IPP-HURRAY! Research Group

Polytechnic Institute of Porto
School of Engineering (ISEP-IPP)

Mechanisms for Reflection-based Monitoring of Real-Time Systems

Ricardo BARBOSA
Luis Miguel PINHO

HURRAY-TR-0420
July-2004

relatório técnico

technical report

Mechanisms for Reflection-based Monitoring of Real-Time Systems

Ricardo BARBOSA, Luis Miguel PINHO

IPP-HURRAY! Research Group
Polytechnic Institute of Porto (ISEP-IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto
Portugal
Tel.: +351.22.8340502, Fax: +351.22.8340509
E-mail: { rbarbosa, lpinho}@dei.isep.ipp.pt
http://www.hurray.isep.ipp.pt

Abstract:
Monitoring is a very important aspect to consider when developing real-time systems. However, it is also important to consider the impact of the monitoring mechanisms in the actual application. The use of Reflection can provide a clear separation between the real-time application and the implemented monitoring mechanisms, which can be introduced (reflected) into the underlying system without changing the actual application part of the code. Nevertheless, controlling the monitoring system itself is still a topic of research. The monitoring mechanisms must contain knowledge about “how to get the information out”. Therefore, this paper presents the ongoing work to define a suitable strategy for monitoring real-time systems through the use of Reflection.

Mechanisms for Reflection-based Monitoring of Real-Time Systems

Ricardo Barbosa, Luís M. Pinho
Polytechnic Institute of Porto, Porto, Portugal
{rbarbosa, lpinho}@dei.isep.ipp.pt

Abstract

Monitoring is a very important aspect to consider when developing real-time systems. However, it is also important to consider the impact of the monitoring mechanisms in the actual application. The use of Reflection can provide a clear separation between the real-time application and the implemented monitoring mechanisms, which can be introduced (reflected) into the underlying system without changing the actual application part of the code. Nevertheless, controlling the monitoring system itself is still a topic of research. The monitoring mechanisms must contain knowledge about "how to get the information out". Therefore, this paper presents the ongoing work to define a suitable strategy for monitoring real-time systems through the use of Reflection.

1 Introduction

Not too long ago, machines were controlled by mechanical systems. Today, almost all airplanes are controlled through fly-by-wire systems and cars have already started to integrate automated driving control systems. The mechanical instruments that once controlled these systems are increasingly being replaced by complex pieces of software [1]. The problem is that the same reliability and safety that the mechanical parts provided is also expected from this software.

To fight this increase in the demand for fault tolerant and reliable software systems, in the last few years an effort was made to create new tools and theories that approach these problems in straightforward way. Fields of research go from testing techniques to software development standards. From all these research fields, one that is particularly important, and that is still much unexploited, is monitoring [1].

In order not only to perform testing for verification and validation of critical software, but also to observe the runtime behaviour of the system after deployment, monitoring services are needed that provide sufficient information about the state of the system [2]. Monitoring must be considered as a key factor in real-time systems, both during the development and deployment phases.

In [3], the motivation for the separation of the monitoring mechanisms from the application is provided. From the development process to the actual design and implementation of both real-time system and monitoring mechanisms, the advantages are considerable and must be taken into account. A clear separation in the real-time application development can be achieved, and through the use of computational reflection the desired monitoring mechanisms can be introduced (reflected) into the underlying system without changes to the application code.

Nevertheless, the concrete strategy to use for the reflection mechanisms must be further researched, in order to better understand their impact on the deployed systems. Although the use of reflection in real-time systems has already been considered [4] in order to deal with dynamic task scheduling, the underlying impact on the determinism of the system is still far from being understood. One of the important issues to also take into account is the necessity to consider features which are usually in the domain of the underlying operating system.

Therefore, this paper presents the ongoing work to define a framework for reflection-based monitoring in real-time systems. The paper is structured as follows. Section 2 presents some basic concepts of monitoring and reflection, which are later integrated in Section 3, where the reflection framework for monitoring is briefly described. Afterwards, Sections 4 and 5 outline the basic mechanisms and strategies that can be used in this framework. Finally, Section 6 provides some considerations on the required further research.

2 Basic Concepts

2.1 Monitoring

Monitoring is the collecting of run time information about the system that cannot be obtained by static analysis [5]. An important aspect to have in mind is the concept of *intrusiveness*. It is important to observe the system without influencing it, meaning that the act of observation cannot disturb in any way the system being monitored. Another

important aspect to have in mind is the non deterministic effect of observing the system through the addition of code lines (software), many times called *Heisenberg uncertainty principle* or *probe effect* [1]. In order to adequately observe the run-time behaviour of the system it is necessary to give particular attention to the impact of any additional monitoring instrumentation, so that it does not interfere with the system's behaviour (or at least that this interference is deterministic).

An important issue to consider is the clear identification of what information to monitor. A large amount of information extracted from the system, may imply extra burden in its functioning, and intrusive issues may arise [6]. On the other hand if too little information is extracted there may be a lack on precision and may not be enough to make a consistent judgement on what is actually happening in the system.

Basically, the information that can (or must) be monitored can be divided into three groups: Data Flow (internal and external), Control Flow (execution and timing) and Resources (memory and execution resources) [1]. Data Flow information concerns the inputs and outputs of each component of the system, also allowing determining what are the intermediately computed values and/or program state that are not visible through the predefined interface. Control Flow information, allows determining at what time and in what order are events received and handled in the system, to determine which, when and in what order tasks are starting, pre-empting and finishing, and to access the kernel specific scheduling and overheads. Finally, Resources information allows determining the kernel internal state, and the utilization of memory, CPU, and other system resources.

After identifying the particular information to monitor, it is also important to determine when to monitor. There are several approaches to deal with this issue, but commonly the collection of data can be triggered by events, since the system under monitoring can be described as a series of state changes. Events like thread creation or termination and context switch can be used to trigger data collection [6].

2.2 Computational Reflection

Reflection is a concept by which a component provides observation and control of its own internal structure and behaviour to the outside world [7][8]. When introducing reflection, two levels of information must be considered: *base level* and *meta or reflective level*.

The structural and behavioural information of an object model is called reified information or meta information. This information is handled by the meta-objects. The set of meta-objects in a reflective system is called the meta level and any changes on the handling of this information by the meta-objects are reflected to the associated object.

The set of objects in a reflective system is called the base level [7]. In order for both levels to work correctly, a protocol for communication must be provided. This "communication" protocol between the meta-object and object is called meta-object protocol [7]. This protocol must be predefined, thus any interaction made between the object and the correspondent meta-object is fixed. The link between the meta-object and the object can either be fixed statically at compile time (or load time), or a more flexible approach can be used by runtime linking [7][8].

Two models for computational reflection are commonly presented [8]. In the Structural model, the meta level is constituted by meta-classes. Meta-classes have the structural description of the objects at base level. When this information is modified, the structure of the objects at the base level is modified accordingly. In the Behavioural model, the objects at meta level (meta-objects) are similar to normal objects and contain all reflective information. A class of a meta-object is called meta-object class. A meta-object is activated when the corresponding base level object is invoked. When this happens, the associated meta-object executes the corresponding meta-method. This method determines the actions to be developed and passes control to the base level object. An interesting concept in this model is the Reflection Tower or N-Metalevel Architecture [7][8]: a meta-object is also an object, thus it can have an associated meta-object.

The important aspect to have in mind is the action of passing control between the object and the meta-object. For this, the messages made to the object at base level must be intercepted [2]: whenever a message is send to a base level object, the object redirects control to the corresponding meta level object. At meta level, the object executes the associated behaviour and then returns control back to the base level object. Three strategies for redirecting control are known [6]: class reflection, method reflection and object reflection.

3 The Reflection-based Framework

The reflection-based framework for monitoring [3] is a direct application of the reflective concept to real-time systems monitoring. As presented in Figure 1, the in the intended monitoring mechanism is separated from the real-time application code. Afterwards, depending on which reflective approach is used, the monitoring features are "inserted" into the system.

This separation diminishes the probability of introducing error-inducing code and also improves any debugging process by removing code. With this framework, emphasis is given to the concept of monitoring, providing a conceptual separation between the real time kernel code and the monitoring code.

Since the reflective mechanisms must be provided either by the underlying operating system [9] or by the programming language itself [10], the monitoring system programmers do not need to know how the underlying system is implemented.

Nevertheless, they do need to know the name of the classes, objects and/or methods, and the type of mechanisms that are available.

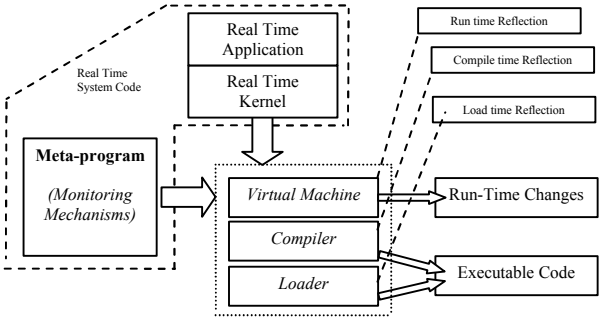

Figure 1. Reflection-based Framework

4 The Monitoring Mechanisms

The meta-program contains all monitoring mechanisms. Optimization issues must be taken into account when developing all these mechanisms. An interesting concept to have in mind is grouping. Higher levels of abstraction can be achieved if similar mechanisms are grouped. Figures 2 to 4 provide examples of the foreseen monitoring mechanisms, grouped by Data Flow, Control Flow and Resources. Each one of these groups contains the actual classes that implement the basic monitoring mechanisms.

Through the used of computational reflection concept of Reflection Tower, it would also be possible to afterwards add user defined requirements-based mechanisms to the already existing (this way actually improving the monitoring model). Nevertheless, it is still necessary to consider the impact on the system's overall timing analysis.

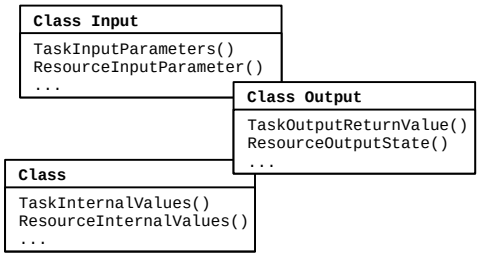

Figure 2. Data Flow group

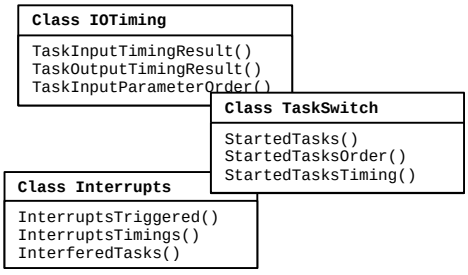

Figure 3. Control Flow group

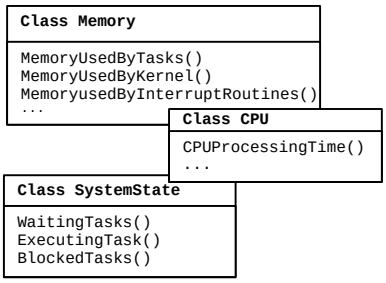

Figure 4. Resources Group

For instance, if it was decided for a particular application to add monitoring features to the Resources group, a class containing the desired mechanisms could be created and incorporated in the group. Another option would be to create a different class and use it instead of one of the already existent.

5 Strategies for Interception

In order to redirect control to the meta level, it is necessary to intercept messages passed between base level objects. This interception mechanism must be implemented either by the programming language or by the underlying (operating) system. An important issue to note is that, based on the monitoring requirements of the actual application (which information to monitor), a decision must be taken on the best fit reflective strategy. For instance, if it is necessary monitoring a particular characteristic of all tasks, class reflection should be used. This way, all tasks will acquire the reflective behaviour and consequently, the monitoring capabilities (Figure 5).

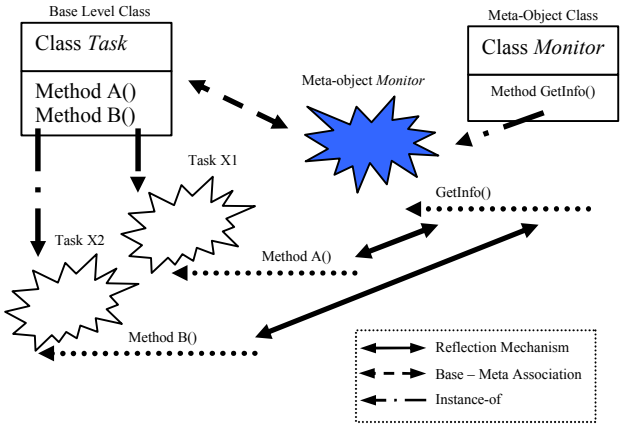

Figure 5. Using class reflection

If instead, a particular method is of interest, method reflection may be used (Figure 6). This way monitoring can be optimized. Finally, if monitoring a particular set of objects, object reflection should be used (Figure 7). For the same reasons of method reflection, this strategy also allows the optimization of code, by monitoring only the desired objects.

Although the most adequate mechanism should be chosen based on the actual subject of monitoring, it is necessary to consider that the overall analysis of the system must also be possible. Therefore, it is also necessary to analyse the used mechanisms in terms of determinism (not only time, but also memory).

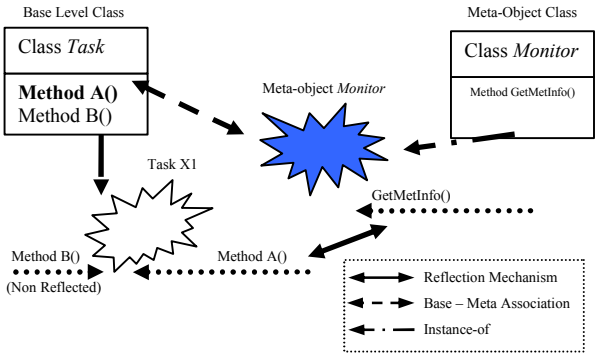

Figure 6. Using method reflection

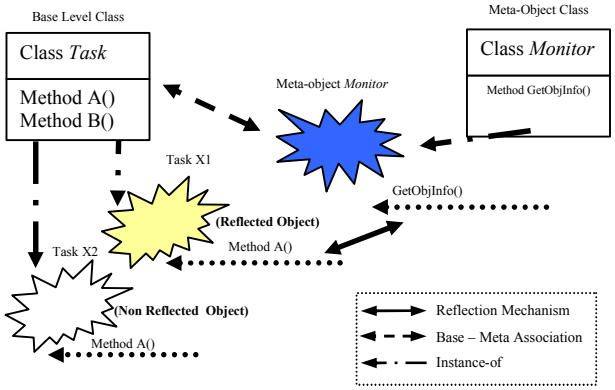

Figure 7. Using object reflection

It is not possible to provide a straightforward heuristic for a generic system. A possible approach would be to handle all objects in the system as equals, monitoring all of them similarly. Nevertheless, a better approach is to increase the quantity and quality of monitoring, based on some particular(s) constraint(s) of the monitored object or set of objects.

For the first case, monitoring strategies may be compiled into a table where, depending on the type of object, a specific strategy may be adopted. For instance, all shared resources of the system could have monitoring capabilities through class reflection strategy. In the second case, more refined criteria are used, in order to optimize the system behaviour. For instance, only particular objects, e.g. critical system tasks, could be provided with full monitoring (using object reflection). The same assumptions can be made for shared resources, etc.

Therefore, particular attributes may have impact on the used strategy. The advantages driven from the choice of strategy must thus be considered for each application and monitoring information.

6 Discussion and Future Work

The work presented in this paper is still the focus of ongoing research. Although not new, the two different concepts: monitoring and reflection, are still far from being integrated in real-time systems. Although, several advantages have already been identified, the impact of the monitoring mechanisms within the application must be further considered. In particular, the deterministic effect has not been tackled yet. It is necessary to provide bounds on the impact of the mechanisms, in order to guarantee the system's timing analysis.

Although not within the focus of this work, the use of the object-oriented approach in real-time systems is still open for discussion. Object-oriented is inherent to reflection, thus its use requires further advances on this separate line of research. It is also obvious that, in the real-time area, there are very few available technologies of use.

The current work is the enhancement of the monitoring requirements, and the design of the monitoring groups and mechanisms. Afterwards, with a full knowledge of the requirements, the needs and impact of the mechanisms will be further researched. The final phase will be the implementation of the framework.

7 References

[1] H. Thane, *Monitoring, Testing and Debugging of Distributed Real Time Systems*, Ph.D. Thesis, MRTC Report 00/15, 2000.

[2] S. Chodrow, F. Jahanian, M. Donner, Run Time Monitoring of Real Time Systems, *Proc. International Real-Time Systems Symposium*, 1991, pp. 74-83.

[3] R. Barbosa, L. M. Pinho, Monitoring of Real Time Systems: a case for Reflection?, *Polytechnic Institute of Porto Technical Report HURRAY-TR-0413*, April 2004. Available online at: http://www.hurray.isep.ipp.pt

[4] S. Mitchell, A. Wellings, A. Burns, "Developping a Real-Time Metaobject Protocol", *Proc. of the IEEE Workshop on Object-Oriented Real-Time Dependable Systems*, 1997.

[5] J. Tsai, Y. Bi, S. Yang, Smith, R., *Distributed Real-Time Systems - Monitoring, Visualization, Debugging, and Analysis*, John Wiley & Sons, New York, USA, 1996.

[6] M. Gergeleit, *A Monitoring-based Approach to Object Oriented Real Time Computing*, Ph.D. Thesis, Dec. 2001.

[7] J.C. Fabre, *Object Orientation And Fault Tolerant Systems – An Overview and Some Examples*, LAAS Report 98088, 1998.

[8] C. Marcos, *Design Patterns as First Class Entities,* Ph.D. Thesis, UNICEN University, 2001.

[9] J. Stankovic, K. Ramamritham, A Reflective Architecture for Real-Time Operating Systems, *Advances in Real-Time Systems*, Prentice Hall, 1993.

[10] S. Chiba, A Metaobject Protocol for C++, *Proc. Object-Oriented Programming Systems, Languages and Applications,* 1995, pp. 285-299.