

# Model-driven Simulation of a Maritime Surveillance System

M. Monperrus<sup>2,3</sup>, F. Jaozafy<sup>1</sup>, G. Marchalot<sup>1</sup>, J. Champeau<sup>2</sup>, B. Hoeltzener<sup>2</sup>,  
JM. Jézéquel<sup>3</sup>

1. Thales Airborne Systems
2. ENSIETA/DTN
3. INRIA/Triskell

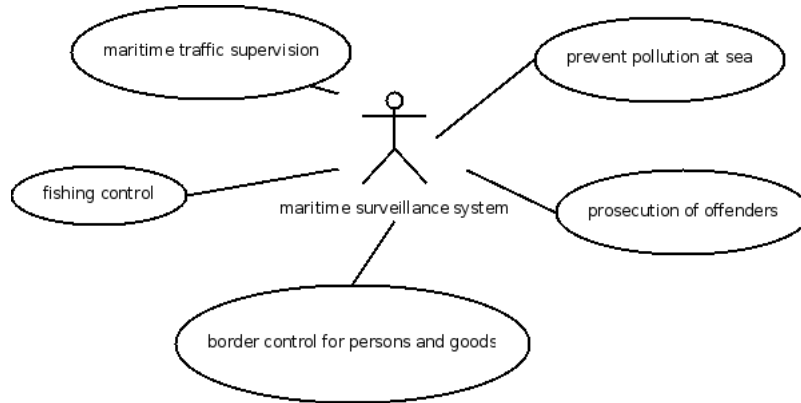
**Abstract.** This paper reports an industrial experiment made at Thales to use Model Driven Architecture (MDA) for system engineering. System engineering processes are currently mainly document-centric. The main experiment goal was to study the applicability of MDA at the system engineering level. The experiment consisted of setting up a model-driven simulation environment for a maritime surveillance system. The simulation is achieved thanks to 3 models conform to 3 metamodels. The implementation uses the Eclipse Modeling Framework and is written in the Java Programming language. This pilot project met the deadline, the budget and the threshold of desired functionalities. We report the main advances given by the MDA approach in the context of simulation for system engineering.

## 1 Introduction

Simulation is a key step in the development of a maritime surveillance system [1]. At the very beginning of the life cycle of the system i.e.; at the system engineering level, simulation is a way to communicate with the customer in order to elicit the needs. Therefore, simulation eases the consistency between the customer requirements and the delivered product. From an engineering point of view, simulation is a way to validate parts of the architecture, as well as the behavior of the product. Last but not least, this permits a better cost estimation and a better planning of technical and human resources.

A maritime surveillance system (MSS) is a multi-mission system. As depicted in figure 1 as a UML use case diagram, it is intended to supervise the maritime traffic, to prevent pollution at sea, to control the fishing activities, to control borders. It is usually composed of an aircraft, a set of sensors, a crew and a large number of software artifacts. The number of functionalities, the relationships between hardware and software components and the communication between the system and others entities (base, other MSSs) indicate the complexity of the system.

Maritime surveillance system designers have been developing simulators for several decades. Current simulation methods, sometimes processed at early stage of design through technico-operational simulations, do not fully warranty the



**Fig. 1.** Use cases of a maritime surveillance system

design quality. Because there is no link between the system model checked by simulation and the system design. Moreover, the techniques used did not meet the required openness for reuse and modifiability. One of the key point of a simulator is to let the end-user, the customer, to test as many versions of the product as needed, and to let the engineers to add easily new features wanted by the customer.

In this paper, we report an industrial experiment we made to develop a new simulator for maritime surveillance systems following a Model Driven Architecture (MDA) approach [2]. The innovation did not reside in 1) new functionalities since the new simulator does not have more functionalities than the legacy ones 2) neither in the modeling activities since simulation has always been based on simulation models. The explored innovation consisted in the use of MDA.

The main goal of this experiment was to study the applicability and the advantages of MDA for system engineering activities. The experiment was a success. We mainly found that the MDA principles enable:

- to keep a complete independence between different concerns that are system architecture, simulation scenarios, and simulation-based analysis.
- to align the system engineering model and the simulation model which can be both explicitly specified.

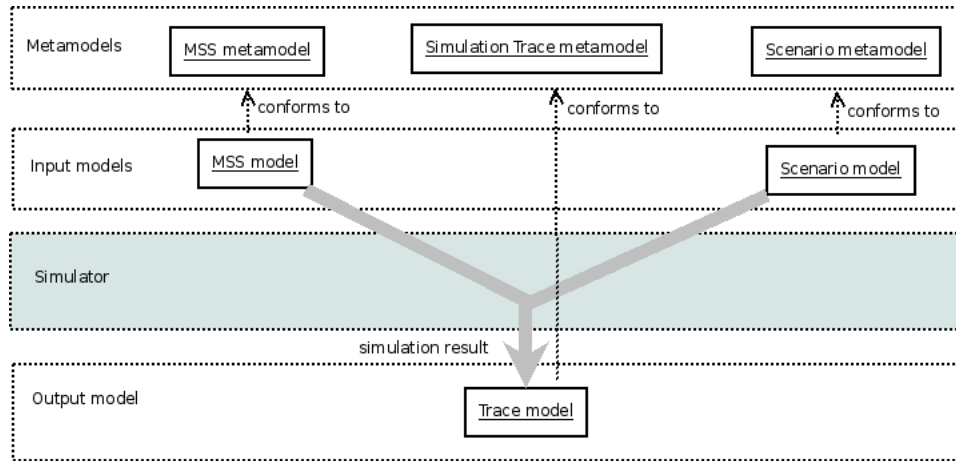
The paper is organized as follows. In section 2, we present the design of the model-driven simulator. We then present the lessons learned in section 3 and conclude.

## 2 The design of the model-driven simulator

Our model-driven simulator was built following the process as follows: 1) identify the domain objects to be modeled; 2) elaborate the corresponding metamodels; 3) design the simulator; 4) develop the simulator.

The remainder of this section does not follow this chronological process. For sake of understandability, we will first present the architecture of the simulator, the metamodels and models used, and we conclude by exploring more in depth how we leverage model orientation in a simulation point of view.

## 2.1 The model-driven simulation process



**Fig. 2.** The MDA architecture of the simulation process

In figure 2 is depicted the architecture of the model-driven simulation process. The backbone of the simulator is the Y pattern depicted as bold lines. The simulator takes as input two models: a model of maritime surveillance system and a model of the tactical situation, called a scenario model. It outputs a model of simulation traces. The three models are specified by three metamodels respectively. The contents of these metamodels are presented in the next sections.

This architecture is built on two model-driven principles: 1) all data are exchanged as models specified by a metamodel 2) the domain concepts are kept independent of implementation concerns. This is slightly different from the Platform Independent Model (PIM) / Platform Model (PM) / Platform Specific Model (PSM) of the seminal MDA paper [2]. However, the architecture follows exactly the same principles. To a certain extent, the MSS model corresponds to a PIM, the scenario model to a PM and the simulation trace model to the PSM.

This architecture has several desirable properties. All the variability of the simulator is encoded into models. For instance, adding a given radar, or changing a feature of the radar to the maritime surveillance system is made by modifying the input MSS model.

The MSS model and the scenario model are not coupled at all. Therefore, we can create several MSS models, several scenario models and study the component of each system in each scenario.

The graphical output of the simulator is absolutely not coupled with the simulation rules. This property enables to easily have many tools that analyze the simulation output. That is to say, we have one and only one interface for plugging modules to the simulator. In such a model-driven architecture, the interface is a model, and is fully, consistently specified by a metamodel, the simulation trace metamodel. This point is further explored in section 2.3.

## 2.2 The metamodels and models used

**Maritime surveillance system metamodel** The maritime surveillance system metamodel is divided in five packages. The criterion used to create the packages is a functional decomposition. The navigation system package contains classes whose roles are to represent routing and positioning components (hardware, software and composite). For instance, there is a GPS<sup>1</sup> class. The attributes of this class are the main characteristics of a GPS.

The detection package contains classes representing detection components of a MSS. For instance, the MSS model can be simulated with a classical radar, a IFF (Identification Friend and Foe), a FLIR (Forward Looking Infra Red). The attributes of a model of the radar include the antenna angle, the rotation speed, the impulse period.

The communication system package explicit the communication type and systems that will be simulated with the model. It ranges from internal communication between the crew members and external communications. The classes covers the communication support (e.g. VHF, IHF, satellite) and communication properties (encryption, protocols).

Finally, the crew package contains classes to specify the number of crew members, their respective skills and their functions. The database package acts like a schema of the embedded database of tactical information (e.g.; radar signatures).

The model mainly used for testing purpose represents a maritime surveillance system composed of an airplane Falcon, embedding a radar and an inertial system.

**Scenario metamodel** The scenario metamodels contains all the needed classes to represent a tactical situation. A model, instance of this metamodel, specifies the surveillance zone, the number and types of objects that are in the zone. For each object is specified a trajectory including the speed of the object. Furthermore, at the system engineering level, and for simulation purposes, we had to specify the concept of surface equivalent radar (SER) per object and the concept of weather for a given zone. During the simulation, the weather is used to compute the quality of the information given by the embedded sensors.

---

<sup>1</sup> Global Positioning System

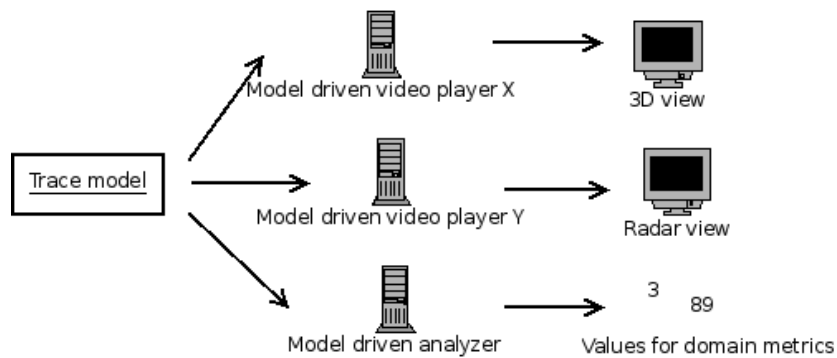
**Simulation trace metamodel** The simulator takes two models as inputs: a MSS model and a scenario model. Then according to the semantics of the simulation, it outputs a simulation trace. Following a model-driven approach, this simulation trace is a model too, specified by a simulation metamodel.

The simulation traces contain all the events that have a semantic with respect to the simulation at the system engineering level. They are :

- the position of the MSS system;
- the position of every objects of the zone;
- the internal attributes of the MSS: fuel, detection field of the radar;
- the semantic events, for instance the detection and identification of the objects achieved by the system core.

In the next section, we show how we use this information to leverage as much as possible of the information given by the simulation.

### 2.3 Leveraging model driven orientation



**Fig. 3.** Usages of the simulation traces

The core of a model-driven simulator is to specify all inputs and outputs with metamodels. To a certain extent, each of these metamodels specify an interface. This enables to introduce a kind of composition between the simulator functions. The MDA architecture of the simulator presented in figure 2 illustrates this point.

However the architecture goes far beyond. The simulation part of figure 2 contains only the simulation rules that represent the interactions between the model and the scenario. The architecture enforces that no analysis is done during this phase.

Our idea is to consider that any treatment on the simulation output could be absolutely decoupled of the rest of the application. The treatments are mainly visualization and measurement. However, there are several possible visualization

as well as several measurements. We encapsulate each of them into a simulation analysis component, that takes as input a simulation trace model specified by the simulation trace metamodel. This is depicted in figure 3. We apply this design principle to several simulation analysis components.

The first component is a video player of the simulation events contained in the simulation trace model. It outputs the tactical view of the zone of surveillance. A screenshot of this player is given figure 4. The main screen is a 2D representation of the zone, with a geographic map, the maritime surveillance system (the yellow dot), the objects of the zone (several boats represented as orange dots), and the detection field of the radar, which is not visible due to the size of the figure. In the upper bar, there are the indicators of the simulation characteristics (time, position and fuel consumption of the MSS). In the left bar is a list of the tactical events that occur, for instance ship detections. The remaining items are widgets to control the simulation (play/pause/stop buttons). The scale that controls the speed of the video player lets to play the scene at a speed different of the simulated time. This simulation component is a demonstrator to communicate with customers as well as to illustrate the impact of a certain system design choice.

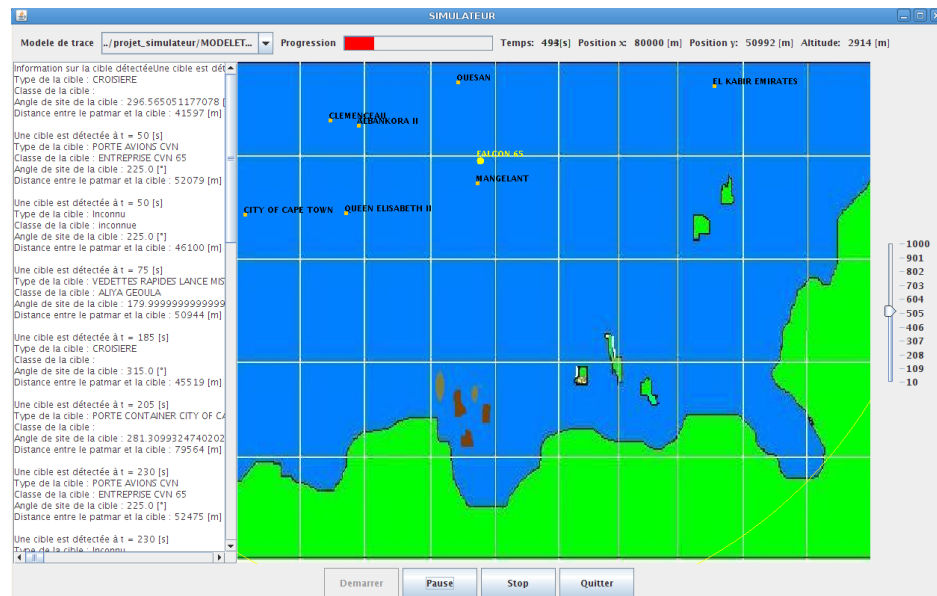


Fig. 4. Screenshot of the model-driven video player

Since we already have human training processes using simulators, we developed another simulation analysis component to represent the view of a radar operator. The last simulation analysis component computes several domain met-

ric values such as the ratio of detected objects or the ratio of identified objects. This component is generated from an abstract specification of domain metrics, thanks to a dedicated framework developed in another project [3].

Finally, since the simulation analysis components are decoupled from the simulation itself, we plan to reuse these components to analyze some traces of real missions. This will enable to elaborate powerful processes for training and debriefing, as well as analyzing real missions in order to improve future versions of the simulator.

### 3 Lessons learned

First of all, this project to apply MDA at the system engineering level was a success. The project met the deadline, the budget and the threshold of desired functionalities. This validates the MDA as an architectural principle for a maritime surveillance system simulator.

*Productivity gain* Our experiment gives some clues about a potential MDA-specific productivity gain. First of all, we obtained the prototype within six months of work of a junior engineer. The technologies used were the Eclipse Modeling Framework [4] coupled to Java to implement the simulator. Thanks to the totally generated EMF model editor, we could right away, and for free, give the system engineers, who are not necessarily computer scientists, a tool to express the models. This is to be compared with the legacy simulators where a specific graphical user interface had to be developed to specify the models involved in the simulation.

*An agility problem* The main disadvantage of EMF is the code generation step. Code generation is not a problem when one does not have to modify or enrich the generated source code (e.g.; a classical mature compiler). But in the EMF case, some methods and method bodies have to be added into the generated code. Our metamodels were very volatile during the prototyping phase, hence required many code re-generation and the associated method modifications. On the one hand, this hampers the agility of the development process. On the other hand, some re-generations were destructive, i.e; destroyed manually added code, due to the number, the complexity and the sensitivity of the generation parameters. This problem was partially addressed with a source revisioning system.

*Separation of concerns* The MDA architecture of the simulation process enables a real separation of concerns. The variability of a MSS simulator lies in the system architecture, the simulation scenarios, the kinds of post-simulation analysis. All the variability of the models is explicitly concentrated into different metamodels. There is no dependency links between the metamodels hence we obtained a total independence between concerns. This architecture is totally open w.r.t. system models, scenarios and analysis; e.g.; we are able to test different scenarios with a same model or perform different analysis from a same simulation.

*Early model alignment* The real innovation of our experiment was the meta-modeling of the domain of maritime surveillance systems. Thanks to this step, the creation of a model involves only domain concepts and is not polluted by implementation concerns. This was perceived as highly valuable: the system engineering model facilitates the communication between the different stakeholders (a kind of mental alignment); it can be linked to a design model; and it is totally aligned with the simulation model by construction.

## 4 Conclusion and perspectives

In this paper, we presented our experiment to apply Model Driven Architecture at the system engineering level. The experiment consisted in developing a model-driven simulator for maritime surveillance systems. This experiment was a success and raises new issues, which will be explored in further R&D projects.

First of all, the domain metamodel can be seen as the specification of a domain specific modeling language (DSML). We will explore the ways to leverage the metamodel with a dedicated human-machine interface that supports an intuitive concrete syntax for the models. Thus, we would have a intuitive and usable modeling backbone to support the system engineering process.

The core of the simulator is an interpreter of a domain model, achieved by dedicated transformation of two models (MSS and scenario) into a simulation trace model. For prototyping and efficiency reasons, it was implemented in Java. We would like to evaluate the relevance of transformation languages in our case, with respect to the requirements specific to simulation.

The product line of maritime surveillance systems is highly driven by the need for composability. We are exploring how to express this composability within the MDA architecture of the simulator. Studies will be made to explore various solutions, e.g.; expressing the composability directly into the maritime surveillance system metamodel, or building a set of composable models of maritime surveillance systems.

Finally, the great promises of MDA for system engineering would have to be compared to other approaches within a controlled comparative experiment, which was not the goal of ours.

## References

1. A. Ince, E. Topuz, E. Panayirci, and C. Isik, *Principles of Integrated Maritime Surveillance Systems*, vol. 527 of *The Springer International Series in Engineering and Computer Science*. Springer, 1999.
2. R. Soley, “Model driven architecture,” tech. rep., Object Management Group, 2000.
3. M. Monperrus, J.-M. Jézéquel, J. Champeau, and B. Hoeltzener, “Measuring models,” in *Model-Driven Software Development: Integrating Quality Assurance* (J. Rech and C. Bunse, eds.), IDEA Group, 2008.
4. F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose, *Eclipse Modeling Framework*. Addison-Wesley, 2004.