Flexible Model-Based Simulation as a System's Design Driver

Jean-Philippe SCHNEIDER[1], Eric SENN[2], Joël CHAMPEAU[1], Loïc LAGADEC[1]

[1] UMR 6285 Lab-STICC, ENSTA Bretagne, 2 rue Franois VERNY 29806 BREST CEDEX 9

[2] UMR 6285 Lab-STICC, Université Bretagne Sud, Rue de Saint-Maudé BP 92116 56321 LORIENT Cedex

Abstract. Complex systems traditionally involve partners from different companies with their own domains of expertise. During design stages, these partners need to exchange pieces of information and to debate around architectural and implementation choices.
Model Driven Engineering for System Engineering simplifies system knowledge sharing, while simulation provides sound results to drive debate. As a consequence, gaining a flexible and dynamic tool that models and simulates the architecture is highly valuable.
In this paper we focus on the functional architecture design and analysis steps of the system engineering process. We identify adaptation to existing system engineering process, tool modularity and interaction with models as three grounding principles for a flexible system model simulation tool. We show that meta-modeling and layered architecture for a simulator are enabling technologies for our three principles. We also demonstrate the use of these technologies by implementing a simulation tool in the context of a sea-floor observatory project.

1 Introduction

System engineering is an interdisciplinary activity during which experts from several domains having an holistic approach look for the near optimal system design to answer to a client's needs [6]. Interdisciplinary approach requires the ability to work with partners with different vocabulary and work techniques. Looking for a near optimal solution implies identifying and comparing multiple design solutions for the system. One of the risks in an interdisciplinary context is that each expert focuses on its domain of expertise and looks for a locally optimal solution (for its domain) that is not necessarily the globally optimal solution (for the system seen as a whole) [10]. Adopting an holistic approach reduces this risk. Holistic approach, interdisciplinarity and near optimal solution are linked problematics. The holistic view of the system must be shared amongst experts. Model-Based System Engineering (MBSE) is an approach in which system engineering artifacts such as the functional architecture are models [4]. Models are abstractions of the system and can be used to share knowledge

between experts [11]. Collaborative design gives all experts the opportunity to express their ideas. It is then possible to take into account the opposite point of views and to find more alternatives of system design. Simulation helps this collaboration by giving a concrete realization to the discussions [3].
The simulation tool used in the collaborative context should be fitted to work across domains of expertise. Each expert should find the main terms defined in its domain. For example, in a sea-floor observatory project, mechanic and software experts are involved. In such a project sensors are deployed underwater and tightness is a major requirement. This requirement has a translation in the mechanic domain and also in the software domain. Mechanically tightness means that the system has seals and is assembled in a manner that ensures that water will not enter into the system. In the software domain, tightness means that there is a way to measure the pressure into the system. In case of an abnormal evolution of pressure indicating a failure in tightness, a message is sent back to supervisors. Mechanical and software models should be put together to enable the simulation of the tightness function. The simulation outputs should be adapted to provide output meaningful for mechanics engineers and also for software engineers.
In this paper, we describe three principles that ground a flexible simulation tool:

- Adaptation to system engineering processes (current, and emergent/future);
- Extensibility of the tooling;
- Interaction with models.

Companies have already defined their system engineering process and use tools to support the process. Our approach is based on the respect of what has already been done in the companies. Multiple domains such as mechanics or software are involved in the design of a system. It is not possible to create a tool taking natively into account all of these domains. Domain experts should be able to add new functionality to the tool and remove those which are not required. Models are prototypes. Domain experts are using them to debate over the design alternatives for the system. They should be able to modify the models according to the result of their discussion. To realize our three principles, we describe a meta-model defining the concepts of a functional architecture. A functional architecture is the definition of the functions implemented in the system, their interactions and their structural layout. The meta-model describes the structure, the communication and the behavior aspects of functions that should be implemented by the system. In a Model-Based System Engineering approach the meta-model ensures independence from existing modeling tools. To obtain a flexible simulation tool, we define a layered and component-based architecture. Layers group tool functionalities according to their degree of specificity to a domain of expertise. Components isolate functionalities and provide a convenient way to reuse them.
The rest of the paper is organized as follows. Section 2 describes some related work. Section 3 first provides a motivating example for this work coming from a sea-floor observatory project we were involved in. Section 4 provides a descrip-

tion of the grounding principles. Section 5 describes the technologies usable to implement a flexible simulation tool.

2 Related Work

Different formalisms can be used to model a system functional architecture such as Enhanced Functional Flow Block Diagram (EFFBD) and SysML models. EFFBDs are an extension of Functional Flow Blocks Diagrams. Functional Flow Blocks Diagrams define functions and their sequence of execution. EFFBD adds data flow to the functional architecture modeling [12] and execution through Timed Petri Nets has been described in [16]. However, once the simulation is started it is not possible to have interaction with the running model.
SysML [17] enables to model a system architecture throughout the system design cycle. Functional architecture is modeled using Block Definition Diagrams for the structure and Activity Diagrams for the behavior. The link between structure and behavior is obtained through an allocation relationship. SysML models can be used as entry model for simulation tools [13] through model transformation. However, in SysML every structural definition (whether implementation or functional) relies on the concept of block. Using the same concepts at the functional and implementation level implies that designers should be careful not introduce implementation details into the functional architecture. The functional architecture should only describe what the system will do with no implementation choices so that multiple architectures are investigated to find the best one.
Two different approaches may be used to enable the simulation of models defined in a modeling tool: extend the modeling or simulation tool to introduce execution capabilities or use one tool to do the modeling, serialize the model in an interchange format which will be imported in the simulation tool.
Extending a modeling tool can be made using a plugin as suggested by Radjenovic and al. [15]. Their approach is structured in three steps with one design model, one simulation model and finally the simulation execution. The simulation tool is extended through a plugin enabling to extend the understanding of multiple input model formalism. However, this approach requires knowledge of the internal functioning of tooling and access through an API, which is not always possible with proprietary tools.
On the opposite, Karsai and al in [9] advocate using an interchange format between modeling and simulation tool. This approach relies on meta-modeling and model transformations. A model transformation is required to transform the architecture model into the interchange format and another one is required to transform the serialized architecture model into a simulation model. The interoperability of the tools relies on the interchange format. The interchange format may be custom as the goal is to provide a model exchange backbone for multiple tools in the same design process. We favored this approach as we do not want to modify existing tools which are known by designers. In our case, we have to adapt to the model serialization format of already used modeling tools.

3 Motivating Example

The MeDON [7] project aims at designing a proof of concept for a sea floor observatory in coastal areas. A sea-floor observatory is made of a set of underwater sensors and of computing servers. The scientific goal of the MeDON observatory is to passively detect sound sources in the area of Brest in France without defining their location. Hydrophones are deployed to acquire underwater sounds. Algorithms were defined to detect contributions to underwater sound above the mean sound level.

During the design phase of the MeDON project, experts from electronic, software and electronic fields among others were involved. Experts worked in their specific field of expertise and had interactions together only during progress study meetings each six month. There was no knowledge repository to store a common view of the observatory design. Decisions impacting every fields of expertise were made according to field specific goals without coordination with other experts. Some of these decisions had huge impacts on work already done. For example, a deployment site was chosen at the beginning of the project. However, this choice had to be modified due to energy supply shortage. This choice was necessary but it implied rework on the deployment of the data computing softwares and on the choice of the servers. With a functional architecture independent from any technology, it would have been possible to reuse a lot of engineering work. Besides, a better communication between experts about the possibility of power supply shortage would have led to the definition of at least two alternatives architectures. So, when the change occurred, the switch of architecture would have been anticipated.

The MeDON project is now being upgraded to include the localization of the sound sources. Learning from our experience, we decide to use a SysML-based system architecture model to define the system architecture and to have a shared view on the system. An algorithm based on the difference of sound arrival time between each hydrophone has been selected to locate sound sources. The sound is acquired by at least three hydrophones in a two dimensional approximation of space. One of the hydrophones is chosen as reference. Each hydrophone acquire sound signals. The acquired sound is then analyzed to detect the presence of a signal higher than the noise. If one is found it is considered as a detection. For each detection on each hydrophone a difference is made between the time of reception on the hydrophone and on the reference. It is then possible to have a location of the sound source [18]. We model a functional decomposition of the algorithm with SysML. The Figure 1 shows the Block Definition Diagram we obtained. A Passive Acoustic Monitoring system *PAMSystem* is made of *Acquisition* and *Computing* functions. The *Acquistion* must contain a *RawAcquisition* function to acquire the raw signal. The *Computing* function must contain a *Localization* function which perform the localization. The *Detection* function analyzes the signal from *RawAcquisition* to check for a value higher than the mean signal value. This function can be grouped either into the *Acquisition* function or the *Computing* function. This is an architecture alternative that should be investigated. We instantiated the architecture in which the *Detection* function

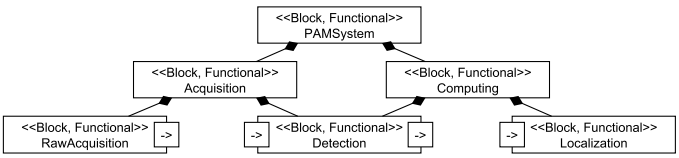

Fig. 1. Block Definition Diagram of a Passive Acoustic Monitoring System

is grouped into the *Acquisition* function. The result is shown Figure 2. In this

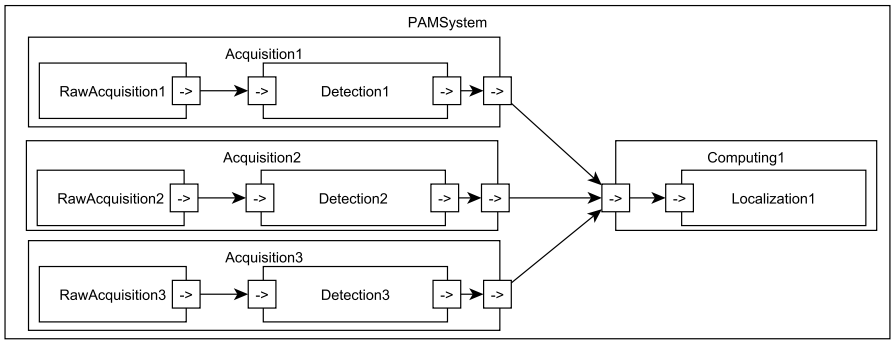

Fig. 2. Internal Block Definition of a Passive Acoustic Monitoring System

work we adopted an approach mixing data scientists' point of view for the block definition and the point of view of software engineers for the data flow. However, this is not enough. We must take into account the point of view of experts from the other domains involved in a sea-floor observatory such as electrical experts. Besides, the tooling is not dynamic and flexible enough to enable users to model and simulate alternative of architecture.

4 Grounding Principles for a flexible simulation tool

In this section, we will introduce three principles underlying the development of our simulation tool. First, we know that industrial companies have well defined system engineering processes. So we think that our tool has to adapt to these processes and not the other way round. Second, we think that the users' needs will continuously evolve so the simulation tool must follow these evolutions. As a consequence, the simulation tool must be extensible. Third, we would like that the model and the simulation become an active support for reflections. This leads to interactions with models.

4.1 Adaptation to Deployed System Engineering Process

The ISO 15288 standard [1] defines the activities performed during the system engineering process. We draw our interest on the functional architecture design step. The functional architecture describe the functions implemented in a system. These functions come from the requirement analysis made with the users. The functions defined through the requirements analysis will be organized in several alternatives of functional architectures. Definition and comparison of functional architecture alternatives are essential as the choices made at this step of the process drive the whole system realization. Simulation is one tool to perform this comparison.
Simulation relies on the modeling activity. Models define the abstraction level of the simulation and serve as entry point. A model driven approach for simulation is made of four steps [2]:

1. Conceptual Modeling: definition of the system model at a given abstraction level.
2. Tool independent simulation modeling: translation of the previous model into a simulation formalism such as Discrete Event. The independence from simulation tool enable reuse of the model.
3. Tool specific simulation model: translation of the previous model into a model using the concepts defined in the chosen simulation tool.
4. Implementation.

In our case, the functional architecture can already be modeled using languages such as SysML [5]. These languages are already used in companies' system engineering process. In order to be adopted, a tool must comply with industrial processes [8]. The modeling of the functional architecture is equivalent to the conceptual modeling step. It is made with existing tools so that we comply with companies' process and tools. As a result, the simulation tool is decoupled from the functional architecture modeling tool while sticking to the system engineering process. A meta-model is provided to describe the elements of the functional architecture to simulate. Intermediate models compliant with the defined meta-model are independent from any simulation tool. The intermediate models can be obtained through model transformations from system engineering tool knowing their meta-models.

4.2 Adaptation to User's Needs

Complex system design requires work from experts coming from multiple domains. Each expert has its own view on the system through its domain vocabulary and also on the metrics given by the simulation. The simulation tool must be able to take into account the differences between domains. Experts need a tool adjusted to the current situation they are facing. Experts should be able to add new functionalities to the simulation tool and remove the ones they are not using.
This requires a modular approach like the one used in the Linux Kernel. The

Linux kernel is made of a set of modules. Each module has an unique purpose such as handling a USB device or printing. A module can be loaded and unloaded. However, some modules are essential to the stability of the system and they can not be unloaded.
Like the Linux kernel, the simulation tool should be made of modules that can be loaded and unloaded by domain experts as required by their current task. Besides, a distinction between modules should be made. Some modules are at the basis of the simulation and so should not be unloaded. Other modules deal with domain specific activities and may be loaded and unloaded at will.

4.3 Interaction with Functional Architecture Models

Interaction with models consists in observing the simulation and its results and in modifying the simulated model. These activities serve the purpose of:

- Enabling to define new alternatives of functional architecture by debating;
- Simulating the different alternatives;
- Comparing the results of the simulation.

One of the goal of functional architecture is to define groupings of functions. In the simulated models the functions groups should be modifiable to perform tests on different alternatives. A function can be seen as an assembly of basic operations. The order in which the basic operations are performed or their nature should be modifiable. Those interactions with models are risky: deadlocks can be created by the modifications. Furthermore, modifications may cause a loss of coherency between the simulated model and the simulation results. The simulation results must be linked to the simulated model. So the simulation engine must take into account all the modifications performed on the model so that the simulation results are still valid.
Tests on the structure of the model should be performed to avoid these risks. At runtime, a deadlock check should also occur. The modeled elements themselves should also provide pieces of information about which modifications users are allowed to perform on them. Unauthorized modifications should be blocked by the model elements.

5 Enabling technologies for the Guiding Principles

In this section, we will introduce the technologies used to implement the three principles. Meta-modeling and model transformation implements the adaptation to system engineering process. Layered architecture helps to implement tool extensibility.

5.1 Meta-Modeling and Model Transformations

We defined a meta-model describing functional architecture models. This meta-model shown Figure 3 is based on the function description in [14]. Functions are

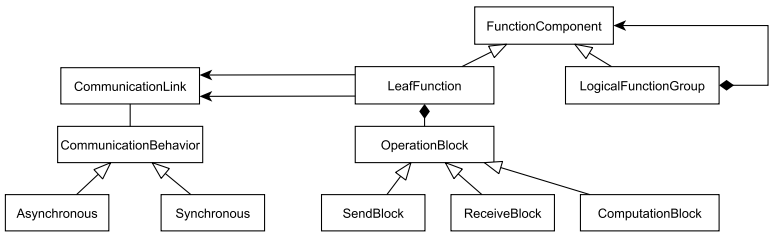

Fig. 3. Meta-model defining a functional architecture

seen as actions performed by a system and are allocated to constituent of the system. We extracted the definition of the functions and elaborate on it. Our meta-model is made of the definition of the functional structure, the communication between functions and the behavior of functions.

The functional structure is described by a Composite pattern between the *FunctionComponent*, *LeafFunction* and *LogicalFunctionGroup* meta-classes. The meta-class *LeafFunction* describes the basic functions of the system. A logical function group (LFG) can be made of other LFGs and of basic functions. in the example of our sea-floor observatory, we have a LFG called Acquisition that groups two *LeafFunction*s *RawAcquisition* and *Detection*. The two latter are *LeafFunction*s as they are not further broken down.

Basic functions can communicate through communication links. Each communication link has a behavior to define if the communication will be synchronous or asynchronous. To bring flexibility in modeling the behavior of functions, we reified the concept of communication link. We decouple the behavior of the communication from the behavior of the function. Changes in the function behavior will not affect the way communications are performed. *LeafFunction*s know each link through an alias. There is one link per exchange between two functions. This mechanism is similar to ports in a component-based modeling. Unlike ports, our modeling of links do not allow broadcast. However, it eases the analyses of the exchanges between functions as exchanges between a sender and multiple receivers must be explicitly modeled.

The behavior of a function can be described as a sequential list of basic operations such as sending or receiving data and performing a computation. The sending operation is described by the name of the communication link on which the data are sent. The receiving operation is described by the name of the communication link from which the data are read. The computation operation is described by the computation duration.

To adapt ourselves to the process and tools used in the industry, the meta-model define an intermediate representation of functional architecture. The functional architecture is modeled using companies' internal modeling tool. Model transformations extract the data relevant to functional architecture from companies' model and create a new model compliant with our meta-model. This new model is used to define the simulation to perform independently from the companies

tools used for modeling the functional architecture at the beginning.
Our meta-model was designed in the context of the sea floor observatory example and uses vocabulary of data processing such as *Computation*. However, it can be easily extended to suit more generic needs. An *Action* meta-class can be created. The *Computation* meta-class will inherit from it. The class *CommunicationBehavior* can be renamed in *LinkBehavior*. Classes detailing the behavior of flows inheriting from *LinkBehavior* can be created.

5.2 Layered Architecture

To obtain tool extensibility we rely on a layered architecture each layer being made of software components. In a layered architecture each layer uses services from the lower level layer and provides services to the upper level one. It is then possible to build a new service providing business oriented data built from tool services. Components have a well-defined interface. Their functional responsibilities are clearly identified. It is possible to switch two components with the same data inputs and outputs. Components may also have the ability to be loaded at runtime. New functionalities can then be added to the tool at runtime.
Using a layered and component-based architecture has several advantages. First, using component enable to co-locate pieces of code having the same role. When a modification is required, locating the area in the code that must be modified is easy. Second, using layers and components requires to clearly identify the interfaces between the components. This enables to write new components and integrate them in the simulator. The only condition is to comply with the interfaces. However, using layers and components for the architecture have some disadvantages. The complexity of the architecture of the simulator may be increased. Information useful in a component may follow a complex path before gaining the targeted component.
We decompose the simulation tool into three layers as shown Figure 4. The *Core*

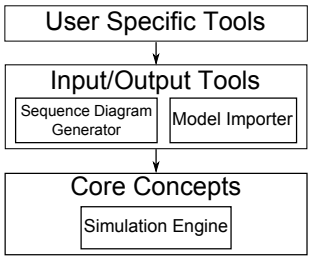

Fig. 4. Decomposition in layers of the tool

Concepts layer contains the simulation engine component. This component contains the implementation of the meta-model.
The functional architecture is modeled outside our simulation tool and model

transformations are performed to import it in our tool. A model importer component should be responsible for performing the model transformation. Besides, the simulation should generate output rendered into an user friendly format. An export module per output format should be implemented. All those components are located into the *Input/Output Tools* layer on top of the *Core Concepts* layer. The *Company Specific Tools* layer will contain the different components developed specifically for the users' needs. For example, a component may be written to compute the global waiting time the different functions and display the obtained value.
An example of component decomposition is the integration of a sequence diagram utilities. Sequence diagrams enable to visualize message exchanges between functions. We defined two components: one that displays directly a sequence diagram and one which creates a log file using a format readable by an external tool in such as the sdedit tool[3]. A component approach is well suited because in both case information about the message exchanges are the same i.e. the identity of the sender, the message name, and the identity of the receiver. Besides, the behaviors of both component are really different, the first manage a display and the second one write data in a file. The link between the execution engine and the components is made through an interface detailing the data structure exchanged between simulator components.

6 Conclusion

In this paper we presented the three necessary principles to obtain a flexible simulation tool at the functional level. First, we think that such a tool should adapt to the system engineering processes already in use in companies and not modify it. Second, we advocate for a tooling able to be adapted to specific needs. Third, the simulation tool must enable to play with the models simulated. We also gave a list of enabling technologies. Meta-modeling and model transformations may support the adaptation to existing system engineering process. Flexibility is achieved through the independence from deployed tooling. Module-based tooling enable the adaptation of the tool to the user needs. Flexibility is achieved by the ability to load and unload modules at will according to the project environment.
A future work is to extend our functional architecture metamodel. We want to reify the composition link between the *LogicalFunctionsGroup* and the *FunctionComponent* metaclasses. It will then be possible to add pieces of information to specify the link such as an alternative identifier. This identifier will ease the process of implementing, simulating and comparing functional architecture alternatives.

Acknowledgments

This work has been done with the financial support of the French Délégation Générale de l'Armement and of the Région Bretagne.

[3] http://sdedit.sourceforge.net/

The authors also wish to thank Zoé Drey and Ciprian Teodorov for their valuable input to this paper.

References

1. Arnold, S.: Iso 15288 systems engineering system life cycle processes. International Standards Organisation (2002)
2. Cetinkaya, D., Verbraeck, A., Seck, M.D.: Model transformation from bpmn to devs in the mdd4ms framework. In: Proceedings of the 2012 Symposium on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium, p. 28. Society for Computer Simulation International (2012)
3. D'Aquino, P., Le Page, C., Bousquet, F., Bah, A.: Using self-designed role-playing games and a multi-agent system to empower a local decision-making process for land use management: The selfcormas experiment in senegal. Journal of artificial societies and social simulation **6**(3) (2003)
4. Estefan, J.A., et al.: Survey of model-based systems engineering (mbse) methodologies. California Institute of Technology, Pasadena, California, USA May **25** (2007)
5. Friedenthal, S., Moore, A., Steiner, R.: A practical guide to SysML: the systems modeling language. Elsevier (2011)
6. Haskins, C., Forsberg, K., Krueger, M., Walden, D., Hamelin, R.D.: Systems engineering handbook. INCOSE. Version **3.2** (2010)
7. Interreg IVA: Marine edata observatory network (2013). http://medon.info/
8. Kapurch, S.J.: NASA Systems Engineering Handbook. DIANE Publishing (2010)
9. Karsai, G., Lang, A., Neema, S.: Design patterns for open tool integration. Software & Systems Modeling **4**(2), 157–170 (2005)
10. Klein, M., Sayama, H., Faratin, P., Bar-Yam, Y.: The dynamics of collaborative design: insights from complex systems and negotiation research. Concurrent Engineering **11**(3), 201–209 (2003)
11. de Lange, D., Guo, J., de Koning, H.P.: Applicability of sysml to the early definition phase of space missions in a concurrent environment. In: Complex Systems Design & Management, pp. 173–185. Springer (2012)
12. Long, J.: Relationships between common graphical representations in system engineering. Vitech white paper, Vitech Corporation, Vienna, VA (2002)
13. McGinnis, L., Ustun, V.: A simple example of sysml-driven simulation. In: Simulation Conference (WSC), Proceedings of the 2009 Winter, pp. 1703–1710. IEEE (2009)
14. Pfister, F., Chapurlat, V., Huchard, M., Nebut, C., et al.: A design pattern meta model for systems engineering. 18th International Federation of Automatic Control (IFAC 2011) (2011)
15. Radjenovic, A., Paige, R.F., Rose, L.M., Woodcock, J., King, S.: A plug-in based approach for uml model simulation. In: Modelling Foundations and Applications, pp. 328–339. Springer (2012)
16. Seidner, C.: Vérification des effbds: model checking en ingénierie système. Ph.D. thesis, Nantes (2009)
17. SysML, O.: Omg systems modeling language (2013)
18. Zimmer, W.M.: Passive acoustic monitoring of cetaceans. Cambridge University Press (2011)