Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 581 – 588

ISBN 978-83-60810-14-9
ISSN 1896-7094

Real-time Task Reconfiguration Support Applied to an UAV-based Surveillance System

Alécio Pedro Delazari Binotto
Fraunhofer IGD / TU Darmstadt – Germany
PPGC UFRGS – Brazil
Email: alecio.binotto@igd.fraunhofer.de

Edison Pignaton de Freitas
IDE – Halmstad University – Sweden
PPGC UFRGS – Brazil
Email: edison.pignaton@hh.se

Carlos Eduardo Pereira
PPGC UFRGS – Brazil
Email: cpereira@ece.ufrgs.br

André Stork
Fraunhofer IGD / TU Darmstadt – Germany
Email: andre .stork@igd.fraunhofer.de

Tony Larsson
IDE – Halmstad University – Sweden
Email: tony.larsson@hh.se

***Abstract*—Modern surveillance systems, such as those based on the use of Unmanned Aerial Vehicles, require powerful high-performance platforms to deal with many different algorithms that make use of massive calculations. At the same time, low-cost and high-performance specific hardware (e.g., GPU, PPU) are rising and the CPUs turned to multiple cores, characterizing together an interesting and powerful heterogeneous execution platform. Therefore, reconfigurable computing is a potential paradigm for those scenarios as it can provide flexibility to explore the computational resources on heterogeneous cluster attached to a high-performance computer system platform. As the first step towards a run-time reconfigurable workload balancing framework targeting that kind of platform, application time requirements and its crosscutting behavior play an important role for task allocation decisions. This paper presents a strategy to reallocate specific tasks in a surveillance system composed by a fleet of Unmanned Aerial Vehicles using aspect-oriented paradigms in order to address non-functional application timing constraints in the design phase. An aspect support from a framework called DERAF is used to support reconfiguration requirements and provide the resource information needed by the reconfigurable load-balancing strategy. Finally, for the case study, a special attention on Radar Image Processing will be given.**

I. Introduction

In addition to timing constraints, several modern applications usually require high performance platforms to deal with different algorithms and massive calculations, varying from monitoring and processing of different data acquired by sensors to cryptography and large image data visualizations and processing. The development of low-cost powerful and application specific hardware (for example, the 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) offer several execution alternatives aiming better performance, programmability and control. The resulting execution platform heterogeneity is intensified with multi-core CPUs, causing problems to achieve simple programming and efficient resource utilization.

Following that direction, low-cost inter-chip hybrid hardware architectures are becoming very attractive to compose adaptable execution platforms and, at the same time, software applications must benefit from that performance powerfulness. This leads to the creation of new methods and strategies to distribute the application's workload (tasks, algorithms, full applications) to execute in the specific processing units in order to better meet application requirements, such as performance and timeliness, without loosing flexibility. In this manner, reconfigurable load-balancing computing is a potential paradigm for those scenarios as it can provide flexibility, improve efficiency, and offer simplicity to high performance heterogeneous processor cluster and multi-core architectures. Figure 1 shows such a theoretical scenario.

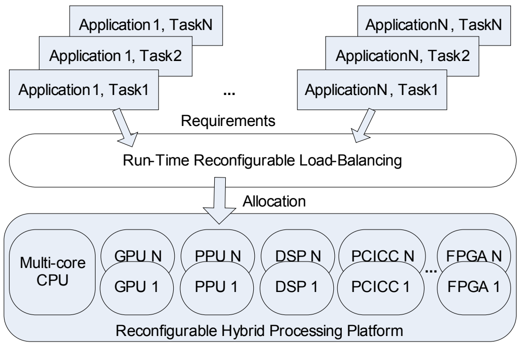

Fig. 1. System overview

The goal is to design proper methodologies, strategies, and support for management of a dynamic load-balancing by means of real-time reconfiguration of involved tasks. To reach this objective, thereby, the main step is to create a self reconfiguration framework that can be used by applications composed of different kinds of algorithms (graphics, massive mathematical calculations, sensor data processing, artificial intelligence, cryptography, etc) which runs on a single personal computer and needs to execute under time constraints and a minimal quality (performance). In addition, during execution time, the framework is intended to keep monitoring the tasks and provides online information in order to a possible new allocation balance, i.e., a reconfiguration of tasks is done if this procedure can promote a better performance for the overall current scenario.

In this paper, the focus is on the very first step in the design method framework: application requirements handling (reconfiguration) in a high-level design. The approach is to extract the requirements to find concurrency in order to base the load-balancing framework towards task parallelization and reallocation. For its accomplishment, the crosscutting concerns related to the real-time non-functional requirements are taken into account. The handling of these concerns by specific design elements called "aspects" (from aspect-oriented paradigm [1] plays an important role to improve understandability and maintainability during task (re)configuration. Based on the support offered by the aspects to monitor and control the above mentioned requirements, a strategy to assign tasks dynamically to units of execution is presented, being these tasks subject to an on-line reconfiguration.

The paper is organized as follows. We start in Section II with a previous work based on aspects and a description of the created ones for requirements identification, modeled using UML notation. Section III follows by a reconfigurable workload strategy implemented by the aspects. Composing these two concepts, Section IV outlines Unmanned Aerial Vehicle Surveillance system as application, focusing on radar image processing tasks. Finalizing, related works, conclusions and future works are presented.

II. Aspect-oriented Handling of Real-time Concerns

In order to provide a reconfigurable solution in runtime with the goal to meet timing requirements, several mechanisms to control and monitor timing parameters must be inserted in the system. Moreover, the mechanisms related to the migration of tasks among processing units, which implements the system reconfiguration itself, also affect several elements in the system in a non-uniform way. Besides, all these mechanisms and controls are not the main goal or functionality of any system, but they must be present in order to achieve reconfiguration. These facts are clear characterized as non-functional crosscutting concerns, which can be successfully addressed by an aspect-orientation [1].

The non-functional requirements handling concerns hinder the system maintainability, reuse, and evolution in current approaches used in task reconfiguration such as those that use pure object-orientation. It occurs because the handling elements (such as timing requirements probes, serialization mechanisms, task migration mechanisms, among others) are not modularized in a single or few system elements, but spread over the system. Any change in one of these elements requires changes in different parts of the system, what besides to be a tedious and error-prone task, do not scale in the development of large and complex applications. The observation of these drawbacks motivates the use an aspect-oriented approach that 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. Moreover, the advantages of the aspect-oriented approach became clearer when applied to the aforementioned task allocation strategies using heterogeneous platforms due to the need of profiling each task in a different hardware, affecting several elements of the application. The use of aspects to address this concern represents an improvement as it helps to cope with the complexity in managing this handling that is spread allover the system.

In the next subsection, the aspects used to address timing related concerns will be presented, which come from the DERAF framework [2] that provides a high-level aspect library to handle Distributed Real-time Embedded (DRE) systems non-functional requirements.

A. Time-based Aspects

In order to support Timing and Precision requirements, the proposal is to use some aspects from the DERAF framework. The packages designed to these types of requirements are presented in Figure 2 and a short description of each one is provided in the following paragraphs. Interested readers are referred to [2] and [3] for more details.

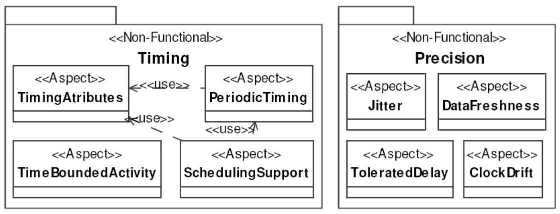
<<Non-Functional>>
Timing
<<Aspect>> TimingAtributes
<<use>>
<<Aspect>> PeriodicTiming
<<use>>
<<use>>
<<Aspect>> TimeBoundedActivity
<<Aspect>> SchedulingSupport
<<Non-Functional>>
Precision
<<Aspect>> Jitter
<<Aspect>> DataFreshness
<<Aspect>> ToleratedDelay
<<Aspect>> ClockDrift

Fig. 2. Timing and Precision packages from DERAF

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

PeriodicTiming: adds a periodic activation mechanism to active objects. 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 scheduling list is schedulable.

TimeBoundedActivity: temporally limits the execution of an activity by adding a mechanism or changing a parameter that can restrict the maximum execution time for an activity (e.g., limit the time which a shared resource can be locked).

Jitter: measures the start/end of an activity, calculates the variation of these metrics and whether the tolerated variance was overran.

ToleratedDelay: temporally limits the beginning of an activity execution (e.g., limits the time which an active task can wait to acquire a lock on a shared resource).

DataFreshness: associates timestamps to data, verifying their validity before using them [4].

ClockDrift: measure the time at which an activity starts and compares it with the expected beginning of this activity; it also checks if the accumulated difference exceeds the maximum tolerated clock drift.

Additional aspects were developed to compose the information provided by the DERAF aspects to inject the reconfiguration strategy. These aspects as described in the following.

B. Aspects for Supporting Reconfiguration

As mentioned before, the task migration support characterizes a non-functional crosscutting concern that affects different parts of the system in different ways. On this way, we propose the use of aspects to address this concern. The new aspects proposed in this work are: **`TimingVerifier`**, **`TaskAllocationSolver`**.

Both use the time parameters inserted by the aspects of the `Timing` package, described previously, and the services provided by the aspects from the `Precision` package. An additional package from DERAF is also used in order to make the reconfiguration decisions take place; it is the `TaskAllocation` package, which is presented in more details latter on in this section. Figure 3 depicts the schema.

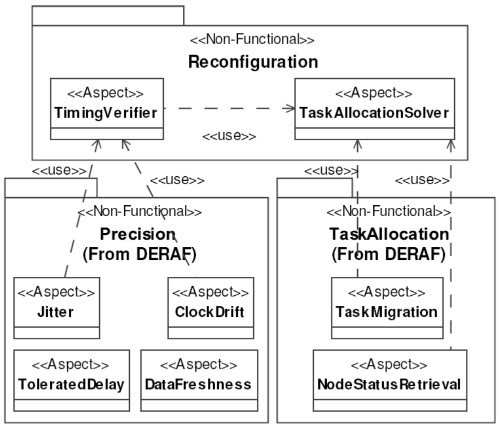

Fig. 3. Aspects for Reconfiguration

The **`TimingVerifier`** is responsible for checking if the processing units are being able to accomplish with the timing requirements specified by the `TimingAttributes`, `PeriodicTiming`, `ToleratedDelay` and `TimeBoundedActivity` aspects. In order to do this, it uses the services from the aspects `Jitter` and `ClockDrift`.

A mechanism to control the meeting of timing attributes is inserted in the beginning and end of each task. This mechanism consists of measuring these times, and comparing them with the requirement specified by the correspondent attribute. As an example, the accomplishment of a specified deadline can be checked by measuring the time in which the task actually ended its computation and comparing it with the time in which it was supposed to finish. It uses the service of the `Jitter` aspects to gather and analyze information about the jitter related to the corresponding requirement. Taking the deadline again as an example, it measures if a non-accomplishment of a deadline is constant or if the measure varies in different executions or in changing the platform scenario. It can be used, for instance, as base information to know if the interaction with other tasks is being responsible for the variance.

The `ClockDrift` aspect is used by the `TimingVerifier` to gather information about the synchronization among the different Processing Units (PUs). It is useful to calculate the cost, in terms of a task migration time. In order to illustrate the idea, consider a task that was migrated from a PU “A” to a PU “B”. The difference in the clock drift between them can result in a waiting time for the result from the PU “B” that does not worth if compared with leaving the task running in the PU “A”.

`TaskAllocationSolver` is the second key aspect. It is responsible for deciding whether a task will be migrated or not and to which of the available PUs. It also has to check possible overload of the PU candidate destinations in order to decide whether it is worthwhile to perform the migration. The `TaskAllocationSolver` uses the measurements available due to the work done by the `TimingVerifier`.

Based on these data, the reasoning about the feasibility or not of a task reconfiguration is done. The explanation of this reasoning is provided in the next section, where the reconfiguration strategy is presented.

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

TaskMigration: provides a mechanism to migrate active objects (tasks) from one PU node to another PU node. It is used by the aspects that control embedded concerns and, in the present work, by the allocation solver aspect `TaskAllocationSolver`.

NodeStatusRetrieval: inserts a mechanism to retrieve information about processing load, send/receive message rate, and/or the PU availability (i.e., the “I'm alive” message). Before/after every execution start/end of an active task, the processing load is calculated. Before/after every sent/received message, the message rate is computed. Additionally, the PU availability message is sent at every “n” message or periodically with an interval of “n” time units.

III. Dynamic Reconfiguration Strategy

A task-based decomposition approach is used, in which each task is an independent algorithm. The tasks are grouped according derivation of the same high-level, simplifying the managing of possible dependencies. Besides, it is coherent to assume that a group of tasks will have the same characteristics or, in other words, the same designed features and hence would be desirable to execute in the same PU. However, this can lead to a non-optimal execution, as it will be discussed in the next sub-sections.

C. First Assignment of Tasks

The strategy is to combine a costly method for assignment problems with a real-time measurement procedure based on the mentioned aspects. For the first assignment of tasks, we do not use the modeled aspects, since we do not have real time measurements of tasks. Therefore, we decided to offer two possibilities: assign all the tasks in the first time step to the CPU and then perform the dynamic reconfiguration; or model the first assignment as a common assignment problem using Integer Linear Programming (ILP), like the generic approach used by the authors of [5]. In this way, a set of tasks can be represented as $T = \{t_{i,j}\}$, where every task i has an implementation based on the specific hardware API (considering each supported hardware) and an execution cost estimation on the PU j. The execution cost $c_{i,j}$ is simply estimated, in this paper, based on a correlation between the input data type used by the task (mapped within a weight scale) and the number of sub-cores presented in the corresponding processing unit. A flag is assigned for tasks that only execute in determined hardware. Note that these costs could not reproduce the reality with fidelity and are just a way to determine a "first guess" to the system.

The total execution time of the application is minimized by finding a schedule solution by means of its tasks execution times. The constraints for the presented model will be the maximum workload of each processing unit (U_{max}), which are represented by

$$U_j = \sum_{i=1}^{n} x_{i,j} c_{i,j} \le U_{j_{\max}} \quad (1).$$

The objective function that minimizes the total application execution time is, then, defined as

$$\min \left\{ \sum_{j=1}^{m} \sum_{i=1}^{n} x_{i,j} c_{i,j} \right\} \quad (2),$$

being the variables $x_{i,j}$ the solution for the modeled ILP.

The ILP problem is considered of complexity NP-hard and it is costly to calculate every period of time to estimate the current optimal assignment. Due to that reason, some approaches concentrate on heuristic-based methods to estimate the best assignment, as the above mentioned work of [5]. Nevertheless, this direction neither considers real execution times nor could represent the best assignment.

In counter part, the approach presented in this paper allows taking into account real execution measurements extracted from the processing units (using aspects) and works with a dynamic reconfiguration module that deals with real execution variables and interferences, leading to a possible better task assignment.

D. Real-time Reconfiguration

After the first tasks assignment, the dynamic reconfiguration strategy enters on scene in application run-time. At this point, we consider the information provided by the created aspects and the first assignment. The assumption is: based on involved migration costs and possible interferences of new loaded tasks, one task can be reconfigured to run in other PU just if the new PU already finished the processing of its tasks (is idle) and the estimated time to execute the task in the new PU will be less than the time to execute in the actual PU, i.e., just if there is a gain. In a simple equation, this relationship can be modeled in terms of the costs:

$$\boldsymbol{T}_{reconfigPUnew} < \boldsymbol{T}_{remainingPUold} - \boldsymbol{T}_{estimatedPUnew} - \boldsymbol{T}_{overhead} \quad (3),$$

where the remaining time ($\boldsymbol{T}_{remainingPUold}$) and the estimated time ($\boldsymbol{T}_{estimatedPUnew}$) are calculated, respectively, for the current PU and for the new candidate PU based on previous measurements or on the first assignment (in the case of first reconfiguration invocation); and an overhead ($\boldsymbol{T}_{overhead}$) that explicit the execution time of the reconfiguration itself. The relationship between $\boldsymbol{T}_{remainingPUold}$ and $\boldsymbol{T}_{estimatedPUnew}$ is considered the partial gain.

The information needed to calculate the reconfiguration will be provided by the `TimingVerifier` aspect and can be modeled without such mathematical formality as:

$$\boldsymbol{T}_{reconfigPUnew} = \boldsymbol{T}_{setupReconfigPUnew} + \boldsymbol{T}_{temporaryStorage} + \boldsymbol{T}_{transferRate} + \boldsymbol{T}_{executionPUnew} + \boldsymbol{L} \quad (4),$$

where $\boldsymbol{T}_{setupReconfigPUnew}$ represents the time for setting up a new configuration on the new PU; $\boldsymbol{T}_{temporaryStorage}$ contributes with the time spent to save temporal data if needed (considering shared and global memory access parameters); $\boldsymbol{T}_{transferRate}$ measures the cost for sending/receiving data from/to the CPU to/from the new PU, which can be a bottleneck on the whole calculation; $\boldsymbol{T}_{executionPUnew}$ symbolizes the measured or estimated cost of the task processed in the new PU; and finally $\boldsymbol{L}$ denotes a constant to represent possible system latency.

During the application execution, the load-balancing module will keep storing the execution times for each type of the tasks with the information gathered by the `TimingVerifier` aspect together with the data provided by the `NodeStatusRetrieval` aspect. These preliminary stored times will be useful as one of the basis to the reconfiguration decision done by the `TaskAllocationSolver`.

An overview about the created strategy is presented on Tables 1 and 2. The behaviors that compose this strategy are inserted in the core of the system by the above mentioned aspects and those used by them, as presented in section II.B.

TABLE I.
TASK REALLOCATION ALGORITHM

```
1:ACQUIRE TIMING DATA ABOUT TASKS EXECUTION;
2:ACQUIRE DATA ABOUT PUS PROCESSING LOAD;
3:CALCULATE THE EXECUTION PRIORITIES FOR ALL TASKS, INCLUDING NEW
    LOADED ONES, BASED ON STEPS 1 AND 2;
4:CALCULATE THE NEW LOAD-BALANCE, TAKING INTO ACCOUNT THE NEW
    PRIORITIES CALCULATED PREVIOUSLY;
5:COMPARE THE NEW LOAD-BALANCE IN ACCORDANCE WITH EQUATION (3)
    AND, CONSEQUENTIALLY, EQUATION (4);
6:PERFORM THE RECONFIGURATION DECISION;
7:MIGRATE TASKS TO PERFORM THE RECONFIGURATION WHEN APPLICABLE.
```

TABLE II.
LOAD-BALANCING MODULE ALGORITHM

```
1:DETECT THE AVAILABLE PUS;
2:CALCULATE THE FIRST ASSIGNMENT USING EQUATION (1) AND (2) OR ASSIGN
    ALL TASKS TO CPU;
3:FOR EACH N TIME-STEP DO:
4:  EXECUTE TASK REALLOCATION ALGORITHM;
5:  STORE MEASURED TIMES;
6:END FOR.
```

IV. UAV-BASED AREA SURVEILLANCE SYSTEM

The presented ideas are illustrated by a case study that consists of a fleet of Unmanned Aerial Vehicles (UAVs) used in area surveillance missions. 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 [6]. We consider a fleet of UAVs that might accomplish missions during all the day and under all weather conditions. The UAVs also must be able to provide different levels of information precision and detail, depending on the required data.

To start, the UAVs receive a mission to survey a certain area and provide required data according to the mission directives. Their movements are coordinated with the other UAVs in the fleet in order to avoid collisions and also provide optimum coverage of the target area. Details about the used coordination algorithm are out of the scope of this paper.

Each UAV is composed of six subsystems that make it able to accomplish its missions and coordinate with the other UAVs. These subsystems are: Collision Avoidance, Movement Control, Communication, Navigation, Image Processing, and Mission Management.

In order to run the tasks described above and meet the high-lighted requirements and constrains modeled on the previous sections, we consider UAVs equipped with the following sensors: Visible Light Camera (VLC); Synthetic Aperture Radar (SAR) and Infra-Red Camera (IRC). In order to support the movement control, communication devices and embedded sensors, each UAV will be equipped with a hybrid processing unit target platform which is used accordingly to the specific needs during the accomplishment of a certain mission, detailed in the following.

E. UAV Subsystems

As mentioned above, each UAV has six subsystems. Each subsystem has a number of tasks that perform specialized activities related to a specific functionality. An UML Use Cases Diagram showing the UAV functionalities is presented in Figure 4.

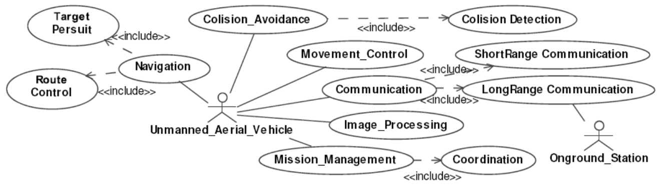

Fig. 4. UAV Use Cases Diagram

Based on the analyses of the UAV functionalities, a summary of the tasks that compose each subsystem is provided in the following paragraphs.

Movement Control: responsible to monitor and control the engines and direction mechanisms, such as flap actuators. It is composed by two tasks. The first is called `Movement-Controller`, which performs the calculus that must be applied in the actuators and engines. The second task is called `MovementEncoder` and it is responsible for the sampling and encoding of the actual values in the engines and actuators, which will be used as feedback data for the `MovementController` task. These tasks are designed to have an implementation based on CPU, GPU, and PPU.

Navigation: control the directions of the UAV movements and sends control information to the Movement Control subsystem. It is composed by the `RouteControl` and `Tar-getPersuit` tasks. The first makes the calculation to guide the UAV through established waypoints, while the second performs the same, but for dynamic waypoints that varies accordingly to a moving object. These tasks have an implementation based on CPU, GPU, and PPU.

Image Processing: this subsystem gathers analog image information and performs its digitalization. It is composed of six tasks. The first is the `CameraController`, which is responsible by the movement of the camera, zoom and focus control of IRC and VLC, and antenna direction of the SAR. The second is the `Coder`, which codifies the analog input into digital data. The third is the `Compressor`, which compresses the digital images. The fourth is the `Reflectifi-cator`, which is responsible for the reflection in the X and Y axis of radar image, as well as the rectification. These two processes are necessary to avoid distortions in the image. The fifth task is called `Filter`, which is responsible for filtering the radar images in order to eliminate the noise due to speckle effect [7]. The last task is called `Pattern-Recongnition` and is responsible to perform image segmentation and recognition of patterns from the previously processed data. These tasks are designed to have an implementation based on CPU, GPU, and PPU.

Communication: communication subsystem has two main tasks, the `LongRangeCom` and `ShortRangeCom`. The first provides connectivity with pair communication nodes in long distances, order of kilometers, while the second provides connectivity in short range, order of meters of distance. Both make use of a third, called `Codec`, which code and decode transmission data based on cryptographic techniques. These tasks have an implementation based on CPU, GPU, and PCICC.

Mission Management: this subsystem has two tasks, the `MissionManager` and the `Coordinator`. The first manages the information about the mission, like required data and mission policy, while the second drives the coordination with the other UAVs to avoid surveillance area overlap. Their implementation are based on CPU, and GPU.

Collision Avoidance: is composed by two tasks, `Colli-sionDetector` and `CollisionAvoider`. The first detects possible collisions with other UAVs of the fleet or non cooperative flying objects, and the second makes the calculus to avoid the collision and send them to the Movement Control subsystem. These tasks have an implementation based on CPU, GPU, and PPU.

F. Reconfiguration Approach

In the presented experiment, the target architecture is composed of a four hybrid PUs platform: one CPU (Intel 2-core) and three types of co-processors, two GPUs (nVidia GeForce8800 GT – 512MB memory - using PCI Express x16 and CUDA-FFT/BLAS toolkit), a PPU (Ageia using PhysX SDK), and a PCICC (IBM using UDX toolkit). Figure 5 shows the desired execution platform, where the Profiling gathers information from the PUs and the Reconfiguration distributes the tasks along the PUs (intra allocation) and also consider sending data to be processed by other UAVs (inter allocation).

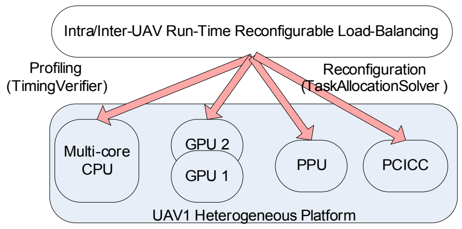
Intra/Inter-UAV Run-Time Reconfigurable Load-Balancing
Profiling (TimingVerifier)
Reconfiguration (TaskAllocationSolver)
Multi-core CPU
GPU 2
GPU 1
PPU
PCICC
UAV1 Heterogeneous Platform

Fig. 5. UAVs Execution Platform with support for Load-balancing

Starting the mission, the UAVs have an initial task allocation throughout the CPU and co-processing units of the target platform according to one of the algorithms presented on sub-section III.A. In the current experiments, it was considered to use the ILP approach for the first distribution of the groups of tasks (and not the tasks individually). This simulation was done using the GLPK toolkit [8] and took in consideration the priorities of abstract tasks and its estimated execution costs in each PU, performing the following distribution: tasks from the Movement Control group (`MovementController`, `MovementEncoder`), and Navigation (`RouteControl`, `TargetPersuit`) are assigned to the PPU; the ones from Collision Avoidance (`CollisionDetector`, `CollisionAvoider`) are assigned to the GPU2; the Image Processing tasks (`CameraController`, `Coder`, `Compressor`, `Reflectificator`, `PatternRecognition` and `Filter`) are all assigned to the GPU1; and the sub-systems Communication (`LongRangeCom`, `ShortRangeCom`) and Mission Management (`MissionManager`, `Coordinator`) are firstly assigned to the CPU.

During execution, the mechanisms injected by the `TimingVerifier` and the aspects used by it - `Jitter` and `ClockDrift` - will start to generate values related to timing measurements. The `Jitter` and `ClockDrift` will take more time to generate more confident values, maybe after 100 executions in other to provide more meaningful measurements. But in an overall view, the `TimingVerifier` aspect will provide data to the `TaskAllocationSolver`, which will take data from the `NodeStatusRetrieval` and with the reasoning mechanisms that were inserted in the subsystems will analyze the data provided by these two aspects according to the algorithm introduced in III.B.

G. SAR Image Processing

A special attention has to be given to the Image Processing sub-system. It is considered to be the group which requires more processing from the execution platform due to work with large data, being a key-factor to influence the dynamic reconfiguration of tasks. Figure 6 depicts the SAR Image Processing workflow. To summarize this figure, the captured data (brute scalar image) must be "adjusted" regarding the SAR position parameters (range and azimuth), followed by Fast Fourier Transforms (FFT) and 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 [9] to get refined explanations about the workflow.

In terms of implementation, this sub-system can normally be developed based on CPU, but it fits better as a general processing on the GPU, since it involves mainly matrices multiplications applied to each scalar of the captured data. In addition, the data is represented using a complex number format (the real -32bits- and the imaginary -32bits- parts express the amplitude and phase of the scalar), generating large data ordering of gigabytes. Then, to optimize its execution time, a common approach is data partitioning. Individual regions can be processed in parallel by the available PUs (GPU, CPU, and PPU on this case) and, at the end, composed together to obtain the final SAR Image.

As a next step, the final image is submitted to a pos-processing phase, i.e., the `PatternRecongnition` task aims to identify certain regions of interest that could contain objects specified in the mission directions as "pattern to be found". For that case, more resolution on those image parts will be needed and, consequently, new data will be generated, demanding more processing from the assigned PU(s) in order to produce the final images and extract new information (patterns). This scenario clearly influences the priority of tasks (old and new ones) since, at that moment, the new high-resolution images will have higher priorities comparing to others that became more "generic".

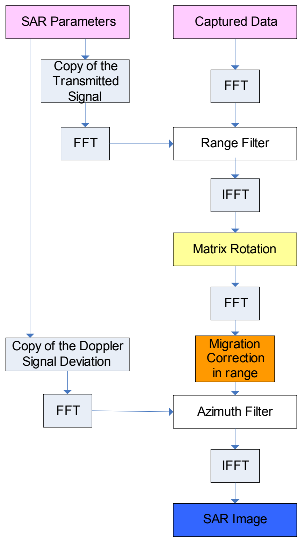

Fig. 6 – SAR Image Processing (based on the notes of [9])

These events cannot be predicted a priory and the verification of such situation require, thus, a smart and dynamic reconfiguration support to reallocate the tasks, accomplishing the timing and requirements budget. On that case, the presented approach considers not just the balance of instructions inside an UAV execution platform, but also the data/image partitioning and interaction with other UAVs serving on the mission area. In this way, the reconfiguration support is applied to sending/receiving data to be processed by other idle UAVs and, at that point, encrypted communication must be applied in short and long range.

H. Previous results

At this step of our research (reconfiguration and application architecture design), the UAV sub-systems algorithms, like Image Processing and Communication, were not implemented. The described scenario was simulated by means of creation of new tasks at run-time, where each task has an estimated cost to execute in each PU and a priority that change "on the fly". Table III exhibits the estimated costs and first priorities for the groups of tasks.

As the `TimingVerifier` aspect gathers online data about the execution of the tasks and the `NodeStatusRetrieval` gathers the PUs load parameters, the `TaskAllocationSolver` decides that the current allocation is possibly not the best configuration for the tasks that are waiting to be processed because it does not minimize the execution time. If confirmed, the reconfiguration takes place by using the `TaskMigration` aspect, which moves the tasks according to the decided new configuration presented by the `TaskAllocationSolver`. In the simplified simulation provided, evidences were gotten that when many new refined images are needed, the load-balancer tends to reallocate the Collision Avoidance tasks from the GPU2 to PPU (and then to CPU) and new instances of the Image Processing group (refined images) are assigned to be processed by GPU2 because of the Priority versus Estimated Cost compromise. In that situation, migration costs were estimated based on the throughput velocity of each PU bus (PCI, PCI-Express, etc.) and the other parameters considered on the equations (3) and (4).

Table III.
Estimation Costs for Tasks Groups

Tasks Group	Estimated Costs (scale: 1 to 6)				First Priority (scale: 1-6)
	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

Moreover, as much as new refined images are required, it was verified assignment of tasks to other UAV that was idle, performing a drastic changing on the Communication task priority. The parameter ***L*** will, then, represent the communication latency between UAVs in short and long range. As it is reasonable to consider a grater ***L*** in long range than in short range, the simulation predicted that allocation in long range is not recommended and in most of the times the waiting time to access its PUs or other UAVs PUs in short range is worthwhile.

Based on that estimation and considering 2 UAVs, Table IV denotes the behavior of the dynamic reconfigurable load-balancer simulator.

The "first guess" represents one instantiation of each group of tasks that is assigned to a PU; and with the dynamic creation of new groups (4, 8, and 12 groups) of the Image Processing tasks, the assignment is changed and optimized, trying to minimize the total execution time. Note that these values cannot represent the best assignment since the simulator did not consider all parameters that influence the whole system. As it is an ongoing work, more accurate data about the reconfiguration will be provided along the refinement of the simulator in order to represent the scenario as reliable as possible.

Table IV.
Assignment of Tasks Groups for an UAV1

Tasks Group	1 st Guess	Dynamic Image Processing Created Tasks		
		4	8	12
Image Processing	GPU1	GPU1 GPU2	GPU1 GPU2 PPU	GPU1 GPU2 UVA2-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

V. Related Works

A set of tools named VEST (Virginia Embedded System Toolkit) [10] 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. This work also provides a library of aspects and has a type of model weaving, making different kinds of analysis to compose the system, such as schedule feasibility. However, it performs statically analysis at compiling time. In our presented proposal, aspects are used to change the system configuration at runtime, adapting its behavior to new conditions faced by the running applications.

Although there are some related works concerning dynamic reconfiguration in cluster computing, like [11] and [12], our approach concentrates on reconfiguration in off-the-shelf single PUs. This way, the work from [5] implements dynamic reconfiguration methods for Real-Time Operating System services running on a Reconfigurable System-on-Chip platform based on CPU and FPGA. The methods, based on heuristics and not on time measurements, take into account the idleness of the processing units and unused FPGA area to perform the load-balance.

In the field of programmability management, [13] gives an overview of the current programming models for multi-core processors, including the RapidMind API [14], which is a commercial tool that provides an interface to the programmer and abstracts specific co-processors development libraries, extracting parallelization automatically from its code. Besides, it supports load-balance in multi-core CPU, GPU, and the Cell, but, to our knowledge, not dynamically.

VI. Conclusion and Future Works

Based on the need of non-functional parameters handling in modern applications and the advection of low-cost multi-core commodity hardware, this paper presents a new strategy of dynamic reconfiguration of tasks, involving aspect-oriented concepts to address the reconfiguration needs. It was presented real-time allocation and migration concepts applied to a modern heterogeneous execution platform.

An UAV surveillance system was used as case study and showed that modern application needs even more performance from “desktop” platforms, which are nowadays composed of several hybrid PUs. Real-time reconfiguration of groups of tasks was applied trough the UAV PUs and also considering the data sending to other UAVs.

Currently, we are defining even more suitable reconfiguration strategies and also working on finishing the implementation of the mentioned aspects that will effectively introduce the mechanisms to perform the reconfiguration in the real system. We plan to run and evaluate the tasks allocation in the platform of Figure 5 with algorithms that represents the real behavior of the tasks and according to the specific co-processors. In the same way, we will work with the tasks individually and not just with its high-level groups and refine the strategy to allocate tasks among UAVs.

Finally, we intend to extract some knowledge about which task fits better on which platform according to different scenarios, i.e., distinct types of surveillance missions established to distinct UAVs platforms, improving the support on planning such missions.

Acknowledgment

A. P. D. Binotto thanks the 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 grant given to him to follow the PhD program in Embedded Real-time Systems in Halmstad University in Sweden in cooperation with UFRGS in Brazil.

References

[1] Kiczales G. et al. “Aspect-Oriented Programming”, *Proceedigns of European Conference for Object-Oriented Programming*, Springer-Verlag, 1997, pp. 220-240.

[2] Freitas E. P., Wehrmeister M. A., Pereira C. E., Wagner F. R., Silva Jr. E. T., Carvalho F. C. DERAF: A High-Level Aspects Framework for Distributed Embedded Real-Time Systems Design. *In: Proc. 10th Int. Workshop on Early Aspects*, Springer, 2007, pp. 55-74.

[3] Wehrmeister M. A., Freitas E. P., Pereira C. E., Wagner F.R. Applying Aspect-Orientation Concepts in the Model-Driven Design of Distributed Embedded Real-Time Systems. *In: Proc. of 10th IEEE International Symposium on Object/component/serrvice-oriented Real-time Distributed Computing (ISORC'07)*, Springer, 2007, pp. 221-230.

[4] A. Burns et al. “The Meaning and Role of Value in Scheduling Flexible Real-Time Systems” *in Journal of Systems Architecture: the EUROMICRO Journal*. vol.46, n.4, 2000, pp.305-325.

[5] Götz Marcelo; Dittmann, Florian; Xie, Tao: Dynamic Relocation of Hybrid Tasks: A Complete Design Flow**.** *In: Proceedings of Reconfigurable Communication-centric SoCs (ReCoSoc'07)*, Montpellier, 2007, pp. 31-38.

[6] Stuart D. M. "Sensor Design for Unmanned Aerial Vehicles" *In Proc of IEEE Aerospace Conference*, 1997, pp. 285-295.

[7] Skolnik, Merrill I., Introduction to Radar Systems, McGraw-Hill, 3rd ed., 2001.

[8] The GNU Project, “GLPK – GNU Linear Programming Kit”, http://www.gnu.org/software/glpk/, Jun. 2008.

[9] Cumming, Ian; Wong, Frank. Digital Processing of Synthetic Aperture Radar Data. Artech House-London, 2005.

[10] Stankovic, J.A. et al., “VEST: Aspect-Based Composition Tool for Real-Time System”, in *Proc. of 9th IEEE RTAS*, 2003, pp. 58-59.

[11] Avresky, Dimiter; Natchev, Natcho; Shurbanov, Vladimir. Dynamic Reconfiguration in High-Speed Computer Clusters. *In Proceedings of the IEEE International Conference on Cluster Computing*, 2001, p. 380.

[12] Avresky, Dimiter; Natchev, Natcho. Dynamic Reconfiguration in Computer Clusters with Irregular Topologies in the Presence of Multiple Node and Link Failures. In *IEEE Transactions on Computers*, 2005, vol. 54, no. 5, pp. 603-615.

[13] McCool, Michael. Scalable Programming Models for Massively Multicore Processors. *Proceedings of the IEEE*, 2008, vol. 96, no. 5, pp. 816-831.

[14] McCool, Michael. Data-parallel Programming on the Cell BE and the GPU using the RapidMind Development Platform. *In Proceedings GSPx Milticore Application Conference*, 2006.