Model-driven performance prediction of distributed real-time embedded defence systems

Katrina Falkner, Vanea Chiprianov, Nickolas J.G. Falkner, Claudia Szabo
School of Computer Science, University of Adelaide
Email: name.surname@adelaide.edu.au

James H. Hill
Department of Computer and Information Science, Indiana University-Purdue University Indianapolis
Email: hillj@cs.iupui.edu

Gavin Puddy, Dan Fraser, Adrian Johnston, Marianne Rieckmann, Andrew Wallis
School of Computer Science, University of Adelaide
Email: name.surname@adelaide.edu.au

Abstract—Autonomous defence systems are typically characterised by hard constraints on space, weight and power. These constraints have a strong impact on the non-functional properties, and performance, of the final system. System execution modelling tools permit early prediction of the performance of model driven systems, however the focus to date has been on understanding the performance of a model rather than determining if it meets performance requirements, and subsequently carrying out analysis to reveal the causes of any requirement violations. In this paper, we propose an integrated approach to performance prediction of model-driven distributed real time embedded defence systems. Our architectural prototyping system supports a scenario-driven experimental platform for evaluating model suitability within a set of deployment and real-time performance constraints. We present an overview of our performance prediction system, demonstrating the integration of modelling, execution and visualisation, and discuss a case study to illustrate our approach.

Prediction of software performance has developed from early approaches based on abstract models to Model-Driven Engineering (MDE) [3] based approaches. MDE techniques are typically applied to the development of application software components, but may also be used to model and solve the configuration and deployment phases, as well as system execution emulation, testing and analysis. One of the main techniques in MDE is the use of Domain Specific Modelling Languages (DSMLs). A DSML is defined in this paper to be a language that offers expressive power focused on a particular problem domain through appropriate notation and abstractions.

System Execution Modelling (SEM) [4], a recent development from research into measurement-based performance prediction, provides detailed early insight into the non-functional characteristics of a DRE system design. It is based upon simple models of resource consumption from the component’s “business logic” [4] and supports detailed performance modelling of software systems, enabling predictions of performance through execution of representative source code of behaviour and workload models deployed upon realistic hardware testbeds.

SEM and MDE may be used in combination to support the emulation of system components and performance models, enabling performance data to be used to redesign and reconfigure the system, prior to any construction of the corresponding real system. However, work in this area is still in its early stages and more work is required to better support early performance prediction. In particular, there is a need to integrate the representation and visualisation of models and performance information to assist in early decision-making based on performance predictions using realistic data sources [5], and to integrate SEM approaches and modelling environments with capabilities for multi-modelling traceability [6].

II. RELATED WORK

A detailed review of model-based performance prediction is provided by Balsamo et al [2]. Early approaches on early performance prediction focus on using abstract modelling such as petri nets [7], queuing networks [8] and Markov chains [9] to provide insight into system structure and high level interactions. Cortellessa and Mirandola [10] extend this work further by developing queuing network models from UML system models.

More recent approaches explore the explicit modelling of
performance. Fritzsche and Johannes [11] introduce a process, Model-Driven Performance Engineering (MDPE) which combines MDE and performance engineering principles. MDPE adopts a software performance process termed Software Performance Engineering. Software Performance Engineering integrates performance engineering based upon two models, the Software Execution Model and the System Execution Model, throughout all phases of the software development lifecycle. The execution models are represented as flow graphs annotated with resource demands, and contention constraints. This approach is integrated directly with MDE, with performance analysis integrated with each stage of model transformation. It is not the intention for developers to directly annotate their models with performance related information but for this information to be either generated from development models or automatically generated performance data. This may limit the practicality of this approach.

Several researchers explore the potential for modelling performance based on an understanding of the system architecture. Hemer and Ding [12] adopt an ADL-based approach for modelling non-functional properties, which may include performance or timing concerns. Edwards et al [5] utilise a dynamic analysis approach for exploring performance scenarios, within a blended ADL/MDE framework. Their focus on scenario-driven experimentation matches our approach, however, they utilise more advanced and specialised simulators to model behaviour, presenting a more complicated approach with the concomitant requirement for greater modelling effort.

III. THE ARCHITECTURE OF THE PERFORMANCE PREDICTION SYSTEM

The performance prediction for a System Execution Model (SEM) is divided into three main steps, as shown in Figure 1: model, execute, and evaluate and predict. In the modelling step, the system expert establishes performance requirements for the System under Study (SUS). These performance requirements are guided by the performance constraints identified by the user but also partly by the structure of the SEM, in terms of its components and their interaction with other systems. Based on the performance requirements we create a scenario model that describes the interaction between the SEM and other systems, such as information about terrain, tactical data links, and time management. In the execution step, the synthetic environment employs scenario information to execute the SEM on various hardware and middleware configurations using the execution engine. In the evaluation and prediction step, data and performance metrics are passed to an evaluation engine. This establishes if the desired performance has been met. Information from the execution and evaluation engines is used in the visualisation component, which offers the user additional insight into the performance of the SUS.

A. Modelling

1) Modelling the SUS - the SEM: The SEM is a middleware and platform-independent model composed of several aspects: the systemic structure of the SUS into main software components and their connections, a high-level description of the functional behaviour of each of these components, a workload model of the resources consumed by the functional behaviour of each of these components, and a model of how the software components are deployed on the distributed hardware testbed.

The tools that we use to create these models are the DSMLs provided by CUTS [13]: the Platform Independent Component Modelling Language (PICML) [14] for Modelling compositions, deployments, and configurations of SEM components; the Component Behavior Modelling Language (CBML) to model component behaviour, and the Workload Modelling Language (WML) to model component workload.

a) Multi-Modelling: With the large number of models contained in the SUS, a challenge remains in capturing the dependencies between them and in enabling change management through traceability. Towards this, we propose a multi-modeling paradigm that facilitates the semantic alignment between entities in different domains. Our approach captures relationships between entities in a relationship meta-model that is used for change traceability [15].

2) Modelling the Scenario: Scenarios capture the interactions of the SUS with other systems and are used to analyse the performance of the SEM across these interactions. The Synthetic Environment that models these interactions will inject data into the SEM, simulating different situations and thus enabling the analysis of the SEM and whether it meets its performance requirements. The conceptual model of the Synthetic Environment is defined by leveraging the existing conceptual model and authoritative domain information of the SISO standard Military Scenario Definition Language (MSDL) [16]. While MSDL includes a large domain model, intended to cover many applications, we specialise it by choosing only the concepts relevant to our scenarios.

B. Executing

1) Executing the SEM: As we are interested in predicting the performance of systems, and not only software, we prototype system architectures. These include four levels of abstraction: application, middleware, operating system and hardware. At the application level, the SEM and scenarios are defined. For middleware, we chose to use the OMG standard Data Distribution Service (DDS) [17], especially due to its extensive support of non-functional properties through QoS policies that support various time and data management mechanisms. The selection of operating systems and hardware is driven by Defence needs. In order to evaluate the performance of the modeled system, different configurations of all levels of abstraction have to be analysed.

2) Executing the Scenario: The scenario initialisation, as specified in the DSML, is processed by the DSML compiler. This generates the main functional entities of the scenario, together with any glue code and configuration files needed. The main entities of the scenarios are distributed.

The platform-independent models produced in the Modeling step are further enriched with information about the platform they will be executing on through middleware and platform-dependent models. From these platform-specific models, code for those middleware and platforms is generated. The generated code is distributed on different hardware machines, and thus it has several components. The system expert constructs a model of how the software components are
deployed on the distributed hardware. We have defined a tool for constructing this model. From the deployment model, configuration files are generated.

C. Evaluating and Predicting

Our performance evaluation engine permits the definition of unit tests to analyze non-functional concerns such as the system throughput and resource utilization among others. As the execution engine is executing the SEM based on the specified scenario, several performance metrics are recorded and passed to the evaluation engine for performance prediction and evaluation. Our execution engine records a large amount of information related to the execution of the SEM. This includes all data passed between the components of the SEM, out-of-band data within the SEM, as well as the data exchanged between the SEM and the Synthetic Environment.

The Metric AGgregation modulE (MAGE) transforms the raw data into aggregated information, according to the procedures specified by the performance requirements. Based on the performance constraint and the aggregated metrics, the evaluation module determines if performance constraints have been met. This information, together with the collected data, is passed to the visualization component. The visualization component completes the performance study by offering the user a more informed overview on the causes of particular performance metric values.

1) Metric Aggregation Module (MAGE): Current aggregation metrics supported by MAGE include simple aggregation such as MIN, MAX and SUM, but also more complex aggregators such as ABS_MEAN (absolute value of mean). To facilitate the calculation of performance measurements we propose more complex aggregations defined as sequences of simple aggregation metrics. For example, consider the utilization of a particular component that could be defined as $u = \frac{\text{service time}}{\text{runtime}}$, that is, the fraction of the total component runtime that the component is in service.

IV. CASE STUDY: EARLY VALIDATION OF PERFORMANCE PREDICTION SYSTEM

In our case study, we analyze the performance of a Unmanned Air Vehicle (UAV) that moves from air to underwater in an ocean environment. The UAV communicates with a Combat Management System (CMS) that could be located on a ship or submarine. The CMS uses a communication link to send to the UAV the coordinates of targets to investigate, whereas the UAV sends back its position and images of the targets of interest. The environment is represented by the air and ocean. The main event that impacts the performance is when the UAV is submerged underwater. This impacts the communication link and therefore the performance of the UAV: when underwater, the communication link has less bandwidth, which implies that the UAV will do more processing, e.g., compressing the data it is sending.

A. Modelling

The UAV is the SUS. Its systemic structure, presented in Figure 2, is made up of three components: a “Controller” (UAV1_CTRL), a “Communication” (UAV1_COMM), and a “GPS” (UAV1_GPS). The “Controller” component does the main processing of the UAV. The data that is processed by the “Controller” is received by the UAV through the “GPS” component, which sends the GPS coordinates. The “Communication” component exchanges control signals with the CMS and is influenced by the available bandwidth.

The main functional behaviour of the UAV is described in the “Controller” component, as a behavioural model as shown...
in Figure 3. The “Controller” exhibits two types of behavior corresponding to the two states of being in the air and underwater, as shown by the upper and lower branches of the workload model respectively. The “cpu” activities of both branches describe the workload the “Controller” has to perform in the two states. This workload is indicated as an annotation of the figure and consists of setting the service time of the “cpu” to 10 msec and 500 msec, corresponding to the UAV being in the air and underwater respectively, and thus to the increased workload when the UAV is underwater.

B. Executing

The distributed components of the structural model of the UAV are deployed on different hardware nodes (Section III). Using our deployment generator tool, the user can select the hardware, software, and middleware configurations.

C. Evaluation and Prediction

Using the SEM execution traces, we evaluate the utilization, as defined in Section III-C1, of the UAV in its states in the air and underwater. We obtain a result of $\text{utilization}_{\text{AIR}} = 4\%$, 4.15%, 4% and respectively $\text{utilization}_{\text{SUB}} = 40\%$ for 100 msec, 59.6% for 150 msec, and 99% for 500 msec. This shows that the utilization of the SUS increases drastically while underwater, as a direct result of the increased processing required for compressing images.

V. CONCLUSION AND PERSPECTIVES

This paper presents a model-driven performance prediction system to address issues related to the integration of realistic data sources, the visualization of the causes of performance issues, as well as the understanding of models and relationships affected by various performance constraints, towards a complete performance prediction system. Our performance prediction system is able to identify performance issues of a SEM that is executed under a variety of conditions, using data obtained from real, emulated, or simulated sources. The performance study of a SEM is divided into three main steps, namely, modeling, execution, and prediction and evaluation.

Our case study shows the feasibility of our system but also highlights new challenges. Towards increased usability and flexibility of our system, we are implementing a graphical Domain Specific Modeling Language (DSML) for defining scenarios. Similarly, DSMs will be defined for capturing performance requirements and for metric aggregations. To enhance the evolution capabilities of our system, we are implementing a DSML for capturing relationships and data exchange traces between different types of models, thus enabling bidirectional analysis of change impact.

REFERENCES