Experiences and Expectations with Model Based System and Software Engineering

Andreas Wortmann, Martin Beet, Dirk Roßkamp
OHB System AG, Bremen, Germany
<Firstname>.<Lastname>@ohb.de

A brief position paper summarizing obstacles and requirements on the application and introduction of model-based engineering from the perspective of and with an emphasis on (space segment) software engineering.

Projects for a long time have engaged ideas of model-based engineering in various flavors in order to optimize quality and efficiency of development. In general, there activities have been carried out in isolated applications not sharing data, structure and processes. At ADCSS 2016 some examples have been presented. They range from flight software development with Rhapsody in C, simulator software based on SMP, SMP2 and ECSS-SMP, hardware/software codesign for specific functions to requirements engineering and systems engineering with SysML.

Common misconceptions and obstacles

While MBSE is around for quite some time there are still quite some misconceptions present. From a language point of view most prominent is the term 'model' itself. With respect to MBS(S)E a model is not a simulator and a model does not refer to a spacecraft model like the EM, PFM or FM. Secondly, MBSE commonly is mistakenly put on a level with using UML or SysML. Many new terms with unclear or overlapping semantics have been introduced, including: digital clone, digital twin, digitalization, digital continuity and many more. Their use sometimes seems rather arbitrary.

The introduction of MBSE methods is a significant change in the way engineers interact and think their projects. It's a long-term process, but there is an implicit expectation that the Return of Invest is quickly achievable in short term. Furthermore, it's blinded to think that everything will be better, more efficient and cheaper, but we don't have to change our way of thinking and invest in related development processes. The assumption that all elements (e.g. software source code, configuration tables) with heritage can be continued to be used without modification in general is not true. Different tools and infrastructure call for different artifacts. On the other side, it's a misconception that replacing artifacts like software code with models inadvertently lead to loss of heritage and previous knowledge. In fact, the opposite is true: If correctly realized the prior knowledge and good design pattern are rigorously applied to all functional code (functions with pre-existing code and new functions) and the heritage is in reuse of implementation concepts and pattern rather than in source code. This is possible by raising the level of abstraction when actually implementing. With the SAVOIR/OSRA ESA is doing exactly this for harmonizing onboard software architecture.

Ongoing activities

At ESA a plurality of activities are carried out striving towards model centric engineering, including the MB4SE and OSMoSE initiatives as well as the various SAVOIR groups and the EGS-CC. At OHB such activities are supported and the internal organization and workflows are aligned. Preceding activities are identified and connect whenever possible in terms of processes and data flow. Various tools and approaches are assessed, including UML based modelling and domain specific languages, and serve a step-wise improvement of established processes and tools.

Essential Requirements

From the experience made so far, some basic requirements against an envisioned model-based engineering environment and platform can be drawn. These are presented in the following.

Todays' systems are large in terms of size and complexity and they are expected to grow even more in future. The engineering platform required must be capable to handle such large systems and keep up with it expected future growth.

Collaboration is essential in large engineering projects. A multitude of different engineering disciplines are working together on a shared model. Two needs arise that at a first glance seem contradicting: On the one hand an engineer requires a stable baseline to base his work on as continuous changes introduced by colleagues while working will stifle progress. On the other hand, an engineer always requires the latest information in order to not design the wrong system. In software engineering transaction-based collaboration tools like git have been introduced and solve these challenges very successfully. In cooperative systems engineering environments such tools will be beneficial as well. In addition to pure transaction based revisioning systems it must be possible (for instance in a concurrent engineering session) to collectively edit a model in a google docs style, where one actually can observe the colleague's cursor.

The tools user interface (UI) is important for acceptance and efficient operation. Overwhelming complexity with 1000-button menus for doing simple jobs are not suitable and error prone. The UI should be scalable with the use case and customized to the engineering task. While the user expects rigorous failure and consistency checks as well as simple analyses to be carried out interactively the tooling must remain live even with very large model being handled. A system that fades for 30 seconds while editing or that requires a "make" button to trigger long-lasting activities that put the engineer on hold will not be accepted in the long term. This specifically holds for models that represent executable systems and test cases.

It's important to model more domain specific aspects of a system than what is achievable when using UML/SysML as these are general purpose languages. While they can be extended via profiles etc. respective models are not very intuitive to read. Multiple paradigms and multiple notations will be required to optimize meta-models and languages for their application in their respective domain. This ranges prose-style (high level requirements) declarative (type system), behavioral (test, math expression) and structural (deployment) languages with textual (requirements), tabular (lookup), graphical (state machine, deployment) or symbolic (math, chemistry) notations.

Outlook

So, how could such an engineering framework look like? Tools including MS Excel and Word that currently are used for connecting artifacts from different engineering disciplines can easily be substituted by a model-based environment. But from the pure amount of highly specialized tools, it seems obvious that there will not be the one "BIG NEW TOOL" that will do everything. Established tools need to collaborate and share a common model. The focus should be on a shared model that can either be directly maintained or that external tools interface with, rather than on an exchange of models among external tool. Such integration calls for a data repository/hub that provides the mentioned collaboration features and it requires the specialized tools to provide interfaces that allow transaction-based or continuous exchange of data.

Due to the specifics of space engineering and peculiarities of the various stakeholder, it is not expected that a suitable out-of-the-box tool will ever be on the market. However, in a collective effort agencies and industry should be able to establish a customizable framework meeting the requirements.