Halmstad University Post-Print

Dynamic Reconfigurable Task Schedule Support towards a Reflective Middleware for Sensor Network

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, Stork A, Larsson T. Dynamic Reconfigurable Task Schedule Support towards a Reflective Middleware for Sensor Network. In: Proceedings of the 2008 International Symposium on Parallel and Distributed Processing with Applications: 10-12 December 2008, Sydney, NSW, Australia. Los Alamitos, Calif.: IEEE Computer Society; 2008. p. 886-891.

DOI: <http://dx.doi.org/10.1109/ISPA.2008.70>

Copyright: IEEE

Post-Print available at: Halmstad University DiVA
<http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-2309>

Dynamic Reconfigurable Task Schedule Support towards a Reflective Middleware for Sensor Network

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

¹ Information Science, Computer & Electrical Engineering School, 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

Sensor networks are being applied in several emerging sophisticated applications due to the use of powerful and high-quality sensor nodes, such as radars and visible light cameras. However, these nodes need additional features to optimally benefit from heterogeneous modern computing platforms. Therefore, reconfigurable computing is a potential paradigm for those scenarios as it can provide flexibility to explore the computational resources on that kind of high performance computing system. This paper presents a reconfigurable sensor node allocation support, based on application requirements, provided by a middleware focused on heterogeneous sensor networks. In order to address this concern, an aspect-orientation paradigm and intelligent agents approach is proposed followed by an UAV case study.

1. Introduction

Sensor network applications are becoming more complex due to the use of different kinds of mobile and sophisticated sensors, providing advanced functionalities [1] to be executed in dynamic scenarios where context-awareness is needed [2]. To support those emerging applications, an adaptable and flexible underlying infrastructure is necessary. Current state-of-the-art approaches suggest the use of middleware, such as the TinyDB [3]. However, this kind of middleware presents the following important drawbacks that could lead to be not useful in such emerging applications: (i) the assumption that the network is composed only by a homogeneous set of basic or very constrained low-end sensors; and (ii) the lack of intelligence and reflection of such network that compromises its required adaptability to face in changing operation conditions, e.g., lack of QoS management and control.

Based on that, adaptability is a major concern that must be addressed due to runtime variations that can

occur in dynamic scenarios where the network is deployed. However, pure adaptation cannot give an effective response to high dynamic behaviors, such as those that occurs during reconnaissance missions in which Unmanned Aerial Vehicles (UAVs) carrying sophisticated sensors are used. During such missions, a quick response to dynamic events, such as changes in user's requirements, changes in environmental conditions and unpredictable processing power demands can occur and must be handled in real-time. The above mentioned characteristics require, though, an in-system intelligence to manage those concerns without human operators' intervention.

This paper presents service schedule reconfiguration support, which is part of a reflective middleware, intended to base sophisticated sensor network applications that must adapt its behavior according to environment and application changing demands. In general, our middleware is based on the missions' specification by the users to be accomplished by the network using a high-level language in which they describe the desired data and constraints on its gathering, representing goals of the mission. Thus, in order to promote the missions accomplishment, the network could change its behavior and nodes have to change their internal task allocation or even ask for additional resources that are idle in other nodes, characterizing a system reconfiguration. Our research proposal aim to use aspect-orientation [4] to weave additional behaviors and offer reconfiguration by intelligent software agents to provide the analysis and reasoning needed to decide about the necessary changes. Based on that scenario, this paper focuses on the middleware schedule reconfiguration support of high performance nodes, such as UAVs provided with radars or visible light cameras, in order to accomplish with system real-time parameters. The text is organized as follows. Section 2 presents an overview of the proposed middleware, followed by Section 3 that

presents the agents approach that provide reflection and adaptation. Section 4 describes related works, and finalizing, Section 5 discuss concluding remarks and directions for future work.

2. Middleware Overview

The general goal of this research is to develop a flexible middleware that can be used to support applications based on heterogeneous sensor networks in terms of timeliness. In that context, *heterogeneity* means that network nodes may have different sensing capabilities, computation power, and communication abilities, executing on different hardware and operating system platforms. The proposed middleware should fit in both *low-end* and *rich* sensors. To accomplish with that achievement, aspect and component oriented techniques will be used in a way similar to the approach presented in [5] and [6], and the mobile multi-agents approach of [7].

Low-end sensors are those with simple capabilities, such as piezoelectric resistive tilt sensors, needing limited processing support and communication resource capabilities. Rich sensors comprehend powerful devices like radar, visible light cameras or infrared sensors that are supported by moderate to high computing and communication resources. Thus, in order to deal with these distinct capabilities, the proposed middleware must be lightweight, while being scalable, flexible, and customizable. For instance, it might handle the node's resource usage in order to assist tasks distribution among different nodes that are capable to accomplish with its processing. The mobility characteristic is also related to the heterogeneity addressed by the middleware.

The input to the sensor network system, coordinated by the proposed middleware, is seen as a “mission” that the whole network has to accomplish. In order to do that, a high-level Mission Description Language (MDL) is being designed. This tool will allow the data specification in which the user is interested, including constraints, such as timing and desired accuracy. This high-level user information will be translated to system parameters, such as QoS related parameters.

The middleware is divided in three layers, indicating they are partially connected (shared functions) in a specific order. Figure 1 presents the layers' overview. The bottom layer is called **Infrastructure Layer** and is responsible for the interaction with the underlying operating system and for the sensor node resources management, like available communication capacities, remaining energy, and sensing capabilities.

The intermediate layer is called **Common Services Layer** Layer. It provides services that are common to

different kinds of applications, such as QoS negotiation and control, and data assurance quality.

The top layer is called **Domain-Services Layer** and has the goal to support problem-domain specific needs, such as data fusion and specific data semantic support.

Figure 1. Middleware Layers Overview

In powerful nodes, i.e., with more energy available, the middleware can provide more complex services aiming at the handling of rich data, such as those related to image processing and pattern matching. This also means that such nodes can take some of the computational and storage burden from meager nodes.

The “smile faces” presented in Figure 1 represent agents that can provide specific services in a certain node at a certain system runtime moment. A special region - called *agents-space* - links the agents vertically throughout the layers, allowing information exchange. For instance, the Infrastructure Layer has just one agent, which is responsible for planning and reasoning of activities.

The “locks” and “keys” in the left-side plane of the figure represent the concept of security, which is a concern that affects all layers. Moreover, crosscutting concerns will be addressed in our middleware using aspect-oriented approach, such as presented in [5] and [6].

In addition, the “lightning bolts” in Figure 1 represent communication among internal elements of the middleware (light lightning bolts) and between applications over the middleware (upper dark lightning bolts). There is also the communication between the middleware and the underlying platform (bottom upper dark lightning bolts).

3. Real-time Reconfiguration Support

Our approach provides reconfiguration support for high performance nodes that process rich data information, such as radar images. The middleware

should monitor the execution of applications' tasks, verifying if the designed timing requirements are being met. When the requirements are not being accomplished, the middleware deploys a dynamic reconfiguration that fits the requirements, moving tasks from a processing unit to another, re-scheduling them.

To monitor the real-time parameters, we use the Timing and Precision packages from the DERAf framework [5], which is an aspect library and it is focused on the handling of Distributed Real-time Embedded non-functional requirements and provide runtime execution control.

The processing loaded in each processing unit (PU) is monitored by the mechanisms provided by the NodeStatusRetrieval aspect included in the DERAf TaskAllocation package. Based on that constant monitoring, an intelligent agent module reasons about the best configuration to be deployed, performing eventual task allocation reconfigurations in order to fulfill the real-time requirements. The reconfiguration itself is done by the use of mechanisms inserted by the TaskMigration aspect (also from DERAf TaskAllocation), as will be explained in the following. Readers are strongly encouraged to get further details about DERAf in [5], since it will be the basis for the next sub-section.

3.1. Aspect-based Real-time Parameters Monitoring

The verification of real-time parameters accomplishment is clearly a non-functional crosscutting concern, which affects different functional elements of a system in distinct parts and ways. This claim is corroborated by the fact that the monitoring activity has to spread measurement control mechanisms in the beginning and end of elements (tasks) as well as compute elapsed times of different events; and be aware about start and end of executions of all affected elements. Based on that fact, the use an aspect-oriented approach is well suitable to promote the appropriate handling of this concern.

In the present approach, an aspect called **TimingVerifier** is designed focusing the handling of the previous mentioned concern. This aspect uses the behaviors of two other ones from DERAf, the Jitter and ClockDrift. The first measures the start/end of an activity (i.e., task execution), calculates the variation of this metric and whether the tolerated variance was overran. The second measures the time at which an activity starts and compares it with its expected beginning; it also checks if the accumulated difference exceeds the maximum tolerated clock drift.

The **TimingVerifier** is responsible for checking if the processing units are being able to accomplish with the timing requirements specified by the Timing package aspects (TimingAttributes, PeriodicTiming, ToleratedDelay, and TimeBoundedActivity). Basically, these aspects concentrate in timing parameters such as deadlines, periods, activation time, and tolerated delay. Though, the **TimingVerifier** checks the accomplishment of timing requirements inserting a mechanism to control the meeting of timing attributes in the beginning and end of each task. This mechanism consists of times measuring, and comparing them with the requirement specified by the corresponding 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 information about the jitter 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.

In addition, the **ClockDrift** aspect is used by the **TimingVerifier** to gather information about the clock synchronization among the different processing units. It is useful to calculate the cost, in terms of a task migration time. As an example, consider a task that was migrated from a PU "A" to a PU "B". The difference in the clock time caused by the 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".

3.2. Agent Reasoning

Agent-based reasoning is responsible to perform the decision about reconfiguration of task allocation. It is assumed that the system has a number of heterogeneous PUs that can better perform a specific type of task. However, in certain situations of overload, the system may require the use of other processing units to balance the overall system load and meet the real-time constraints specified by the application requirements.

The planning-agent, located inside the agents-space of the middleware Infrastructure Layer, perform the reasoning. The reasoning consists in a "first guess" to configure the system during its startup and in a dynamic task schedule reconfiguration during system runtime, as presented in the following.

3.2.1. First Assignment of Tasks

For the first assignment of tasks, we do not use the modeled aspects, since the tasks' real time measurements are unknown. Therefore, two possibilities are offered: assign all the tasks in the first time step to the CPU and then perform the dynamic reconfiguration; or a more reasonable model is to perform the first assignment in a pre-processing phase as a common assignment problem using Integer Linear Programming (ILP), like the generic approach used by the authors of [8]. In that approach, a set of tasks has an implementation and an execution cost estimation on each supported PU. The total execution time of the application is minimized finding a schedule solution by means of its tasks execution times.

However, 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, as the above mentioned work of [8]. 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 PUs (using aspects) and works with a dynamic reconfiguration module that deals with real execution variables, possibly leading to a better task assignment.

3.2.2. Dynamic Reconfiguration

After the first assignment, information provided by the created aspects is considered. 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 $T_{reconfigPUnew} < T_{remainingPUold} - T_{estimatedPUnew} - T_{overhead}$ (3),

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 of 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:

$$T_{reconfigPUnew} = T_{setupReconfigPUnew} + T_{temporaryStorage} + \frac{T_{transferRate}}{T_{executionPUnew}} + L \quad (4),$$

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; $T_{executionPUnew}$ symbolizes 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 (from DERAf). These preliminary stored times will be useful as one of the basis to the reconfiguration decision done by other created aspect, 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 3.1.

Table 1 - 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 2 - 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.

4. Case Study

The application scenario is a surveillance sensor network in which a fleet of UAVs make part as mobile nodes carrying sophisticated sensor, such as Visible Light Camera (VLC), Synthetic Aperture Radar (SAR), and Infra-Red Camera (IRC). The UAVs must be able to provide different levels of information definition and detail, depending on the users' requirements.

At the beginning, 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. Each UAV is composed by six subsystems that make it be able to accomplish its mission and coordinate with the others UAVs. These subsystems are: Collision Avoidance, Movement Control, Communication, Navigation, Image Processing, and Mission Management. Although details about the used algorithms in each subsystem are out of the scope of this paper, we give later a special attention to the Image Processing subsystem.

Moreover, in order to support the system described above and meet the high-lighted requirements and constraints, each UAV will be equipped with a hybrid processing unit platform which is used accordingly to the specific needs during the accomplishment of a certain mission, detailed in the following.

4.1. Execution Platform

The target architecture is composed of a four heterogeneous PUs platform: one CPU and three types of co-processors, two GPUs (CUDA-FFT/BLAS toolkit), a PPU (using PhysX SDK), and a PCICC (UDX toolkit). Figure 2 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).

Figure 2. Execution platform

4.2. Reconfiguration Approach

Starting the mission, the UAVs have an initial task allocation throughout the CPU and the PUs platform according to one of the approaches presented on subsection 3.2.1. In the current experimentation, it was

considered to use the ILP approach for the first distribution using the GLPK toolkit [9]. Table 3 exhibits the estimated costs and first priorities to feed the GLPK-based simulation of task scheduling.

Table 3 – Task Estimation Costs

Tasks	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

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 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 provided data according to the algorithm introduced in 3.2.2.

Taking attention to the Image Processing subsystem, it is considered to be the group which requires more processing from the execution platform due to work with large data and due to new instantiations created “on the fly”. In summary, the captured data must be “adjusted” regarding the SAR position parameters to produce the final image. Afterwards, the final image is submitted to a pos-processing phase 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 new images and extract new information (patterns). This scenario clearly influences the priority of tasks since, at that moment, the new high-resolution images will have higher priorities comparing to others that became more “generic”.

These events cannot be predicted a priori and the verification of such situation require, thus, a smart and dynamic reconfiguration support to reallocate the tasks, accomplishing the timing and requirements budget.

Thus, considering 2 UAVs, Table 4 denotes the behavior of our dynamic reconfigurable load-balancer simulator. The “first guess” represents one instantiation of each group of tasks that is assigned to a PU; and with dynamic creation of new groups (4, 8, and 12) of the Image Processing tasks, the assignment is changed

and optimized to minimize the total execution time. Note that these values cannot represent the best assignment since our 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 our simulator in order to represent the scenario as reliable as possible.

Table 4 – Task Assignment

Tasks	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

5. Related Work

The AWARE project [10] proposes a middleware to provide integration of the information gathered by different types of sensors, including WSN (Wireless Sensor Network) and mobile robots. In addition to that approach, our work also deals with QoS concerns and runtime reflection to address changes during system runtime.

One method for schedule reconfiguration of tasks was presented in the work of [8], but 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 runtime time measurements, take into account the idleness of the processing units and unused FPGA area to perform the load-balance. Besides, they are not designed within a Middleware, as in the present proposal.

6. Conclusion and Future Works

An automatic and runtime task schedule reconfiguration strategy provided by a reflective middleware for heterogeneous sensor networks was presented. It allows an efficient resources use offered by the heterogeneous computer platform and provides compliance with the real-time constraints specified by the application requirements; faced that runtime conditions changes due to the scenario dynamicity. For this achievement, a proposal using aspect-orientation in conjunction with intelligent agents was designed to address the needs for adaptation under concern.

Future directions focuses the refinement of the reconfiguration strategy implemented by the intelligent

agent, as well as aims to provide more complete simulations, considering a large range of runtime parameters and real UAV subsystem algorithms, emphasizing the Image Processing dynamicity.

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] D. Culler, D. Estrin, and M. Srivastava, "Overview of sensor networks," *IEEE Computer*, vol. 37, no. 8, pp. 41–49, 2004.
- [2] K. Henriksen and J. Indulska. "A software engineering framework for context-aware pervasive computing". *Proc. of PerCom*, IEEE Computer Society, pp. 77–86, 2004.
- [3] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. "TinyDB: An acquisitional query processing system for sensor networks". *ACM Transactions on Database Systems*, 30(1):122–173, 2005.
- [4] Kiczales, G. et al. "Aspect-Oriented Programming", *Proc. of European Conference for Object-Oriented Programming*, Springer-Verlag, 1997, pp 220-240.
- [5] E. P. Freitas, M. A. Wehrmeister, C. E. Pereira, F. R. Wagner, E. T. Silva Jr., F. C. Carvalho. "DERAF: A High-Level Aspects Framework for Distributed Embedded Real-Time Systems Design". In: *Proc. of 10th International Workshop on Early Aspects*, Springer, 2007, pp. 55-74.
- [6] A. Tesanovic, et al. "Aspects and Components in Real-Time System Development: Towards Reconfigurable and Reusable Software", *Journal of Embedded Computing*, IOS Press, v.1, n.1, 2005.
- [7] E. P. Freitas, P. Söderstam, W. O. Morais, C.E. Pereira, T. Larsson. "Adaptable Middleware for Heterogeneous Wireless Sensor Networks". *Proc. of 10th EASSS'08*, 2008.
- [8] Götz, Marcelo; Dittmann, Florian; Xie, Tao. "Dynamic Relocation of Hybrid Tasks: A Complete Design Flow". In: *Proc. of Reconfigurable Communication-centric SoCs (ReCoSoc'07)*, Montpellier, 2007, pp. 31-38.
- [9] The GNU Project, "GLPK – GNU Linear Programming Kit", <http://www.gnu.org/software/glpk/>, Jun. 2008.
- [10] P. GIL et al. "Data centric middleware for the integration of wireless sensor networks and mobile robots". In *Proc. of 7th ROBOTICA'07*. 2007.
- [11] D. C. Schmidt et al. "A Decision-Theoretic Planner with Dynamic Component Reconfiguration for Distributed Real-Time Applications". *Proc. of 8th ISADS'07*. 2007. pp.461-472.