Towards Accurate Monitoring of Extra-Functional Properties in Real-Time Embedded Systems

Mehrdad Saadatmand
XDIN AB
Stockholm, Sweden
mehrad.saadatmand@xdin.com

Mikael Sjödin
Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University, Västerås, Sweden
mikael.sjodin@mdh.se

Abstract—Management and preservation of Extra-Functional Properties (EFPs) is critical in real-time embedded systems to ensure their correct behavior. Deviation of these properties, such as timing and memory usage, from their acceptable and valid values can impair the functionality of the system. In this regard, monitoring is an important means to investigate the state of the system and identify such violations. The monitoring result can also be used to make adaptation and re-configuration decisions in the system as well. Most of the works related to monitoring EFPs are based on the assumption that monitoring results accurately represent the true state of the system at the monitoring request time point. In some systems this assumption can be safe and valid. However, if in a system the value of an EFP changes frequently, the result of monitoring may not accurately represent the state of the system at the point when the monitoring request has been issued. The consequences of such inaccuracies can be critical in certain systems and applications. In this paper, we mainly introduce and discuss this practical problem and also provide a solution to improve the monitoring accuracy of EFPs.

Index Terms—Monitoring, Extra-Functional Properties, Real-Time, Embedded, Accuracy.

I. INTRODUCTION

Successful design of real-time embedded systems depends heavily on how they behave with respect to their extra-functional properties. This is mainly due to the constraints and limitations of these systems in terms of available resources [1]. Therefore, a real-time embedded system needs to achieve its functionality under these limitations and constraints. This implies that the extra-functional properties of the system should remain within an acceptable range and violations in this aspect should be managed and prevented. For example, due to memory constraints, different components of a system should not consume more memory than expected. Similarly, in terms of timing properties, a task may not execute more than its allowed execution time budgets; otherwise, it can affect the overall behavior and functionality of the system.

Monitoring extra-functional properties is an important means in this regard not only to detect and identify violations but also to provide necessary information for runtime adaptation and reconfiguration in systems [2], [3]. However, accuracy of the monitored values plays a significant role in the correct identification of violations and also making appropriate decisions for performing runtime adaptation. The term accuracy in this paper is used in the sense that to what extent a monitored value represents the actual state of the system and the extra-functional property that is monitored at a certain time point.

One of the factors that contribute to the accuracy of monitored extra-functional properties is the time difference between the point when a request for monitoring an extra-functional property is issued and the time point when it is actually monitored and its value is obtained. This time difference (referred to as ‘monitoring time difference’ in this paper) is especially critical in systems where frequent changes of extra-functional properties can happen. An example could be when CPU load in a system changes constantly due to frequent termination and execution of different jobs or when memory usage varies rapidly because of frequent allocation and deallocation of memory. For this reason, it is also deemed necessary to consider timestamps in monitoring of extra-functional properties to be able to judge their accuracy and validity. A point to remember though is that how this time difference affects the accuracy of monitored values can change from one system and even execution scenario to another. For example, in one system, rapid changes in CPU load may happen while memory usage can remain at a certain level. In such a system the aforementioned monitoring time difference is critical for validity and accuracy of CPU load measurements while it may not be so for monitoring memory usage. In another system the situation could be the opposite of this case.

In this paper, we focus on this problem and how we can improve the accuracy of monitored values by having more control over the time difference between a request for monitoring and the actual act of monitoring. We introduce an approach which enables to reduce this time difference and thus helps with the accuracy of the monitored extra-functional properties. This is achieved by considering priorities for performing monitoring of different extra-functional properties. The approach works by performing the task of monitoring properties with higher frequency of changes (for which the monitoring time difference can greatly affect the accuracy of the obtained values) at a higher priority level than other properties. We have implemented the approach using OSE Real-Time Operating System (RTOS) [4] which is a commercial RTOS used heavily in telecommunication systems and is embedded in millions of devices around the world.

The remainder of the paper is structured as follows. In
Section II describes a short introduction to the OSE RTOS. Section III describes the general approach along with its implementation. In Section IV, an evaluation of the implemented approach is done. Discussions on several important points related to the suggested approach are done in V. In Section VI related works are mentioned, and finally in Section VII conclusions are made and future directions are explained.

II. OSE Real-Time Operating System

OSE is a real-time operating system developed by Enea designed from the ground for use in fault-tolerant distributed systems that are commonly found in telecommunication domain, ranging from mobile phones to radio base stations. It provides preemptive priority-based scheduling of tasks. OSE offers the concept of direct and asynchronous message passing for communication and synchronization between tasks, and OSE’s natural programming model is based on this concept. The Interprocess Communication Protocol (IPC) in OSE, allows tasks to run on different processors or cores, utilizing the same message-based communication model as on a single processor. This programming model provides the advantage of not needing to use shared memory among tasks. The runnable real-time entity equivalent to a task is called process in OSE, and the messages that are passed between processes are referred to as signals (thus, the terms process and task in this paper can be considered interchangeably). Processes can be created statically at system start-up, or dynamically at runtime. Static processes last for the whole life time of the system and cannot be terminated. Types of processes that can be created in OSE are: interrupt process, timer interrupt process, prioritized process, background process, and phantom process. A process can be in one of the following states: ready, running or waiting. One interesting feature of OSE is that the same programming model based on signal passing can be used regardless of the type of a process. OSE also has a tool for debugging, profiling and monitoring called Optima. Among many other features, Optima enables to monitor properties such as CPU load and memory usage (both heap and stack usage), total number of generated signals in the system, number of signals belonging to a certain process, etc. can be monitored. When an application (e.g., a monitoring and profiling tool) needs to obtain the value for any of the properties, it just sends a signal containing a code identifying the desired property to the monitor process. Upon the receipt of this signal, the monitor process which is in waiting state, starts and obtains the property value and sends it back to the requesting process. In this scenario, the monitor process needs to compete with other processes in the system based on their assigned priority levels to obtain CPU and perform its job. Therefore, it is natural that there will be some time difference until the monitor process can successfully obtain the value of a property. This scenario is shown in Figure 1. We aim to reduce the time difference which is denoted in the figure by t for properties whose values can change during this time and also this change is considered important and sensitive.

![Fig. 1. Monitoring a property and the time difference between the request for monitoring and the reply](image)

As stated before, if a property in the system changes so frequently then this time difference affects the accuracy of its read value. In this case, if the application requesting this value needs to perform some calculations and make decisions based on the value of the property at the exact time point that it issued the request for monitoring, the accuracy of the monitored value can affect the correctness of such calculations and decisions.

B. Priorities for Extra-Functional Properties

As described so far, the frequency of change in the value of properties in the system can be different from one property to another. Considering this fact, the user may prefer to fetch the value of a specific property with a shorter monitoring time difference to have better accuracy for it. To capture this need, we let different priority levels be assigned for different properties for performing monitoring. This way, the monitoring of CPU load, for example, can be done at a higher priority level than the monitoring of memory usage. It is achieved by dynamically adjusting the priority of the monitoring process for each property, instead of always invoking the monitoring process at the same fixed priority level. Therefore, the system becomes more flexible and monitoring of different properties does not have to be done at the fixed and default priority level of the monitoring process.

C. Implementation

The preferred way of communication between processes in OSE is through signals. Send and Receive APIs are provided in OSE for this purpose. To send a signal, a pointer to the signal structure along with the PID of the receiver is specified as: `send(sig, pid)`. In our implementation, the signal contains an identifier code to tell the monitor process which priority to monitor, which is defined in the system in the form of an `enum` structure in C/C++.
To allow the specification of priority levels for properties, we have defined a wrapper around the send API as: `send_w_prio(&sig, pid, prio)`. What `send_w_prio` does is basically that it first sets the priority of the monitor process (which is in the waiting mode to receive a signal) to the value of `prio`, and then sends to it the rest of the parameters using the normal send API. This way, it can be triggered to execute at a lower or higher priority level than its default value.

**IV. EVALUATION**

To test the suggested approach, a simulation environment has been setup. A monitor process is implemented which waits to receive a signal containing the identifier of a property based on which it then obtains the value for that property from the system and sends back the result to the requester. Another process (referred to as P) is also implemented which constantly changes the value of a dummy property (represented by a variable called N) to a random number and logs it in a file for later inspections and comparisons. The purpose of this process is to simulate a property whose value changes very frequently and constantly in the system. Two additional processes are defined in the system with priority levels below that of process P but above the default priority of monitor process (we call them I1 and I2). A shell command (in the OSE shell) called App has also been created which when executed sends a request for monitoring the property N to the monitor process.

When a request for monitoring the property N is initiated using the added shell command, the execution of the monitor process is delayed by higher priority processes. This situation is demonstrated in the Table I where log records corresponding to such cases (extracted from the log files generated by the P and monitor processes) are listed.

**TABLE I**

<table>
<thead>
<tr>
<th>MRTP* (µs)</th>
<th>Monitoring Done at (µs)</th>
<th>Monitored Value</th>
<th>Value at MRTP*</th>
</tr>
</thead>
<tbody>
<tr>
<td>5140013</td>
<td>5150001</td>
<td>838</td>
<td>3200</td>
</tr>
<tr>
<td>6630025</td>
<td>6630026</td>
<td>3500</td>
<td>5900</td>
</tr>
<tr>
<td>7300011</td>
<td>7300001</td>
<td>3883</td>
<td>9600</td>
</tr>
<tr>
<td>11460011</td>
<td>13470001</td>
<td>5032</td>
<td>1600</td>
</tr>
</tbody>
</table>

* MRTP: Monitoring Request Time Point - from the start of the system.

**DISCREPANCY BETWEEN PROPERTY VALUES AT MONITORING REQUEST TIME AND WHEN ACTUAL MONITORING IS DONE.**

The first row in the table, for example, shows that the request for monitoring property N is issued at 5140013 (µs) from the start of the system. Due to the interferences from the two other processes (I1 and I2) in the system, the actual monitoring is performed at time point 5150001 (µs) which results in obtaining 838 as the value for the property (remember that when P executes, it repeatedly generates a random number and sets it as the value of N). However, by consulting the generated log file from process P, we can see that the actual value of the property at the time when monitoring request has been issued was 3200.

In the next step, we initiate again a request for monitoring but this time with a priority higher than those of I1 and I2 processes. Part of the result from the generated log files is shown in Table II.

**TABLE II**

<table>
<thead>
<tr>
<th>MRTP* (µs)</th>
<th>Monitoring Done at (µs)</th>
<th>Monitored Value</th>
<th>Value at MRTP*</th>
</tr>
</thead>
<tbody>
<tr>
<td>5970013</td>
<td>5970014</td>
<td>6804</td>
<td>6804</td>
</tr>
<tr>
<td>8630001</td>
<td>8630002</td>
<td>6698</td>
<td>6698</td>
</tr>
<tr>
<td>12500011</td>
<td>12500002</td>
<td>9629</td>
<td>9629</td>
</tr>
<tr>
<td>13500018</td>
<td>13500009</td>
<td>9600</td>
<td>9600</td>
</tr>
</tbody>
</table>

* MRTP: Monitoring Request Time Point - from the startup of the system.

From the result of the second case, we can see that more instances of monitoring have managed to obtain accurate results for the property. The discrepancies which are still observed in this scenario, such as the second row from the bottom of Table II are due to the re-activations of process P which still has higher priority than the monitor process (the 1µs difference between the monitoring request time point and actual time of performing monitoring is the natural increment of internal clock of the system).

**V. DISCUSSIONS**

In the previous section, it was shown how by allowing to assign priority levels for monitoring of different properties it becomes possible to achieve better accuracy. However, there are some important points that need careful attention. If the priority of the monitor process is increased, it also means that other processes in the system can suffer delays due to preemption by the now higher priority monitor process. The consequences of such delays are of course dependent on the nature and responsibilities of those processes in the system. Therefore, the user initiating a monitoring request at a high priority level should also consider its consequences.

Another factor that contributes to the monitoring time difference and thus accuracy of monitored properties is the time length of the monitoring process itself. Implementation efficiency of the monitor process can thus help with the reduction of monitoring time difference.

A point which can be specific to our implementation is that the shell daemon in OSE is a system process which also has an assigned priority level. This can be important for knowing how the shell command that we implemented is interpreted by the shell daemon and executed and how the daemon behaves in terms of getting CPU time to execute comparing its priority level to other processes in the system. In our settings, since the priority of the shell daemon was the same default value in both cases, its effect on monitoring was simply ignored.

Here we mainly discussed the frequency of change in the value of a property. Another aspect that can also be added to the picture is the magnitude of such changes and its importance for accurate monitoring. For example, the value of a property can change quite frequently in small increments from 1.0 to 5.0. But depending on the use of the monitored value, the difference between 1.45 and 1.80 may not be significant at all. Therefore, deciding for which properties
we need more accurate monitoring is dependent on the use of the monitored values as well as the importance, tolerance and negligible fluctuations in the value of those properties. Moreover, in this paper we did not discuss how exactly the values of different extra-functional properties are calculated and determined. For some properties such as CPU load or the number of signals (in case of OSE and signal-based systems) it may be straightforward to obtain such values, while for some other properties such as reliability, it might be very cumbersome and complicated. This topic, however, is out of the scope of the paper.

VI. RELATED WORK

The work done in [5] discusses the issue of accuracy in monitoring performance of a system using the hardware counters and registers that are available on modern microprocessors. One of the differences between this work and ours is that it compares the accuracy of monitoring using hardware counters versus hardware sampling while we basically discussed the issue of accuracy by targeting the monitoring time difference. It also touches on the effects of adding monitoring features by acknowledging the fact that “as in any physical system, the act of measuring perturbs the phenomenon being measured” [5]. [6] which introduces a very interesting approach to reduce energy consumption in real-time distributed embedded systems, defines the concept of monitoring intervals for cyclically monitoring the system to identify and exploit dynamic slacks. It discusses the overhead of monitoring and how to find an optimal interval value for it. However, it does not deal with the accuracy of monitoring results versus constant changes in the state of the system.

As for the application of monitoring, [7] serves an example for the use of monitoring results where models of real-time systems are synthesized based on the monitoring information that is collected. In [3], we have introduced and implemented an adaptive approach using monitoring information about timing behaviors of encryption algorithms to balance security and timing requirements in real-time systems. Regarding the implementation of monitoring mechanisms, in [2], we have discussed the challenges and practicalities of monitoring timing properties in real-time systems, especially in industrial RTOSes, and provided a solution to improve monitoring of real-time events. The work in [8] provides a framework for monitoring of timing properties and a model for the specification of timing constraints. The work in [8] provides a framework for monitoring of timing properties and a model for the specification of timing constraints. The disturbances of adding monitoring features and issues related to probe-effect are discussed in detail in [9].

VII. CONCLUSION AND FUTURE WORK

In this paper, we discussed the issue of accuracy in monitoring EFPs and its importance in real-time embedded systems. Mainly, the monitoring time difference has been discussed as an important factor that contributes to the accuracy of monitoring results. An approach was introduced to reduce this time difference. An implementation of the approach was also provided and it was evaluated and shown how the approach reduces the monitoring time difference.

By implementing the approach as part of OSE and Optima, more flexibility can be provided for monitoring the system in terms of accuracy levels, and also regarding the inclusion of timestamps for monitored values, the users can get a better idea about the monitoring time difference. One point to remember is that adding monitoring features bring along their own costs and effects which should also be taken into account in the design of systems. In our evaluation example, the monitoring request was always initiated with a priority less than the priority of the main process P. If not so, the monitoring could preempt process P which could represent a main task in the system such an engine control task, and thus impair the functionality of the system. Therefore, there is a type of trade-off between the monitoring accuracy and other functions of the systems. Considering this issue especially on multi-core systems could be an interesting direction of this work.

As a continuation of this work, we are working on building a monitoring framework to provide a flexible environment to the user to tune the monitoring of different properties based on different factors that contribute to its accuracy. Towards this goal, we are also investigating other possible factors that can affect the accuracy of monitoring results.

VIII. ACKNOWLEDGEMENTS

This work has been supported by the CHESS European Project (ARTEMIS-JU100022) [10] and XDIN AB [11] through the ITS-EASY program.

REFERENCES