguration. Since the 1990s, this issue has gained the attention of both the academic and manufacturing fields. Thus, many methodologies, middleware and frameworks have been proposed [3][4][5]. One important field of application of this paradigm is developing applications of adaptive control systems. In fact, these systems are self-adapting systems known by the flexibility to adapt their behavior to the environmental dynamic changes [6]. So, this feature can be satisfied based on context-awareness.

Developing a context-aware framework for adaptive control systems is a challenging task. In fact, these systems require a set of particular exigencies. First of all, they have to adapt their behavior properly without losing system effectiveness. Reconfigurations must always be done safely without conflicts or break downs. To clarify, logical relations between the tasks of reconfiguration processes such as rejection rules must be absolutely respected. Similarly, precedence constraint as well as using some shared resources ought to be guaranteed. Just as respecting these relationships, managing the allocation and de-allocation of the used resources has to be insured as well. No doubt, providing the services before their deadlines is of great importance otherwise the services lose their meaning.

It is true that the available literature on context-aware frameworks has evolved over time. Particularly, providing solutions for real-time as well as functional constraints have gained a great attention from researchers. However, these two points of interest are still not developed in clear and efficient way [10] [13] [14]. For this reason, we propose in this paper a new tool for developing context-aware frameworks to solve the aforementioned constraints. It is called Reconfigurable Real-Time Context-Aware (R2TCA) framework. This new tool is dedicated to developing reconfigurable systems running under real-time and functional constraints. It enables to develop applications following a layered architecture composed of four layers [21]. Every layer has a specific role in the adaptation process. We took an example of baggage handling systems as adaptive system in order to prove the suitability of R2TCA. Moreover, system response time as well as the memory utilization rate is calculated so that we prove R2TCA robustness.

This paper is structured in five main sections. In Section 2, we present the state of the art. Section 3 describes the new tool. Section 4 shows the application of the new tool to a case study. Finally, Section 5 concludes the paper.

II. BACKGROUND

Many works have proposed different context-aware frameworks. In this section, we give an overview of these achievements.

Forkan et al. [7] have proposed the Cloud-oriented context-aware middleware in ambient assisted living (CoCaMAAL). They focused on developing a scalable and context-aware framework in order to facilitate both data collection and processing. Forkan et al. [8] have performed a Big Data for Context-aware Monitoring (BDCaM) that is an extension of CoCaMAAL. They proposed a discovery-based approach enabling systems to adapt their behavior at run-time. It enables finding context information using big data. Mcheick et al. [9] have proposed a context-aware architecture for health care systems in which they focused on abstracting the context. They consider that scalability and inter-operability are the key features towards the abstraction. They added an extension for the addition of sensors so that they simplify dealing with sensed data. Edwin and Alvin [10] have defined CAMPUS, which is a middleware for making automated decisions of adaptation at run-time. Their aim was to reduce the effort made by developers by getting rid of the need to predict and maintain adaptation rules. Lei et al. [11] have proposed a tool called PerDe. They focused on designing a domain-specific language. Also, they provided a set of graphical tool-

 ISBN: 978-1-61208-505-0

kits covering development steps for ubiquitous computing applications. Balland and Consel [12] have introduced DiaSuite which is a tool dedicated to drive the development processes on specific domains of Sense/Compute/Control (SCC). It has a compiler responsible for generating customized basis for every development step. Liu and Cheng [13] have proposed a middleware framework called MARCHES. It aims to support time-critical adaptive vehicle systems. Also, they tried to improve reconfiguration efficiency for these systems in changing environments. Papadopoulou et al. [14] have proposed an approach of development of pervasive systems based on the notion of Personal Smart Space (PSS). In this work, they addressed the issue of sharing resources.

These existing concepts and tools have been developed in order to satisfy adaptive systems requirements. Although, most of them do not propose a clear strategy to handle real-time services. In addition, there are no solutions that have treated functional constraints like managing dependencies relations. In addition, resource sharing was not considered in the majority of the existing works. Moreover, coherence rules, such as inclusion and exclusion rules, were not considered either. That is why we propose in this paper a new context-aware framework in order to overcome these constraints. We implemented a four layers architecture where every layer has its specific role. This new tool is able to satisfy what the existing works do not.

III. R2TCA: New Tool for Reconfigurable Real-Time Context-Aware Systems

The four layers architecture is given as follows: (i) Reconfiguration layer, (ii) Context control layer, (iii) Services layer, and (iv) Communication layer. The technical description and details of the internal models and behavior of each layer is available in a research report at [21]. The goal of this paper is to present the implemented tool and to show its execution.

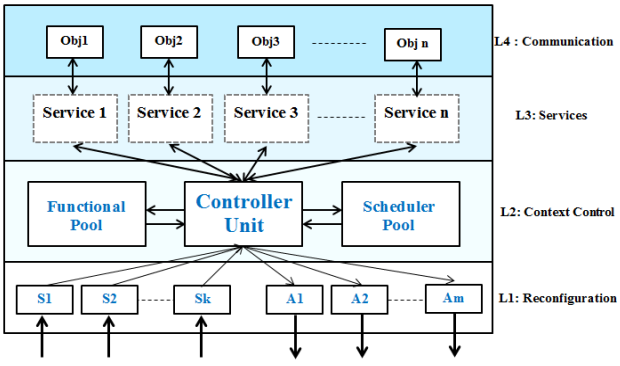

Figure 1. The modules of R2TCA layers.

Fig. 1 presents the superposition of the framework's four layers. The description of layers and their modules is depicted in the next paragraphs.

A. Reconfiguration Layer

This is the first layer in the architecture (see Fig. 2). It is responsible for collecting triggered events in the environment. This collection is ensured thanks to sensors and the layer considers these events as reconfiguration requests. Then, it forwards the reconfiguration requests to the upper layer. The second role of this layer is to transfer commands coming from the upper layer after having checked the constraints to the actuators of the connected devices.

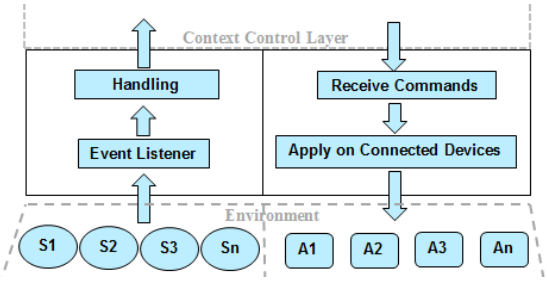

Figure 2. Reconfiguration layer modules.

Fig. 2 shows the composition of this layer. It is composed of four main modules. *Event Listener* is listening for the events. *Handling* is responsible for treating the sensed events and forwarding them to the upper layer. *Receive Commands* is responsible for switching the changes to the environment. *Apply on Command Devices* is responsible for delivering the needed actions to the actuators.

B. Context Control Layer

This is the second layer in the architecture. It is the key layer in the entire proposed framework since it is responsible for controlling the execution of the adaptation process. It is composed of three components, as follows: controller unit, functional pool and scheduler pool.

1) Controller Unit: It is the component responsible for the collaboration between the layers of the architecture and between the components of the context control layer (see Fig. 3). Its role is to receive reconfiguration requests sent by the reconfiguration layer and: (i) decides the modification level that should be processed (whether loading a new service or updating some objects of a service or updating some data of a service), (ii) sends functional constraints to the functional pool to be checked, (iii) sends tasks to be executed to the scheduler pool.

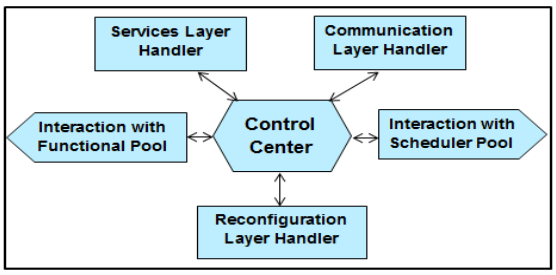

Figure 3. Controller unit modules.

2) Functional Pool: This pool is responsible for checking functional constraints that can be present in adaptive systems. It guarantees the control of dependencies such as using shared resources. Also, it offers the possibility to order the services according to their importance and precedence relation through a priorities table. In addition, it keeps coherent execution of the services by checking inclusion and exclusion rules. Moreover, it provides the ability to allocate and de-allocate resources to be used. Fig. 4 depicts the test levels that guarantee a correct functional behavior.

 ISBN: 978-1-61208-505-0

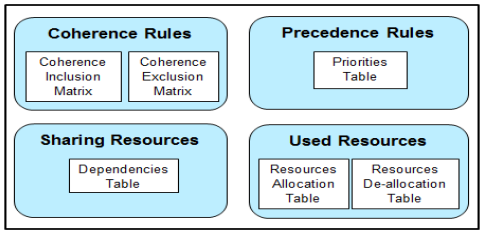

Figure 4. Functional pool modules.

3) Scheduler Pool: This pool is responsible for scheduling real-time tasks. It offers the possibility to employ different categories of scheduling protocols (see Fig. 5). It takes in protocols for scheduling periodic tasks with fixed priorities (Rate Monotonic [15] and Deadline Monotonic [16]) and dynamic priorities (Earliest Deadline First [17] and Least Laxity First [18]). Also, it takes in scheduling of aperiodic tasks thanks to Polling, Deferrable and Sporadic servers [19]. Not only that, but it enables scheduling with regard to sharing resources through Priority Ceiling Protocol [20].

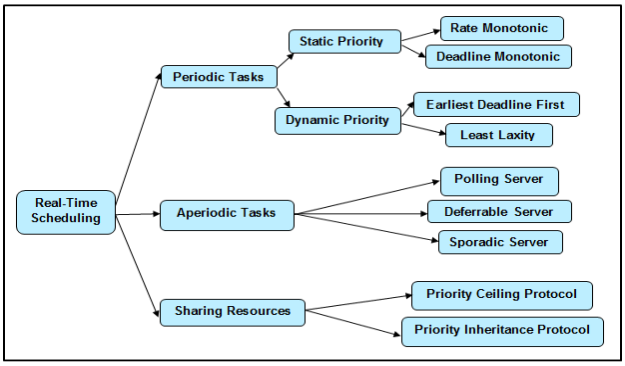

Figure 5. Scheduler pool modules.

C. Services Layer

This is the third layer in the architecture. It contains services containers. In fact, a service is the set of tasks to be accomplished by the system. These tasks are translated in the form of code organized in objects (these objects are classes from the object oriented programming).

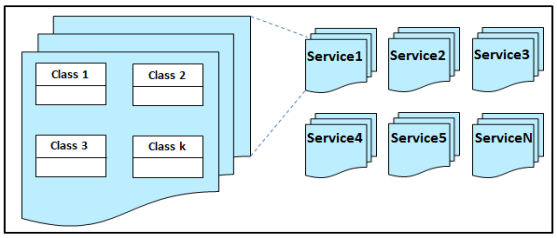

Figure 6. Services layer modules.

Fig. 6 depicts the services containers. Every container is composed of a set of classes including the tasks of the system.

D. Communication Layer

This is the fourth and the last layer in the proposed framework. It represents the interface of the framework to the developers (Fig. 7). It contains a set of interfaces (from object oriented programming) that encapsulates the tasks offered by the services. Each interface represents one service.

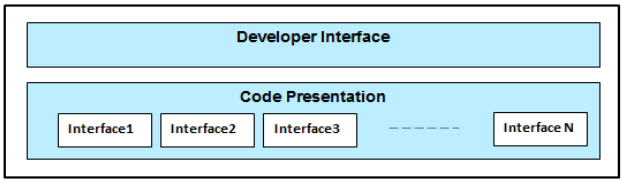

Figure 7. Communication layer modules.

R2TCA is distinguished by the high level of control during leading reconfigurations. It provides a whole component to handle the correctness of the processes execution. Neither deadlocks nor blocking will be faced at run-time thanks to the functional pool. R2TCA solves the problem of system feasibility by including a scheduler pool. This pool guarantees that all services will be executed before exceeding their deadlines.

IV. Application

Based on the R2TCA description presented in the previous section, we have implemented our framework using C# programming language. We have developed the four layers. We proved the suitability of our work by developing a control application of an airport baggage handling system.

A. Baggage Handling System Model and Design

A baggage handling system (BHS) is composed of different devices like conveyor sections, Radio Frequency IDentification (RFID) readers, X-ray sections, pushers, etc. The main target behind using such system is to transport passenger baggage to the right destination at the right time. Fig. 8 depicts an example of BHS. There are various services accomplished by this system: up-stream, down-stream, merging, diverting, tracking, and stop services.

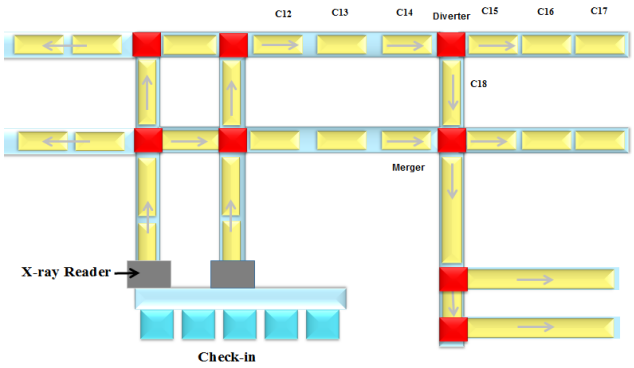

Figure 8. The baggage handling system structure.

To be executed as expected, the BHS has to overcome some problems. First, this system can face reconfigurations at run-time. These reconfigurations can be caused by different reasons like maintenance, failures, human intervention, actions in the environment. So, the system should adapt its behavior accordingly. This adaptation is not a trivial task since putting the system offline or causing some blockage are not acceptable. Thus, controlling functional aspects of reconfiguration processes is really a crucial need. Moreover, conveying the

 ISBN: 978-1-61208-505-0

baggage from check-in to the right destination must not take a long time, otherwise the system loses its effectiveness. In this context, we implemented R2TCA a new tool providing the possibility to overcome these problems.

B. Contribution: the new tool R2TCA

In this subsection, we present our developed tool R2TCA. We provide a description of the implementation of the main functionality.

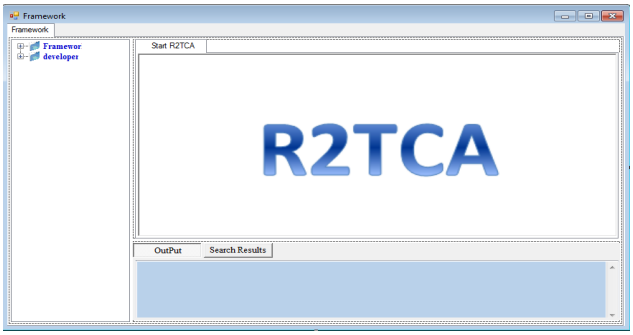

Figure 9. The developers interface of R2TCA.

We have implemented R2TCA which includes the framework services. Fig. 9 depicts its interface. This interface enables developers to access the code of the framework and to use it in order to implement control applications. The folder *Framework* shown in Fig. 10 contains all the packages and classes of our framework. Developers have to use these classes by means of inheritance. They have the possibility to edit the folder *Developer* and create their own project.

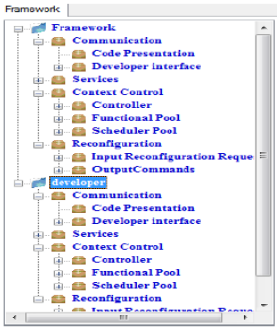

Figure 10. The packages of R2TCA.

In their new projects, developers are free to use the scheduling protocol that fits their needs, they are also free to modify, add, and delete some services (see Fig. 11):

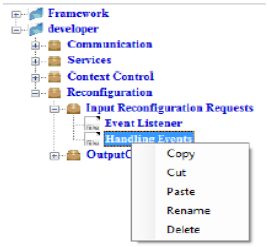

Figure 11. The developer project files.

We have implemented all the scheduling protocols aforesaid in the section of scheduler pool. Feasibility tests and simulations were performed. Let us see how this tool can be used with a baggage handling system.

C. Simulation of the baggage handling system

Before we go to the simulation and test, let us present the developed services and the relations between them.

TABLE I. SERVICES FEATURES.

Name	ID	Priority	included services	excluded services	Resources	Type
Stop	0	1	4	1,2,3,5	R	Real-time
Upstream	1	4	4	0,2	R,Q	Real-time
Downstream	2	4	4	0,1	R,Z	Real-time
Merge	3	3	4	0,5	-	Real-time
Track	4	2	-	-	R	Real-time
Divert	5	3	4	0,3	R,Q	Real-time

As mentioned in Table 1: Service stop has the highest priority, service track has a lower priority, both merge and divert services have equal priorities which is less than track priority, up-stream and down-stream also have the same priority, which is the lowest priority.

The relation between these services can be defined as follows: (i) Services up-stream and down-stream are in exclusion (because they are opposite). (ii) Services merge and divert are also in exclusion. (iii) Service track is in inclusion with all the rest of services. (iv) Service stop is in exclusion with the rest of services. We mentioned also the shared resource R and Q and the type of services (whether they are real-time or not).

We consider the BHS shown in Fig. 8. Baggage has to be transported from the check-in point to the departure gates. Initially, service down-stream and track are running. A reconfiguration scenario arises when the conveyor 16 is broken down. Baggage has to be routed another way, so the control application will load the stop service to stop the down-stream and then to run up-stream until reaching a diverting section (see Fig. 12).

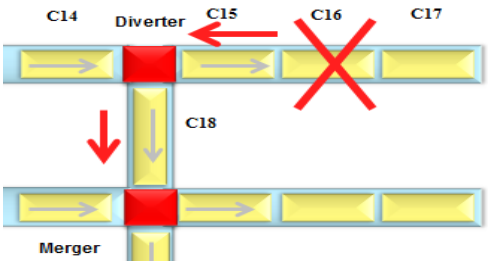

Figure 12. The reconfiguration scenario.

The controller unit begins by selecting down-stream and track service from the services layer (see Fig. 13).

```
Begin : Control Unit
Controller Extract services from the Periodic Task of the service DownStream
Services extacted is  : Downstream ;  id Extracted = 2;
End : Control Unit
```

Figure 13. The controller selection of downs-stream service

After that, the controller sends a functional request to the functional pool in order to verify the needed restrictions. Once

called, the functional pool creates the necessary tables. It begins with creating the exclusion matrix (this matrix helps to determine which services can work together and which services cannot work together, as we mentioned in Table 1). Similarly, it creates the inclusion matrix that helps to determine which services can be executed simultaneously. Then, it creates priorities, dependencies and resources tables.

After creating all the necessary control matrices and tables, the functional pool starts the verification of the functional requirements of the down-stream service. First of all, it will check the priority of the selected service from the priorities table. The priority of down-stream is 4. The functional pool will verify inclusion and exclusion rules thanks to the created matrices. Then, it checks the dependencies rules using the dependencies table. It verifies if there are shared resources between the services. As shown in Fig. 14, service down-stream uses the resources R and Q, and service track uses the resource R. So, it returns that there are shared resources and Priority Ceiling Protocol has to be used to control these resources. Finally, resources allocation table will be checked to allocate the other needed resources (see Fig. 15).

```
||||||||||||||||||||||Begin : Functional Layer|||||||||||||||||||||||
Functional Pool Receive a new request from Control Unit : ID of service Recieved = 2
||||||||||||||||||||||Begin : Functional Layer/Check Priority|||||||||||||||||||||||
ID of the service Downstream = 2
Functional pool : verifying priority ....
id of the service 2
Priority of the service 4
Verification of Priority : done !
||||||||||||||||||||||End : Functional Layer/Check Priority|||||||||||||||||||||||
||||||||||||||||||||||Begin : Functional Layer/Check Exclusion|||||||||||||||||||||||
Functional pool : verifying Exclusion Matrix....
ID of the service Downstream = 2
Services Downstream and Stop are in exclusion
Services Downstream and Upstream are in exclusion
Services Downstream and Downstream are not in exclusion
Services Downstream and Merge are not in exclusion
Services Downstream and Track are not in exclusion
Services Downstream and Divert are not in exclusion
Verification of Exclusion Matrix : done !
||||||||||||||||||||||End : Functional Layer/Check Exclusion|||||||||||||||||||||||
||||||||||||||||||||||Begin : Functional Layer/Check Inclusion|||||||||||||||||||||||
Functional pool : verifying Inclusion Matrix....
ID of the service Downstream = 2
Services Downstream and Stop are not in Inclusion
Services Downstream and Upstream are not in Inclusion
Services Downstream and Downstream are not in Inclusion
Services Downstream and Merge are not in Inclusion
Services Downstream and Track are in exclusion
Services Downstream and Divert are not in Inclusion
Verification of Inclusion Matrix : done !
||||||||||||||||||||||End : Functional Layer/Check Exclusion|||||||||||||||||||||||
```

Figure 14. Checking functional constraints (first part).

```
||||||||||||||||||||||Begin : Functional Layer/Check Dependencies|||||||||||||||||||||||
Functional pool : verifying Dependencies ....
  Service with th ID 2      Use ressource : R
  Service with th ID 2      Use ressource : Z
  Service with th ID 4      Use ressource : R
services with ID = 2 and 4 share the resource R
 There are Shared Ressource : PCP have to be used
Verification of Dependencies : done !
||||||||||||||||||||||End : Functional Layer/Check Dependencies|||||||||||||||||||||||
||||||||||||||||||||||Begin : Functional Layer/Check Resources Allocation|||||||||||||||||||||||
Functional pool : verifying Resource Allocation....
initialization of resource is done R UnLocked
Adding the resource R in the list
initialization of resource is done Z UnLocked
Adding the resource Z in the list
Resource : R exists already
Verification of Resource Allocation : done !
||||||||||||||||||||||End : Functional Layer/Check Resources Allocation|||||||||||||||||||||||
||||||||||||||||||||||End : Functional Layer|||||||||||||||||||||||
||||||||||||||||||||||Begin : Control Unit|||||||||||||||||||||||
 Type of Service : Real Time Service
//////////////////Begin : Control Unit
Controller Send Periodic Task DownStream to scheduler Pool
End : Control Unit//////////////////
```

Figure 15. Checking functional constraints (second part).

Once all these constraints are verified, the functional pool sends a positive feedback to the controller. At this level, the controller verifies if the services type is real-time or not. Down-stream is defined as real-time service in our system, so it will be sent to the scheduler pool to be scheduled based on the chosen protocol (in our case, Rate Monotonic is selected to handle periodic tasks and the polling server to handle aperiodic tasks).

The same steps of checking will be repeated with the service track. As seen in Table 1, the downstream and track services are real-time services, so they will be switched to the scheduler pool. The scheduler pool starts the execution of services. At the period of Polling Server (Fig. 16), it checks the list of aperiodic events (reconfiguration events).

```
Period of Polling Server
Sum of  WCET in PS = 0
At the moment 29/05/2016 13:34:04 No ready aperiodic event, The polling server is disabled
////** t=2**////At the moment  29/05/2016 13:34:04 execute the Service DownStream/////////
Running
```

Figure 16. Checking queue of aperiodic tasks.

While our system is running and conveyors are advancing, a problem arises with the motor responsible for conveyor 16, making it unable to execute any command. So, our system should find another valid path to transport baggage to the target destination. This change is picked-up by R2TCA when sensors send an event to stop down-streaming and run up-streaming until reaching a diverting section. Here, stop and up-streaming will be inserted in the queue of aperiodic tasks. This change is considered a reconfiguration scenario.

Before any verification, the controller checks the feasibility of the system and decides if this event will be accepted or not. After verifying this condition, the controller verifies that once accepted, this new event will not cause the lower priority events accepted and not yet executed by the system to miss their deadlines.

After accepting the new event, the controller extracts the ID of the service stop (ID=0), then sends a request to the functional pool. As Table 1 shows, service stop has the priority 0 (the highest priority) and is in exclusion with all the other services except the service track. On the other hand, service stop includes track service. So, the track will not be deleted from the list of competitive services. The functional pool verifies dependencies and resources rules then returns the feedback to the controller. The controller sends the aperiodic event stop to the scheduler pool by putting it in the queue of aperiodic tasks (see Fig. 17).

```
Begin : Control Unit
Controller Send Aperiodic Task Aperiodic-Stop Event to scheduler Pool
End : Control Unit
////** t = 8**//////////////At the moment 29/05/2016 13:34:10 No ready Periodic event //
```

Figure 17. Controller sends aperiodic tasks to scheduler pool.

The stop service is used in order to stop the down-stream of conveyors. After stopping down-stream, the reconfiguration layer sends a second request to load the service up-stream. Conveyors should move back until reaching a divert point. The controller checks the feasibility, accepts up-stream event, creates a functional request and sends it to the functional pool.

When sensors detect that conveyors reach the diverting position, the controller loads the service divert. Then, it creates a functional request and communicates with the functional pool to verify the functional rules. After that, the controller sends diverting event to the scheduler pool which will add it to the queue of aperiodic events and execute it at the first activation of the polling server. When the target point is achieved, the system should resume its ordinary work.

 ISBN: 978-1-61208-505-0

D. R2TCA Performance

The contribution of R2TCA is that it improves the reconfiguration process and ameliorates the adaptation of system behavior. R2TCA helps systems to shorten the execution time and so to hasten the response thanks to the functional pool. In fact, this pool helps to avoid the time lost in starting the execution of a wrong process. By checking the coherence rules and tasks priorities, developers are confident in the correctness of tasks execution. In addition, the dependencies table plays an important role in minimizing the conflicts, avoiding deadlocks and getting rid of the problem of priorities inversion. We have performed a comparison between two simulation scenarios: with and without the functional pool. The results are shown in Fig. 18.

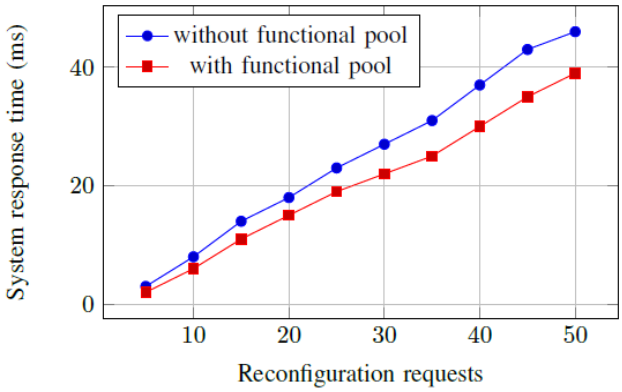

Figure 18. The system response time.

Fig. 18 shows that the red curve (with functional pool) has smaller values than the blue curve (without functional pool). This means that the system response time is faster when using the functional pool. Also, it shows that, by the increasing number of reconfiguration requests, more time is saved compared to using only a few number of requests. Therefore, using the functional pool is an efficient way to save time.

Comparing R2TCA with other related work is beneficial in terms of underlying the differences and highlighting the offered advantages. To this end, we compared R2TCA with CAMPUS [10] development tool. CAMPUS was proposed with the aim of automating context-aware adaptation decisions at run-time. We choose to compare our tool with CAMPUS since its logic of the control process is similar to the logic of R2TCA.

In order to execute an adaptation process when using CAMPUS, if a service is composed of nine tasks and if these tasks are composed of two tasklets, the developers have to prepare 2*9= 18 adaptation rules. The number of these rules will increase with increasing number of services in the system and also with increasing number of taskelts composing each service. In addition, the performed rules have the risk to be tied to a specific application and so it will be less flexible. Instead, by using R2TCA the same number of adaptation rules will be applied in all cases. The functional pool of R2TCA offers the possibility to check six main rules by means of the six tables (Priority, Precedence, Resources Allocation, Resources De-allocation, Inclusion Matrix, and Exclusion Matrix). Table 2 shows a clear view of the gain in terms of adaptation rules provided by R2TCA.

TABLE II. NUMBER OF ADAPTATION RULES OF R2TCA Vs CAMPUS.

	Tasks number	Tasklets number	CAMPUS adaptation rules	R2TCA adaptation rules
Service1	1	6	1*6=6	6
Service2	2	3	2*3=6	6
Service3	8	2	2*8=16	6
Service4	6	3	6*3=18	6
Service5	4	7	4*7=28	6
Service6	3	9	3*9=27	6
Service7	5	3	5*3=15	6
Service8	7	2	7*2=14	6

According to Table 2, CAMPUS and R2TCA have the same number of adaptation rules only in the simple cases where the service has only one task composed of six tasklets or in the case where the service has two tasks composed of three tasklets. Otherwise, R2TCA provides a lower number of functional rules which makes it easier for developers to develop the adaptation process. No doubt, every adaptation rule will use system resources (such as the processor time, the memory, the energy). The number of these resources depends on the complexity of the rule and the controlled entities.

Fig. 19 contains an approximate calculation of the system response time when using R2TCA and CAMPUS.

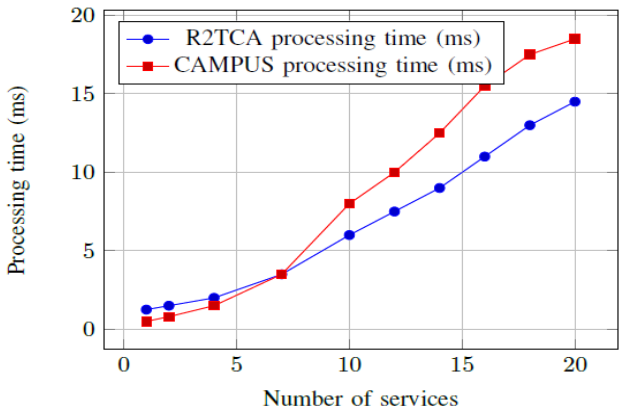

Figure 19. The processing time.

Fig. 19 shows that the processing time of the services is almost the same when using R2TCA and CAMPUS for eight services or less. But when the number of services is higher, R2TCA has a lower processing time compared with CAMPUS.

After that, we wanted to make sure that by adding the extra control pools (functional and scheduler) the system will not be lead to a point of congestion or bottleneck. Processor utilization as well as energy resources have to be absolutely sufficient. No doubt, the memory consumption at run-time is of great importance and it represents a key factor to get a correct reconfiguration process. That is why we have calculated the memory utilization rate when there is an increasing number of reconfiguration requests.

Fig. 20 shows that by increasing the number of reconfiguration requests (receiving 60 requests simultaneously), the system uses 74% of the available memory. This value is considered acceptable since it does not cause the system to be very loaded.

 ISBN: 978-1-61208-505-0

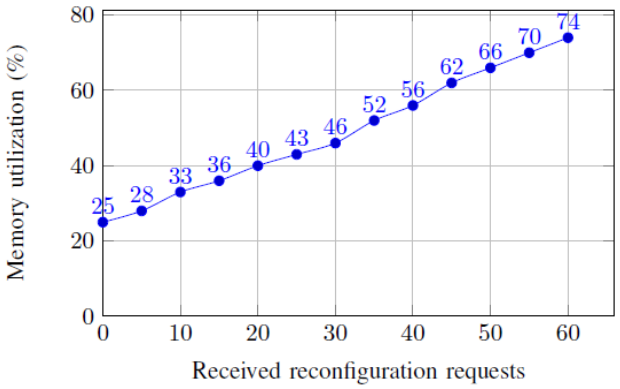

Figure 20. The memory utilization.

V. Conclusion

In this paper, we have presented the new tool R2TCA, a context-aware framework devoted to the development of control applications of adaptive systems. This new tool provides developers with the opportunity to respect both functional and real-time constraints. On one hand, the functional pool helps developers to check the inclusion and exclusion rules as well as to handle the resources to be allocated, de-allocated and shared. On the other hand, the scheduler pool guarantees the feasibility execution of services by offering various types of scheduling protocols. The robustness of R2TCA was proved by implementing our case study (the baggage handling system). R2TCA helps improve the response time of the system. Not only that, but also it is an interesting tool maximizing the resources utilization. In further works, R2TCA can be extended in order to include other control components. We plan to empower our framework by adding artificial intelligence so that we enable the prediction of context. Also, we consider to add a component responsible for quality of service issues.

References

[1] G. D.Abowd, et al. "Towards a better understanding of context and context-awareness." International Symposium on Handheld and Ubiquitous Computing. Springer Berlin Heidelberg, pp. 304-307, 1999.

[2] D. Salber, A. K. Dey, and G. D. Abowd, "Ubiquitous computing: Defining an hci research agenda for an emerging interaction paradigm.", GVU Technical Report;GIT-GVU-98-01, 1998.

[3] X. Li, M. Eckert, J.-F. Martinez, and G. Rubio, Context aware middleware architectures: Survey and challenges, Sensors, vol. 15, no. 8, pp. 20 57020 607, 2015.

[4] U. Alegre, J. C. Augusto, and T. Clark, Engineering context-aware systems and applications: a survey, Journal of Systems and Software, vol. 117, pp. 5583, 2016.

[5] S. Sukode, S. Gite, and H. Agrawal, Context aware framework in iot: A survey, International Journal of Advanced Trends in Computer Science and Engineering, vol. 4, no. 1, pp. 01-09, 2015.

[6] W. Lepuschitz, A. Zoitl, M. Vallee, and M. Merdan, Toward selfreconfiguration of manufacturing systems using automation agents, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, vol. 41, no. 1, pp. 5269, 2011.

[7] A. Forkan, I. Khalil, and Z. Tari, Cocamaal: A cloud-oriented contextaware middleware in ambient assisted living, Future Generation Computer Systems, vol. 35, pp. 114127, 2014.

[8] A. Forkan, I. Khalil, A. Ibaida, and Z. Tari, Bdcam: Big data for context-aware monitoring-a personalized knowledge discovery framework for assisted healthcare, 2015.

[9] H. Mcheick, H. Sbeity, H. Hazimeh, J. Naim, and M. Alameh, Context aware mobile application architecture (camaa) for health care systems, in Humanitarian Technology Conference-(IHTC), 2014 IEEE Canada International, pp. 15, 2014.

[10] E. J. Wei and A. T. Chan, Campus: A middleware for automated context-aware adaptation decision making at run time, Pervasive and Mobile Computing, vol. 9, no. 1, pp. 3556, 2013.

[11] L. Tang, Z. Yu, H. Wang, X. Zhou, and Z. Duan, Methodology and tools for pervasive application development, International Journal of Distributed Sensor Networks, vol. 10 no. 4 516432, 2014.

[12] B. Bertran, et. al Diasuite: A tool suite to develop sense/compute/control applications, Science of Computer Programming, vol. 79, pp. 3951, 2014.

[13] S. Liu and L. Cheng, A context-aware reflective middleware framework for distributed real-time and embedded systems, Journal of Systems And Software, vol. 84, no. 2, pp. 205218, 2011.

[14] E. Papadopoulou, S. Gallacher, N. K. Taylor, and M. H. Williams, A personal smart space approach to realising ambient ecologies, Pervasive and Mobile Computing, vol. 8, no. 4, pp. 485499, 2012.

[15] J. Lehoczky, L. Sha, and Y. Ding, The rate monotonic scheduling algorithm: Exact characterization and average case behavior, in Real Time Systems Symposium, 1989., Proceedings. IEEE, pp. 166 171, 198.

[16] N. C. Audsley, A. Burns, and A. J. Wellings, Deadline monotonic scheduling theory and application, in Control Engineering Practice, vol. 1, no. 1, pp. 7178, 1993.

[17] M. Spuri and G. C. Buttazzo, Efficient aperiodic service under earliest deadline scheduling, in Real-Time Systems Symposium, 1994., Proceedings. IEEE, pp. 211, 1994.

[18] J. Hildebrandt, F. Golatowski, and D. Timmermann, Scheduling coprocessor for enhanced least-laxity-first scheduling in hard real-time systems, in Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on. IEEE, pp. 208215, 1999.

[19] B. Sprunt, L. Sha, and J. Lehoczky, Aperiodic task scheduling for hard-real-time systems, Real-Time Systems, vol. 1, no. 1, pp. 2760, 1989.

[20] L. Sha, R. Rajkumar, and J. P. Lehoczky, Priority inheritance protocols: An approach to real-time synchronization, Computers, IEEE Transactions on, vol. 39, no. 9, pp. 11751185, 1990.

[21] http://www.aut.uni-saarland.de/mitarbeiter/frey/publications/

 ISBN: 978-1-61208-505-0