Scalable Computing: Practice and Experience
Volume 10, Number 3, pp. 277–289. http://www.scpe.org
ISSN 1895-1767
© 2009 SCPE

TOWARDS TASK DYNAMIC RECONFIGURATION OVER ASYMMETRIC COMPUTING PLATFORMS FOR UAVS SURVEILLANCE SYSTEMS

ALÉCIO P. D. BINOTTO*, EDISON P. DE FREITAS†, MARCO A. WEHRMEISTER‡, CARLOS E. PEREIRA§, ANDRÉ STORK¶, AND TONY LARSSON‖

Abstract. High-performance platforms are required by modern applications that make use of massive calculations. Actually, low-cost and high-performance specific hardware (e.g. GPU) can be a good alternative along with CPUs, which turned to multiple cores, forming powerful heterogeneous desktop execution platforms. Therefore, self-adaptive computing is a promising paradigm as it can provide flexibility to explore different computing resources, on which heterogeneous cluster can be created to improve performance on different execution scenarios. One approach is to explore run-time tasks migration among node's hardware towards an optimal system load-balancing aiming at performance gains. This way, time requirements and its crosscutting behavior play an important role for task (re)allocation decisions. This paper presents a self-rescheduling task strategy that makes use of aspect-oriented paradigms to address non-functional application timing constraints from earlier design phases. A case study exploring Radar Image Processing tasks is presented to demonstrate the proposed approach. Simulations results for this case study are provided in the context of a surveillance system based on Unmanned Aerial Vehicles (UAVs).

Key words: reconfigurable computing, dynamic scheduling, aspect-oriented paradigm, unmanned aerial vehicles

1. Introduction.

In addition to timing constraints, modern applications usually require high performance platforms to deal with distinct algorithms and massive calculations. The development of low-cost powerful and application specific hardware (e.g., GPU—Graphics Processing Unit, the Cell processor, PPU—Physics Processing Unit, DSP—Digital Signal Processor, PCICC—PCI Cryptographic Co-processor, FPGA—Field Programmable Gate Array, among others) offers several alternatives for execution platforms and application implementation, aiming at better performance, programmability and data control. The resulting heterogeneity in the execution platform can be considered as an asymmetric multi-core cluster. This cluster's processing power is intensified with the new generation of multi-core CPUs, being a challenge to program applications that use efficiently all available resources and Processing Units (PU).

In this sense, low-cost hybrid hardware architectures are becoming attractive to compose adaptable execution platforms. Thus software applications must benefit from that powerfulness. This leads to the creation of new strategies to distribute applications' workload (tasks, algorithms, or even full applications that must run concurrently) to execute in asymmetric PUs in order to better meet application's requirements, such as performance and timeliness, without loosing flexibility. Dynamic and reconfigurable load-balancing computing (by means of task allocation reconfiguration, i. e., rescheduling) is a potential paradigm for those scenarios, providing flexibility, improving efficiency, and offering simplicity to program an (balanced) application on heterogeneous and multi-core architectures. Fig. 1.1 shows such a theoretical scenario of a desktop-based platform composed of several devices.

An important step towards the usage of the above mentioned hybrid platforms is to create a real-time workload self-rescheduling framework to balance the resource usage by applications composed of different algorithms (graphics, massive mathematical calculations, sensor data processing, artificial intelligence, cryptography, etc.), executing on top of such hybrid platforms under time constraints, in order to achieve a minimal Quality of Service. In addition, it has to be predicted that during execution time, new tasks can arise and influence the whole system. In this manner, such framework must keep monitoring the tasks' performance to provide online information for a possible new allocation balance, indicating that task rescheduling may be necessary to promote a better performance for the overall current scenario.

In this paper, the focus is on the very first step in the reconfiguration framework: application requirements handling (rescheduling) in a high-level design phase. The approach is based on application requirements, like

*Fraunhofer IGD/TU Darmstadt, Darmstadt, Germany / Informatics Institute, Federal University of Rio Grande do Sul, Porto Alegre, Brazil (alecio.binotto@igd.fraunhofer.de).
†School of Information Science, Computer and Electrical Engineering, Halmstad University, Sweden / Informatics Institute, Federal University of Rio Grande do Sul, Porto Alegre, Brazil (edison.pignaton@hh.se).
‡Department of Automation and Systems Engineering, Federal University of Santa Catarina, Florianópolis, Brazil (marcow@das.ufsc.br).
§Electrical Engineering Department, Federal University of Rio Grande do Sul, Porto Alegre, Brazil (cpereira@ece.ufrgs.br).
¶Fraunhofer IGD/TU Darmstadt, Darmstadt, Germany (andre.stork@igd.fraunhofer.de).
‖School of Information Science, Computer and Electrical Engineering, Halmstad University, Sweden (tony.larsson@hh.se).

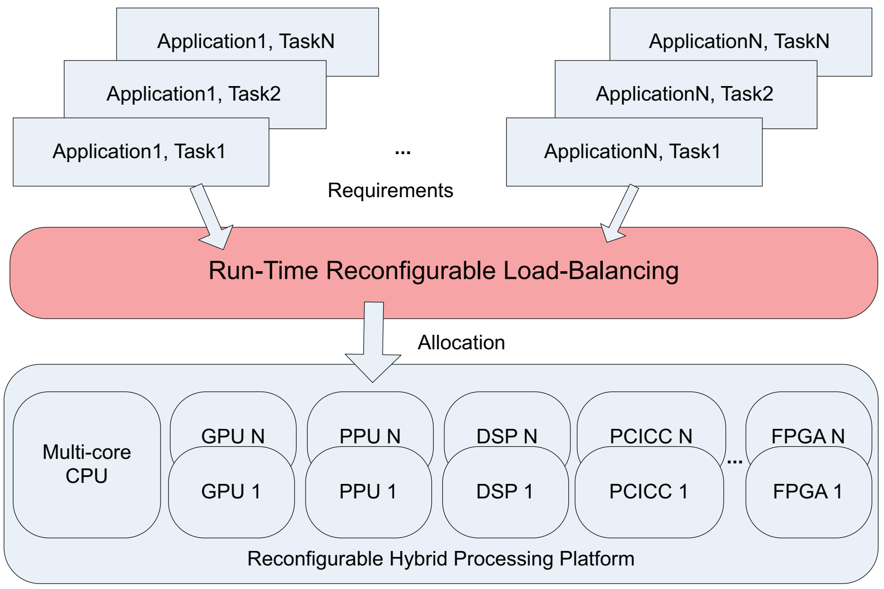

Fig. 1.1. *System overview.*

task/application deadlines, in order to find the current allocation balance that minimizes the whole application(s) execution time. This information is used by the framework to balance the computation over the execution platform. For its accomplishment, crosscutting concerns related to real-time non-functional requirements are taken into account. Handling these concerns by specific design elements called "aspects" (from aspect-oriented paradigm [11]) plays an important role for understandability and maintainability of the system design, as those concerns may influence different parts of the system in different ways. Then, based on the support offered by the aspects to monitor and control the resource usage (profiling), a strategy to assign tasks dynamically is presented, which is submitted to a run-time rescheduling when it is needed.

The paper is organized as follows. Section 2 starts with a previous work on aspects and requirements identification, modeled using UML. Section 3 follows with the dynamic workload strategy implemented by the created aspects. Composing these two concepts, Section 4 outlines an UAV surveillance system as case study, focusing on RIP (Radar Image Processing) tasks, which are dynamically created at run-time. Finalizing, related work, conclusions and future directions are exposed.

2. Handling Timing Concerns Using Aspects. In order to achieve dynamic rescheduling to improve tasks load-balancing, we investigate the use of aspect-oriented paradigms to cope with the modern system's crosscutting concerns, which are usually related to Non-Functional Requirements (NFR). Such requirements must be effectively handled already from requirements analysis to implementation phases to enhance system understandability during design. The context addressed by this work is similar to the one presented in the design of Distributed Real-time Embedded (DRE) systems, i. e. performance and timing NFR are very important during all application development phases. In this sense, we have adopted the taxonomy published in [6].

Traditional approaches, such as object-orientation, do not provide adequate means to deal with NFR handling. It occurs due to the inefficient modularization for NFR handling elements (timing requirements probes, serialization mechanisms, task migration mechanisms, among others), i. e. they are not modularized in a single or few system elements, but spread allover the system. Any change in one of these elements requires changes in different parts of the system, leading to a tedious and error-prone task that does not scale in the development of large and complex applications. The observation of these drawbacks motivates the use of an aspect-oriented approach, which makes possible to address such concerns in a modularized way. It separates the handling of the non-functional concerns in specific elements, increasing the system modularity, diminishing the coupling among elements, and though affecting positively the system maintainability, reuse and evolution [19]. Moreover specifically, the advantages of an aspect-oriented approach became clearer when applied to the task allocation strategies using heterogeneous platforms due to the need of profiling each task in different hardware, affecting several elements of the application. The use of aspects to address this concern represents an improvement since it helps to cope with the complexity in managing this concern spread through the whole system.

The next subsection presents a brief description of the NFR taxonomy presented in [6]. Following, a brief description of some aspects from an aspect framework, called DERAF [7], are presented to demonstrate how to deal with time-related NFR. Afterwards, an extension to DERAF, in terms of new aspects to deal with dynamic reconfigurable load-balancing, are is presented.

2.1. Non-functional Requirements. DRE systems domain presents a large set of NFR. Depending on the application domain, some requirements are more important than others. The same can be said about NFR handling: some of NFR are mandatory handled, while others are not. In this sense, Fig. 2.1 shows NFR taxonomy presented in [6], which focus on some of these very important requirements of DRE systems domain.

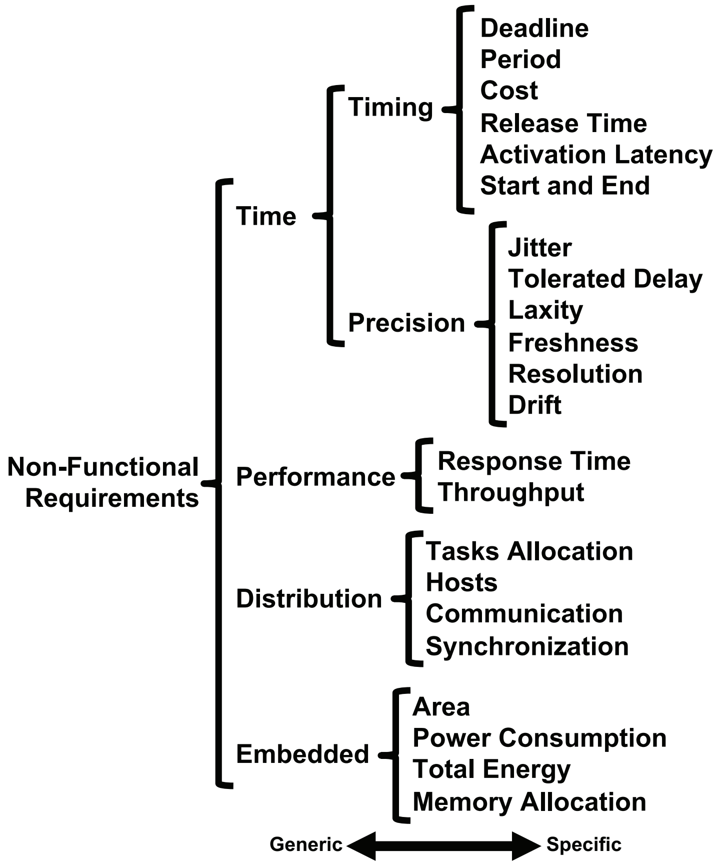

FIG. 2.1. *NFR requirements for DRE systems.*

The real-time concerns are captured by the requirements within the *Time* classification, which is divided in *Timing* and *Precision* requirements. The former presents time-related characteristics of system's tasks, activities, and/or action, e.g. deadlines or periodic executions. The later denotes constraints that affect the temporal behavior of the system in a "fine-grained" way, determining whether a system has hard or soft time constraints. An example is the *Freshness* requirement, which denotes the time interval within which a value of a sampled data is considered updated. Another key requirement is the *Jitter*, which directly affects system predictability since large variance in timing characteristics affects system determinism.

Performance requirements are not only tightly related to those presented in the *Time* classification, but also to those concentrated in the *Distribution* classification. They usually represent requirements employed to express a global need of performance, like the end-to-end response time for a certain activity performed by the system, or the required throughput rate in term of sending/receiving messages.

Distribution classification presents requirements related to the distribution of DRE system's activities, which usually execute concurrently. For instance, these concerns address problems such as task allocation over different PUs, as well as the synchronization and communication needs and constraints. Concerns related to embedded systems generally present requirements related to memory usage, energy consumption, and required hardware area size. *Embedded* classification gathers these concerns.

In this paper, the interest is to provide a runtime reconfigurable solution aiming at meeting time-related requirements. All mechanisms related to tasks migration among different PUs are non-functional crosscutting concerns, which are tightly related to system reconfiguration. In this sense, tasks migration is not only an expected final behavior of any system, but it also affects several functional elements in different ways and in different parts of the system.

2.2. Time-related Aspects. In order to address the mentioned *Time* and *Precision* requirements, this work (re)uses aspects from the *Distributed Embedded Real-time Aspects Framework* (DERAF) [7]. DERAF's Timing and Precision packages are presented in Fig. 2.2. A short description of each aspect is provided in the following paragraphs. Interested readers are pointed also to [7] for more details about DERAF.

TimingAttributes: adds timing attributes to active objects (e.g. deadline, priority, WCET, start/end time, among others), and also the corresponding behavior to initialize these attributes.

PeriodicTiming: controls execution of active objects by means of a periodic activation mechanism. This improvement requires the addition of an attribute representing the activation period and a way to control the execution frequency according to this period.

SchedulingSupport: inserts a scheduling mechanism to control the execution of active objects. Additionally, this aspect handles the inclusion of active objects into the scheduling list, as well as the execution of the feasibility test to verify if the active objects list is schedulable.

TimeBoundedActivity: limits the execution of an activity in terms of a deadline for finishing this activity, i. e. it adds a mechanism to restrict the maximum execution time for an activity, e.g. it limits the time which a shared resource can be locked by an active task. Jitter: measures the variance of activities' timing characteristics by means of measuring their start/end time, and calculating the variation of these metrics. If the tolerated variance was overran, corrective actions can be performed.

ToleratedDelay: restricts maximum latency for the beginning of an activity execution, e.g. limits the maximal duration in which a task can wait to acquire a lock on a shared resource.

DataFreshness: controls system data's expiration by means of associating timestamps to them, and also by verifying data validity before using them. Every time controlled data are written, their associated timestamps must be updated. Similarly, before reading these data, their timestamps must be checked [2].

ClockDrift: controls deviation of clock references in different PUs. It measures the time at which an activity starts, comparing it with the expected time for the beginning of this activity; it also checks if the accumulated difference among successive checks exceeds the maximum tolerated clock drift. If this is the case, some corrective action is performed.

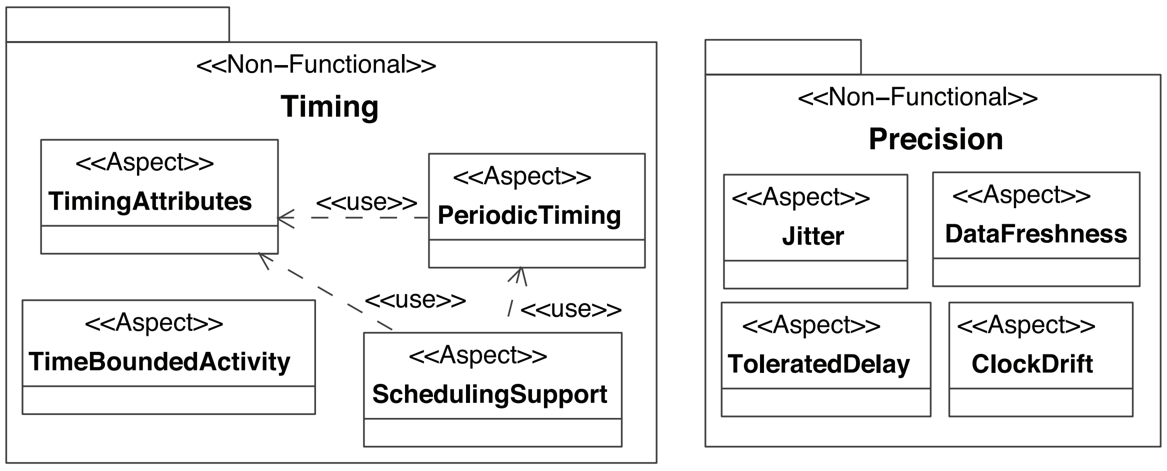

FIG. 2.2. *Timing and Precision packages from DERAF.*

2.3. Aspects to Support Tasks Self-Rescheduling. As mentioned before, task migration support characterizes a non-functional crosscutting concern in dynamic tasks rescheduling, spreading its handling mechanisms over several system's elements in a non-standard way. Therefore, we propose to use aspects to deal with this concern, and hence, two new aspects have been incorporated in DERAF: $TimingVerifier$, $TaskAllocationSolver$.

TimingVerifier and *TaskAllocationSolver* aspects use time parameters inserted by Timing package's aspects, and also services provided by aspects from the Precision package. To keep DERAF logical organization, both aspects have been included in an additional package, named TaskAllocation package, is included. Fig. 2.3 depicts the rescheduling-related package.

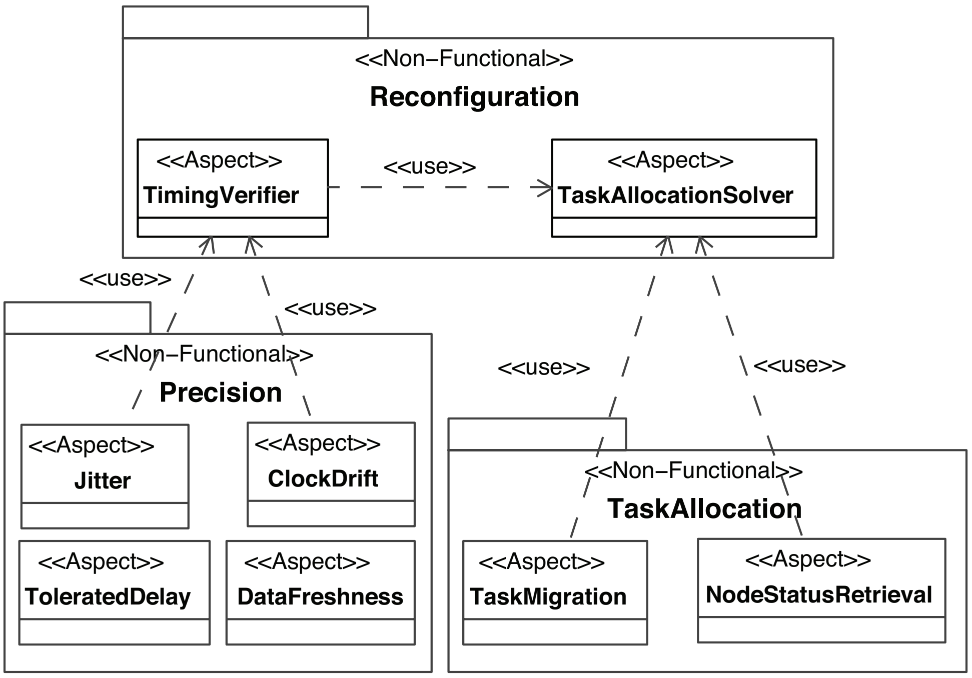

Fig. 2.3. *Aspects for Reconfiguration included in DERAF.*

TimingVerifier aspect is responsible for checking if PUs are being able to fulfill with timing requirements specified by *TimingAttributes*, *PeriodicTiming*, *ToleratedDelay* and *TimeBoundedActivity* aspects. In addition, *TimingVerifier* uses services provided by *Jitter* and *ClockDrift*.

To perform this checking, a mechanism, which controls if timing attributes are being respected, is inserted in the beginning and in the end of each task. More specifically, this mechanism consists in measuring the current time, comparing it with requirements specified by the correspondent attributes. For example, tasks deadlines' accomplishment can be checked by measuring the time in which a task actually finishes its computation, comparing this value with the time in which this task was supposed to finish. *TimingVerifier* uses the service of the *Jitter* aspects to gather information about the jitter related to analyzed requirement (in the mentioned example, the task's deadline). Moreover, considering the deadline again as example, *TimingVerifier* checks if the non-accomplishment of a task's deadline is constant, or if it varies in different executions or in the changing the platform scenario. In this sense, *TimingVerifier* can be used as base information, for instance, to know if the interaction among task is the responsible for the variance in tasks execution time.

ClockDrift aspect is used by *TimingVerifier* to gather information about synchronization among the different PUs, which is used, in addition to the time spent for task migration between PUs, to calculate the overall migration cost. To illustrate this idea, let's consider a task that has been migrated from a PU "A" to a PU "B", which is faster than PU "A" and potentially more capable of executing this task. The difference in the clock reference between these PUs could lead to an additional delay for this task's outcome (coming from PU "B") that would not be worth in comparison with letting the task to run in the PU "A".

The second key aspect for tasks dynamic rescheduling is the *TaskAllocationSolver*. It is responsible for deciding if a task will be migrated or not, and also for selecting to which PU this task is migrated. For that, *TaskAllocationSolver* checks the overload status of all destination PUs and the time spend for task migration, in order to decide if it is worthwhile to perform the migration. Hence, *TaskAllocationSolver* uses the measurements provided by *TimingVerifier* aspect. Based on these data, the reasoning about task reconfiguration feasibility is performed, as explained in the next section.

The reconfiguration itself and the retrieval of PUs status are performed by two other aspects from DERAF: *TaskMigration* and *NodeStatusRetrieval*. This way, reasoning and execution of tasks reconfiguration are decoupled, allowing that changes performed by one aspect do not affect the other one. A brief summary of the *TaskMigration* and *NodeStatusRetrieval* aspects is provided in the following.

TaskMigration: provides a mechanism to migrate active objects (tasks) from one PU to another one. It was originally used by aspects that control embedded concerns and, in the present work, is extended to provide services needed by *TaskAllocationSolver* aspect.

NodeStatusRetrieval: inserts a mechanism to retrieve information about processing load, send/receive messages rate, and/or the PU availability (i. e. "I'm alive" message). Before/after every execution of affected active objects (tasks), the processing load is calculated. Before/after every sent/received message, the message rate is computed. Additionally, PU availability message is sent at every "n" message or periodically with an interval of "n" time units. All of these information are taken into account by *TaskAllocationSolver* during the tasks migration decision process.

3. Dynamic Tasks Self-Scheduling.

A task-based approach is then used, in which each task is designed to be an independent algorithm. They are grouped according derivation of the same high-level class, simplifying the managing of possible dependencies. Besides, it is coherent to assume that a group of tasks will have similar characteristics and hence would be desirable to execute in the same PU. However, this can lead to a non-optimal execution performance and must also be considered in the dynamic strategy discussed in the next sub-sections.

3.1. First Assignment of Tasks.

For the first assignment of tasks, we do not use the modeled aspects, since tasks' real time measurements are unknown on first execution. One possibility is to perform the first schedule as a common assignment problem using Integer Linear Programming (ILP) and application timing requirements, similar to the approach used by [9]. This way, a set of tasks $i = 1$ to n have an implementation x and an execution cost estimation c on each PU j; and the allocation was following designed: the task i is not allocated on the processor j when $x_{i,j} = 0$ and the task i is allocated on the processor j when the $x_{i,j} = 1$. The constraints for the model were the maximum workload for the PUs. Bellow, the constraint of each processing unit j ($U\max$), based on [9]:

$$U_j = \sum_{i=1}^{n} x_{i,j} c_{i,j} \leq U_{j_{\max}} \tag{3.1}$$

The best allocation was, then, found using the objective function that minimizes the resource utilization (percentage of occupancy for the PUs), defined as:

$$\left\{ \sum_{j=1}^{n} \sum_{i=1}^{n} x_{i,j} c_{i,j} \right\} \tag{3.2}$$

being the assignment variables $x_{i,j}$ the solution for the modeled ILP, m the number of computing units and n the number of considered tasks.

The mentioned ILP problem is of NP-hard complexity and become more complex in the scope of this work when dealing with more than two computing units and several tasks. To optimize the assignment calculation, some approaches concentrate on heuristics, as presented on [13].

However, this direction of estimating costs neither considers real execution times nor could represent the best assignment since a great number of estimations is used. This way, a second step of assignment allows taking into account real execution measurements extracted from the processors as well as dealing with the constraints presented by the NFRs. Based on that, the following dynamic module deals with real performance execution variables and possibly leads to a further better task assignment.

3.2. Task Scheduling Reconfiguration.

After the first assignment, information provided by the profiling aspects is consideredconsidered. Based on involved estimated costs (previously calculated using the pre-processing approach of the first guess) and possible "interferences" of runtime conditions and new loaded tasks, one task can be rescheduled to run in other processing unit just if the estimated time to be executed in the new hardware will be less than the time in the actual unit, i. e., just if there is a gain. Simply, this relationship can be modeled in terms of the costs:

$$T_{reconfigPUnew} < T_{remainingPUold} - T_{estimatedPUnew} - T_{overhead} \tag{3.3}$$

where the remaining time ($T_{remainingPUold}$) and the estimated time ($T_{estimatedPUnew}$) are calculated, respectively, for the current PU and for the candidate unit based on previous measurements (or on the first assignment in the case of first rescheduling invocation). An overhead ($T_{overhead}$) is considered to calculate the execution time of the reconfiguration itself. The relationship between $T_{remainingPUold}$ and $T_{estimatedPUnew}$ is, then, the partial gain.

The information to calculate the rescheduling will be then provided by the TimingVerifier aspect and can be modeled as:

$$\begin{aligned} T_{reconfigPUnew} &= T_{setupReconfigPUnew} + T_{temporaryStorage} + \\ &\quad T_{transferRate} + T_{executionPUnew} + L \end{aligned} \tag{3.4}$$

where $T_{setupReconfigPUnew}$ represents the time for setting up a new configuration on the new processor; $T_{temporaryStorage}$ is the time spent to save temporal data (considering shared and global memory access); $T_{transferRate}$ measures the cost for sending/receiving data from/to the CPU to/from the new processing unit, which can be a bottleneck on the whole calculation; $T_{executionPUnew}$ symbolizes the measured or estimated cost of the task processed in the new unit; and **L** denotes a constant to represent possible system latency.

Reinforcing the concepts, this approach deals with runtime conditions, like input emphdata type and amount to be processed, tasks assignment, and instantiation of new tasks "on the fly". All these runtime parameters that could not be known a priori can influence the execution of the system and must be evaluated periodically, leading to a large number of reconfiguration analysis and decisions. Then, supposing that a determined task is going to be executed n times in a determined time window, the strategy bellow reschedules the formed queue of task instantiations, giving a new relation of gain, just if the following assumption occurs:

$$T_{reconfigPUnew} < \sum_{i=1}^{n} \Big(T_{taskPUold_i} + T_{transferPUold_i} - T_{taskPUnew_i} - T_{transferPUnew_i} \Big) \tag{3.5}$$

where **TreconfigPUnew** is the time to perform the reconfiguration (mainly data transfer from the current processing unit to the new one if the task needs the calculated data done until the time of rescheduling), **TtaskPUold** is the time performance of the task in the current unit, **TtransferPUold** is the time for transferring data from CPU to the current computing unit (via bus), **TtaskPUnew** is the assumed time performance of the task in the candidate processing unit, and **TtransferPUnew** is the time for transferring data from CPU to the candidate unit (via bus).

Algorithm 1, bellow, describes the task reallocation module. It is also important to mention that the heuristic needs improvements along future works.

Algorithm 1 Task Reallocation Heuristic.

```
 1: Acquire Timing Data (Performance) about Previous Tasks Execution, storing them in a performance
    Database (Initialized according to First Assignment Phase and Regularly Updated);
 2: Acquire information about PUs;
 3: if Task has never been not executed then
 4:     Allocate it to a PUs according to First Assignment, storing it on the Database;
 5: end if
 6: Calculate Equation 3.5 according to Performance Data;
 7: Execute Load-Balancing Algorithm;
 8: Perform Reconfiguration Decision;
 9: Reschedule Task to perform the Reconfiguration when applicable;
10: Store Performance Data in the Database;
```

4. Case Study: UAV-based Area Surveillance System. The use of the presented ideas is illustrated by a case study that consists of a fleet of Unmanned Aerial Vehicles (UAVs) in the context of area surveillance missions. This kind of system has several kinds of applications, such as military surveillance, borderline patrolling, and civilian rescue support in cases of natural disasters, among others. Fig. 4.1 illustrates a military surveillance usage scenario where the UAVs can also communicate with each other.

Such UAVs can be equipped with different kinds of sensors that can be applied, depending on the weather conditions, time of the day and goals of the surveillance mission [16]. In this case study, it is considered a fleet

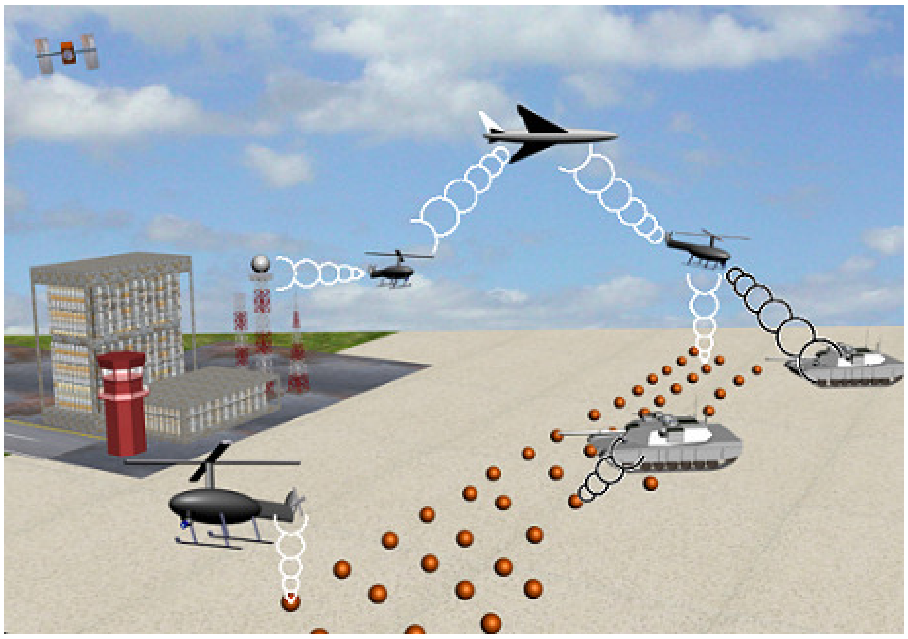

FIG. 4.1. *UAV-based Area Surveillance System.*

of UAVs that might accomplish missions during all the day and under whatever weather condition. UAVs must be able to provide different levels of information definition and detail, depending on the required data.

The UAVs receive a mission to survey a certain area, providing required data according to mission directions. Their movements are coordinated with the other UAVs in the fleet to avoid collisions and also to provide optimum coverage of the target area.

Each UAV is composed by six subsystems, making it capable to accomplish its mission and alto to coordinate with the others UAVs. These subsystems are: Collision Avoidance, Movement Control, Communication, Navigation, Image Processing, and Mission Management.

At this point, it is important to highlight the trade-off regarding cost, weight and size, and effectiveness of each UAV. The device, as a whole, may not be too big nor too heavy, in order to avoid unnecessary fuel consumption, as well as to be less susceptible of detection by counter forces sensors. Additionally, it may not have an enormous cost that could forbids the project. However, the UAV must be effective enough to provide the required data within an affordable cost and time budget. For more details about this trade-off discussion we address the readers to [16].

Another UAV's interesting feature is the possibility to apply different policies to missions, depending on user final intentions and specific requirements. There are two extremes for these policies: (i) Device Preservation Anyhow and (ii) Mission Accomplishment Anyhow. The first one consists of preserving UAVs even if the mission is not accomplished. It is especially applied in cases in which the devices can be destroyed and the information gathered by it is not worth compared to the cost of its destruction. On the other hand, in Mission Accomplishment Anyhow policy, the information gathered by the UAVs (and transmitted to the base station) is highly critical and overcomes the value of device loss. Within these policies, there are a variety of other factors that imposes different constraints to mission accomplishment and device preservation. Depending on the mission policy adopted, more resources can be (re)directed to tasks related to the movement control (when the UAV is escaping from a dangerous situation) or data gathering and processing (when information gathering has the highest priority).

In order to run the tasks described above, meeting the highlighted requirements and constrains modeled on the previous sections, we consider UAVs equipped with the following sensors: Visible Light Camera (VLC); SAR Radar (SARR) and Infra-Red Camera (IRC). To support the movement control and devices communication, each UAV is equipped with a hybrid "desktop"-based target platform which is used according to specific needs during the accomplishment of a certain mission, as detailed on section 4.2.

In this sense, each mentioned subsystem has a number of tasks to perform specialized activities related to a specific functionality, as depicted in the use cases diagram presented in Fig. 4.2. Based on the analyses of the

UAV functionalities, a summary of these tasks is provided in the following paragraphs.

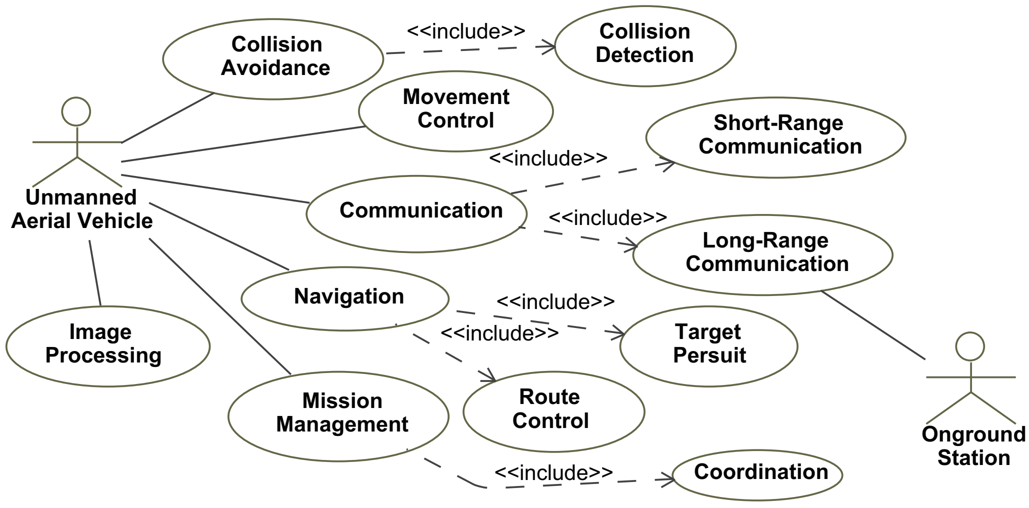

Fig. 4.2. *UAV Use Cases Diagram.*

Navigation guides the UAV movements, sending control information to the Movement Control subsystem. It is composed by the RouteControl and TargetPersuit tasks. The first task makes the necessary computation to guide the UAV through established waypoints, while the second one performs the same, but for dynamic waypoints that can be modified according to a moving object.

Image Processing gathers analog images, digitalizing them for further processing. It is composed by five tasks: (i) CameraController, which is responsible for camera movement, zoom and focus control of IRC and VLC, and antenna direction of SARR; (ii) Coder, which codifies the analog input into digital data; (iii) Compressor, which compresses the digital images; (iv) Reflectificator, which is responsible for the reflection in X and Y axis of radar image, as well as the rectification, that are necessary to avoid distortions in gathered images; (v) Filter, which filters radar images to eliminate the noise due to speckle effect [14].

Communication it has two main tasks: LongRangeCom and ShortRangeCom. The first task provides connectivity with pair communication nodes in long distances (of the order of kilometers), while the second one provides connectivity in short range distances (of the order of meters). These two tasks uses a third one, called Codec, which code and decode data transmissions.

Mission Management has also two tasks: MissionManager and Coordinator. The first one manages the information about the mission, such as required data, mission policy and resource autonomy control (e.g. remaining fuel). On the other hand, the second one drives the coordination with the other UAVs to avoid overlapping in the surveillance area.

Collision Avoidance is composed by two tasks: CollisionDetector, which detects possible collisions with other UAVs of the fleet or non cooperative flying objects; and CollisionAvoider, which calculates UAV's collision escape directions, sending them to the Movement Control subsystem.

4.1. Execution Platform. The target architecture of each UAV is composed of a four heterogeneous PUs platform: one host (the CPU), two GPUs, a PPU, and a PCICC. Fig. 4.3 shows the desired platform, where the *Profiling* gathers information from PUs (tasks performance) and the *Reconfiguration* distributes the tasks along them (intra allocation) according to the presented algorithms. It also consider sending data to be processed by other UAVs (inter allocation).

4.2. Reconfiguration Approach. Starting the mission, the UAVs have an initial task allocation throughout the CPU and the PU devices according to sub-section 3.1. In the current experimentation, it was considered to use the ILP approach for the first distribution using the GLPK toolkit [8]. Table 1 exhibits estimated costs (based on [3]) and first tasks' priorities that feed the GLPK-based simulation of task scheduling.

During execution, the mechanisms injected by TimingVerifier and the aspects Jitter and ClockDrift will start to generate information related to timing measurements. The TimingVerifier aspect will provide data to TaskAllocationSolver, which will get data from NodeStatusRetrieval. With the reasoning mechanisms, it will periodically analyze the provided information according to the algorithm introduced in 3.2.

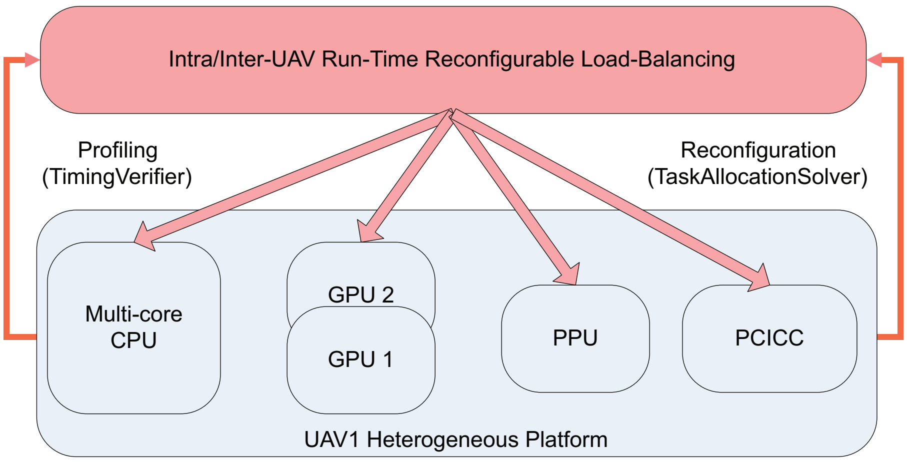
Intra/Inter-UAV Run-Time Reconfigurable Load-Balancing
Profiling (TimingVerifier)
Reconfiguration (TaskAllocationSolver)
Multi-core CPU
GPU 2
GPU 1
PPU
PCICC
UAV1 Heterogeneous Platform

Fig. 4.3. *Execution platform.*

Table 4.1
Task Estimation Costs.

Task	**Estimated Cost (scale: 1 to 6)**				**First Priority (scale: 1 to 6)**
	GPU	**PPU**	**CPU**	**PCICC**	
Image Processing	1	4	6	—	1
Collision Avoidance	3	2	5	—	2
Movement Control	2	2	3	—	1
Navigation	1	1	2	—	3
Communication Short/Long range	4/6	—	3/5	1/2	4
Mission Management	5	—	1	—	6

Emphasis is given to the RIP subsystem, which is considered to be the group that requires more processing due to the handling of large data and new instantiations created dynamically. The RIP workflow is depicted on Fig. 4.4. Theoretically, tasks associated to RIP should execute with better performance on a GPU device when the application is not aware about the context of the whole execution scenario, i. e. executing stand alone. Thus, initial demands of all tasks should be executed in the GPU. Shortly, the captured data (raw scalar image) must be "adjusted" regarding the SAR position parameters (range and azimuth), followed by Fast Fourier Transform (FFT), image rotation, and other corrections to produce the final image. This process can be performed individually in the range and azimuth directions and it consists basically in a data compression on both directions using filters that maximize the relation between the signal and the noisy. Readers are addressed to [5] to get refined explanations about the workflow.

Afterwards, in the explored surveillance system, the final image is submitted to a pos-processing in order to identify regions of interest that could contain objects specified in the mission directions as a "pattern to be found" or a "target". In this case, more resolution on specific areas will be needed and new data will be generated, demanding more processing from the assigned PU(s) in order to produce new images and extract relevant information (patterns).

Based on that description, this dynamic scenario clearly influences the tasks' priority since, at a moment, the new high-resolution images will have higher priorities if compared to others that became more "generic". These events cannot be predicted a priory and the verification of such situation require a smart, context-aware, and dynamic reconfiguration support to balance the workload, accomplishing the timing requirements and budget.

4.3. Results. Considering 2 UAVs in the case study, Table 4.2 denotes the behavior of the dynamic reschedule load-balancer simulator. The "first guess" represents one instantiation of each group of tasks assigned to a PU; and with dynamic creation of new groups (4, 8, and 12) of RIP tasks, the assignment is changed and optimized to minimize the total execution time. Note that these values cannot represent the best assignment since the version of the simulator did not consider all parameters that influence the whole system. As it is an

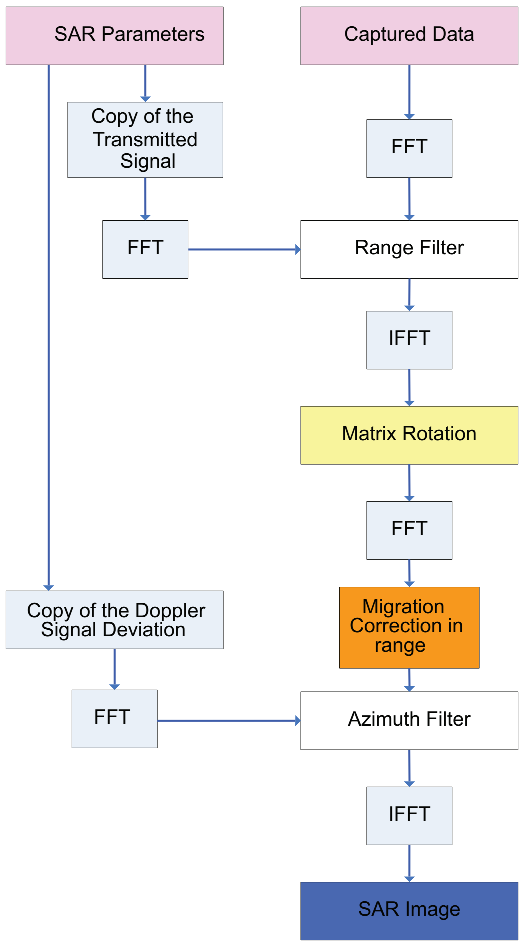

Fig. 4.4. *SAR Image Processing (based on the notes of [5]).*

Table 4.2
Task Assignment.

Task	**1st Gess**	**Dynamic Image Processing Created Tasks**		
		4	**8**	**12**
Image Processing	GPU1	GPU1 GPU2	GPU1 GPU2 PPU	GPU1 GPU2 UAV2-GPU1
Collision Avoidance	GPU2	PPU	CPU	CPU
Movement Control	PPU	PPU	CPU	CPU
Navigation	PPU	PPU	PPU	CPU
Communication	CPU	CPU	CPU	PCICC
Mission Management	CPU	CPU	CPU	CPU

ongoing work, more accurate data about the reschedule must be provided along the simulator's refinement in order to represent the scenario as realistic as possible.

5. Related Work. VEST (Virginia Embedded System Toolkit) [15] is a set of tools that uses aspects to compose a distributed embedded system based on a component library. Those aspects check the possibility of composing components with the information taken from system models. It provides analysis such as task schedule feasibility. However, it performs statically analysis at compiling time. In our proposal, aspects are used dynamically to change the system configuration at runtime, adapting its behavior to new operating conditions.

Although there are some related works concerning dynamic reconfiguration in cluster computing, like for example ([18]; [12]; [1]), our approach concentrates on single desktop platforms composed by different processing units where the reconfiguration is performed within these devices. In this way, the work presented by [9] implements dynamic reconfiguration methods for Real-Time Operating System services running on a Reconfigurable System-on-Chip platform based on CPU and FPGA. The method, based on heuristics, take into account the idleness percentage of the computing units and unused FPGA area (calculated as pre-processing) to perform the load-balancing and to decide about a reconfiguration of tasks in runtime by means of task migration. Our approach complements this related work, developing generically methods that comprise more than two processing units and that work with dynamic performance data.

Targeting GPUs, the work of [17] presented a programming framework to achieve energy-aware computing. On the proposed strategy, the compiler translates the framework code to a C++ code for CPU and a CUDA code for GPU. Then, a runtime module dynamically selects the appropriate processor to run the code taking into account the difference in energy efficiency between CPU and GPU based on energy consumption estimation models. However, it does not take into account runtime energy measurements (runtime profiling), which is an important module of our work.

Another approach, focusing performance improvement of spheres collision detection simulation, was proposed by [10], in which some strategies have been presented to perform data balancing over CPU and GPU, both in an automatically and manually options. That work takes into account the performance of a kernel implemented on the CPU and GPU. After the execution starts, both versions of the programs are executed with equally input data and time performance is verified. More data are then dynamically assigned to the processor that executed faster the previous data, indicating that the approach uses data decomposition instead of task decomposition. Our work concentrates on task decomposition and its dynamic assignment according to estimated or profiled performance.

The work presented in [4] published a study to accelerate compute-intensive applications using GPUs and FPGAs, listing some of their pros and cons. The work performed a qualitative comparison of application behavior on both computing units taking into account hardware features, application performance, code complexity, and overhead. Although GPUs can offer a considerable performance gain for certain application, that work's results showed that FPGAs can be an interesting computing unit and could promote a higher performance compared to GPU when applications require flexibility to deal with large input data sets. However, using FPGAs comes with cost of hardware configuration before using it as a computing unit, a task usually oriented to experienced users. Thus, task reconfiguration frameworks, as the one presented in this work, could provide a higher abstraction layer to assist developers during system design.

6. Conclusions and Future Work. This paper presents a methodology to address the problem of efficient task assignment in runtime targeting hybrid computing platforms. It allows the use of resources offered by an asymmetric computer platform, providing compliance with dynamic changes in timing requirements and constraints, and also runtime conditions. In order to achieve the proposed goals, our proposal uses an aspect-oriented framework in conjunction with a dynamic task self-rescheduling strategy, in order to address the dynamic runtime scenarios under concern.

A UAV-based surveillance system simulation has been used to show the need for workload adaptation required by sophisticated applications, running on top of hybrid computers, which face dynamic execution scenarios. Real-time task rescheduling was applied on UAV PUs, focusing on RIP. Results indicate that rescheduling contributes to a more appropriate system resource usage, and hence towards performance improvement. Sending/receiving data between UAVs was also considered, but details about specific problems related to these interactions, such as delays in the communication between the UAVs, have not been focused by this text.

Future directions lead to refine the scheduling strategy to provide complete simulations, considering a larger range of runtime parameters, including the reconfiguration costs itself; and real algorithms for UAV's subsystems, emphasizing RIP dynamicity. Heuristics to predict the future allocation of tasks based on its recent use seems to be a good strategy, and will possibly avoid unnecessary reconfigurations in a specific time-window.

Acknowledgments. A. P. D. Binotto thanks the partial support given by DAAD fellowship and the Programme Alßan, the European Union Programme of High Level Scholarships for Latin America, scholarship no. E07D402961BR.

E. P. Freitas thanks the Brazilian Army for the given grant to pursue the PhD program focused on Embedded Real-time Systems at Halmstad University in cooperation with UFRGS (Brazil).

REFERENCES

[1] B. B. Brandenburg, J. M. Calandrino, and J. H. Anderson, *On the Scalability of Real-time Scheduling Algorithms on Multicore Platforms: a case study*, Proceedigns of Real-Time Systems Symposium, (2008), pp. 157–169.

[2] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, F. Di Gi, B. Omenico, K. Ramamritham, J. Stankovic, and L. Strigini, *The Meaning and Role of Value in Scheduling Flexible Real-Time Systems*, Proceedigns of Early Aspects: Current Challenges and Future Directions, 46 4 (2000), pp. 305–325.

[3] G. Cai, K. Peng, B. M. Chen, and T. H. Lee, *Design and Assembling of a UAV Helicopter System*, Proceedigns of International Conference on Control and Automation, (2005), pp. 697–702.

[4] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, *Accelerating Compute-Intensive Applications with GPUs and FPGAs*, Proceedigns of Symposium on Application Specific Processors, (2008), pp. 101–107.

[5] I. Cumming and F. Wong, *Digital Processing of Synthetic Aperture Radar Data*, Artech House-London, 2005.

[6] E. P. Freitas, M. A. Wehrmeister, C. E. Pereira, F. R. Wagner, E. T. Silva Jr., and F. C. Carvalho, *Using Aspect-Oriented Concepts in the Requirements Analysis of Distributed Real-Time Embedded Systems*, Proceedigns of International Embedded Systems Symposium, (2007), pp. 221–230.

[7] E. P. Freitas, M. A. Wehrmeister, C. E. Pereira, F. R. Wagner, E. T. Silva Jr., and F. C. Carvalho, *DERAF: A High-Level Aspects Framework for Distributed Embedded Real-Time Systems Design*, Proceedigns of Early Aspects: Current Challenges and Future Directions, (2007), pp. 55–74.

[8] The GNU Project, *GLPK—GNU Linear Programming Kit*, In http://www.gnu.org/software/glpk/. Access in Jun. 2008.

[9] M. Götz, F. Dittmann, and T. Xie, *Dynamic Relocation of Hybrid Tasks: A Complete Design Flow*, Proceedigns of Reconfigurable Communication-centric SoCs, (2007), pp. 31–38.

[10] M. Joselli, M. Zamith, E. Clua, A. Montenegro, A. Conci, R. Leal-Toledo, L. Valente, B. Feijo, M. Dornelas, and C. Pozzer, *Automatic Dynamic Task Distribution between CPU and GPU for Real-Time Systems*, Proceedigns of the IEEE International Conference on Computational Science and Engineering, (2008), pp. 48–55.

[11] G. Kiczales, J. Irwin, J. Lamping, J. M. Loingtier, C. Videira Lopes, C. Maeda, and A. Mendhekar, *Aspect-Oriented Programming*, Proceedigns of European Conference for Object-Oriented Programming, (1997), pp. 220–242.

[12] M. Linderman, J. Collins, H. Wang, and T. Meng, *Merge: A Programming Model for Heterogeneous Multi-core Systems*, ACM Sigplan Notices, 43 3 (2008), pp. 287–296.

[13] M. McCool, *Scalable Programming Models for Massively Multicore Processors*, Proceedings of the IEEE, 96 5 (2008), pp. 816–831.

[14] M. I. Skolnik, *Introduction to Radar Systems*, Third ed., McGraw-Hill, 2001.

[15] J. A. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, and B. Ellis, *VEST: Aspect-Based Composition Tool for Real-Time System*, Proceedigns of Ninth IEEE Real-Time and Embedded Technology and Applications Symposium, (2003), pp. 58–69.

[16] D. M. Stuart, *Sensor Design for Unmanned Aerial Vehicles*, Proceedigns of IEEE Aerospace Conference, (1997), pp. 285–295.

[17] H. Takizawa, K. Sato, and H. Kobaiashi, *SPRAT: Runtime Processor Selection for Energy-aware Computing*, Proceedigns of the IEEE International Conference on Cluster Computing, (2008), pp. 386–393.

[18] P. Wang, J. Collins, G. Chinya, H. Jiang, X. Tian, M. Girkar, N. Yang, G. Y. Lueh, and H. Wang, *EXOCHI: Architecture and Programming Environment for a Heterogeneous Multi-core Multithreaded System*, Proceedigns of the ACM SIGPLAN conference on Programming language design and implementation, (2007), pp. 156–166.

[19] M. A. Wehrmeister, E. P. Freitas, D. Orfanus, C. E. Pereira, and F. Ramig, *A Case Study to Evaluate Pros/-Cons of Aspect- and Object-Oriented Paradigms to Model Distributed Embedded Real-Time Systems*, Proceedigns of 5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software, (2008), pp. 44–54.

Edited by: Janusz Zalewski
Received: September 30, 2009
Accepted: October 19, 2009