Dynamic Activity and Task Allocation Supporting UAV Teams in Surveillance Systems

Edison Pignaton de Freitas*, Alécio Pedro Delazari Binotto***,**
Carlos Eduardo Pereira** André Stork*** Tony Larsson***

**IDE – Halmstad University – Sweden*
(e-mail: {edison.pignaton, tony.larsson}@hh.se).
***PPGC UFRGS – Brazil (e-mail:*
cpereira@ece.ufrgs.br)
**** Fraunhofer IGD / TU Darmstadt – Germany,*
(e-mail: {alecio.binotto, andre.stork}@igd.fraunhofer.de)

Abstract: The use of Unmanned Aerial Vehicles is increasing in the field of area patrolling and surveillance. A great issue that emerge in designing such systems is the target workload distribution over a fleet of UAVs, which generally have different capabilities of sensing and computing power. Targets should be assigned to the most suitable UAVs in order to efficiently perform the end-user initiated missions. To perform these missions, the UAVs require powerful high-performance platforms to deal with many different algorithms that make use of massive calculations. The use of COTS hardware (e.g., GPU) presents an interesting low-cost alternative to compose the required platform. However, in order to efficiently use these heterogeneous platforms in a dynamic scenario, such as in surveillance systems, runtime reconfiguration strategies must be provided. This paper presents a dynamic approach to distribute the handling of targets among the UAVs and a heuristic method to address the efficient use of the heterogeneous hardware that equips these UAVs, with the goal to meet also mission timing requirements. Preliminary simulation results of the proposed heuristics are also provided.

1. INTRODUCTION

Modern surveillance systems are employing Unmanned Aerial Vehicles (UAVs) in conjunction with ground sensor nodes in order to provide meaningful information to the end-user [1]. A challenge that emerges from this trend is the dynamic allocation of targets detected by ground sensors to UAVs, which will handle the targets by performing a given activity such as tracking. In general, the UAVs used in this kind of system have different capabilities in terms of computing platform resources and sensing devices, which make them suitable to carry out missions, handling the targets presented in the surveillance area. This problem can be interpreted as inter-node allocation, in which several activities, i.e., the activities that handle different targets, have to be allocated to certain UAVs that compose the system.

Moreover, another challenge is the so-called intra-node task allocation in UAVs, since an activity is composed of several tasks. This is an important issue as each UAV is supported by heterogeneous computing units and different tasks can be allocated and processed by different units. These characteristics make impossible a pre-defined and static allocation, not only because of the hardware heterogeneity, but also because of the dynamics of the runtime scenario, where the UAVs have to perform different activities, which require new tasks to be processed. This problem becomes more challenging by the fact that the mentioned tasks are constrained by timing requirements, which crosscut different parts of the system, increasing the allocation complexity at runtime.

In fact, the handling of real-time concerns requires several mechanisms to control and monitor timing parameters and properties spread over different parts of the system. Additionally, the mechanisms related to runtime intra-node task allocation, which implements a dynamic load balancing; also affect several elements in a non-uniform way. Besides, all these mechanisms and controls are not the main goal or functionality of any system, but must be present in order to achieve a good scheduling and thus providing a better performance. As these characteristics can be classified as non-functional crosscutting concerns, they are addressed by an aspect-oriented approach presented by the authors in a previous work [2].

As a continuation of the work developed and reported by the authors in [2], this paper presents a proposal to handle the problems related to the inter- and intra-node allocation. A strategy to distribute the handling of targets among UAVs (inter-node) is described and, once the target is assigned to a UAV, a heuristic needed to manage the dynamic task allocation among heterogeneous computing units (intra-node) is proposed. The goal is to fulfill tasks' timing requirements and increase the system performance.

The reminding of the text presents in Section II the scenario description and the problem statement. In Section III, it is described the strategy to distribute alarms among UAVs (inter-node problem), followed by Section IV that presents the approach to perform task allocation over the hardware platform (intra-node problem). Simulation results are provided in Section V and a discussion about related works in Section VI. Finalizing, Section VII provides conclusions and directions towards future works.

2. SCENARIO DESCRIPTION AND PROBLEM STATEMENT

Surveillance systems are useful for different purposes, depending on the target applications. The current work focuses on area surveillance applications, used for instance in border line patrolling or for security assurance of critical areas. Based on these applications, the formal definition of the scenario and the problem specification are outlined in the following subsections.

2.1 Scenario Overview

The scenario assumed in the following is simply a large contiguous area, which is modeled as a square having an area of A km^2 partitioned in a grid containing X columns by Y rows, where X = Y (quadratic). Each cell is identified by its Cartesian coordinates (x and y), covering a given part of the surveillance area. All elements of the scenario (targets and sensors) are supposed to occupy only one cell of the grid at a time. One cell may be occupied by more than one element at the same time.

In the surveillance area, targets are considered as non-authorized vehicles or persons, or groups of them, which appear in a non-deterministic way (modeled as a Poisson distribution **P(r)**, where **r** is the number of new targets that enter in the surveillance area in a certain t time instant). A given target τ_i^k is said to be of kind "k" and having an identifier "i", which represents its entrance order in the surveillance area. Up to K possible kinds of targets may appear in the surveillance area, from k = 1,…, K. Targets are supposed to be non-static, but some may stop in a given part of the area and stay there for a non-deterministic period of time (modeled as a random time) before they continue to move. At each time "t", a target is located in a cell, is represented by ($x_{\tau i}(t)$, $y_{\tau i}(t)$). For each new target that enters in the area, its type is randomly chosen. A snapshot of the area at a given time "t" will provide a given number of targets, which is represented by k_t. The movement of the targets is considered to be with a constant speed $\|v_{\tau i}\|$, however, different targets may have different speeds. In addition, targets may randomly change the direction of their movement.

The entire surveillance system is composed by heterogeneous sensors, which have different sensing and movement capabilities. There are "S" static sensors on the ground (sn_i, i = 1,…, S) and "N" UAVs flying over the area (u_i, i = 1, …, N). It is assumed that a static sensor node on the ground is capable to detect a target when it passes in its cell. When it occurs, an alarm is issued, which is heard by all sensors nodes (static or carried by UAVs) that are positioned a given number of cells away from the alarm issuer node. This is a tunable parameter according to the range of the communication technology used by the sensor nodes. The alarms contain a timestamp with the time in which the target was detected and the position of the node. The alarms are kept issued until at least one UAV receives them.

The UAVs move autonomously over the surveillance area, according to a given movement pattern described by the final user when establishing the mission. The goal of this work is not on the movement pattern itself and readers are referred to [3] for more details about different approaches that may be applicable. The idea is that the UAVs move according to a predefined movement pattern and respond to the events and handle detected targets - by them or by the sensors on the ground - in an efficient way, combining their abilities (e.g. capability to perform a given task over a target depending of its type) and available resources.

When it comes to the available resources, UAVs that took responsibility for handling of a given target may need additional computational resources during that work. This characterizes unpredictable runtime timing conditions. This way, they can communicate with other UAVs, requiring performing part of the work that is overloading its platform, for instance, part of the image processing tasks.

In each time "t", an ui is located in a cell, represented by ($x_i(t)$, $y_i(t)$). There is a set of actions (AC) that can be performed by it in each cell (x, y) at a given time "t", which is represented by:

$$AC = \{search\ ,analyze\ ,track\ \}$$

The "search" action is associated with what the UAV has to perform in order to detect a new target. The "analyze" action is taken in order to gather detailed information about the detected target, which is in fact the processing of the sensor data, such as radar image processing. The "track" action makes the UAV capable to keep track of the targets' movement, if that kind of target was chosen to be tracked by the commands established in the mission directions.

During its movement, an UAV is continuously making decisions, communicating or performing one of the defined actions. If an UAV is neither analyzing nor tracking a detected target, it is by default searching for new targets. The UAVs can communicate with nodes up to a number of cells distant from itself. It is also a tunable parameter that may vary depending on the communication technology used in the UAVs. When an UAV detects a new target or receives an alert, it communicates with the other UAVs that are in nearby range in order to decide which one is the most likely to respond that event.

2.2 Problem Statement

Based on the described scenario, the goal is to tackle two problems that emerge from the scenario dynamicity: the decision of which UAV will respond a given issued alarm (ac-

tion/activity); and the resource allocation on each UAV platform in order to process the tasks related to the UAV action. This way, the first problem is the target handling distribution over UAVs and the second is the load-balancing of tasks intra-UAV.

As the UAVs carry different sensors that are more or less suitable for different kinds of targets and different weather conditions, the sum of these factors have to be considered to allocate a given target to the more appropriate UAV. In addition, taking into account that the UAV platform is generally composed by heterogeneous hardware, the best allocation of tasks to the available computing units need to be decided in runtime since conditions vary according to the UAV activities performed in a given time.

In order to support the solution for the considered problems, a formal definition of the sensor applicability is required for the first problem and a generic model of the hardware platform have to be provided for the second; both are formulated in the following.

1. *Sensor Applicability*

Based on the type of target, the sensor device type and status, and the weather conditions, it is possible for a UAV to determine its applicability to perform a given activity to handle a given target. This is expressed by an "applicability function", which translates the capability of a UAV (i) to employ its sensor (type j) to perform a given activity over a certain type of target (k), in a specific time instant (t).

$$\theta_{i,j,k}(t) = \begin{cases} \varsigma_i^j(t) - We_{i,j}(t), & if \quad j = k \\ 0, & otherwise \end{cases} . \qquad (1)$$

where ςji(t) represents the sensor devices type and its status over time, which is described by a linear function with parameters dictated by the type of the sensor; and is a function that estimates the degree of degradation in the measurements, offered by a sensor of type "j", due to the current weather conditions (temperature and humidity) at time "t". This function maps the weather conditions parameters into a value representing the measurement degradation factor.

2. *UAVs' Hardware Platform*

UAV systems often require high performance platforms to deal with massive calculations. Due to the enhancement of low-cost powerful hardware, such as GPUs, and the increasing power offered by multi-core CPUs, an approach that combines these processors aiming to build a UAV computing platform can offer a good solution in terms of cost and performance. However, in order to take advantage of a hybrid solution, task allocation is a challenging problem.

A good strategy to obtain a performance gain in such heterogeneous platforms is to distribute the tasks of a UAV according to its timing requirements, allocating in specific processing units, exploring parallelism. Dynamic reconfigurable load-balancing computing seems to be a potential paradigm for these scenarios since it can deal with varying runtime conditions as to (re)adapt the work balance according to dynamic changes and timing requirements. It will provide flexibility and explore the high computational performance of hybrid and multi-core architectures. Figure 1 depicts a generic overview of a hybrid computing platform, presenting the load-balancer module responsible for task allocation.

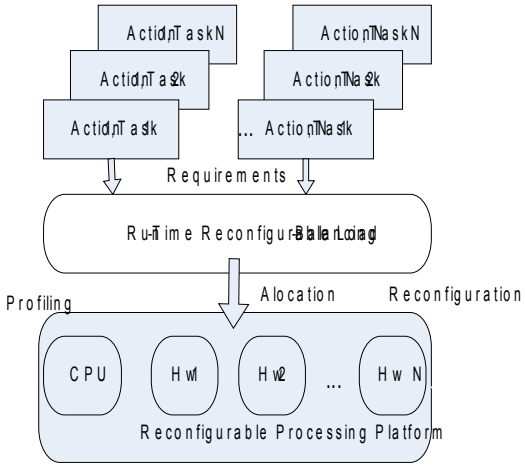

Fig. 1 – Intra-node task allocation overview.

3. INTER-NODE ALARM HANDLING DISTRIBUTION

In order to tackle the first problem, a heuristic based on the evaluation of the utility in employing a given UAV to perform a given activity for a target is used. This problem is mapped to a decision making problem, which is solved by maximizing a multi-attribute utility function. This modeling of the problem presents the three elements of a decision making theory presented in [4] and [5]. It is considered as a game in which the decision maker gathers information about the environment state and, by taking advantage of the knowledge about the conditions offered by the environment, proposes an action to be performed, which is assessed in terms of its utility, i.e., the gains that are originated from the consequences of the decision maker choice of action.

The utility function is used to evaluate the task to be performed over a target, which is defined by the commands established in the mission directions. Then, a comparison of the results provided by the utility function is assessed for each alternative. Taking the one that has the maximum value of these results, it is decided which UAV that will engage in performing the task. In the long run, this approach will maximize the use of the entire system. This reduces the problem to a maximization problem:

$$U_{\max}^{AC_j}(t) = \max\left(U_i^{AC_j}(\theta_{i,j,k}(t), C(e_i(t), p_i(t), p_j), L_i(t)\right) \qquad (2)$$

w here θ is the sensor applicability function explained above, C is the cost evaluation of remaining energy ($e_i(t)$), $p_i(t)$ is the current UAV position and p_j is the position of the target provided by the alarm. The last term, $L_i(t)$, is the current load of the hardware platform that supports the UAV, in terms of the used percentage of the total capacity.

As the computation of (1) carries a certain degree of uncertainty, due to the imprecision or incomplete information about the weather conditions, as well as the precise location of the target used in the computation of C , due to its unpredictable (unknown) movement pattern, the utility computation is done using a risk profile model, based on [6].

According to the referred work in [6], which models the behaviour of investors in the stock market using utility functions, the investors can be classified in different profiles. These profiles represent investors more or less prone to the risk when performing their trades, and they are represented by different types of functions. The complete theory includes additional details, such as coefficients to tune the degree of the risk aversion and concerns about the most suitable types of functions depending on other factors. However, in the present approach a simplified model is adopted without all the elements presented in the original theory.

The metaphor used the current proposal is to associate the idea of risk profiles of the investors to profiles that can be assigned to the UAVs in the sense that they can be more or less prone to take risks when estimating their utility to handle a given target. The UAVs that have better resource conditions and powerful capabilities are more likely to risk in computing their utility face the uncertainty of the input data, as they can expect good results (due to the good conditions), i.e. really be useful to handle a given target. On the other hand, UAVs that are "weaker" in the sense of having less capabilities and lower resources are more likely to use a more conservative utility function.

This study considers the use of two functions to express the profiles for the UAVs, a logarithmic one for the risk tolerant UAVs and a quadratic one for the conservative ones. Equation (3) shows the version for the more risk tolerant UAVs, while (4) presents the one for those less risk tolerant.

$$U_i^{AC_j}(\theta_{i,j,k}, C) = \begin{cases} \ln(\theta_{i,j,k} + e - 1) - \ln(C + e - 1) - L, \ if \ \theta_{i,j,k} > 0 \\ 0, \ otherwise \end{cases} \quad (3)$$

$$U_i^{AC_j}(\theta_{i,j,k}, C) = \begin{cases} \frac{(\theta_{i,j,k})^2 - C^2}{4} - L, \ if \ \theta_{i,j,k} > 0 \cdot \\ 0, \ otherwise \end{cases} \quad (4)$$

w here the valid interval for $\theta_{i,j,k}$, C and L is [0, 1].

4. INTRA-NODE DYNAMIC RECONFIGURATION STRATEGY

The strategy to balance tasks over the aforementioned processing units does not only consider a first balance – when the application starts – but mainly evaluates the tasks during the execution phase, i.e. at the system runtime. Executing in the background, it should profile timing performance parameters of tasks in the computing units, in order to find possible optimal allocations. This leads to a dynamic scheduling and reconfiguration due to changes in run-time conditions. The strategy is divided in two main modules: profiling and reconfigurable balancing.

4.1 Profiling

In order to achieve dynamic reconfigurable load-balancing, some specific non-functional requirements play an important role in tasks' profiling. This work proposes an aspect-oriented [7] approach to handle such requirements, as described in [2] and summarized in the following.

The profiling to support load-balancing addresses specific application requirements, which are related to the monitoring and verifying of timing parameters focused on pre-defined tasks. The handling of these requirements is spread over the system, intertwined with several other features, characterized as non-functional requirements. However, the handling of non-functional requirements will make the system maintainability, reuse, and evolution in current approaches such as pure object-oriented more difficult. It occurs since the handling elements (such as timing parameters 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 the 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 applications. The observation of these drawbacks motivates the use of an aspect-oriented approach that makes possible to address such concerns in a modularized way, which 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.

The work presented in [2] proposes the following aspects to deal with concerns related to profiling: **`TimingVerifier`**, **`NodeStatusRetrieval`** . The `TimingVerifier` aspect is responsible to measure the time performance of a task in terms of processor clocks (using the variables such as *delay* and *jitter*). `NodeStatusRetrieval` estimates information about the processing load of the each computing unit. It is implemented using a metric that gives a weight (meaning percentage of use) for each task.

4.2 Reconfigurable Balancing

The reconfiguration to balance to l oad of processing units uses the information provided by the profiling, presented above, and based on it, implements a scheduling strategy to reallocate tasks according to the new scenario characteristics.

As the reconfiguration mechanism also involves non-functional requirements, they are also handled by aspects. The first is the **`TaskAllocationSolver`** , which is responsible for the scheduling strategy and also decides if a task needs to be rescheduled to another available co-processor, using the measurements taken by

`TimingVerifier` and `NodeStatusRetrieval` . For its accomplishment, the heuristic approach (described in the following) analyses a first assignment when the application starts, and periodically performs the profiling and the reconfiguration analysis. A stability analysis is also made in order to avoid endless or costly reconfigurations. The second aspect used is the **`TaskMigration`** , which is responsible for the mechanisms that will actually perform the migration of the tasks from one processing unit to another. Figure 2 shows the relationship between the mentioned aspects; for more details, interested readers are referred to [2].

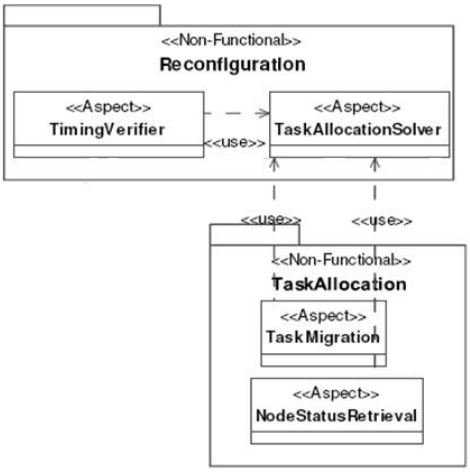

Fig. 2 – Aspects to support task allocation.

For the first assignment of tasks, the strategy is to pre-calculate the performance of each task in each processing unit in a standalone way, i.e., just one task is processed at a time in one processing unit in a pre-processing phase. In this case, estimated costs of a task in every execution unit will be gathered – but, without considering the runtime environment – and will be stored in a *Performance Database* . Then, the first schedule can be performed as a common assignment problem using Integer Linear Programming (ILP), similar to the approach used in [8].

However, the ILP problem is of NP-hard complexity and thus costly to be calculated every time in order to estimate the optimal assignment. Aiming to optimize the calculation of the assignment, some approaches concentrate on heuristics, as presented in [9] and [8].

After the first assignment, information provided by the profiler (designed using the mentioned aspects) is considered. Based on estimated costs (previously calculated using the pre-processing approach), possible changes in the runtime conditions, and new loaded tasks, one task can be rescheduled to run in another processing unit, if the estimated time to be executed in the new hardware is less than the time to be processed in the current unit.

The information used to calculate the rescheduling will then be provided by the `TimingVerifier` aspect. All these runtime parameters that could not be known beforehand may influence the execution of the system and must be evaluated periodically, leading to a large number of reconfiguration analysis and decision tasks. Then, supposing that a determined task is going to be executed *n* times in a given time window, the strategy bellow reschedules the formed queue of task instantiations, giving a relation of gain, if the following assumption occurs:

$$TreconfigPUnew < \sum_{i=1}^{n} \left(\begin{array}{l} TtaskPUold_i + TtransferPUold_i - \\ TtaskPUnew_i + TtransferPUnew_i \end{array} \right) \quad (5)$$

where *TreconfigPUnew* is the time to perform the reconfiguration (composed mainly by data transfer from the current processing unit to the new one if the task needs the data processed until the time of the 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)

5. SIMULATION RESULTS

Simulations for the proposed surveillance system were developed using Shox [10], which is a Java-based wireless network simulator, and GPLK [11], which is a tool intended to solve large scale linear programming problems. The first tool was used to perform the simulations of the alarm assignment to the UAVs, while the second was used to simulate the intra-UAV task allocation. The second tool used the output of the first in order to take the information about when a UAV was assigned with a new alarm, which triggered the execution of new tasks over the supporting platform.

5.1 Simulation Setup

In the conducted simulation, the considered surveillance area has dimensions 15 Km x 15 Km, in which 20000 ground sensor nodes are randomly deployed with independent uniform probability (homogeneous Poisson point process in two dimensions, which generates a geometrical random graph). This distribution gives around 70% probability of the nodes in the network to form a connected graph [12], for a communication range of 500 meters. Nine UAVs of three different types, equally distributed, patrol the area, having a communication range of 2 Km, and flying at speeds from 100Km/h up to 150Km/h. Two sets of simulations were performed: one with ten targets entering in the surveillance area; and another with twenty targets. A total of twenty executions were performed for each of these sets. The target entrance in the area is done according to the scenario specification provided in Section II, and it can be of five different types, randomly chosen.

The energy resources are randomly initiated with values between 90% and 100% for all UAVs. These resources are consumed according to a decreasing linear function per parts, having the time as parameter, and weighted by the current speed of the UAV in each of its parts. Sensor status is randomly started for each UAV, starting from values between

70% and 90%, and may decrement after the employment of the UAV on handling a target. It was considered that UAVs with less than 30% of remaining energy resources or less then 30% of sensor capability use the utility function presented in (4), while the others use the one presented in (3).

In the presented experiment, the computing architecture that equips the UAVs is composed of one CPU and two types of co-processors, two GPUs running CUDA. Figure 3 shows the execution platform, where the Profiling gathers information from the computing units and the Reconfiguration distributes the tasks.

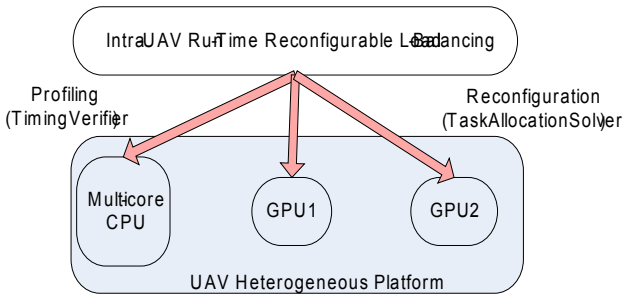

Fig. 3– UAVs Execution Platform with support for load-balancing.

5.2. Inter-node Alarm Distribution

In order to evaluate the efficiency of the alarm handling distribution among the UAVs, the mean utility achieved by the use of the proposed approach described in Section III was compared with the optimum value, which represents the best utility fit UAV-target, i.e. use the most suitable UAV to handle a given target. This is achieved by means of a global knowledge of the system (an oracle view).

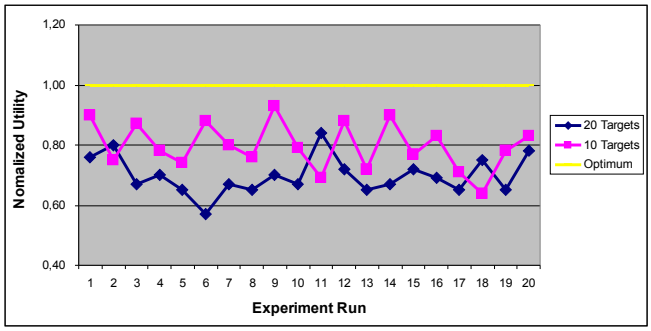

Fig. 4– Normalized utility.

Figure 4 presents a plot of achieved values for each execution, normalized in relation to the optimum value. It is possible to observe that the presented heuristic achieved good results in both sets of runs. In average, the set of ten targets achieved 80% of the optimum allocation, while the set of twenty achieved 70%. This difference can be explained by the fact that with more targets coming into the area, it is probable that more UAVs are busy by the time of a new target appearance. However, this behavior was expected and it does not represent a problem, as it is a matter of making the right tuning of the system in terms of number of UAVs and expected target entrance ratio.

5.3 A.Intra-node Resource Allocation

At the current stage of the project (reconfiguration and application architecture design), the actual UAV sub-systems algorithms, like Image Processing and Communication, were not implemented. The described scenario was simulated by creating new tasks at run-time, where each task has an estimated cost to execute in each processing unit and a priority, which changes "on the fly". Table I exhibits the estimated costs and first priorities for the groups of tasks.

Table 1 . Estimation Costs for Tasks Groups

Tasks Group	Estimated Costs (scale: 1 to 6)			Priority (scale: 1-6)
	GPU 1	GPU2	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

As the `TimingVerifier` aspect gathers online data about the execution of the tasks and the `NodeStatusRetrieval` gathers the computing units' load parameters, the `TaskAllocationSolver` decides that the current allocation is possibly not the best one for the tasks that are waiting to be processed. If confirmed, the reconfiguration takes place by using the `TaskMigration` aspect, which moves the tasks according to the new configuration decided by the `TaskAllocationSolver`. In the provided simulation, evidences were gotten that when many new refined images are needed, the load-balancer tends to reallocate the Collision Avoidance tasks from the GPU1 to GPU2 (and then to CPU) and new instances of the Image Processing group (refined images) are assigned to be processed by GPU1 due to the analysis "Priority versus Estimated Cost". In this situation, migration costs were estimated based on the throughput velocity of each computing unit bus (e.g., PCI, PCIExpress) and the parameters considered on the equations (3) and (4). For that estimation, Table II denotes the behavior of the dynamic reconfigurable load-balancer simulator in one of the UAVs that compose the fleet.

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 2 . Estimation Costs for Tasks Groups

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

6. RELATED WORKS

In [13], a proposal to handle the problem of balance between target search and response by a team of UAVs is provided. The work evaluates the tradeoff between search and response within a framework, presenting a predictive algorithm that provides a great balance between these tasks. The first difference between our approach and this related work is that we handle only the response to and motion following alarms, abstracting the UAVs movement planning to perform the search for new targets. This means that our focus is in the work load distribution among the UAVs, which is represented by the distribution of the alarms among them. Besides, this difference is also motivated by the peculiarity of the two different missions in the context of the two works. We focus in area surveillance, while they focus on target acquisition. In the first, all the area most be covered, without the assumptions of preferred locations to move, which is true in the target acquisition they address

Although there are works on dynamic reconfiguration in cluster computing, like [14], our approach concentrates on task rescheduling in single desktop PUs. In this field, the work of [8] implements dynamic reconfiguration of operating system services for a platform composed of CPU and FPGA. The methods, based on heuristics, take into account the idleness of the PUs and unused FPGA area to perform the load-balance.

In addition, the programming models described on [9] give an overview of current approaches targeting multi-core processors, including the commercial RapidMind API. It is a C-like model that abstracts the specific co-processors' APIs, generating target code dynamically for the supported PUs (CPU, GPU, and the Cell) just before the first execution of the application starts. At this time, it also automatically balances the workload (RapidMind parts of code) through the available PUs using heuristics that assume certain PU idleness, increasing the application performance. Our approach is complementary; introducing dynamic rescheduling strategies designed using aspects orientation and based on PUs profiling and task performance over the application lifetime.

8. CONCLUSION AND FUTURE WORKS

This paper presented heuristics to solve two problems presented in surveillance systems: the distribution of targets among the UAVs; and the intra-UAV load-balancing of tasks that are required to perform the activities related to the target handling. Simulation results show the suitability of the proposed approach to address the related problems as it can promote a better task scheduling over the processing unit at runtime and assign dynamically the alarms to UAVs that are able to efficiently handle it.

Currently, we are working in unifying the two modules of the simulation in a unique tool and trying to define other reconfiguration strategies that can be even more suitable to address the described problems. Moreover, the distribution of tasks related to the handling of a given target across UAVs is being analyzed. This feature would make it possible for a UAV to send a task to be performed by another idle UAV in its communication range. Preliminary simulation results of this task distribution are reported in [15], but this is a feature simulated separately, and thus it is not included in the framework described in the current paper.

ACKNOWLEDGMENT

E. P. Freitas thanks the Brazilian Army for the grant to follow the PhD program in Embedded Real-time Systems at Halmstad University in cooperation with UFRGS.

A. P. D. Binotto thanks the support given by DAAD fellowship and the Programme Alßan scholarship no. E07D402961BR.

REFERENCES

[1] Erman, A.T., Hoesel, L., Havinga, P.: *Enabling Mobility in Heterogeneous Wireless Sensor Networks Cooperating with UAVs for Mission-Critical Management.* IEEE Wireless Communications. Vol. 15, Issue 6, 2008, pp. 38-46.

[2] Freitas, E. P., Binotto, A. P. D., Pereira, C. E., Stork, A. and Larsson, T.: *Dynamic reconfiguration of tasks applied to an UAV system using aspect orientation,* Proc. of The 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA-08), Sydney, Australia, Dec. 10-12, 2008.

[3] Gaudino, P., Schargel, B., Bonabeu, E., Clough, B. T.: *Swarm Intelligence: a New C2 paradigm with an Application to Control of Swarms of UAVs.* Proceedings of 8th ICCRTS Command and Control Research and Technology Symposium, 2003.

[4] Keeney, R. L., Raiffa, H.: *Decision with Multiple Objectives: Preferences and Value Trade-offs.* John Wiley & Sons, 1976.

[5] Raiffa, H.: *Decision Analysis.* Addison-Wesley, 1970.

[6] Luemberger, D.: *Investiment Science,* Oxford, 1998.

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

[8] Götz, M., Dittmann, F., Xie, T.: *Dynamic Relocation of Hybrid Tasks: A Complete Design Flow.* In: Proceed-

ings of Reconfigurable Communication-centric SoCs (ReCoSoc'07), Montpellier, 2007, pp. 31-38.

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

[10] Lessmann, J., Heimfarth T., Janacik, P.: ShoX: *An Easy to Use Simulation Platform for Wireless Networks.* In Proceedings of Tenth International Conference on Computer Modeling and Simulation, 2008, pp. 410-415.

[11] The GNU Project, "*GLPK – GNU Linear Programming Kit",* http://www.gnu.org/software/glpk/, Jun. 2008.

[12] Bettstetter, C.: *On the minimum node degree and connectivity of a wireless multihop network.* in MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing, ACM, New York, NY, USA, 2002, pp. 80–91.

[13] Jin, Y. Liao, Y., Minai, A. A., Polycarpou, M. M.: *Balancing Search and Target Response in Cooperative Unmanned Aerial Vehicle (UAV) Teams,* IEEE Transactions on System, Man, Cybernetics-Part B: Cybernetics, vol. 36, nr. 3, p. 571-587, 2006.