Timing analysis tools in a model-driven development environment
Jorge Garrido Juan A. de la Puente Juan Zamorano
Miguel A. de Miguel Alejandro Alonso

Abstract: This paper discusses the use of timing analysis tools on software systems developed with model-driven engineering methodologies. Model-driven development is mainly based on model transformations and automatic code generation. However, a deep understanding of the internal structure and behaviour of the automatically generated code is required in order to conduct later phases of the lifecycle, including validation of non-functional requirements such as real-time properties. The paper describes the integration of a timing analysis tool with TASTE, a model-driven environment developed under the auspices of the European Space Agency. A study of the influence of using an MDE approach on timing analysis techniques, along with figures on the proportion of automated and human-written code in a representative example of a spacecraft attitude control system is included in the paper.

Keywords: Real-time systems, high-integrity software, model-based development, schedulability analysis, Ada, timing analysis.

1. INTRODUCTION

The continuous increase in the complexity of real-time, safety-critical embedded systems, as well as their heterogeneity, has forced an evolution in the development processes of these systems. The original document-to-code process can no longer scale to the size and complexity required from modern embedded systems. Model-driven engineering (MDE) (Schmidt, 2006) is a software development approach aimed at tackling the development of complex systems by means of abstract models encompassing all relevant design phases and levels. Models provide support for analysing the system properties, describing the system behaviour, and reasoning about the system at an abstract level. Different kinds of models with different levels of abstraction can be used for a single system, providing support for a development process going from abstract definitions to detailed design and implementation code. Indeed, in order to ensure the consistency of the models, a set of rigorous model transformations has to be defined.

Although many benefits can be obtained from using models, some low-level aspects of a system may be hidden by automated model transformations. While some system properties are less dependent on low level details as long as the implementation meets the requirements (mainly functional aspects), other properties may require a deeper understanding of the system internals. One of these areas is timing analysis, which requires a detailed description of the task structure, scheduling methods, and processor operation that is not always available in higher-level models. The notion of a computational model (see e.g. Panunzio and Vardanega, 2014) has arisen as a means to abstract the low-level properties of the implementation platform so that they can be incorporated into high-level descriptions. However, some real-time parameters that are required for timing analysis, such as the worst-case execution time (WCET) of individual operations, have still to be obtained from the implementation code. Although preliminary values can be provided prior to implementation stages (mainly based on previous experience), final WCET values can be only obtained from final implementation code for a specific execution platform.

In previous work we have shown how a WCET analysis tool can be used on a representative real-time system (Garrido et al., 2012, 2013), and how an MDE approach can be used to develop software for the same system (de la Puente et al., 2014). In this paper we extend this work by showing how a timing analysis tool can be integrated into an MDE development cycle and discussing the results obtained in an industrial-size real-time software system. The rest of the paper is organized as follows: section 2 introduces the TASTE environment which has been used in this work; section 3 describes how a timing analysis tool has been integrated into it. A case study is described in section 4, and the results of using the integrated environment on it are discussed. Finally, section 5 contains some conclusions and guidelines for future work.

2. THE TASTE TOOLSET

TASTE is a software development environment based on MDE principles (Perrotin et al., 2012). It is one of
the outcomes of the ASSERT project (Conquet, 2008), an FP6 project led by the European Space Agency (ESA) which was aimed at improving the quality of the system-and-software development process for critical embedded real-time systems. A model-based software development process was defined in the project, with a number of model views that support software development using an iterative scheme (figure 1):

- First, the functional behaviour, the logical structure of the system, and the relationships among its components are abstracted in a platform-independent model (PIM).
- The properties of the platform on which the software is to run are abstracted in a deployment model view. This comprises the hardware specification of the different nodes of the system (TASTE supports distributed systems) and the connections among them.
- Then, a concurrency model view is derived, which provides an abstract description of the tasking and real-time structure of the system. Real-time and other non-functional properties can be analysed on this view, and changes to other model views can be made if needed in order to ensure a proper behaviour.
- Implementation code is automatically generated from the concurrency view. Source code-based tools (compilation chain, debuggers, etc.) can be used to produce an executable code image and load it on the platform. Makefiles and other configuration files can also be generated from the concurrency view.

![Fig. 1. ASSERT process phases](image)

TASTE uses AADL (Feiler, 2012) as a core modelling language, which integrates model views using different tools and languages, e.g. ASN.1 (ITU, 2008) for data modelling, SDL (ITU, 2011) for event-driven functional modelling, and Simulink (Mathworks, 2013) for continuous-time functional modelling.

3. TIMING ANALYSIS IN TASTE

Since the end of the original ASSERT project, ESA has kept on supporting and funding the development and improvement of the TASTE toolset. In particular, the Model-based Software Development Lifecycle (MBSDL) project, aimed at assessing the suitability of the MDE approach for developing space software systems, has been recently completed. Requirements elicitation, modelling and traceability, validation and verification activities, and the influence of the use of models in project documents were the main topics of the study.

The particular area of interest in this paper is the validation of real-time requirements. Following a preliminary study on the integration of timing analysis tools in TASTE (Pérez et al., 2011), the Rápita Verification Suite (RVS) including the RapiTime worst-case execution time (WCET) analysis tool (Bernat et al., 2002; Wilhelm et al., 2008) has been used to perform basic timing analysis in the project. The integration of RVS tools in TASTE allows the developer to use its analysis facilities in the same transparent way as the other tools in the toolkit. RVS directly extracts the information required for the analysis from the concurrency view in the AADL model and from the implementation source code.

RVS performs two main kinds of analysis of the source code:

- **Structural analysis**, which yields a set of possible execution paths from the static program structure.
- **Dynamic analysis**, which is based on the execution of instrumented code, providing a set of execution traces with timing data. This data can be used to estimate WCET values, as well as other verification data such as test coverage measurements. By using small individual code blocks, the combination of WCET of blocks allow to produce a probabilistic WCET analysis of greater blocks of code (i.e. subprograms). The WCET value may then not have been observed in the measurements (Bernat et al., 2002), but be composed of the WCET of its composing blocks.

The overall system structure can be derived by performing model transformations on the functional and interface views of the PIM. The information on how the system functionality is mapped into a set of threads can then be extracted from the concurrency view. Since the mapping is automatically done by Ocarina, an AADL toolset that is used as a core engine in TASTE (Lasnier et al., 2009), structural data can be automatically extracted from the model by modifying the Ocarina transformation scripts to also output the relevant structural information during the transformation process. These data are then fed to the RVS tool, which combines them with information extracted by the compiler preprocessor in order to produce a detailed structural analysis of the code. The analysis results include identification of source code files, allocation of functions to source files, sequential and conditional blocks inside functions, and call trees.
The information generated in the structural analysis is used to instrument the code by placing specific function calls in relevant sections of the code in order to generate a trace at execution time. These relevant sections of code are entry and exit points of functions, as well as conditional execution sections of code, such as if-then-else statements, loops, and in general any potential point of branching in the code. Once the code has been compiled and executed, the generated traces are analysed. As part of this process, execution time values are matched with structural information by means of identified timestamps. As a result several execution time values, statistics and predictions, including the measured worst case execution times for each function in the system, are produced.

Fig. 2. Extended ASSERT process with structural analysis and WCET values fed back to concurrency view.

All this information can be fed back to the system model so as to refine the concurrency view and feasibility analysis, and iterate over the system design if required, as shown in figure 2. Also, and as one of the main points, the resultant information can be used for real-time validating purposes proving safe WCET measurement values as well as providing testing code coverage evidences.

The measured WCET values are fed back to the concurrency view of the system, where they replace preliminary values. Then, the timing analysis of the system can be completed by performing schedulability analysis using the relevant tools integrated in TASTE: Cheddar (Singhoff, et al., 2004), Marzhin and MAST (González Harbour et al., 2001).

4. A CASE STUDY

4.1 The UPMSat2 ADCS

The RVS tools integrated in TASTE have been used on a case study in order to assess the suitability of the approach to real system development. The UPMSat2 satellite ADCS (Attitude Determination and Control System) (de la Puente et al., 2014) has been chosen because of the availability of a complete TASTE model of the system from which the implementation source code has been generated. This also shows that the analysis can be still performed even considering only a subset of an overall system, thus providing a mean to early testing implementation-ready subsets of a bigger system. This consideration of only a subset of the system does not affect the validity of the approach, as scheduling algorithms deal with execution times of tasks independently and then each calculates the timing effects of their interaction. Structural and dynamic timing analysis have been carried out on the system, and the results are discussed in sections 4.2 and 4.3.

The ADCS is in charge of maintaining the attitude of the satellite within specified values. In order to accomplish this function, the system periodically gets estimates of the satellite orientation with respect to the Earth using magnetic field measurements read from magnetometers. A control algorithm (Cubas et al., 2015) computes corrective actions which are effected through magnetorquers, which are activated by a PWM task (Zamorano and Garrido, 2015). The control algorithm has been designed using Simulink.

Preliminary WCET data for a previous version of the ADCS software were obtained using RapiTime (Garrido et al., 2012) and aiT, a static analysis tool (Garrido et al., 2013). The analysis described in this section has been performed on the TASTE model and mostly auto-generated code, thus making it possible to analyse the influence of the model-driven approach on the results.

The system is built by executing a script that orchestrates the different tools performing the model transformation, code generation and compiling activities required. As a result, an executable file is generated, along with a number of source and configuration files. Although TASTE is
Fig. 4. Sample results of the static structural analysis of the periodic control task.

As figure 4 shows, the periodic task may execute up to 43 functions considering all possible paths, comprising 253 lines of code and 240 sequential code blocks. This is summed up and organised in the report call tree shown in figure 5, where the sequence of calls from the task root is represented. As can be seen in the figure, there are two procedures encapsulating the user-implemented procedure `adcs_PI_Periodic_Control-U`, which in turn includes three calls for getting the magnetometer measurements (`vm_adcs_Get_MGM_Value-U`), computing the control action (`adcs_RLControl_Step-U`), and actuating on the magnetorquers (`adcs_RLSet_Activation-U`). For each of these procedures, some glue code functions are also executed. The glue code subprograms are auto-generated by TASTE in order to enable transparent interoperability between functions written in different languages, for different processors or even running on different computer nodes with a different architecture, as TASTE also supports distributed system development. While easing the development, the glue code adds complexity to the system, as figure 5 shows (grey shadowed bullets), having also an influence on the timing analysis and execution time results, as will be discussed in section 4.3.

![Fig. 5. TASTE implementation periodic task call tree.](image)

As an additional remark, it can be said that the amount and complexity of the code automatically generated from the TASTE model (TASTE-d) is significantly larger than that the original hand-written code analysed in (Garrido et al., 2012) and (Garrido et al., 2013), as shown on table 1. When the TASTE system is built without debugging extra procedures (as for production), the structural complexity is notably reduced (TASTE-nd).

<table>
<thead>
<tr>
<th>Component</th>
<th>TASTE-d</th>
<th>TASTE-nd</th>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada files</td>
<td>50</td>
<td>45</td>
<td>8</td>
</tr>
<tr>
<td>C files</td>
<td>89</td>
<td>53</td>
<td>5</td>
</tr>
<tr>
<td>AADL files</td>
<td>19</td>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>Python files</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Directories</td>
<td>28</td>
<td>26</td>
<td>3</td>
</tr>
<tr>
<td>Total lines</td>
<td>74 032</td>
<td>10 437</td>
<td>814</td>
</tr>
</tbody>
</table>

Table 1. Structural analysis comparison.
4.3 Timing analysis

The structural complexity of the ADCS code makes it difficult to estimate the WCET of the different parts of the code. However, the integration of the timing analysis tool in the toolset makes this task easier. As already mentioned, the structural analysis results are used by RapiTime to include instrumentation points in relevant sections of the code. The instrumentation code generates a trace of timestamps registering the identification and clock value at each time the execution reaches an instrumentation point. The traces are used to compute execution-time parameters after each test run:

- **Execution time values.** The execution time of a section of code is the difference between the entry and exit instrumentation point timestamps. Instrumentation points execution times are previously measured to be subtracted at trace analysis time.
- **Worst case execution times measurements.** WCET values are measured by adding the worst case values of each section of the code executed in the worst case path of the code being analysed.
- **Coverage analysis.** As information gathered from traces can be incrementally added to the databases containing the analysis results, RapiTime also keeps a record of how many times has each block been executed in tests, which blocks have never been executed, etc.

The summary results of the ADCS timing analysis are shown in table 2. For the original implementation, the timing analysis method has been also a RapiTime based analysis. It can be seen that the code generated by TASTE has slightly longer execution times than the original handwritten code, which could be expected from the greater complexity of the auto-generated code. However, the increment in WCET values is only about 10%, which means that the overhead of using the MDE approach is acceptable. The values are the same for the debug and not debug version, as the extra checking code is not executed by the main executable and is only meant to be executed in tests.

<table>
<thead>
<tr>
<th></th>
<th>TASTE-d &amp; TASTE-nd</th>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control algorithm</td>
<td>9 131 cycles</td>
<td>7 074 cycles</td>
</tr>
<tr>
<td>Control task</td>
<td>33 206 cycles</td>
<td>30 812 cycles</td>
</tr>
</tbody>
</table>

It should be noted, however, that the overhead is not the same for all sections of code. The overhead of the control algorithm, which is purely sequential C code generated by the Simulink model, is significantly higher, about 30%. This is due to two main factors. First, the ratio of glue code over user code is higher in this case, and second, the call to the control algorithm includes more parameters, and more complex ones, than other subprogram calls, making the execution of the glue code last longer.

A deeper analysis of the higher structural complexity in the MDE approach reveals that it comes mostly from the mentioned glue code, which includes several types definitions and transformations, which require a significant amount of code lines but are not executed. This explains in part why the structural complexity is not reflected in proportional execution time values.

5. CONCLUSIONS

The MBSDL project has provided the required guidelines for a full model-driven lifecycle for the aerospace sector. The influence of using models in each development phase has been addressed, revealing different specific phases or tasks specially affected. Structural and timing analysis have been both identified as affected by the use of models and code auto-generation, motivating the integration of the latest version of the RVS toolsuite, selected among other candidates, in TASTE, an ESA-supported MDE development environment.

In this paper a preliminary analysis on the effect of models on a specific case study has been presented. The analysis has been carried out by comparing two implementations of the same subsystem of the UPMSat-2 satellite. The attitude determination and control system has been implemented by using a model based development, and compared the structural and timing analysis performed with the results obtained analysing an equivalent non model driven implementation.

The analysis shows that, although the structural complexity of the system generated using models is several times higher than the traditional approach, the worst case execution values are only increased in the range of 10% to 30% in the studied subsystem.

Finally, it is worth to mention that, although the time required to implement both approaches is not significantly different for an experienced programmer, the model-based approach has a leaner learning curve, as well as being less error prone, reducing the amount of time required for reviewing the system and fix defects, provided that this techniques are well understood and addressed inside organizations (Schmidt, 2006; Terrier and Gérard, 2006; Mohagheghi and Dehlen, 2008; Hutchinson et al., 2011; Torchiano et al., 2013). The exact values presented here may vary for different timing analysis tools available in the market. As they differ on a wide range of approach solutions, generalization can not be made on expectable integration effort required, as well as for the pessimism induced by specific tools or approaches. The main outcomes of this paper are the report of a successful integration of a timing analysis tool in a MDE environment, the description of the approach followed, and the assessment on how safety-critical developments can benefit from the combination of these tools and environments on several development phases: integrated design, code generation, analysis, evaluation and testing. Future work includes performing these analysis on other systems with different characteristics as well as with different MDE approaches, in order to find evidence of general case tendencies.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the support of AdaCore and Rapita Systems for this work.
REFERENCES


