New Adaptive Middleware for Real-Time Embedded Operating Systems

Fethi Jarray
Laboratoire CEDRIC, CNAM,
292 rue St-Martin 75141,
Paris cedex 03, France
Email: fethi_jarray@yahoo.fr

Hamza Chniter
INSAT Institute
University of Carthago, Tunisia
FST faculty-University of Elmanar, Tunisia
Email: hamza.chniter@yahoo.fr

Mohamed Khalgui
INSAT Institute
University of Carthago, Tunisia
Email: khalgui.mohamed@gmail.com

***Abstract*—The paper presents a middleware implemented in Java, RT-MED, which corresponds to a software layer to be placed above the operating system. This software component is designed to execute and evaluate the performance, reliability and correctness of some real-time scheduling approaches which are theoretically validated. It describes a transition from the theory to the actual implementation of the proposed solutions. These solutions are based on a combinatorial optimization approach to solve the problem of feasibility in a system which is dynamically reconfigurable. RT-MED also presents a patch between the system and its environment under different constraints such as time and energy. It offers a set of adjustable parameters to control the flow of the execution. The middleware can be integrated into many operating systems and provides good quality both in terms of execution time and energy consumption. The implementation of this tool is based on java technology with embedded and real-time systems supported by the Real-Time Specification for Java. We have used a UML profile to describe various states and run-time reconfiguration of the embedded system. Results show that the middleware can effectively maintain the control and the stability of the system.**

I. Introduction

An embedded system (ES) is a device which is dedicated for a specific functions. It includes hardware and software parts which are designed to operate without human intervention. Modern embedded systems are often based on microcontrollers and integrated into more complex systems to perform time-sensitive and specific applications. There are usually linked with real-time constraints that determine their reliability and accuracy [1]. These systems are designed to control many devices to meet a wide range of user needs. The correctness of a real-time embedded system depends not only on giving the correct result, but also upon providing this result on the desired time. The first challenge of them is to interact continuously with their environment under various functional and extra-functional (e.g. temporal) constraints [2]. These ES must be able to react and solve temporarily any disturbance due to a run-time reconfiguration of the system. A reconfiguration is any operation allowing the addition-removal-update of OS tasks in order to adapt the system to its environment under well-defined constraints [3]. Several studies and solutions based on DVS (Dynamic Voltage Scaling) technology [4] have been performed in this context. This technology aims to dynamically change the processor speed during the execution of the system's tasks. The objective is to choose the suitable scaling factors for the processor speed to ensure the best compromise between the execution time and the energy consumption where all temporal constraints are respected. In this context, two combinatorial optimization solutions based on integer programming and heuristics have been developed and presented in [5] and [6]. The first exposes an integer programming model and an heuristic which tries to act to the processor speed during the execution of the OS tasks. The goal is to fix the optimal execution speed of each task to overcome the problem of feasibility after applying a run-time reconfiguration scenario to the system. The second, tries to develop an extended tasks model which is able to determine the favorable processor speed and it includes a mechanism based on deadlines adjustment. These mechanisms must be applied when all the available speeds of the processor are unable to satisfy the requirement and the system lies infeasible. Experimental results have shown that our approaches perform well in terms of maintaining the stability and correctness of the system and minimizing the energy consumption. These theoretical solutions give good results to solve the problem of real-time scheduling but this is still far from all that is practical. They present challenges for testing and require a complex environment. The goal now becomes increasingly oriented towards a practical and functional purpose of all these solutions which can prove practically, relevance and good operation of software which integrate these theoretical mechanisms. Seeing the solution which operate and respect all functional constraints in a real-time environment, will be essential to validate these theories and to judge their reliability and correctness.

A middleware is a software layer located above the operating system. It communicates with it and exploits its functionality to support the development of many reliable solutions. However, the architectures of most of the middleware solutions, do not offer the predictability required to support the real-time behavior in new complex systems, or the reconfigurability required for these middleware solutions to be integrated into various OS. RT-MED presents a middleware implemented in Java and describes the transition from the theory to the actual implementation. It presents a patch between the system and its environment under different constraints such as time and energy. It also offers a set of adjustable parameters to control the execution flow. The software can be integrated into different operating systems and provides good quality both in terms of execution time and energy consumption. The implementation of this middleware is based on java technology supported by the Real-Time Specification for Java (RTSJ). The

978-1-4799-8679-8/15/$31.00 copyright 2015 IEEE
ICIS 2015, June 28-July 1 2015, Las Vegas, USA

UML profile is used to describe various states, operational constraints and run-time reconfiguration.

The remainder of this paper is organized as follows. We discuss in section 2 the originality of the paper by studying the state of the art. Section 3 exposes a case study to explain the solutions of our developed approaches. In section 4, we present a proposition for modeling the system and the proposed solutions. Finally, an overview of the soft real-time application is presented exposing some graphical interfaces and operating parameters from RT-MED.

II. State of the art

The problem of real-time scheduling is classically to ensure the execution of all the tasks at run-time without missing the deadlines where the total energy consumption is minimized [1]. The use of maximum speed of the processor can accelerate the execution time of all tasks but it can produce significant energy consumption that exceeds the system capacity, hence the fact to vary the processor speed during execution becomes a need. A new technology known as DVS(Dynamic Voltage Scaling) [4] is integrated in the new processors for this purpose. Several studies have been performed in this context such as the work in [5], [6], [7], [8] to solve the scheduling problem in real-time systems. The real-time reconfigurable embedded systems are required to produce more flexible and adaptive solutions for a true real-time schedule under power constraints. However, development and design of high quality of scheduling middleware for real-time environment is difficult and complex, as it demands several requests such as system implementation, validation and optimization. The recent advance of middleware technologies, that enables communication and coordination in a computing system provides the perfect way to implement real-time middleware solutions. Several works have been performed in this context. ES such as avionics mission computing [9], unmanned flight control systems [10], and autonomous aerial surveillance [11] increasingly rely on real-time ORB middleware to meet challenging requirements such as communication and processing timeliness. Several middleware are emerging as fundamental building blocks for these systems. ORB services such as the TAO Real-Time Event Service [12] offer higher-level services for managing functional and real-time properties of interactions between application components. In [13], authors have developed SAFRAN middleware to enable applications to self adaptation to their contexts. It provides reactive adaptation policy for components using an aspect-oriented approach. Moreover, this middleware does not address timely configuration of components across the elastic resources of cloud computing. Gridkit [14] is a middleware framework that supports reconfigurability of applications dependent upon the condition of the environment and the functionality of registered components. In [15], authors presented PolyORB middleware that can support distribution with different personalities such as CORBA, RT-CORBA. The middleware proposed in [16] also implements real-time data distribution, and includes a specific architectural element which supports the adaptation to mobile networks. SALES [17] represents a middleware for data distribution which aimed at large-scale mobile systems. It does not take advantage of all real-time and QoS support. In [18], authors implemented an agent middleware application P-based networking. Additional benchmark applications have been implemented on a Java-based platforms to estimate the stabilities and real-time capabilities on real embedded systems. An architecture of quality of service middleware and corresponding algorithms was developed to support specified service in dynamic environments [19]. This middleware is able to monitor and adjust to a changing environment in distributed real-time and embedded systems. Work in [20] provides two contributions to the study of how autonomic configuration of pub/sub middleware can provision and use on-demand cloud resources effectively. Authors in [21] studied The deployment and configuration middleware that satisfy multiple QoS properties to meet their real-time and fault tolerance properties while consuming significantly less resources. They present the design of a middleware-agnostic configuration framework that uses allocation decisions to establish application components/replicas and configure automatically the underlying middleware. A new solution a middleware for enabling real-time interaction between nodes allowing time-bounded reconfiguration was presented in [22]. This middleware addresses these characteristics supporting timely reconfiguration in distributed real-time systems based on services. A data-centric distribution middleware that supports the development of predictable applications have been presented in [23]. In this context, heavy concepts defined in the Modeling and Analysis of Real-Time and Embedded systems has been integrated in order to allow using Model-Driven Engineering (MDE) and schedulability analysis techniques. Another work presents a Fleet Tracking and Management application built upon a developed middleware to show performance results for a large number of vehicles [24]. This work aims at developing context distribution and reasoning services for mobile collaboration in large scale systems with real-time monitoring, communication, and coordination of mobile nodes movements and activities. In [25], the authors proposed a tool for analyzing and simulating the safe behavior of critical hard real-time applications. It included a graphical user interface for controlling and displaying results. In [26], Jakovljevic presented a research about application of Java language to develop real-time middleware. The presented proposals require the use of a specific modeling language to define the attributes of tasks and their parameters. It exists a large number of works based on high abstraction levels and Model-Driven Engineering that use SysML or MARTE for modeling of real-time embedded systems. MARTE profile was used in the MoPCoM project [27] for modeling and code generation of reconfigurable embedded systems. The project uses the Rhapsody2 UML modeling tool and presents two distinct productions for the system modeling and schedulability analysis. In [28], the authors provide a mixed modeling approach based on SysML and the MARTE profiles to take into account from the design strategy. This approach provides just experimental results and no an actual implementation ones.

The cited works are interesting and important. However, none of the existing middleware architectures supports the concept of reconfigurable software, which is the key idea in our approach. The originality of our developed middleware is based on the fact that RT-MED addresses to the reconfigurable systems that can dynamically change their behaviors in real time. Indeed, this OS sub-layer can give not only snapshot and effective solutions but also optimal ones with operational research approaches. This middleware allows also to manage

reconfigurations that are unpredictable and depend on the reaction between user, system and environment. The proposed solutions in this middleware affect both the hardware part by acting in the processor speed during execution and the software part by changing the basic parameters of OS tasks such as deadlines and periods. In this paper, we try to expose the different steps from modeling to implementation to achieve a real-time middleware that presents a practical solution based on our approaches previously developed and presented in [5] and [6].

III. Case Study

Let we suppose a system which is composed of a set of tasks, each task is characterized by four parameters according to [1]. Firstly by its release time r_i, i.e each task T_i cannot begin execution before r_i. Secondly by its absolute deadline constraint d_i, i.e. each task should finish before d_i. Thirdly by its computation time at the normalized processor frequency C_{ni}. Finally by its period which is equal to the deadline. It is assumed that the WCET(Worst Case Execution Time) is constant and that the tasks will be executed on a single processor with a variable operating frequency according to the EDF scheduling policy. We denote respectively by f_n and V_n the normalized frequency and the voltage of the system. The actual execution time of the task is prolonged when the voltage is decreased to save the energy. The reason is that the frequency of the processor is approximately linearly proportional to the supply voltage [29]. We see that reducing voltage cuts down the energy dissipation, but the operating frequency will decrease accordingly. We can see that the task execution time is inversely proportional to the voltage. In order to provide the required system performance, the supply voltage should be scaled as low as possible to minimize the energy consumption, while guaranteeing the temporal properties. We suppose that each task T_i is executed at frequency F_i and at voltage V_i. We denote by η_i the reduction factor of voltage when T_i is executed, $V_i = \frac{V_n}{\eta_i}$. So the WCET is equal to $C_i = C_{ni}\eta_i$. The power consumption is $P = CV^2F$ where C is a constant related to the circuit type of the processor ensuring that P_i has the dimension of a power. Hence, if the system is running over x times, the energy consumption is $E = Px$. The problem is then to allow a low-power and real-time optimal scheduling of reconfigurable tasks after each reconfiguration scenario. We assume a simplified model of power, i.e. The power P_i consumed by the task T_i is :

$$P_i = C{V_i}^2F_i = C\frac{V_nf_n}{{\eta_i}^3}.$$

The energy E_i consumed by the task T_i is:
$E_i = P_iC_i = C\frac{V_nf_nC_{ni}}{{\eta_i}^2} = K\frac{C_{ni}}{{\eta_i}^2}$ with $K = CV_nF_n$.
So the total energy consumption of the system is:

$$E = \sum_{i=1}^{n} E_i = K\sum_{i=1}^{n}\frac{C_{ni}}{{\eta_i}^2} \quad (1)$$

The CPU utilization U is calculated by:

$$U = \sum_{i=1}^{n}\frac{C_i}{d_i}. \quad (2)$$

where n denotes the number of tasks in the system and C_i, d_i are respectively the execution time and the deadline of the task i. We assume a real-time embedded system to be composed of 5 tasks as depicted in Table I. The current system is

TABLE I: Current system configuration

Tasks	Release time	WCET	deadline	period
T_1	0	13	80	80
T_2	0	6	70	70
T_3	0	39	90	90
T_4	0	13	110	110
T_5	0	26	100	100

feasible since the CPU utilization is equal to 0.959. The energy consumption is equal to $2.112J = 2112mW$. We assume a reconfiguration scenario by adding 3 additional tasks (Table II). The new system becomes infeasible where some tasks miss their deadlines and the CPU utilization factor is equal to 1.327. The energy consumption is also increased and becomes $2,952J = 2952mW$. To overcome the problem of non

TABLE II: New system configuration

Tasks	Release time	WCET	deadline	period
T_1	0	13	80	80
T_2	0	6	70	70
T_3	0	39	90	90
T_4	0	13	110	110
T_5	0	26	100	100
T_6	0	10	85	85
T_7	0	11	94	94
T_8	0	14	105	105

feasibility, We propose in [6] to modify the processor speed by using integer programming model or simulated annealing to meet the corresponding deadlines and to optimize the energy consumption. Our model was applied to the previous system. It computes for each task, start time, finish time and the new WCET after changing the scaling factors of the processor speed. The results are presented in Table III.

TABLE III: Applied model for WCET reconfiguration

Tsks	Release time	Last WCET	New WCET	Scaling factor	Start time	Finish time	deadline
T_1	0.00	13.00	9.105	0.70	0.00	9.105	80
T_2	0.00	6.00	4.202	0.70	9.105	13.307	70
T_3	0.00	39.00	27.316	0.70	13.307	40.623	90
T_4	0.00	13.00	9.105	0.70	40.623	49.729	110
T_5	0.00	26.00	18.210	0.70	49.729	67.939	100
T_6	0.00	10.00	7.004	0.70	67.939	74.944	85
T_7	0.00	11.00	7.704	0.70	74.944	82.648	94
T_8	0.00	14.00	9.805	0.70	82.648	92.454	105

We apply to the same system, the model presented in [5]. The results are depicted in Table IV. Now we assume that a reconfiguration scenario has been applied to the system and the parameters of the tasks have been updated as depicted in Table V. We see that the maximum speed cannot overcome the problem of non feasibility since no available scaling factor can fulfill the needs. The system becomes infeasible because the factor of the CPU utilization is equal to $U = 1.2909 \geq 1$. The solution presented in [5] tries to adjust the deadlines of tasks as a flexible solution to meet the corresponding requested constraints with a minimum energy consumption. The computational results are shown in Table VI.

TABLE IV: Applied model for WCET reconfiguration

Tasks	WCET	New WCET	Start time	Finish time	Deadline	Scaling factor
T_1	13.00	10.40	0.00	10.40	80.00	0.80
T_2	6.00	4.80	10.40	15.20	70.00	0.80
T_3	39.00	31.20	36.20	67.40	90.00	0.80
T_4	13.00	10.40	99.60	110.00	110.00	0.80
T_5	26.00	20.80	15.20	36.00	100.00	0.80
T_6	10.00	10.00	67.40	77.40	85.00	1.00
T_7	11.00	11.00	77.40	88.40	94.00	1.00
T_8	14.00	11.20	88.40	99.60	105.00	0.80

TABLE V: New system configuration

Tasks	Release time	WCET	deadline	period
T_1	0	73	80	80
T_2	0	65	70	70
T_3	0	83	90	90
T_4	0	103	110	110
T_5	0	96	100	100
T_6	0	75	85	85
T_7	0	81	94	94
T_8	0	100	105	105

TABLE VI: Applied model for WCET reconfiguration and deadline adjustment

Tasks	WCET	Start time	Finish time	scaling factor	Last dead-line	New dead-line	New WCET
T_1	73.00	95.40	110.00	0.2	80.00	119.80	14.60
T_2	65.00	53.20	79.20	0.4	70.00	104.80	26.00
T_3	83.00	00.00	33.20	0.4	90.00	134.80	33.20
T_4	103.00	144.20	164.80	0.2	110.00	164.80	20.60
T_5	96.00	125.00	144.20	0.2	100.00	149.80	19.20
T_6	75.00	110.00	125.00	0.2	85.00	125.30	15.00
T_7	81.00	79.20	95.40	0.2	94.00	140.80	16.20
T_8	100.00	33.20	53.20	0.2	105.00	157.3	20.00

IV. UML-BASED DESIGN OF FEASIBLE RECONFIGURABLE SYSTEMS

A. Middleware Overview

This section provides an overview on RT-MED middleware that works with reconfigurable embedded systems. Fig. 1 exposes two parts: the first is a software part including the operating system and related applications, the second describes the environment with which the system interacts. This environment requires several constraints that the system must comply. Power consumption, user requests and hardware reactions represent the main sources of these constraints. Any reconfiguration in the system affects automatically these constraints. The system manages a database of tasks which are ready to be executed. It communicates with the application layer to ensure reliable execution. RT-MED takes the role to control and manage the reconfiguration that appear on the system. Through its modules, it handles the resolution of feasibility problems caused by the reconfiguration scenarios. This system sub layer uses operational research methods based on mathematical programming and heuristics presented in [5] and [6] to produce the optimal solution. This solution may guarantee the proper functioning of the system where ensuring temporal and environmental requirements. The middleware intervenes, for example, to change the speed of the processor when needed, adjust the basic parameters of the system, delete or add tasks etc.

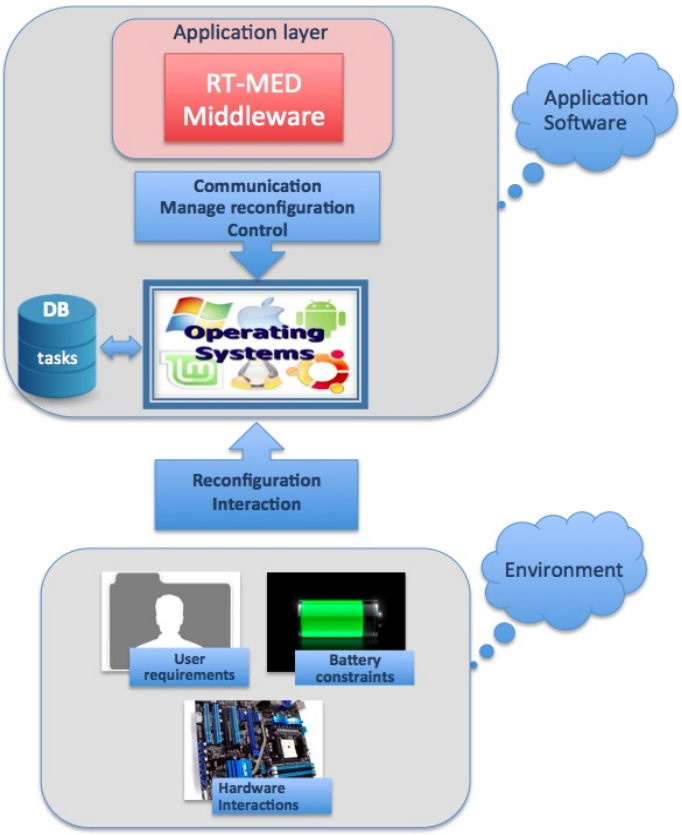

Fig. 1: Middleware overview

B. UML-based Modeling for Middleware Component

A set of classes and modules is defined to ensure the different agents which can act in the execution phase. The class "system" represents the operating system (Fig. 2), it handles the management and execution of a set of tasks which represent the system features. The system is in communication with its environment which requires multiple constraints such as the capacity of the battery power source, the hardware reactions and the intervention of the user. The class "tasks" represents a system task characterized by a set of parameters such as execution time, the deadline and periods. The system contains a software sublayer represented by the "middleware" class which forms the basis of our work. This middleware has a "listener" that takes the role of an event controller on the system, its objective is to detect any reconfiguration that appears. The class "reconfiguration scenario" presents the reconfiguration that the system can undergo. This reconfiguration is assumed to be an addition, delete or update of tasks in the system. The class "middleware" is composed by a set of modules and sub-modules. These modules represent the agents that will be called by the middleware when reconfigurations occur to retrieve the functional parameters and find solutions that stabilize the system. These modules are based on "Integer programming" and "heuristics" approaches. Some of important modules are presented in follow:

- *Change processor speed()*: tries to change the processor speed to ensure the execution of all tasks where minimizing the energy consumption,

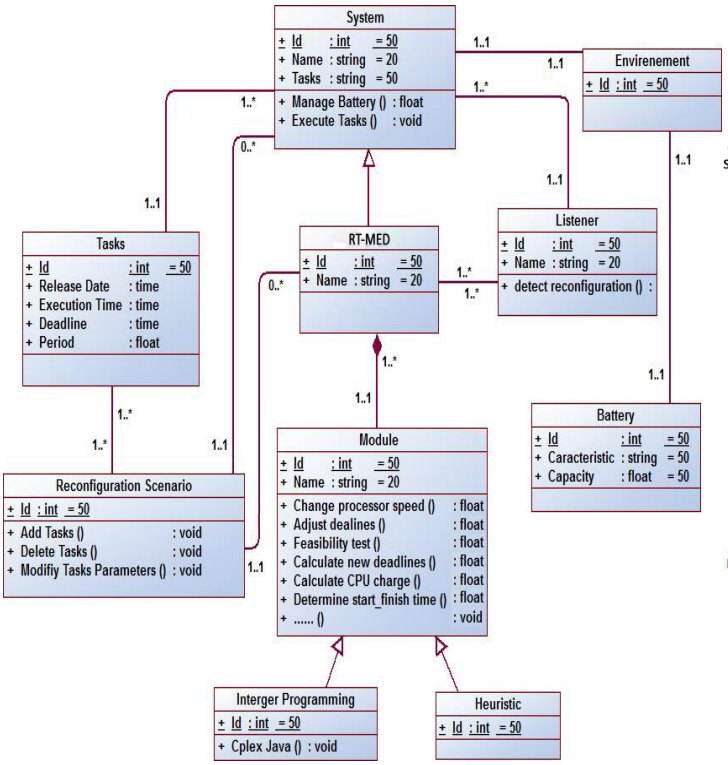

Fig. 2: Class diagram

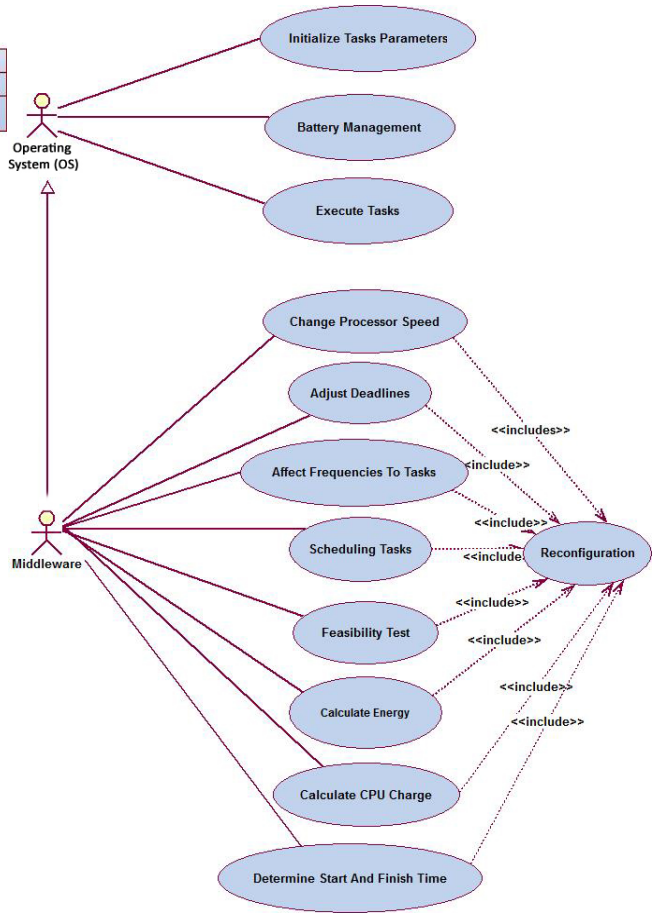

Fig. 3: Use cases diagram

- *Adjust deadlines()*: change the basic tasks parameters such as the deadlines in order to provide a new parallelism that guarantees the system feasibility,
- *Feasibility test()*: measures the feasibility of the system at run-time,
- *Calculate new deadlines()*: calculates new tasks parameters after modification,
- *Start_finish time()*: produces the execution sequence of the tasks (start and finish time) for each found solution,
- *Display results()*: takes care of the dynamic display which describes the flow of execution and found results,
- *CPLEX java()*: uses the modules CPLEX java which are responsible to execute mathematical programming models and recovery results.

The system deals with several features, such as the management of the energy consumption and the execution of the requested tasks to ensure proper operation and achieve the needs of the user in accordance with all environmental constraints. RT-MED middleware is integrated into an under layer of the system to intervene when any reconfiguration occurs. It can act in different circumstances to apply multiple actions such as changing the processor speed, calculating the new deadlines, evaluating the energy consumption and scheduling the system's tasks (Fig. 3). The sequence diagram describes a sequence of events and communications between the actors. Fig. 4 presents the sequencing of the interaction between the middleware on one hand and the system from the other. The main goal is to reach the required input and give the best solution that keeps the system in favorable functional conditions. The Middleware, through its agents, remains ready and listening to any reconfiguration that can be triggered in the system (Fig. 5). Once it detects the reconfiguration, it tests the feasibility of the system. If the system is feasible under the new configuration, it returns to its listening state, if not, it applies the first method which acts on the processor speed. The feasibility test is done dynamically during the solution search. Once the solution is found, the middleware provides the new functional parameters to be applied and returns to its listening state. If this method still unable to find the solution, the middleware calls the second method that propose the change tasks parameters such as the deadlines. The process restarts in the same way until the determination of the desired solution. The middleware continues waiting for new interactions to intervene.

V. RT-JAVA BASED IMPLEMENTATION OF FEASIBLE RECONFIGURABLE SYSTEMS

The implementation of this middleware is based on java technology with embedded and real-time systems. The Java programming language and its extensions for real-time systems is usually used for the development of predictable applications. The real-time Specification for Java (RTSJ) [30] is an open

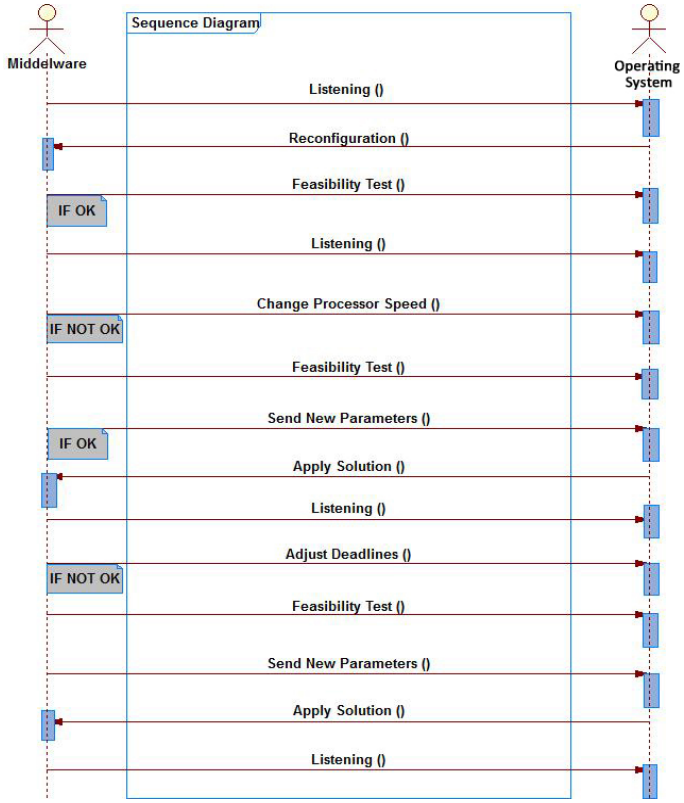

Fig. 4: Sequence diagram

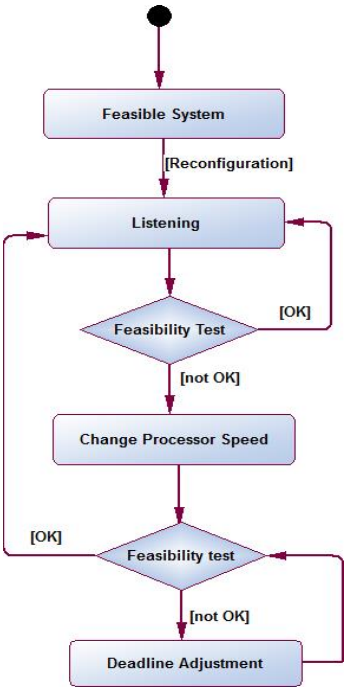

Fig. 5: Activity diagram

specification that augments the Java language to open the door more widely to using the language to build real-time systems. Implementing with the RTSJ requires support in the operating system such as JRE and the Java Class Library (JCL). It addresses several problematic including memory management, threading, synchronization and time. To implement the Integer Programming approaches, we have used CPLEX through Concert Technology for Java. It gives an overview of a typical application program, and describes procedures for creating, solving model and providing results. ILOG CPLEX Java provides an API for Java applications that use CPLEX. It allows the Java application to call CPLEX directly, through the JNI (Java Native Interface). The Java interface is built on top of ILOG Concert Technology for Java and supplies a rich functionality allowing the use of Java objects to build optimization model. It also provides functionality for solving Mathematical Programing (MP) problems and accessing informations about solution.The developed middleware is used to apply and exploit our proposed solutions for real-time scheduling problem. It implements a set of scheduling algorithms which aim to push the system to a feasible state when any disturbance occurs. It also allows the user to specify tasks structures and their timing requirements such as parameters and reconfiguration scenarios. The results (energy, cpu utilization,system state) can be dynamically visualized during the execution on this interface. The dynamic results indicate the passage of the system from one state to another describing its behavior against the reconfiguration scenarios.

The main interface is divided into three panels:

1) A panel on the left allows the configuration and the setup of the system. It contains several components,
   - A Button Parameters which displays a window containing the current system configuration, the chosen policy based implementation and the predefined available scaling factors of processor speed. The user can add, remove or modify heavy tasks,
   - A Button Configuration which is used to manually create a scenario of reconfiguration. The user must enter for each task, the release time, the WCET, the deadline and the period,
   - The Start and Stop Button for run and turn off of the system,
2) A panel at the top that says in text information about the current system and consists of the following information,
   - An area to display the current value of CPU utilization,
   - An Energy area to display the actual value of the power consumption,
   - An Energy area to display the number of rejected tasks after reconfiguration,
3) A panel on the right allowing to draw curves representing the following elements,
   - A graphic space to represent the curves,
   - A Button energy can draw dynamically the shape of the curve representing the current energy,
   - A Button utilization witch aims to draw the CPU utilization one.

When the middleware is running at first time, some files are generated. Some of them include the results to be displayed in the graphical interface. The others ones contain the current

system configuration. We start with a system composed of 5 tasks (Fig. 6). When the parameter are depicted and

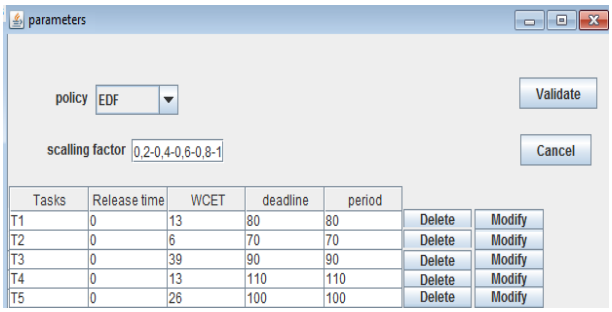

Fig. 6: Basic system

validated from the middleware, no problems were detected. The current system is feasible since the CPU utilization is equal to $0,959$ and the energy is equal to $2,112$ (Fig. 7). The

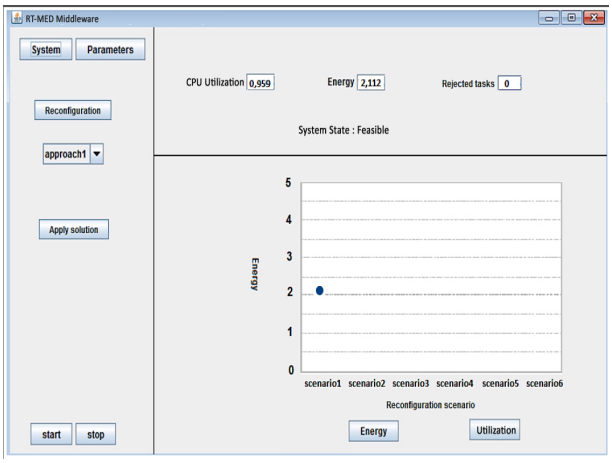

Fig. 7: Visualized Result

reconfiguration can be done manually on the setting interface or previously planned to be executed at run-time when the system is working. Now, we apply a reconfiguration scenario which consists in adding 3 new tasks to the system (Fig. 8). Before applying our solutions presented in [5] and [6], the

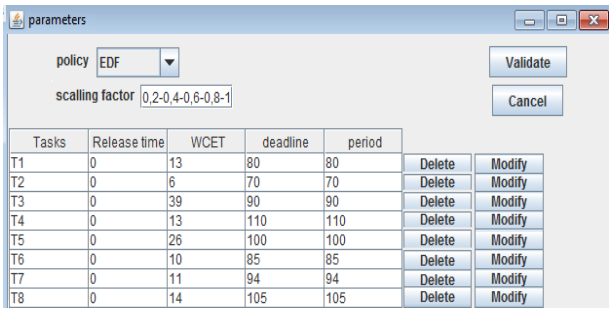

Fig. 8: First configuration scenario

middleware reports that the CPU utilization is equal to $1,327$ and the energy is $2,952$ which push to an infeasible state of the system (Fig. 9). By applying the developed approaches,

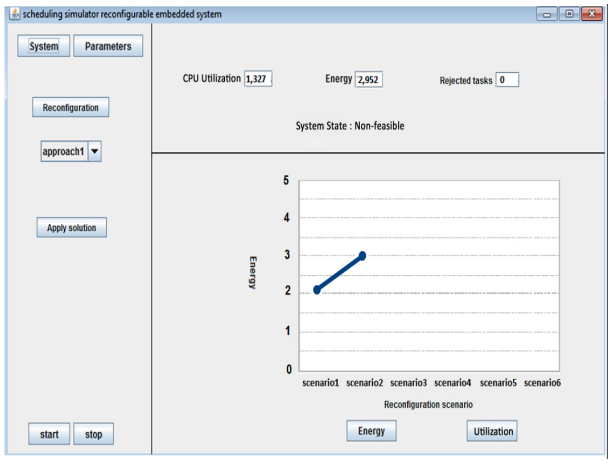

Fig. 9: Visualized Result

the problem is solved. The CPU utilization becomes $0,9808$ (Fig. 10). Now let's try to examine the behavior of the system

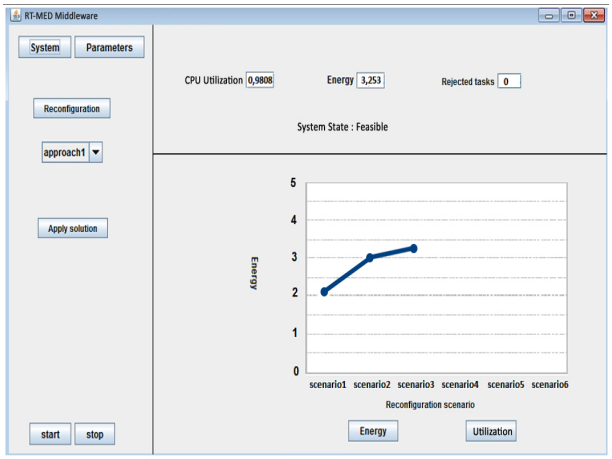

Fig. 10: Visualized Result

when another scenario is applied as shown in Fig. 11. If

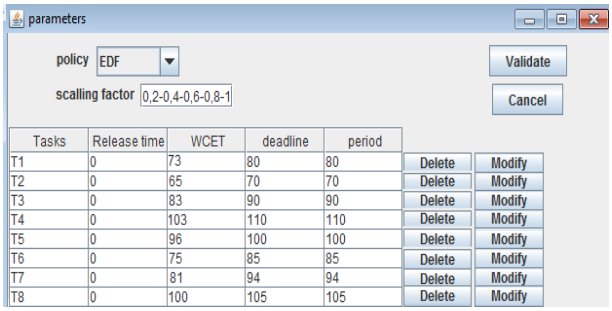

Fig. 11: Second configuration scenario

we apply the approach presented in [6], the system remains infeasible (Fig. 12). We go to apply the approach presented in [5] which is based on the deadline adjustment. The results are displayed in Fig. 13. To make tests with dynamic planned reconfigurations, a list of scenarios is programmed to be

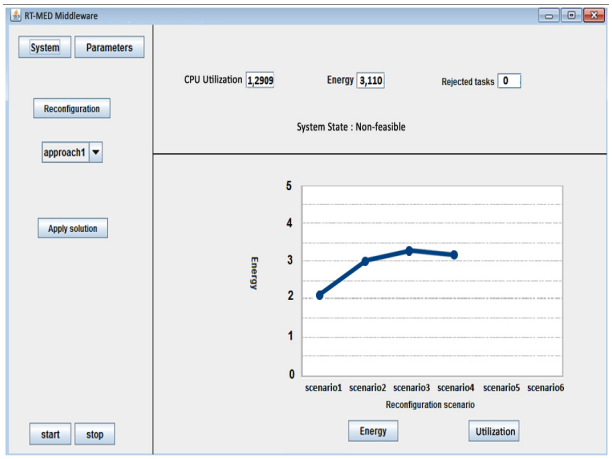

Fig. 12: Visualized Result

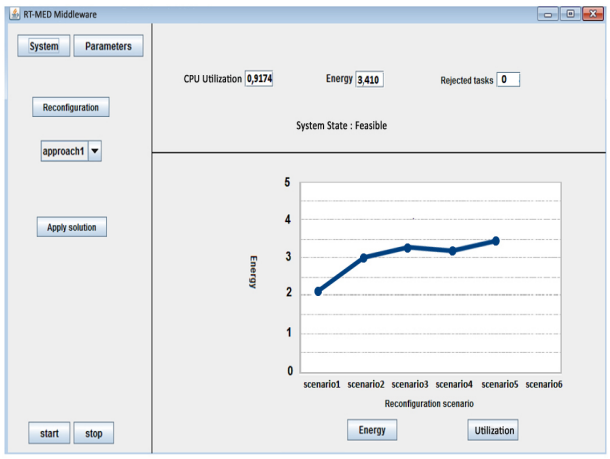

Fig. 13: Visualized Result

successively executed at run-time. Each scenario will have its own parameters and the date of starting time. Fig. 14 shows the reaction of the system before these reconfigurations and dynamically displays the results of each ones. Fig. 15 shows

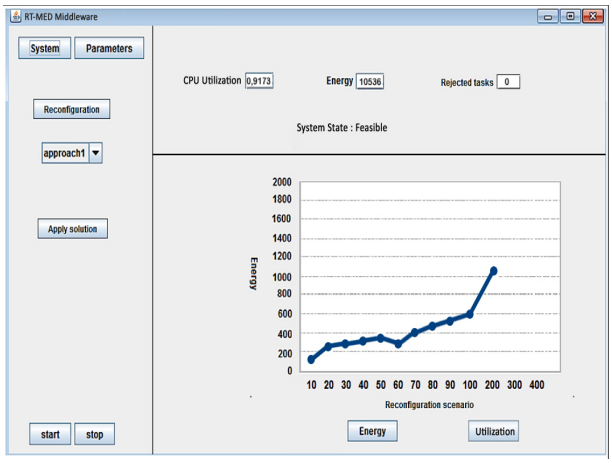

Fig. 14: Results of planned reconfigurations

the number of deadline adjusted after the execution of each of reconfiguration scenario. In Fig. 16, we show the resolution

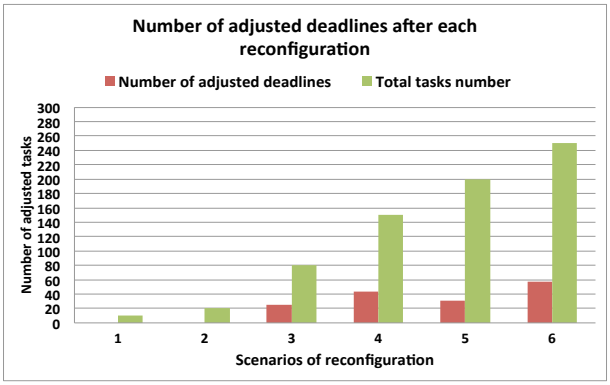

Fig. 15: Number of adjusted deadlines

time of the problem after each reconfiguration.

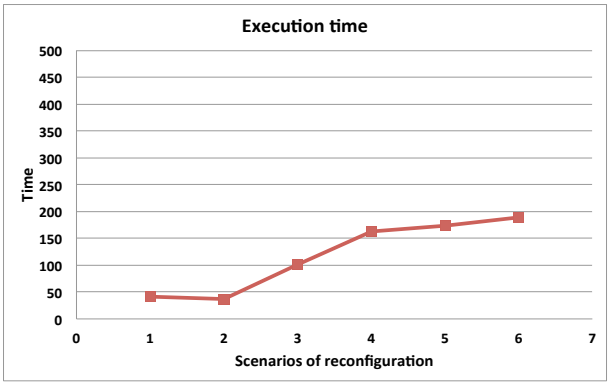

Fig. 16: Execution time

VI. Conclusion

Many solutions and scheduling algorithms have been proposed, however, few of them are implemented in real-time systems. To be used and evaluated, theoretical solutions must be deployed into a real-time system. Really Implementation is another alternative to evaluate them. Unfortunately, just few real-time scheduling middleware have been developed to date and most of them require the use of a specific simulation language. In this paper we have presented a real-time scheduling middleware which aims to implement and evaluate our developed approaches. RT-MED tries to produce optimal flexible and adaptive solutions for a true real-time schedule under power constraints. It includes a graphical interface to manage functional parameters and to view results which are dynamically displayed. It can be integrated into many operating systems. In Future, this middleware could be improved and enriched to schedule real-time systems constituted of periodic and aperiodic tasks with shared resources. Also, the support for multiple types of schedulers and multiprocessor architectures. It also will be tested with large size instances for more pertinence and reliability.

REFERENCES

[1] Liu, J.W.S. and Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM, 20(1):46-61, May 1973.

[2] GAUJAL, B., AND NAVET, N. Ordonnancement sous contrainte de temps et d'énergie. In ACs de l'éecole d'été temps réel (ETR03) (9-12 Septembre2003), pp. 263 UTF2013276.

[3] Imran Rafiq Quadri, Abdoulaye Gamatié, Pierre Boulet, Samy Meftali, Jean-Luc Dekeyser: Expressing embedded systems configurations at high abstraction levels with UML MARTE profile: Advantages, limitations and alternatives. Journal of Systems Architecture - Embedded Systems Design 58(5): 178-194 (2012).

[4] GRUIAN F., Energy-centric scheduling for real-time systems, PhD thesis, Lund Institute of Technology, 2002.

[5] Chniter H., Jarray F. and Khalgui M. (2014). Adaptive embedded systems: New composed technical solutions for feasible low-power and real-time flexible os tasks. 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO).

[6] Chniter H., Jarray F. and Khalgui M. (2014b). Combinatorial approaches for low-power and real-time adaptive reconfigurable embedded systems. International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS).

[7] Chantem, T., Hu, X. S., and Lemmon, M. D. (2009). Generalized elastic scheduling for real-time tasks. IEEE Trans, Computers 58(4):480-495.

[8] Dwivedi, S. P. (2012). Adaptive scheduling in real-time systems through period adjustment. CoRR abs/1212, (3502).

[9] D. Corman, WSOA weapon systems open architecture demonstration using emerging open system architecture standards to enable innovative techniques for time critical target prosecution, in: IEEE/AIAA Digital Avionics Systems Conference (DASC), October 2001.

[10] T.F. Abdelzaher, E.M. Atkins, K.G. Shin, QoS negotiation in real-time systems and its application to automated flight control, IEEE Transactions on Computers 49 (11) (2000).

[11] J. Loyall et al., Comparing and contrasting adaptive middleware support in wide-area and embedded distributed object applications, in: IEEE International Conference on Distributed Computing Systems, April 2001.

[12] R. Klefstad, D.C. Schmidt, C. ORyan, Towards highly configurable real-time object request brokers, in: the IEEE/IFIP International Symposium on Object- Oriented Real-time Distributed Computing (ISORC), March 2002.

[13] David, P.C., Ledoux, T.: Software Composition, chap. An aspect-oriented Approach for developing self-adaptive fractal components, pp. 8297. Springer LNCS, Berlin / Heidelberg (2006)

[14] Ostermann S., Prodan R. and Fahringer T. : Extending grids with cloud resource management for scientific computing. In: 10th IEEE/ACM International Conference on Grid Computing, 2009. pp. 42 49 (13-15 2009)

[15] T. Vergnaud, J. Hugues, L. Pautet, and F. Kordon. PolyORB: a schizophrenic middleware to build versatile reliable distributed applications. Proc.of the 9th International Conference on Reliable Software Technologies, Palma de Mallorca (Spain), in LNCS, Vol. 3063, June 2004.

[16] K.-J. Kwon, C.-B. Park, and H. Choi, A proxy-based approach for mobility support in the DDS system, 6th IEEE International Conference on Industrial Informatics, 2008. INDIN 2008, 2008.

[17] A. Corradi, M. Fanelli, and L. Foschini, Adaptive context data distribution with guaranteed quality for mobile environments, 2010 5th IEEE International Symposium on Wireless Pervasive Computing (ISWPC), pp. 8, 2010.

[18] Bridges, C.P. and Vladimirova, Tanya. Real-time agent middleware experiments on java-based processors towards distributed satellite systems, Aerospace Conference, 2011 IEEE,March 2011 : 1-10

[19] Joe Hoffert, Aniruddha S. Gokhale, Douglas C. Schmidt: Timely autonomic adaptation of publish/subscribe middleware in dynamic environments. IJARAS 2(4): 1-24 (2011)

[20] Joe Hoffert, Douglas C. Schmidt, Aniruddha S. Gokhale: Adapting Distributed Real-Time and Embedded Pub/Sub Middleware for Cloud Computing Environments. Middleware 2010: 21-41

[21] Jaiganesh Balasubramanian, Aniruddha S. Gokhale, Abhishek Dubey, Friedhelm Wolf, Chenyang Lu, Christopher D. Gill, Douglas C. Schmidt: Middleware for resource-aware deployment and configuration of fault-tolerant real-time systems. IEEE Real-Time and Embedded Technology and Applications Symposium 2010: 69-78

[22] Marisol Garca-Valls, Iago Rodrguez Lopez, Laura Fernndez-Villar: iLAND: An Enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems. IEEE Trans. Industrial Informatics 9(1): 228-236 (2013)

[23] Hctor Prez Tijero, J. Javier Gutirrez: On the schedulability of a data-centric real-time distribution middleware. Computer Standards & Interfaces 34(1): 203-211 (2012)

[24] Lincoln David, Rafael Vasconcelos, Lucas Alves, Rafael Andr, Gustavo Baptista, Markus Endler: A communication middleware for scalable real-time mobile collaboration. WETICE 2012: 54-59

[25] K. Lahmar, R. Cheour, M. Abid. Wireless sensor networks: trends, power consumption and simulators. Modelling Symposium, pp. 200-204, 2012.

[26] G. Jakovljevic, Z. Rakamaric, and D. Babic. Java simulator of real-time scheduling algorithms. In Proceedings of the 24th International Conference on Information Technology Interfaces, volume 1, pages 411- 416, Cavtat, Croatia, June 2002.

[27] A. Koudri et al, Using MARTE in the MOPCOM SoC/SoPC Comethodology, in MARTE Workshop at DATE08, 2008.

[28] H. Espinoza et al, "Challenges in combining sysML and MARTE for model-based design of embedded systems," in ECMDA-FA09. Springer-Verlag, 2009, pp. 98-113.

[29] Zhu, Y. (2005). Dynamic voltage scaling with feedback EDF scheduling for real-time embedded systems. Master's thesis, North Carolina State University.

[30] Michael H. Dawson, "Challenges in implementing the real- time specification for java (RTSJ) in a commercial real-time Java virtual machine," isorc, pp.241-247, 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC), 2008