A Real-time Scheduling of Reconfigurable OS Tasks with a Bottom-up SPL Design Approach

Hamza Gharsellaoui[1,2], Jihen Maazoun[3], Nadia Bouassida[3], Samir Ben Ahmed[1,4] and Hanene Ben-Abdallah[5]

[1]*LISI-INSAT Laboratory, Carthage University, Carthage, Tunisia*
[2]*Al-Jouf College of Technology, TVTC, Al Jouf, Kingdom of Saudi Arabia*
[3]*Mir@cl Laboratory, Sfax University, Sfax, Tunisia*
[4]*FMPNS, Tunis El Manar University, Tunis, Tunisia*
[5]*FCIT, King Abdulaziz University, Jeddah, Kingdom of Saudi Arabia*

Keywords: Real-time Scheduling, Reconfigurable Embedded Systems, SPL Design, UML Marte.

Abstract: Several real-time embedded system must be dynamically reconfigured to account for hardware/software faults and/or maintain acceptable performances. Depending on the run-time environment, some reconfigurations might be unfeasible,*i.e.*, they violate some real-time constraints of the system. In this paper, we deal with the development of dynamically reconfigurable embedded systems in terms of the production of execution schedules of system tasks (feasible configuration) under hard real-time constraints. More specifically, we propose an approach that starts from a set of reconfigurations to construct a Software Product Line that can be reused in a predictive and organized way to derive real-time embedded systems. To make sure that the SPL offers various feasible reconfigurations, we define an intelligent agent that automatically checks the system's feasibility after a reconfiguration scenario is applied on a multiprocessor embedded system. This agent dynamically determines precious technical solutions to define a new product whenever a reconfiguration is unfeasible. The set of products thus defined by the agent can then be unified into an SPL. The originality of our approach is its capacity to extract, from the unfeasible configurations of an embedded system, an SPL *design* enriched with real-time constraints and modeled with a UML Marte profile. The SPL design can assist in the comprehension, reconfiguration as well as evolution of the SPL in order to satisfy real-time requirements and to obtain a feasible system under normal and overload conditions.

1 INTRODUCTION

The ubiquity of portable computerized systems has diversified the application domains of embedded, real-time systems. This diversity is accompanied by an increasing number of new, complex deployed software whose development still faces several challenges. On the one hand, most available software development models do not take into account the specifics of embedded systems, *e.g.*, hard timing constraints, limited memory and power use, predefined hardware platform technology, and hardware costs. On the other hand, the new generations of this system type are imposing new criteria, *e.g.*, run-time flexibility and agility (Gharsellaoui et al., 2012). These challenges call for appropriate tools and methodologies for embedded software engineering and dynamically reconfigurable embedded control systems as an independent discipline. In this work, we aim to provide for the development of these systems through predictive and organized reuse. More specifically, we propose an approach to construct Software Product Lines (SPL) that can be reused for the development of dynamically reconfigurable, embedded, real-time systems.

In this paper, we examine the development of reconfigurable, embedded real-time systems in terms of scheduling their tasks so as to meet a set of timing constraints. As such, we consider an embedded system as a set of n real-time tasks that must be scheduled. Thus, the general development goal is to ensure that any reconfiguration scenario ψ_h that changes the system implementation does not violate its real-time constraints; that is, the system is feasible and meets its real-time constraints even if its implementation is changed after a reconfiguration scenario.

DOI: 10.5220/0005342501690176
In *Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2015), pages 169-176
ISBN: 978-989-758-100-7

To meet this development goal, we propose an approach that starts from a set of unfeasible reconfigurations to construct a Software Product Line that can be reused in a predictive and organized way to reconfigure a system to as to meet its timing constraints. To make sure that the SPL offers only various feasible reconfigurations, we define an intelligent agent that automatically checks the system's feasibility after a reconfiguration scenario is applied on a multiprocessor embedded system. This agent dynamically determines precious technical solutions to define a new product whenever a reconfiguration is unfeasible. The set of reconfigurations thus defined by the agent can then be unified into an SPL. The originality of our approach is its capacity to extract, from the unfeasible configurations of an embedded system, an SPL *design* enriched with real-time constraints and modeled with a UML Marte profile. The SPL design can assist in the comprehension, reconfiguration as well as evolution of the SPL in order to satisfy real-time requirements and to obtain a feasible system under normal and overload conditions.

The proposed approach adapts the bottom-up SPL construction method proposed in (Maazoun et al., 2013) for software products. In the original method, an SPL is constructed from variant products' source code; it models a family of product solutions that can used to derive new products in the SPL domain. The variability is represented through features (defined as an end user, visible system characteristic (Kang et al., 1990)) that are related with constrained relationships to guide their reuse. Compared to existing SPL methods (*cf.*, (Ziadi, 2004), (Ziadi et al., 2012), (She et al., 2011), (Al-Msie'Deen et al., 2012)), the method of (Maazoun et al., 2013) has the merit of producing all mandatory and variable features along with all the constraints (and, or, require, etc) relating them. In our adaptation of this method, the SPL construction starts from the tasks' variable parameters of the failed (unfeasible) system, which we consider as a feature. It produces an SPL representing a set of feasible reconfigurations. To model the SPL, we use extend the SPL-UML profile (Maazoun et al., 2013) with timing constraints as proposed in the UML Marte profile; we call the resulting profile SPL-UML-Marte.

The remainder of the paper is organized as follows. In Section 2, we overview work pertinent to scheduling with a focus on our optimal scheduling algorithm which is used for the reconfiguration construction. Section 3 presents the SPL-UML Marte as a profile for software product lines. Section 4 presents the SPL construction approach which produces a set of feasible reconfigurations from unfeasible ones. Section 5 summarizes the main results and outlines their intended future extensions.

2 RELATED WORK

As discussed in the introduction, our approach tackles the development of reconfigurable, embedded real-time systems in terms of task scheduling. In our approach, a reconfiguration scenario entails the construction of a new SPL starting from the current (sporadic/periodic) tasks' parameters. A sporadic task is described by a minimum inter-arrival time P_i^{p,ψ_h} which is assumed to be equal to its relative deadline D_i^{p,ψ_h}, and a worst-case execution time (WCET) C_i^{p,ψ_h} for each reconfiguration scenario ψ_h and on each processor p. Indeed, a hard real-time system typically has a mixture of off-line and on-line workloads and assumed to be feasible before any reconfiguration scenario ψ_h. According to (Brocal et al., 2011), a hyperperiod is defined as $HP = [\zeta, 2*LCM+\zeta]$, where LCM is the well-known Least Common Multiple of the tasks periods and ζ is the largest task offset. This algorithm, in our original paper assumes that sporadic tasks span no more than one hyperperiod of the periodic tasks for each reconfiguration scenario ψ_h on each processor p.

In this section, we first present works dealing with reconfigurations and real-time scheduling of embedded systems. Second, we illustrate the scheduling algorithm we adopted in the proposed SPL construction approach. In recent works, many real-time systems rely on the EDF scheduling algorithm in the case of uniprocessor scheduling theory. This algorithm has been shown to be optimal under many different conditions. For example, for independent, preemptive tasks, on a uni-processor, EDF is optimal in the sense that if any algorithm can find a schedule where all tasks meet their deadlines, then EDF can meet the deadlines (Dertouzos, 1974).

To address demands for increasing processor performance, silicon vendors no longer concentrate wholly on the miniaturization needed to increase processor clock speeds, as this approach has led to problems with both high power consumption and excessive heat dissipation. Instead, there is now an increasing trend towards using multiprocessor platforms for high-end real-time applications (Brocal et al., 2011).

For these reasons, we will use in our work the case of real-time scheduling on homogeneous multiprocessor platforms. Before presenting our original contribution, we will present some definitions below. According to (Gharsellaoui et al., 2012), each periodic task is described by an initial offset a_i (activation time), a worst-case execution time (WCET) C_i, a relative deadline D_i and a period T_i.

According to (Buttazzo and Stankovic, 1993), each sporadic task is described by minimum inter-arrival time P_i which is assumed to be equal to its relative deadline D_i, and a worst-case execution time (WCET) C_i. Hence, a sporadic task set will be denoted as follows: $Sys_2 = \{\sigma_i(C_i,\ D_i);\ \text{i = 1 to m}\}$. Reconfiguration policies in the current paper are classically distinguished into two strategies: static and dynamic reconfigurations. Static reconfigurations are applied off-line to modify the assumed system before any system cold start, whereas dynamic reconfigurations are dynamically applied at run-time, which can be further divided into two cases: manual reconfigurations applied by users and automatic reconfigurations applied by intelligent agents (Gharsellaoui et al., 2012). This paper focuses on the dynamic reconfigurations of assumed mixture of off-line and on-line workloads that should meet deadlines defined according to user requirements. The extension of the proposed algorithm should be straightforward when this assumption does not hold.

Running Example
In this section, we illustrate the performance of our proposed approach for both periodic synchronous and asynchronous tasks, and sporadic tasks. The simulation was conducted on our tool RT-SPL-Reconfig and proven by the real-time simulator Cheddar (Legrand and Singhoff, 2004) with a task set composed of old tasks (ξ_{old}) and new tasks (ξ_{new}^{p,ψ_h}) on the processor p for each reconfiguration scenario ψ_h. We illustrate with a simplified example to ease the understanding of our approach. The task set considered for this example is given in table 1 and is composed of 10 tasks. The sum of utilization of all tasks is equal to 426.1%. We have 3 identical processors in our system to schedule these tasks. In this case, we assume that each task's deadline is less than or equal to its period. The worst case execution times, deadlines, and the time periods of all tasks are generated randomly. This algorithm, in our original paper assumes that sporadic tasks span no more than one hyperperiod of the periodic tasks $HP^{(p,\psi_h)}$ = $[\zeta^{(p,\psi_h)}$, 2*LCM + $\zeta^{(p,\psi_h)}]$, where LCM^{p,ψ_h} is the well-known Least Common Multiple of tasks periods and (ζ^{p,ψ_h}) is the largest task offset of all tasks τ_k^{p,ψ_h} for each reconfiguration scenario ψ_h on each processor p. Also, in this experiment, our task set example is initially implemented by 5 characterized old tasks ($\xi_{old} = \{\tau_1; \tau_2; \tau_3; \tau_4; \tau_5\}$). These tasks are feasible because the processor utilization factor U = 1.19 $\leq$ 3. These tasks should meet all required deadlines defined in user requirements and we have $Feasibility(Current_{\xi_{old}}(t)) \equiv True$.

Firstly, tasks are partitioned; task τ_1 is partitioned on first processor, τ_2 and τ_3 are partitioned on processor 2 while task τ_4 and τ_5 are partitioned on processor 3. We have three sets of local tasks. As there is only one task on first processor then task τ_1 utilization factor is the same as the first processor 1 utilization factor ($u^{1,0}$ = 0.285 $\leq$ 1) while utilization factors of processor 2 and processor 3 are calculated as follows:

$$U^{2,0} = \sum_{i=1}^{(2)^2} \frac{C_i^2}{T_i^2} = 0.372 < 1,$$

$$U^{3,0} = \sum_{i=1}^{(2)^3} \frac{C_i^3}{T_i^3} = 0.533 < 1,$$

We suppose that a first reconfiguration scenario ψ_1 (h = 1) is applied at time t_1 to add 5 new tasks $\xi_{new}^{\psi_1} = \{\tau_6; \tau_7; \tau_8; \tau_9; \tau_{10}\}$. The new processor utilization becomes U^{ψ_1} = 4.261 $>$ 3 time units. Therefore the system is unfeasible. $Feasibility(Current_{\xi}^{\psi_1}(t1)) \equiv False$. Indeed, if the number of tasks increases, then the overload of the system increases too. Our optimal earliest deadline

Table 1: Task Parameters.

Task	C_i	D_i	$T_i = P_i^*$
τ_1	2	9	7
τ_2	3	21	20
τ_3	2	9	9
τ_4	2	13	10
τ_5	3	15	9
τ_6	14	21	19
τ_7	10	24	16
τ_8	8	18	18
τ_9	13	16	17
τ_{10}	5	11	12

first (OEDF) algorithm is based on the following Guarantee Algorithm which is presented by Buttazzo and Stankovic in (Buttazzo and Stankovic, 1993). Indeed, OEDF algorithm is an extended and improved version of Guarantee Algorithm that usually guarantees the system's feasibility.

3 SPL-UML MARTE: A PROFILE FOR REAL-TIME SYSTEMS SPL

To model the constructed SPL for a real-time embedded system, we propose the UML Marte profile named SPL-UML Marte.

3.1 Extensions for Class Diagrams

In the context of SPL, the intelligent agent (IA) of our SPL construction method proposes to introduce different types of variability that are modeled using the following stereotypes:

- <<optional>> specifies optionality of a class, package, attribute or operation.
- <<recommended>> denotes recommendation of classes only.
- <<mandatory>> indicates obligatory classes, packages, attributes or operations.
- <<mandatory_association>> specifies obligatory relation between classes. It is represented by a bold line.
- <<optional_association>> specifies optional relation between classes. It is represented by dashed line.
- <<Xor_association>> indicates an alternative relation between classes.
- <<Feature_Name>> specifies in which feature the tagged element (class, package, attribute or operation) is present.

3.2 Extensions for Sequence Diagrams

In our profile, the UML Marte sequence diagram is extended with the following stereotypes:

- <<optional>> denotes optional objects, actors or operations.
- <<mandatory>> specifies obligatory objects, actors or operations.
- <<mandatory_relation>> marks obligatory relations between objects, actors. It is represented by bold line.
- <<optional_relation>> specifies optionality between objects, actors. It is represented by dashed line.
- <<Xor_relation>> indicates an alternative relation between objects, actors (ifelse). It is represented by solid line.
- <<Feature_Name>> specifies in which feature the tagged object/actor is present.

4 SPL DESIGN CONSTRUCTION FOR REAL-TIME RECONFIGURABLE SCHEDULING

Besides the SPL-UML Marte profile definition, our second contribution consists in the adaptation of the SPL design process of (Maazoun et al., 2013)through the addition of an intelligent agent (IA). This latter derives from a set of unfeasible tasks' parameters a feasible configuration that is considered as one product of the SPL. In other words, our SPL factorizes the commonalities and variability among the harmonized task set parameters; it represents a family of feasible reconfigurations. That is, any reconfiguration that is derivable from the SPL (while respecting the SPL constraints) can be used as a new feasible reconfiguration of the failed system.

In this section, we will first present the extraction of the feature model and design corresponding to the SPL for real-time reconfigurable scheduling. Second, we illustrate the approach through an example we constructed with our tool SPL-design tool for SPL extraction.

4.1 Feature Model and Design Extraction

Our bottom-up process extracts, from the task set parameters of multiprocessors embedded systems, the SPL design enriched with real-time constraints extracted from the feature model. Similar to the original process (Maazoun et al., 2013), our process operates in the following four steps:

1. **Name Harmonization.** This pre-processing step identifies the semantic correspondences among the names of packages, classes, methods and attributes names through interrogating WordNet. The semantic relations are examined in the following order: the equivalence (Synonyms), the string extension (str_extension), and then the generalization (Hypernyms)(Maazoun et al., 2013). At the end of this step, all semantically related names would be harmonized and can then be analyzed through the FCA in the features identification step.
2. **Reverse Engineering.** In the second step, the approach reverse engineers the code to construct the class and sequence diagrams required in the feature extraction step of our process. A class diagram contains all classes and enumerates the relationships between them (association, inheritance,

composition, aggregation). A sequence diagram contains Lifelines, message, operation, object...

3. **Feature Model Extraction.** In this step, the approach uses FCA and LSI to extract the commonalities and variability among the harmonized task set parameters. In our approach, the data represents the task set parameters being analyzed and recalculated by the IA in order to handle real-time requirements; the data description is represented through a table where the processors constitute the rows while tasks and their states (packages, classes, methods, attributes) constitute the columns of the table. From this table, a concept lattice is derived. The concept lattice first allows the definition of commonalities and variations among all processors: The top element of the lattice indicates that certain objects have elements in common (i.e., common elements), while the bottom element of the lattice show that certain attributes fit common objects (variations). Common blocks and blocks of variation are composed of atomic blocks of variation representing only one feature. Besides the blocks, the lattice also indicates the relationships among elements. The Mandatory, Optional, Xor, Require and AND relationships can be automatically derived from the sparse representation of the lattice and presented to the analyst.

   In our work, we suppose that the task set parameters, recalculated by the IA, are implemented by different developers. Consequently, the processors may have different structures. For example, a class in one processor can be replaced in a second processor with two classes where the attributes and methods of the original class are distributed. Moreover, since all the processors belong to the same embedded system, there are semantic relationships among the words used in the names. To define the similarity, we apply LSI. It allows to measure the similarity degree between names for packages, classes, methods and attributes. Consequently, the approach uses LSI and FCA to identify features based on the textual similarity. LSI uses each line in the block of variations as a query to retrieve all lines similar to it, according to a cosine similarity that equals to 0.70 (Binkley and Lawrie, 2011). The result of LSI used as input parameters for the FCA to group the similar tasks based on the lexical similarity and on the feasibility criteria. Thus, any task that hasn't a similar task feasibility criteria will be ignored. This process permits to identify atomic blocks of variations (*Feature*).

   The next step in our process determines the hierarchy and constraints among features and finalizes the feature model construction. This phase has two-fold motivations. First, the features which are composed of many elements (tasks) (package, classes, attributes, methods) are renamed based on the frequency of the names of tasks. In addition, the organization and structure of the features are also retrieved based on the semantic criteria. In fact, to retrieve the organization of the features, we use the semantic criterion:

   - *Hypernyms(FeatureN1, FeatureN2)* $\longrightarrow$ *FeatureN1 is the parent of FeatureN2*
   - *Str_extention(FeatureN1, FeatureN2)* $\longrightarrow$ *FeatureN1 OR FeatureN2*
   - *Synonyms(FeatureN1, FeatureN2)* $\longrightarrow$ *FeatureN1 XOR FeatureN2*

   In fact, Hypernyms allow to construct the hierarchy of feature models and Str_extention and Synonyms permit to determine constraints. Finally, the constraints between the different features that are extracted with FCA and LSI are verified and some others are added based on the semantic criteria and on the real-time properties. At the end of this last step, all the features would be collected by the IA in a feature model to specify the variations between the task set parameters.

4. **Design Elements Extraction and SPL Design Extraction.** The organization and structure of the SPL design is retrieved based on the following construct rules:

   - **R1.** Each mandatory class will be presented with her mandatory elements (attribute, method).
   - **R2.** If a relationship is mandatory, then the association ends are mandatory and it will be present in the design.
   - **R3.** If a relationship has a startAssociation or an endAssociation mandatory, will be present in the design and the optional startAssociation or endAssociation will be present and stereotyped ”recommended”.
   - **R4.** The rest of the optional element will be present in the design according to the degree of its presence in all the class' diagrams.

   Finally, the IA proposes to represent the design of the SPL using our UML Marte profile enriched with the information and real-time constraints extracted from the feature model generated in order to re-obtain the feasibility of the embedded systems under study.

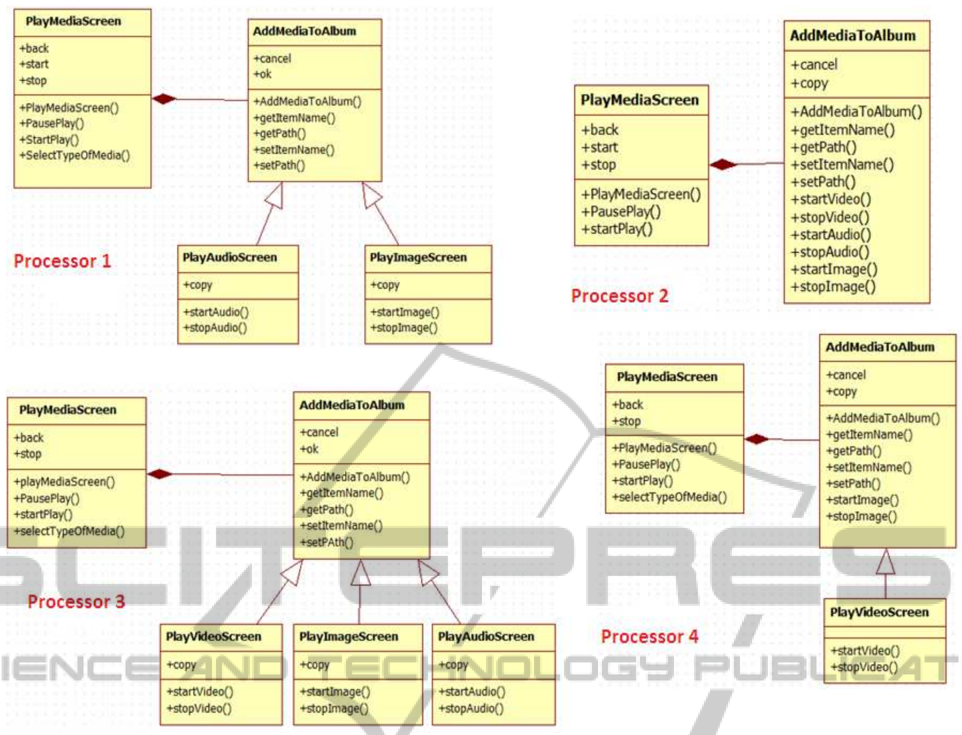
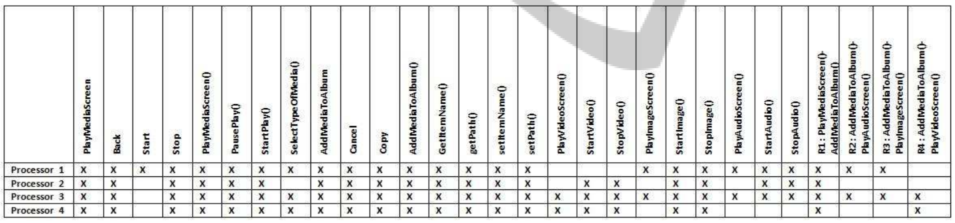


Figure 1: Class diagrams of four processors.



	PlayMediaScreen	Back	Start	Stop	PlayMediaScreen()	PausePlay()	StartPlay()	SelectTypeOfMedia()	AddMediaToAlbum	Cancel	Copy	AddMediaToAlbum()	GetItemName()	getPath()	setItemName()	setPath()	PlayVideoScreen()	StartVideo()	StopVideo()	PlayImageScreen()	StartImage()	StopImage()	PlayAudioScreen()	StartAudio()	StopAudio()	R1 : PlayMediaScreen()-AddMediaToAlbum()	R2 : AddMediaToAlbum()-PlayAudioScreen()	R3 : AddMediaToAlbum()-PlayImageScreen()	R4 : AddMediaToAlbum()-PlayVideoScreen()
Processor 1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X	X	X	X	X	X	X	X	X	
Processor 2	X	X		X	X	X	X		X	X	X	X	X	X	X	X		X	X		X	X		X	X	X			
Processor 3	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Processor 4	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X				X			X

Figure 2: Extract of the formal context describing mobile systems by class diagram elements.

4.2 Example

To illustrate our adaption of the SPL construction, let us consider a set of mobile phone software products. In the first step, the user chooses the source code file, then the design tree will be extracted and saved in an XML document. The XML document contains to the name of elements of the parsed code (packages, classes, methods, attributes). To extract the class and sequence diagrams, we reverse engineered the code. The class diagram of products after reverse engineering is shown in figure 1. In order to tolerate some differences among the class diagram, we apply the FCA to analyze elements of class diagrams. The data description is represented through a table where the processors constitute the rows, while the classes diagrams elements (packages, classes, methods, attributes) and relationship between classes constitute the columns of the table (see figure 2). After that,

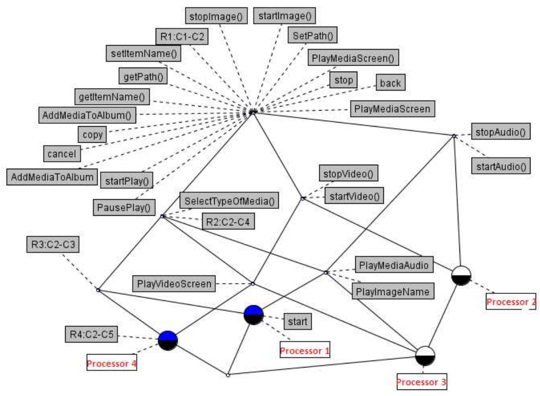


Figure 3: The lattice for the formal context.

a concept lattice is derived (see figure 3). This later defines the commonalities and the task set variations among all processors. The top elements of the lattice indicate that some objects have features in common.

These latter are mandatory in the class diagram, however, the others are optional. Applying FCA, LSI and semantic criteria, all atomic blocks are identified and collected in a feature model to specify the variations between these processors (Maazoun et al., 2013).
In the second step, we perform the SPL design construction. In fact, all mandatory classes will be present with their mandatory methods and attributes. In the mobile phone example, the class ”PlayMediaScreen” and its attributes (tasks) ”back” and ”stop” and methods ”PlayMediaScreen()”,”PausePlay”, ”startPlay” are mandatory. Then, all mandatory relationship will be presented like ”R1:C1-C2” (see figure 2). Every relationship having an associationEnd mandatory will be presented like ”R2:C2-C4” and ”R3:C2-C3”. Finally, the class and sequence diagrams corresponding to the SPL is derived (see figure 4).

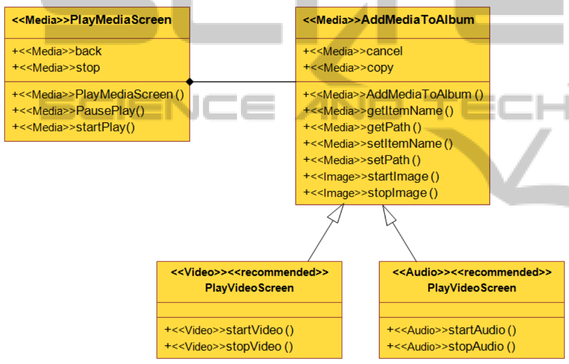


Figure 4: The class diagram of the SPL represented with SPL-UML Marte.

Now, the IA applies these output results as input parameters to the embedded system under study and consequently the performance of the real-time scheduling becomes more efficient. Indeed, the intelligent agent should define the different solutions generated by the SPL feature model as input parameters. In this case, the best solution that satisfies functional requirements and gives the minimum utilization factor of the system was chosen. This solution ameliorates also the response time and hence the chances of executing more new tasks are increased as well. These results were suggested by the tool RT-Reconfig and give a feasible system which is proven also by the real-time simulator Cheddar (Singhoff et al., 2004).

5 CONCLUSION

In this paper, we first reviewed existing works for reconfigurable homogeneous multiprocessor systems to be implemented by sporadic and periodic OS tasks that should meet real-time constraints. In this work, we propose to use an optimal scheduling algorithm based on the EDF principles and on the dynamic reconfiguration for the minimization of the response time of constrained deadline real-time tasks and proven it correct. Secondly, we proposed how a Software Product Line (SPL) design approach can be used to produce various feasible reconfigurations (based on EDF); these latter can reused when needed to dynamically reconfigure a failed system. The proposed process starts from the current tasks' parameters to identify a product line in a bottom-up approach. Our new original approach extracts from the unfeasible embedded system, the SPL design enriched with real-time constraints presented by the critical embedded systems. The enriched SPL is modeled with a UML Marte profile that assists in the comprehension, reconfiguration as well as evolution of the SPL in order to satisfy real-time requirements.

Finally and in our future works, we are examining how to take advantage of the SPL-UML Marte notation to propose a maintenance process for SPL with the Reconfigurable real-time embedded systems. We plan also in future works to study reconfigurations of real-time tasks of distributed embedded systems to define a new Software Product Lines satisfying real-time requirements.

REFERENCES

Al-Msie'Deen, R., Seriai, A., Huchard, M., Urtado, C., Vauttier, S., and Salman, H. (2012). An approach to recover feature models from object-oriented source code. In *Day Product Line 2012*.

Binkley, D. and Lawrie, D. (2011). Information retrieval applications in software maintenance and evolution. *In Encyclopedia of Software Engineering*, pages 454–43.

Brocal, V., Balbastre, P., Ballester, R., and Ripoll, L. (2011). *Task period selection to minimize hyperperiod, Emerging Technologies & Factory Automation (ETFA), IEEE conference on, pp.1-4, 2011.* doi: 10.1109/ETFA.2011.6059178, Toulouse, France, 16th edition.

Buttazzo, G. and Stankovic, J. (1993). *RED: Robust Earliest Deadline Scheduling*. Int. Workshop On Responsive Computing Systems, Austin, 3rd edition.

Dertouzos, M. (1974). *Control Robotics: The Procedural Control of Physical Processes*. Proceedings of the IFIP Congress.

Gharsellaoui, H., Khalgui, M., and Ahmed, S. B. (2012). *Feasible Automatic Reconfigurations of Real-Time OS Tasks*. IGI-Global Knowledge, USA, isbn13: 9781466602946 edition.

Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A. (1990). Feature-oriented domain analysis (foda) feasibility study,. *Technical report CMU/SEI-90-TR-21, Software Engineering Institute,Carnegie Mellon University,.*

Legrand, J. and Singhoff, L. (2004). *Cheddar : a Flexible Real Time Scheduling Framework*. ACM SIGAda Ada Letters, volume 24, number 4, pages 1-8. Edited by ACM Press, ISSN:1094-3641.

Maazoun, J., Bouassida, N., Ben-Abdallah, H., and Seriai, A. (2013). Feature model extraction from product source codes based on the semantic aspect. Reykjavik, Iceland. Proceedings of the 8th International Joint Conference on Software Technologies.

She, S., Lotufo, R., Berger, T., Wesowski, A., and Czarnecki, K. (2011). Reverse engineering feature models. pages 461–470.

Singhoff, F., Legrand, J., Nana, L., and Marce., L. (2004). Cheddar : a flexible real time scheduling framework. *ACM SIGAda Ada Letters, volume 24, ACM Press, New York, USA. December 2004, ISSN:1094-3641. F. Singhoff and A. Plantec. What is Cheddar, at AADL scheduling home site.*

Ziadi, T. (Decembre, 2004). Manipulation de lignes de produits en uml. *These de doctorat, Universite de Rennes 1*.

Ziadi, T., Frias, L., da Silva, M. A. A., and Ziane, M. (2012). Feature identification from the source code of product variants. pages 417–422.