Halmstad University Post-Print

Dynamic Reconfiguration of Tasks Applied to an UAV System Using Aspect Orientation

Edison P. Freitas, Alécio P. D. Binotto, Carlos E. Pereira, André Stork and
Tony Larsson

N.B.: When citing this work, cite the original article.

©2008 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Pignaton de Freitas E, Binotto A, Pereira C, Larsson T. Dynamic Reconfiguration of Tasks Applied to an UAV System Using Aspect Orientation. In: International Symposium on Parallel and Distributed Processing with Applications, 2008. ISPA '08.: . IEEE; 2008. p. 292-300.

DOI: <http://dx.doi.org/10.1109/ISPA.2008.69>

Copyright: IEEE

Post-Print available at: Halmstad University DiVA

<http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-2254>

Dynamic Tasks' Reconfiguration for UAV System using Aspect Orientation

Edison P. Freitas^{1,3}, Alécio P. D. Binotto^{2,3}, Carlos E. Pereira³, André Stork² and Tony Larsson¹

¹ *Sektionen för Informationsvetenskap, Data- och Elektroteknik, Halmstad University, Sweden*

² *Fraunhofer IGD / Technische Universität Darmstadt, Germany*

³ *Informatics Institute, Federal University of Rio Grande do Sul, Brazil*

{edison.pignaton, tony.larsson}@hh.se, {alecio.binotto, andre.stork}@igd.fraunhofer.de, cpereira@ece.ufrgs.br

Abstract

Many modern applications require high-performance platforms to deal with a variety of algorithms requiring massive calculations. Moreover, low-cost powerful hardware (e.g., GPU, PPU) and CPUs with multiple cores have become abundant, and can be combined in heterogeneous architectures. To cope with this, reconfigurable computing is a potential paradigm as it can provide flexibility to explore the computational resources on hybrid and multi-core desktop architectures. The workload can optimally be (re)distributed over heterogeneous cores along the life-cycle of an application, aiming for best performance. As the first step towards a run-time reconfigurable load-balancing framework, application requirements and crosscutting concerns related to timing play an important role for task allocation decisions. In this paper, we present the use of aspect-oriented paradigms to address non-functional application timing constraints in the design phase. The DERAf aspects' framework is extended to support reconfiguration requirements; and a strategy for load-balancing is described. In addition, we present preliminary evaluation using an Unmanned Aerial Vehicle (UAV) based Surveillance System as case study.

1. Introduction

In addition to timing constraints, several modern applications require high performance platforms to deal with advanced algorithm based calculations, varying from monitoring and processing of large quantities of data acquired from sensors for visualizations, image processing, etc.. The enhancement of these applications and the development of low-cost powerful and application specific hardware (for example, the GPU - Graphics Processing Unit, the Cell processor, the PPU - Physics Processing Unit, FPGA - Field Programmable Gate Array, among others) offer several execution alternatives aiming for better performance,

programmability and control. This heterogeneity is intensified with the recent multi-core CPUs.

Following this direction, low-cost inter-chip hybrid hardware architectures can also come to be used to compose adaptable execution platforms. However, at the same time, software applications must benefit from the extended performance. This requires the creation of new methods and strategies to distribute the application's workload (tasks, algorithms, application modules or other granularity) over the different processing units in order to better meet application requirements, such as performance and timeliness, without losing flexibility. In this way, reconfigurable load-balancing computing is a potential paradigm as it can provide flexibility, improve efficiency, and offer simplicity to such a hybrid multi-core architecture. In figure 1 a hypothetical system scenario is illustrated.

Fig. 1 - System overview

Assuming this new emerging challenge our aim is to design proper methodologies, strategies, and support for management of dynamic load distribution, by means of reconfiguration of “costly” tasks. The more specific intention 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, etc) which runs on a single multi-core desktop and needs to execute under time constraints and a minimal quality (performance). In addition, during execution time, the framework is

intended to monitor the tasks and provide online information giving a possible new allocation balance, i.e., a reconfiguration of tasks is done if this procedure gives a better performance for the overall scenario.

In this paper, our focus is on the very first step needed in developing the framework, i.e. handling of application requirements in a high-level design process. The approach is to take into account the requirements to find concurrency in order to tailor the load-balancing module towards optimal task distribution. For its accomplishment, we give attention to the non-functional requirements, e.g., crosscutting concerns. The handling of this kind of concerns is done by specific design elements called “aspects” (from aspect-oriented paradigm [1]) that plays an important role to improve understandability and maintainability during task (re)configuration. We complement the work with a simple strategy to assign tasks dynamically to the Processing Units (PUs), assuming that these tasks are subject to a runtime reconfiguration.

The paper is organized as follows. We start with a review of related work. Section 3 describes the requirements identification and the proposed aspects modeled using UML notation, followed by a reconfigurable workload strategy in Section 4. Composing these two concepts, Section 5 outlines an Unmanned Aerial Vehicle Surveillance system as application. Finalizing, conclusions and future work are presented, discussing the contributions.

2. Related Work

A set of tools named VEST (Virginia Embedded System Toolkit) [2] uses aspects to compose a new 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 uses a sort of model weaving, making different kinds of analysis to compose the system, such as schedule feasibility. However, it performs static analysis at compile time. In the proposal presented in our paper, 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 [3] and [4], our approach is aimed to handle a hybrid processing platform composed by different processing units where reconfiguration is performed both over and within the hardware. In this way, the work presented on [5] implements dynamic reconfiguration methods for Real-Time Operating System services running on a

Reconfigurable System-on-Chip platform. The methods, based on heuristics, take into account the idleness of the processing units and unused FPGA area to perform the load-balance and to decide about a reconfiguration of tasks in runtime.

3. From Requirements to Design

In order to achieve dynamic reconfigurable load-balancing, we investigate the use of aspect-oriented paradigms in order to address this concern from the early phases of system development, for example in a design phase related to identifying concurrency. The focus of this phase is the task decomposition and time-dependency analysis.

To support load-balancing, some key concepts are important to address specific requirements, which must be well defined to provide a better understanding of the system as a whole. We concentrate on the non-functional requirements (NFR) related to execution times, as depicted in Figure 2. Considering this approach and having the correlated context of Distributed Real-time Embedded (DRE) systems, we adopt the taxonomy published in [6], which is briefly presented in next sub-section.

Fig. 2 - NFR as basis for load-balancing

The handling of non-functional requirements and concerns hinder the system maintainability, reuse, and evolution in current approaches used for task reconfiguration. It occurs because the handling elements (such as timing requirements probes, serialization mechanisms and task migration mechanisms) are not modularized in a single or a few system elements, but spread over the system. Any change in one of these elements requires changes in different parts of the system, that besides to be a tedious and error-prone task, do not scale in the development of large and complex applications.

The observation of above mentioned problems motivates the use of 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 task allocation strategies for heterogeneous platforms due to the need of profiling each task in a different hardware scenario, affecting several elements of the application.

3.1. Non-functional Requirements

DRE systems have key non-functional requirements concerning their development. Figure 3 presents some of these key requirements, which are mainly based on the study presented in [8], on the IEEE glossary [9], and on the SEI glossary [10].

Fig. 3 - NFR requirements for DRE systems

Real-time concerns are captured by the requirements stated in the Time classification, which is divided in Timing and Precision requirements. The first concerns the specification of temporal limits for system activities execution, such as established deadlines and periodic activations. Requirements classified as Precision denote 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 sampled data is considered updated. Another key requirement is Jitter, which directly affects the system predictability since a large variance degrades system determinism.

The Performance requirements are tightly related to those presented in the Time classification and with those concentrated in the Distribution classification, leading us to decide on putting them in a distinct category. They represent requirements usually employed to express a global need of performance, like the end-to-end response time and throughput rate.

The goal of Distribution classification is to identify key requirements related to the distribution of DRE system activities, which usually execute concurrently.

For instance, these concerns address problems such as task allocation over the system nodes, as well as the communication needs and constraints.

The Embedded classification deals with concerns related to embedded systems present requirements such as memory usage, energy consumption, and required hardware area size.

It is clear that all the mechanisms related to the migration of tasks among processing units, which implements the system reconfiguration, are non-functional crosscutting concerns. This means that the migration of tasks is not a final functional behavior of any system, but it affects several elements in different ways and in different parts of the system.

Design aspects to handle Time classification issues are used to model the introduction of time parameters in the system elements. Additionally, a contribution is added by proposing new aspects that extend the ideas referred to handle the reconfiguration concerns.

3.2. Time-based Aspects

In order to support Timing and Precision requirements, the proposal is to use aspects from the DERAf framework, presented in [7] and [7]. The packages designed for these types of requirements are presented in Figure 4 and a short description of each one is provided in the following paragraphs. DERAf *Distributed Embedded Real-time Aspects Framework* is an extendable high-level aspects framework for distributed embedded real-time systems design that provides a set of aspects that covers the needs of handling of NFR described above in sub-section 3.1.

The aspects in the Timing package handles problems related to the insertion of timing attributes to active objects, such as deadline, priority and period, and the mechanisms that use this parameters, such as frequency control. Besides, it does also provide the scheduling mechanisms needed to control the execution of active objects, as well as the required mechanisms to limit the usage of shared resources.

Fig. 4 – Used packages from DERAf

The Precision package provides support for handling of concerns like jitter, by measuring the start/end of an activity and calculating the variation of this metrics, and clock drift. It also provides means to

associate time stamps to mark the validity of controlled data and control temporal limits, such as delay.

3.3. Aspects for Supporting Reconfiguration

As mentioned before, the task migration support characterizes a non-functional crosscutting concern that spread the handling mechanisms over several system elements in a non-standard way. The new aspects proposed in this work, to address this concern, are: **TimingVerifier**, **TaskAllocationSolver**.

Both use time parameters inserted by aspects part of the Timing package, described previously, and services provided by aspects from the Precision package. An additional DERAf package is used for reconfiguration decisions to take place; it is the TaskAllocation package, presented latter in this section. Figure 5 depicts the schema.

The TimingVerifier is responsible for checking if the processing units are being able to accomplish the timing requirements specified by the TimingAttributes, PeriodicTiming, ToleratedDelay and TimeBoundedActivity aspects. In order to do this, it uses 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, 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 jitter information related to the corresponding analyzed 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.

Fig. 5 - Reconfiguration Package included

The ClockDrift aspect is used by the TimingVerifier to gather information about the synchronization among the different PUs. It is used 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 “B” that does not make it worth if compared with leaving the task running in “A”.

The TaskAllocationSolver is responsible for deciding if a task will be migrated and if so to which of the available PUs. It also has to consult overload of the PU candidates, in order to decide if it is worth to perform migration. TaskAllocationSolver uses the measurements available due to the work done by the TimingVerifier.

Based on these data, reasoning about the feasibility or not of a task reconfiguration is performed. This reasoning is explained in section 4.2, where the reconfiguration strategy is shortly approached.

The reconfiguration itself and the retrieval of PUs (nodes) status are done by two other DERAf aspects, TaskMigration and NodeStatusRetrieval. 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 these two 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., via the “I’m alive” message. Before/after every execution start/end of an active object (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.

4. Reconfigurable Load-balancing

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

better, same designed features and hence would be desirable to work in the same PU.

4.1. First Assignment of Tasks

The strategy is to combine a costly method for task assignment problems with a real-time measurement procedure. For the first assignment of tasks, the modeled aspects are not used, since there are no real time measurements of tasks by this moment. Therefore, it was 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 the second approach, a set of tasks can be represented as $T = \{t_{i,j}\}$, where every task i has an implementation (considering each supported hardware) and an execution cost estimation on the PU j . The execution cost $c_{i,j}$ is in this approach simply estimated 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 PU. 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 finding a schedule solution by means of its tasks’ execution times. The constraints for our model will be the maximum workload of each processing unit (U_{max}), which are represented by

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

An objective function to minimize the total application execution time is defined, 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 as to estimate the current optimal assignment. Due to that reason, some approaches concentrate on heuristic-based methods to estimate the best assignment. 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. It works with a dynamic reconfiguration module that deals with real execution variables and interferences, leading to a possible better task assignment and consequently better performance.

4.2. Dynamic Reconfiguration

After the first assignment of tasks, the dynamic reconfiguration strategy enters an application scene in run-time. At this point, we to start with 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 another 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:

$$T_{reconfigPUnew} < T_{remainingPUold} - T_{estimatedPUnew} - T_{overhead} \quad (2),$$

where the remaining time ($T_{remainingPUold}$) and the estimated time ($T_{estimatedPUnew}$) 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 first reconfiguration invocation); and an overhead ($T_{overhead}$) that explicit the execution time of the reconfiguration itself. The relationship between $T_{remainingPUold}$ and $T_{estimatedPUnew}$ 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:

$$T_{reconfigPUnew} = T_{setupReconfigPUnew} + T_{temporaryStorage} + T_{transferRate} + T_{executionPUnew} + L \quad (3),$$

where $T_{setupReconfigPUnew}$ represents the time for setting up a new configuration on the new PU; $T_{temporaryStorage}$ contributes with the time spent to save temporal data if needed (considering shared and global memory access parameters); $T_{transferRate}$ 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; $T_{executionPUnew}$ is the measured or estimated cost of the task processed in the new PU; and finally 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 Table 1. 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 3.3. The calculation is evaluated according to available PUs and for every determined time-step just after the first assignment of tasks.

Table 1 – Task Reallocation Module Algorithm

1:ACQUIRE TIMING DATA ABOUT TASKS EXECUTION;
2:ACQUIRE DATA ABOUT PU 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 (2) AND, CONSEQUENTIALLY, EQUATION (3);
6:PERFORM THE RECONFIGURATION DECISION ;
7: MIGRATE TASKS TO PERFORM THE RECONFIGURATION WHEN APPLICABLE.

5. UAV-based Area Surveillance System

The use of the presented ideas are illustrated by a case study that consist 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 [11]. In the case study it is considered a fleet of UAVs that might accomplish missions during all the day and under whatever weather condition. The 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 and provide required data according to the mission directions. Their movements are coordinated with the other UAVs in the fleet in order to avoid collisions and also provide optimum covertures of the target area. Details about the used coordination algorithm are out of the scope of this paper. However, we assume that it requires a considerable computation effort.

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

At this point, it is important to high-light the trade-offs regarding cost, weight and size, and effectiveness of each UAV. The device as a whole may not be too big or too heavy in order to avoid much fuel

consumption and be less susceptible to be detected by counter forces sensors. It may neither have a cost that makes the project unfeasible; however, it 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 to [11].

Another interesting feature that UAVs must provide is the possibility to apply different policies to the missions, according to user intentions and other specific requirements. There are two extremes of these policies: *Device Preservation Anyhow* and *Mission Accomplishment Anyhow*. The first consists of preserve the UAVs even the mission is not accomplished. It is especially used in cases in which the devices can be involved in dangerous situations, may be even destroyed, and the information that they could gather do not worth by the cost of its destruction. In the opposite side is the *Mission Accomplishment Anyhow* policy, in which the information that the UAVs can gather and transmit to a base station is very critical and overcomes the value of the possibly loosed device. In the middle of these policies, there are a variety of other factors that imposes different levels of limits to the mission accomplishment and device preservation. Depending on the mission policy adopted, more resources can be directed to tasks linked with the movement control (when the UAV is running way from a dangerous situation) or data gathering and processing (when the information gathering is in the highest priority).

In order to run the tasks described above and meet the high-lighted requirements and constraints 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). In order to support the movement control, communication devices and embedded sensors, each UAV will be equipped with a hybrid multi core target platform which is used accordingly to the specific needs during the accomplishment of a certain mission, detailed on section 5.2.

5.1. UAV Modelled 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. A Use Cases Diagram showing the UAV functionalities is presented in Figure 6.

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

Movement Control: responsible for monitoring and control of engines and direction mechanisms, such as

flaps actuators. It is composed by two tasks. The first is called *MovementController*, which performs the calculus of the values 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.

Fig. 6 - UAV Use Cases Diagram

Navigation: control the directions of the UAV movements and sends control information to the Movement Control subsystem. It is composed by the *RouteControl* and *TargetPersuit* tasks. The first make the necessary calculation to guide the UAV through established waypoints, while the second does the same thing, but for dynamic waypoints that varies accordingly to a moving object.

Image Processing: this subsystem gathers analog image and its digitalization. It is composed by five 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 SARR. 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 *Reflectificator*, 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 last task is called *Filter*, which is responsible for filtering the radar images in order to eliminate noise due to speckle effects [12].

Communication: communication subsystem has two main tasks, the *LongRangeCom* and *ShortRangeCom*. The first provides connectivity between pairs of communication nodes over long distances, order of kilometers, while the second provides connectivity in short range, order of meters of distance. These two tasks make use of a third, called *Codec*, which code and decode transmission data.

Mission Management: this subsystem has two tasks, the *MissionManager* and the

Coordinator. The first manages the information about the mission, like required data, mission policy and resource autonomy control (e.g. remaining fuel) while the second drives the coordination with the other UAVs to avoid surveillance area overlap.

Collision Avoidance: contains two tasks, the *CollisionDetector* 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.

5.2. Reconfiguration Approach

In the experiment, the target architecture is designed to be composed of a three hybrid PUs platform: one CPU and three co-processors, two GPUs and a PPU. Figure 7 shows the desired execution platform.

Fig. 7 - Execution platform 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 4.1. In the current experimentation, 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 [13] and took into consideration the priorities of abstract tasks and its estimated execution costs in each PU, performing the following distribution: tasks from the Movement Control (*MovementController*, *MovementEncoder*), and the 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*, *Filter*) are all assigned to the GPU1; and the subsystems 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 referent 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. It occurs because of the nature of the results provided by their mechanisms, which depends on the comparison between sampled values. The TimingVerifier aspect will provide data to the TaskAllocationSolver, which will take data from the NodeStatusRetrieval and with the reasoning mechanisms that it inserted in the subsystems; it will analyze the data provided by these aspects according to the algorithm presented in 4.2.

It is interesting to emphasize the Image Processing sub-system behavior. This module can be a key-factor to influence the dynamic reconfiguration of tasks. Depending on the image processing methods utilized, the captured radar images could be, for example, divided in parts to process in parallel, increasing the total number of tasks. However, the most important factor for a dynamic load-balancing is that this module tries to identify certain regions of interest on the images, which could contain an object of interest, specified as a pattern in the mission directions. In that case, more resolution on those parts of the image will be required and, consequently, more processing will be demanded from the assigned PU in order to access and extract information. As a result, new images will be generated, partitioned (if it is the case), and processed.

The generation of new tasks mentioned above influences the priority of other tasks since at that moment, these images have higher priorities compared to the others that became more “generic”. For instance, one possibility is to assign the new images to the GPU2 and the Collision Avoidance from GPU2 to PPU (which is partially idle on first assignment). These events cannot be predicted a priori and the verification of such situation require, thus, a smart and dynamic reconfiguration in order to reallocate the tasks to accomplish with the timing and requirements budget. This situation was also simulated without implementing the image processing algorithm itself. It simulated the creation of new tasks with new priorities and high computation costs “on the fly”.

During the accomplishment of the mission, different types of processing are required for distinct demands. More image processing “power” is demanded if more details are required from certain gathered images in some situations. In other situations, more processing power related to movement control processing may be required for example in the case in which the UAV mission is governed by a device preservation policy and it is involved in a situation that represents a threat

to is integrity. As the TimingVerifier aspect gather online data about the execution of the tasks and NodeStatusRetrieval gathers the PUs load parameters, the TaskAllocationSolver decides that the current allocation is not the best configuration for the tasks that are waiting to be processed because it does neither minimize the execution time, nor meets the specified timing requirements. Thus, the reconfiguration takes place by using the TaskMigration aspect, which inserts the mechanisms that will move the tasks according to the decided new configuration presented by the TaskAllocationSolver. In the simplified simulation provided in the present work, it was taken into account a situation in which image processing required more processing power, and it was observed that as much as new refined images are needed, the load-balancer tends to reallocate the Collision Avoidance tasks from the GPU2 to PPU and new instances of the Image Processing group (refined images) were assigned to be processed by GPU2. Table 2 shows the estimated execution costs on the PUs and its priority for each group of tasks. The costs express the parameters considered on the equations (2) and (3).

Table 2 –Cost Estimations for Tasks Groups

Tasks Group	Estimated Costs (scale: 1 to 6)			Priority (scale: 1-6)
	GPU	PPU	CPU	
Image Processing	1	4	6	1
Collision Avoidance	3	2	5	2
Movement Control	2	2	3	1
Navigation	1	1	2	3
Communication	4	4	1	4
Mission Management	5	5	1	6

Table 3 – Assignment of Tasks Groups

Tasks Group	1 st Guess	Dynamic Image Processing Created Tasks		
		4	8	12
Image Processing	GPU1	GPU1 GPU2	GPU1 GPU2	GPU1 GPU2 PPU
Collision Avoidance	GPU2	PPU	PPU	CPU
Movement Control	PPU	PPU	CPU	CPU
Navigation	PPU	PPU	PPU	CPU
Communication	CPU	CPU	CPU	CPU
Mission Management	CPU	CPU	CPU	CPU

Based on above cost estimation, Table 3 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 due that the present simulator did not consider all parameters that influence the whole system. As this 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 realistic as possible.

6. Conclusions and Future Work

Based on the needs of modern applications and to enable use of abundant low-cost multi-core hardware, we introduced aspect-oriented concepts applied in such a scenario. The aspects enable us to apply a new strategy based on dynamic reconfiguration of tasks. Therefore, we also presented allocation and migration concepts applied to the hybrid execution platform. Additionally, we discussed a new strategy of dynamic reconfiguration which was implemented with simulation of tasks, i.e., the reconfiguration strategies were then simulated using estimation costs for each UAV algorithm (tasks).

Although more ambitious task reallocation can be imagined in the presented context, our contribution is focused on the importance of aspect orientation on the field of parallel and distributing processing in comparison with tradition software techniques, which do not provide a modularized and organized way to address the related issues. Looking deeply in this contribution, the use of aspects make possible to extract the handling of the measurements and migration concerns from the core system tasks, putting them in separate design elements, which, besides providing a decoupling of these crosscutting concerns from the core system elements, makes possible the applicability of scalable and reusable strategies. This can get importance as core functional elements can be added or removed from the system without side effects to the others. Besides, these functional elements can be reused in other projects without the need for changes related to concerns that are not the focus of the elements, contributing also with design patterns for parallel programming, cited in [14].

Currently, we are defining more suitable reconfiguration strategies and also implementing the aspects that introduce mechanisms to perform reconfiguration in the real system. We plan to run and evaluate the tasks allocation with algorithms that represents the real behavior of the tasks. In addition, we would like to analyze the share of processing within the UAVs (communication between UAVs).

7. Acknowledgements

A. P. D. Binotto thanks the support given by DAAD fellowship and the Programme Alþan, 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 follow the PhD program in Embedded Real-time Systems in Halmstad University in cooperation with UFRGS in Brazil.

8. References

- [1] Kiczales, G. et al. "Aspect-Oriented Programming", *Proceedings of European Conference for OOP*, Springer-Verlag, 1997, pp 220-240.
- [2] Stankovic, J.A. et al., "VEST: An Aspect-Based Composition Tool for Real-Time System", in *Proc. of 9th IEEE RTAS*, 2003, pp. 58-59.
- [3] Avresky, Dimitry; Natchev, Natcho; Shurbanov, Vladimir. Dynamic Reconfiguration in High-Speed Computer Clusters. In *Proceedings of the IEEE International Conference on Cluster Computing*, 2001, p.380.
- [4] Avresky, Dimitry; 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.
- [5] Götz, Marcelo; Dittmann, Florian; Xie, Tao: Dynamic Relocation of Hybrid Tasks: A Complete Design Flow. In: *Proceedings of ReCoSoc'07*, Montpellier, 2007, pp. 31-38.
- [6] Freitas, E.P., Wehrmeister, M.A., Pereira, C.E., Wagner, F.R., Silva Jr., E.T., Carvalho, F.C. "Using Aspect-Oriented Concepts in the Requirements Analysis of Distributed Real-Time Embedded Systems", *Proc. of IESS'07*, 2007, pp. 221-230.
- [7] Freitas, E.P., Wehrmeister, M.A., Pereira, C.E., Wagner, F.R., Silva Jr., E.T., Carvalho, F.C. DERAFA: A High-Level Aspects Framework for Distributed Embedded Real-Time Systems Design. In: *Proceedings of 10th International Workshop on Early Aspects*, Springer, 2007, pp. 55-74.
- [8] Burns, A., Wellings, A. "Real-time systems and programming languages", Addison-Wesley, 2 ed., 1997.
- [9] IEEE, "IEEE Standard Glossary", Sep. 2006, http://standards.ieee.org/catalog/olis/arch_se.html.
- [10] Carnegie Mellon Software Engineering Institute, "Online Technical Terms Glossary", Sep. 2006, <http://www.sei.cmu.edu/str/indexes/glossary/>.
- [11] Stuart, D. M. "Sensor Design for Unmanned Aerial Vehicles" In *Proc of IEEE Aerospace Conference*, 1997, pp. 285-295.
- [12] Skolnik, Merrill I., Introduction to Radar Systems, McGraw-Hill, 3rd ed., 2001.
- [13] The GNU Project, "GLPK – GNU Linear Programming Kit", Jun. 2008, <http://www.gnu.org/software/glpk/>.
- [14] Mattson, Timothy; Sanders, Beverly; Massingill, Berna. Patterns for Parallel Programming. Addison Wesley, 2005.