Modeling scenarios for the performance prediction of distributed real-time embedded systems

Katrina Falkner, Vanea Chiprianov, Nickolas Falkner and Claudia Szabo
School of Computer Science, University of Adelaide
Adelaide, South Australia, Australia
firstname.lastname@adelaide.edu.au

Gavin Puddy
Defence Science and Technology Organisation
Department of Defence
Edinburgh, Australia
gavin.puddy@defence.gov.au

***Abstract*—Autonomous defence systems are typically characterized by hard constraints on space, weight and power. These constraints have a strong impact on the non-functional properties and especially performance of the final system. System execution modelling tools permit early prediction of the performance of model driven systems; however they are intended for one shot analysis, not for repeatable, interactive use. In this paper we propose a Domain Specific Language for describing scenarios to repeatedly test a system execution model within a Synthetic Environment. We exemplify it by describing and executing a scenario involving an UAV and a CMS.**

Keywords-component; formatting; style; styling; insert (key words)

I. INTRODUCTION

Mission-critical Distributed Real-time and Embedded (DRE) systems, such as naval combat systems or mission-systems, can have life-cycles that can be counted in the decades [1]. The design complexity and cost of such long-lived, large systems continue to grow while business owners continue to seek improvements in the return on investments for such projects. Over the time that these systems have been developed, fundamental development philosophies have changed and the use of monolithic and proprietary design has made way for modular designs, code reuse, and for the adoption of product lines approaches [2]. Moreover, mission-critical DRE systems are frequently in active use, evolve over time, are rarely built from scratch, and integrate legacy systems [3]. While an understanding of both functional and non-functional aspects of the system design is important, issues associated with the *non-functional aspect* of the design are of greater concern for resource constrained platforms, such as submarines or autonomous vehicles. With strict budget allocations for space, weight and power for various systems installed in such platforms, any early insight into the performance of these systems, and their corresponding deployment and budget considerations, becomes crucial.

One approach to allow the early checking of meeting non-functional performance requirements is *performance prediction*. We adopt the definition of *software performance* introduced by Balsamo et al [4]: 'the process of predicting (at early phases of the life cycle) and evaluating (at the end) based on performance models, whether the software system satisfies the user performance goals'. Similarly, we require that our understanding of performance prediction be based upon the provision and evaluation of an existing *performance model*.

Prediction of software performance has developed from early approaches based on abstract models to *Model-Driven Engineering (MDE)* [5] 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) [6], 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'' [6] 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 repeatedly and interactively exert the system execution model, in order to obtain more reliable and significant test results [7]. In this paper we describe a domain specific language to describe scenarios that can be used to repeatedly test a system execution model, building upon our previous work, developed collaboratively between the University of Adelaide and DSTO, in the development of a model driven framework for performance prediction of DRE systems [8].

II. Related Work

A detailed review of model-based performance prediction is provided by Balsamo et al [4]. Early approaches on early performance prediction focus on using abstract modeling such as petri nets [9], queuing networks [10] and Markov chains [11] to provide insight into system structure and high-level interactions. Cortellessa and Mirandola [12] extend this work further by developing queuing network models from UML system models.

Edwards et al [7] 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. On 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 visualization component, which offers the user additional insight into the performance of the SUS.

A. Modeling

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 defining the 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 test-bed.

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 Behaviour Modelling Language (CBML) to model component behaviour, and the Workload Modelling Language (WML) to model component workload.

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) [15]. While MSDL includes a large domain model, intended to cover many applications, we specialize it by choosing only the concepts relevant to our scenarios (cf. Section IV).

B. Executing

1) Executing the SEM

We prototype system architectures, which 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

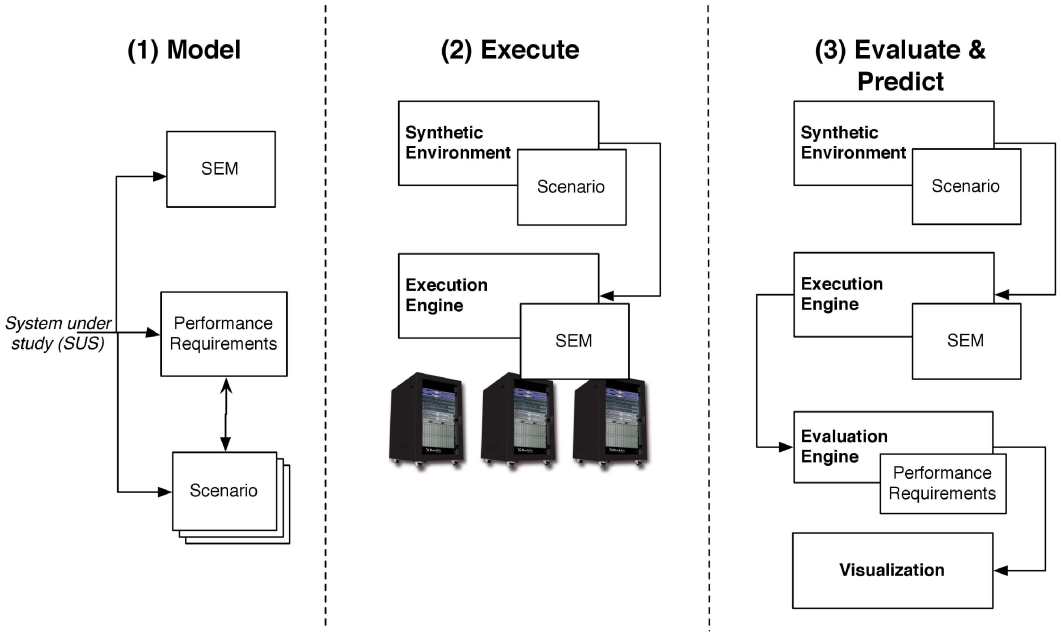

Figure 1. The Performance Prediction process for a System Execution Model

Data Distribution Service (DDS) [16], 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 modelled system, different configurations of all levels of abstraction have to be analysed.

2) Executing the Scenario

The scenario initialisation, as it 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 *Modelling* step are further enriched with information about the platform they will be executing on through middleware and hardware 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 [8]. From the deployment model, configuration files are generated.

C. Evaluating and Predicting

Our performance evaluation engine permits the definition of unit tests to analyse non-functional concerns such as the system throughput and resource utilization among others [8]. 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.

IV. The Architecture of the Scenario-driven Synthetic Environment

The process we use to develop the Synthetic Environment is Distributed Simulation Engineering and Execution Process (DSEEP) [17]. DSEEP is a waterfall-like process, with seven steps. We present here the results of the third step - *Design Simulation Environment*. In the first three steps we develop the following important artefacts: scenarios, simulation environment requirements, a conceptual model and a simulation environment design. These artefacts are described below.

A. Synthetic Environment Requirements

The key requirements of the Synthetic Environment, to ensure that our system is exerted correctly against a scenario, are:

- *Repeatability*: The Synthetic Environment should exercise the SEM in a repeatable way.
- *Time management*: Its interaction with the SEM needs a mechanism to manage the time of messages exchanged between them.
- *Data distribution management*: The exchanged messages themselves need a mechanism for data distribution management.
- *Simulation interoperability*: These messages will mainly be generated by the Synthetic Environment to simulate real-world data sources. Depending on the level of fidelity desired, it may be necessary at later stages to integrate emulators or even data captured from real hardware components.
- *Scenario specification*: The way in which messages are passed between the Synthetic Environment and the SEM simulates interactions between the SEM and other systems. As discussed above, these interactions are specified in scenarios. Scenario specification consists in describing [18]: the environment - in our case the terrain as grid, the movement of the entities interacting with the SEM, the tactical data links, the data, and the scenario initialisation.
- *Interactive simulation*: The scenario can be specified either in a repeatable, scripted manner, or in an interactive manner, in which the user manipulates simulation models at run-time.

B. Synthetic Environment Conceptual model

The conceptual model is defined, as recommended by DSEEP, by leveraging the existing conceptual model and authoritative domain information of the SISO standard Military Scenario Definition Language (MSDL) [15]. While MSDL includes a large domain model, intended to cover many applications, we specialise it by choosing only the concepts

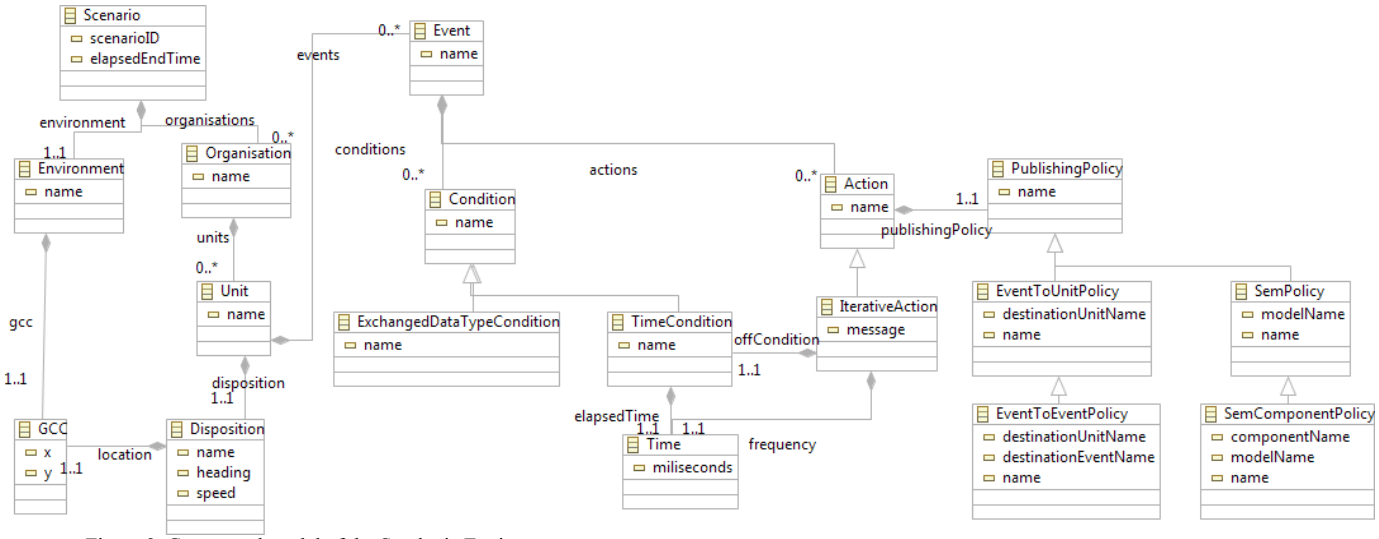

Figure 2. Conceptual model of the Synthetic Environment

relevant to our scenarios. The concepts from MDSL relevant to us are (cf. Figure 2): *Scenario*, which contains several *Organisation*s that model the actors, each of them containing several *Unit*s (e.g. UAV, ship), each with a *Disposition* modelled by a *GCC-Geocentric coordinate* in the *Environment*.

We add two groups of concepts: one related to modelling events, the other related to publishing policies. The event

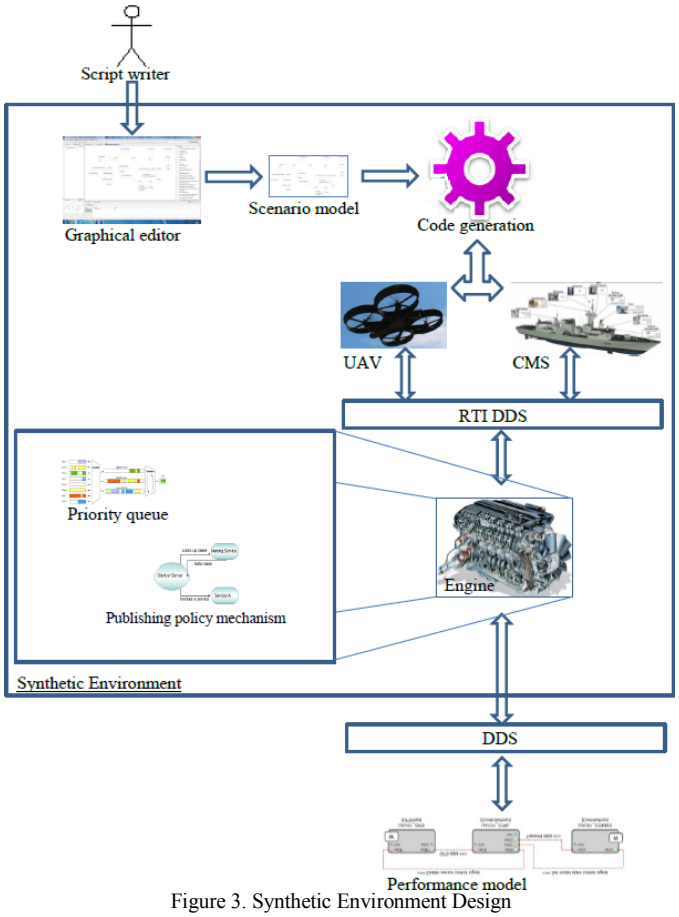

Figure 3. Synthetic Environment Design

mechanism was chosen to model the behaviour of *Unit*s and their response to the actions of other *Unit*s. The concepts related to the event mechanism are: *Event*, which has several *Condition*s that have to be met in order for it to be triggered. Conditions can be of different types, either on the data that is exchanged between events - *ExchangedDataTypeCondition*, or on the time when an event is triggered - *TimeCondition*. Once the event has been triggered, it performs *Action*s. These can be one shot actions, or *IterativeAction*s, in which case a *frequency* with which the repetition is happening and an *offCondition* need to be indicated.

A second group of entities that we add to the conceptual model is related to publishing policies. These are needed to indicate how the messages are routed between Units inside the Synthetic Environment, e.g. *EventToUnitPolicy*, *EventToEventPolicy*, or between Units inside the Synthetic Environment and SEMs – *SemPolicy* – or SEM components – *SemComponentPolicy*.

C. *Synthetic Environment Design*

The design of the Synthetic Environment, shown in Figure 3, has to accommodate different levels of fidelity of simulation, starting from periodically generating events, going through injecting data, to the use of more complex simulators for some of the components and eventually to the integration of real data-generating components, such as radars in a combat management system (CMS).

The design is based on a series of architectural decisions. The first decision was to introduce a (time and data management) bus – DDS – between the Synthetic Environment and the SEM Performance model, to manage the *time* and the *data* exchange between them answering thus the respective requirements.

The second decision was to introduce a time and data management bus between the components of the Synthetic Environment – RTI DDS, thus making the Synthetic Environment a distributed virtual environment. The rationale was to accommodate the integration of units, emulators and real data sources as components, in accordance with the *Simulation interoperability* requirement. Shown in Figure 3 are

two units: UAV and CMS, and the Synthetic Environment Engine. They are all distributed entities. The Engine uses a priority queue to manage the order of the events, both between entities inside the Synthetic Environment and Units inside the Environment and SEMs outside of it. The Engine also has a mechanism for managing the publishing policies.

The next decision was to define a DSML for *scenario specification* in order to hide the complexities of scenario specification, to enable generation of code and configuration files. The DSML leverages the conceptual model presented previously. The decision to leverage MSDL was based both on the DSEEP recommendation to reuse existing conceptual models and authoritative domain information, and on the desire to remain compatible with the standard for specifying scenarios. The DSML offers the *Script Writer* a *Graphical editor*, with which a *Scenario model* can be described. The Graphical Editor was generated form the conceptual model using Eclipse EMF and GMF [19]. From the Scenario model, C++ code is generated through a model-to-text transformation written in Eclipse Xpand.

Lastly, we introduced an event-condition-action mechanism through which the behaviour of each unit and its response to the actions of other units is specified. This mechanism, by ensuring the fact that actions happen in the Synthetic Environment only when a set of conditions are met, ensures the *repeatability* necessary for exerting the SEMs multiple times. This decision also enables taking into account events generated by humans, as part of an *interactive simulation*.

DSEEP recommends that several scenarios be developed together with defence stakeholders. One such scenario is presented in the next section, to illustrate the use of the Synthetic Environment.

V. Case Study: Early Validation of the Scenario-driven Synthetic Environment

Following the data flow from Figure 3, a Scenario model is described by the Script writer using the Graphical editor. An abridged version of such a model is presented in Figure 4. It

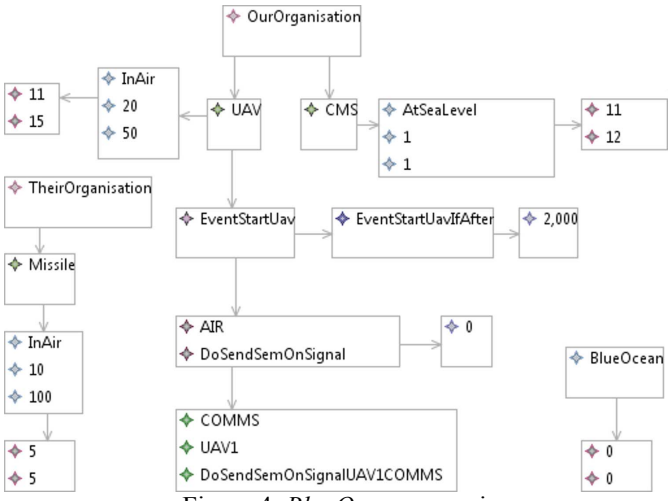

Figure 4. *BlueOcean* scenario

consists of two organisations, *OurOrganisation* and *TheirOrganisation*, and a *BlueOcean* environment.

The *BlueOcean* has no detail modelled, only a GCC of *0,0*. OurOrganisation has two units: a *CMS*, with an *AtSeaLevel* disposition, at a certain GCC *11,12*, and an *UAV* with an *InAir* disposition, at a certain GCC *11,15*. TheirOrganisation has only one unit, a *Missile*, with an *InAir* disposition, at a certain GCC *5,5*.

In this scenario, the UAV has four events, from which only the first one is shown in Figure 4. The first event starts the UAV in air - *EventStartUav*. The event has a triggering time condition – *EventStartUavAfter* – with a time value of *2000*ms. The *DoSendSemOnSignal* action associated with this event tells the UAV SEM to start in the *AIR* state. The communication with the UAV SEM is controlled through the use of a *DoSendSemOnSignalUAV1COMMS* SemComponentPolicy, which indicates the name of the SEM model – *UAV1* – and the name of the SEM component inside it – *COMMS* – to which the communication is to be relayed. The second event starts sending GPS data from the UAV to the CMS. The CMS discovers the threat of the Missile, and commands the UAV to go underwater – the third event. Finally the UAV is sent a signal to stop – the fourth event.

The next step in the data flow in Figure 3 is generating code. A snippet of generated C++ code for this scenario is presented in Listing 1. It shows the four events waiting to be executed in a while loop. For each event, there is a check if all the triggering conditions have been met, e.g. eventstartuav.areConditionsFulfilled(), then all actions associated to that event are executed, e.g. eventstartuav.executeActions(parameter_list). Some details, like the parameter_list of the actions have been omitted in this listing. The generated code contains a thread for each of the three units of this scenario, thus making them distributed entities.

A simplified trace of events and messages being exchanged in the Synthetic Environment and between the units and their respective SEMs can be followed as changes in the state of the UAV1 SEM, in Figure 5. The state is initially OFF, but when EventStartUav is triggered, the OFF state stops and the AIR state starts. When the event to go underwater is triggered, the AIR state stops and the SUB state starts. Finally, when the stop event is triggered, the SUB state stops and the OFF state starts again.

```
while ( ! (endLoop [0] && endLoop [1] && endLoop [2] && endLoop [3]) ){
  if (eventsendgps.areConditionsFulfilled(sc) && ! endLoop[0] ){
    eventsendgps.executeActions(parameter_list);
    endLoop [0] = true;}
  if (eventstartuav.areConditionsFulfilled(sc) && ! endLoop[1] ){
    eventstartuav.executeActions(parameter_list);
    endLoop [1] = true;}
  if (eventgounderwater.areConditionsFulfilled(sc) && ! endLoop[2] ){
    eventgounderwater.executeActions(parameter_list);
    endLoop [2] = true;}
  if (eventstopuav.areConditionsFulfilled(sc) && ! endLoop[3] ){
    eventstopuav.executeActions(parameter_list);
    endLoop [3] = true;}
}
```

Listing 1. Generated code for the *Blue Ocean* scenario

Data returned:

lid	timeofday	severity	hostname	thread_id	message
303	2013-04-26 14:41:57	4	ubuntu	-1361384640	Component state OFF stop 4093905915 at 38898717.120629
304	2013-04-26 14:41:57	4	ubuntu	-1361384640	Component state AIR start 4074704161 at 38898717.120745
310	2013-04-26 14:42:26	4	ubuntu	-1361384640	Component state AIR stop 4074704161 at 38898746.102186
311	2013-04-26 14:42:26	4	ubuntu	-1361384640	Component state SUB start 4074704162 at 38898746.102238
313	2013-04-26 14:42:36	4	ubuntu	-1361384640	Component state SUB stop 4074704162 at 38898756.130313
314	2013-04-26 14:42:36	4	ubuntu	-1361384640	Component state OFF start 4074704163 at 38898756.130366

Figure 5. Trace for the *Blue Ocean* Scenario

VI. Conclusion and Perspectives

Analysing and predicting the performance of distributed real-time embedded defence systems involves repeated executions of the performance models. For this, we introduce in this paper a Domain Specific Modelling Language (DSML) for describing scenarios and a Synthetic Environment to execute them. The DSML offers a method to describe scenarios at a high abstraction level, using concepts familiar to the domain user and hiding complexities of code generation. The models are then executed by the Synthetic Environment Engine, which provides the means to ensure interoperability between simulators, data and time management both inside the Synthetic Environment and between this and the SEMs.

The Synthetic Environment architecture allows for integrating simulators, e.g. for radars, for (wireless) network connections. In the future we will integrate such simulators. We will also develop an interactive interface, which will allow interactive control and visualization of the Synthetic Environment. This interface will be integrated into our visualization framework.

References

[1] L. Merola, "The COTS software obsolescence threat," in *Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems*, 2006.

[2] K. Balasubramanian, J. Balasubramanian, J. Parsons, A. Gokhale and D. Schmidt, "A Platform-Independent Component Modeling Language for Distributed Real-time and Embedded Systems," *Journal of Computer and System Sciences,* vol. 73, pp. 171-185, 2007.

[3] N. Weiderman, J. Bergey, D. Smith and S. Tilley, "Approaches to legacy system evolution," Software Engineering Institute, Carnegie-Mellon University, Tech. Rep. CMU/SEI-97-TR-014, 1997.

[4] S. Balsamo, A. D. Marco, P. Inverardi and M. Simeoni, "Model-based performance prediction in software development: a survey," *IEEE Transactions on Software Engineering,* vol. 30, no. 5, pp. 295-310, 2004.

[5] S. Beydeda, M. Book and V. Gruhn, Model Driven Software Development, Springer-Verlag, 2010.

[6] J. Hill, D. Schmidt and J. Slaby, "System Execution Modeling Tools for Evaluating the Quality of Service of Enterprise Distributed Real-time and Embedded Systems," in *Designing Software-Intensive Systems: Methods and Principles*, IGI Global, 2008, pp. 335-371.

[7] G. Edwards, S. Malek and N. Medvidovic, "Scenario-driven dynamic analysis of distributed architectures," in *10th International Conference on Fundamental Approaches to Software Engineering*, 2010.

[8] K. Falkner, V. Chiprianov, N. Falkner, C. Szabo, J. Hill, G. Puddy, D. Fraser, A. Johnston, M. Rieckmann and A. Wallis, Model-driven performance prediction of distributed real-time embedded defence systems. Accepted for the 18th International Conference on Engineering of Complex Computer Systems (ICECCS 2013), July 2013.

[9] C. Ramamoorthy and G. Ho, "Performance evaluation of asynchronous concurrent systems using petri nets," *IEEE Transactions on Software Engineering,* vol. 6, no. 5, pp. 440-449, 1980.

[10] B. Spitznagel and D. Garlan, "Architecture-based performance analysis," in *Proceedings of the Conference on Software Engineering and Knowledge Engineering*, 1998.

[11] G. Bolch, S. Greiner, H. de Meer and K. S. Trivedi, Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications, Wiley, 2006.

[12] V. Cortellessa and R. Mirandola, "PRIMA-UML: a performance validation incremental methodology on early UML diagrams," *Science of Computer Programming,* vol. 44, pp. 101-129, 2002.

[13] J. Hill, D. Schmidt, J. Edmondson and A. Gokhale, "Tools for continuously evaluating distributed system qualities," *IEEE Software,* vol. 27, no. 4, pp. 65-71, 2010.

[14] J. Hill, D. Schmidt, A. Porter and J. Slaby, "CiCUTS: Combining System Execution Modeling Tools with Continuous Integration Environments," in *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, 2008.

[15] SISO, "Military Scenario Definition Language (MSDL). SISO-STD-007-2008," 2008.

[16] OMG, "Data Distribution Service for Real-time Systems Version 1.2," 2007.

[17] IEEE, "IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP) IEEE Std 1730-2010," 2011.

[18] A. Tolk, Engineering Principles of Combat Modeling and Distributed Simulation, Wiley, 2012.

[19] R. C. Gronback, Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit, Addison-Wesley Professional, 2009.