

does not itself significantly contribute directly to the yield, so the weapons are still basically fission devices.

⁴ One way to ensure this is to design the bomb's core so that, notwithstanding its implosion, reaching criticality still requires the boost neutrons that are released following fusion of the surrounding lithium and deuterium gas. The mechanism that activates this fusion is independent of that which detonates the high explosives. Consequently, an accidental activation of the high explosive implosion system will always be insufficient, on its own, to instigate a nuclear reaction.

⁵ For example, short circuits can result from melted solder, charred plastic or from arcing between newly proximate wiring. Errant currents might then energise the firing mechanism. Such risks are minimised by isolating the firing mechanism

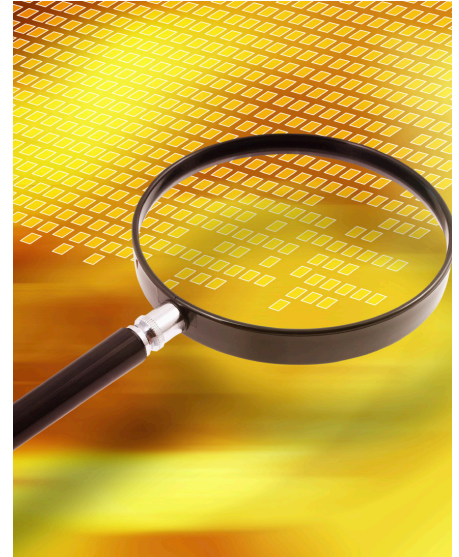
within a physical barrier and then using so-called 'strong links' to connect to external components. 'Weak links' within the isolated section are then designed to predictably fail 'open' in abnormal environmental conditions, long before the strong links have chance to fail in a potentially unpredictable fashion.

⁶ By 'protection', I mean ensuring that your nation's imminent destruction shall not go unnoticed.

⁷ The U.S. terminology for nuclear mishaps is provided by DoD Directive 5230.16. Apparently, an unauthorised launch of an ICBM, resulting in detonation, is an example of a so-called 'NUCFLASH'. I wonder how long it took them to come up with that one.

⁸ To expedite launch procedures, the security code required to activate the Minuteman missile was set to '00000000' in all silos.

John Ridgway is a retired analyst who spent most of his career working for Serco Transportation Systems. Having failed to change the world, he now lives in quiet retirement with his wife and two dogs. He can be contacted at j.ridgway4@ntlworld.com



Requirements in the Loop: The Future of Model-Based System Engineering

by Yves G enevaux, Ken Nathan, Fabien Gaucher & Bruno Joffret

In this article, we consider the level of maturity of Systems Engineering as compared with other phrases of embedded systems development. Although much progress has been made over the years with the introduction of Requirements Management tools, Traceability tools and support for organising the elaboration of an architecture with Model-Based System Engineering (MBSE), essentially through various SysML flavours, there still remains a vast area for improvement. It is

known that something of the order of 70% of the errors introduced during the various activities of an embedded system development project are actually introduced during the specification phase, whereas only some 4% of the errors are discovered during this phase. Figure 1 illustrates the percentage of errors typically introduced and detected during the development process. Given that the general consensus is that the earlier you find the errors, the more efficient and cost-

effective it is, it is clear that the processes and tools used during the specification phase need to be improved.

One blocking factor to improvement is that most of the information relating to the expected behaviour of the system and of its components remains in a number of requirements written in natural language. Imagine dozens, or hundreds, of requirements that are to be considered in parallel, where any of them can be 'activated' at any

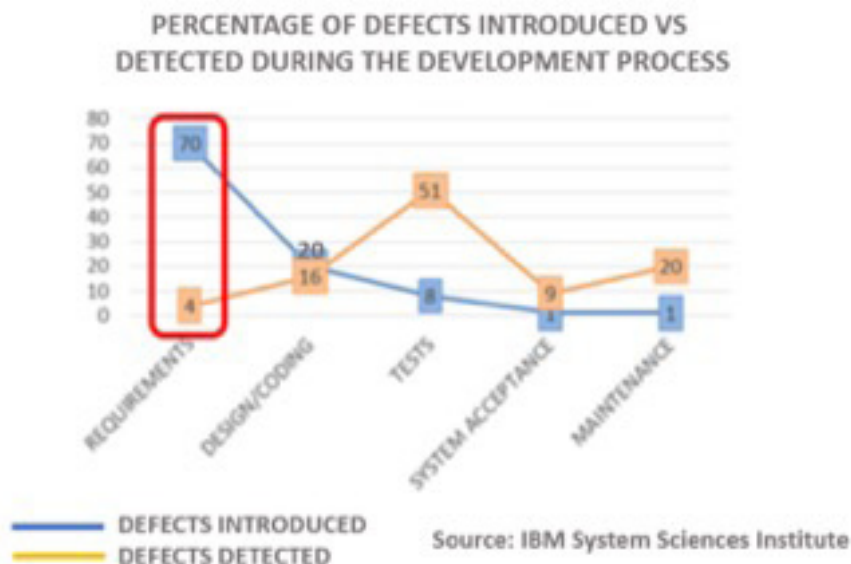


Figure 1: Percentage of defects introduced versus percentage detected by development process phase

moment. The human brain is simply not designed to handle such parallelism. Conflicts, missing requirements, ambiguities and semantics that do not actually reflect the designers' intentions are numerous. Even if MBSE is applied rigorously, a very

large amount of critical functional information is simply out of the model, radically reducing the ambition one can have for the accuracy and correctness of the model. Figure 2 captures this mismatch and its implications.

How can we improve this situation? What fundamental principles should we apply in order to make the specification less error-prone, more effective and supportable by easy-to-use tools?

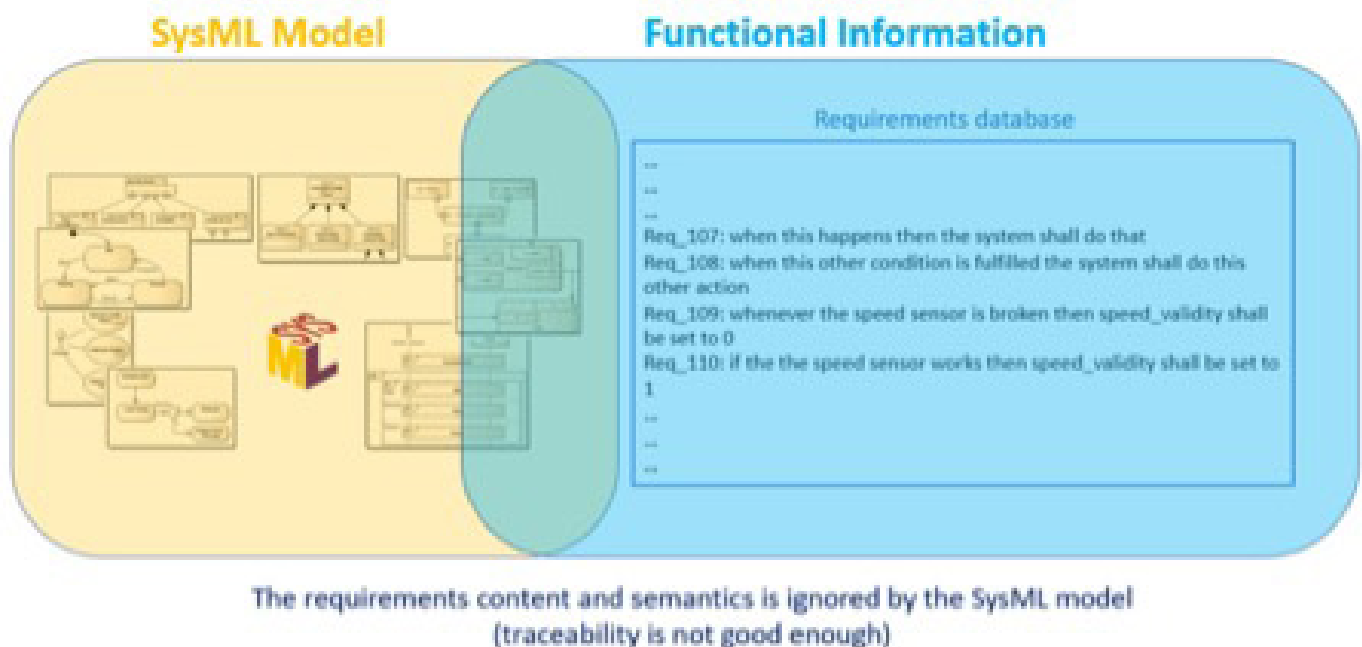


Figure 2: Model-Information mismatch and its implications for requirements

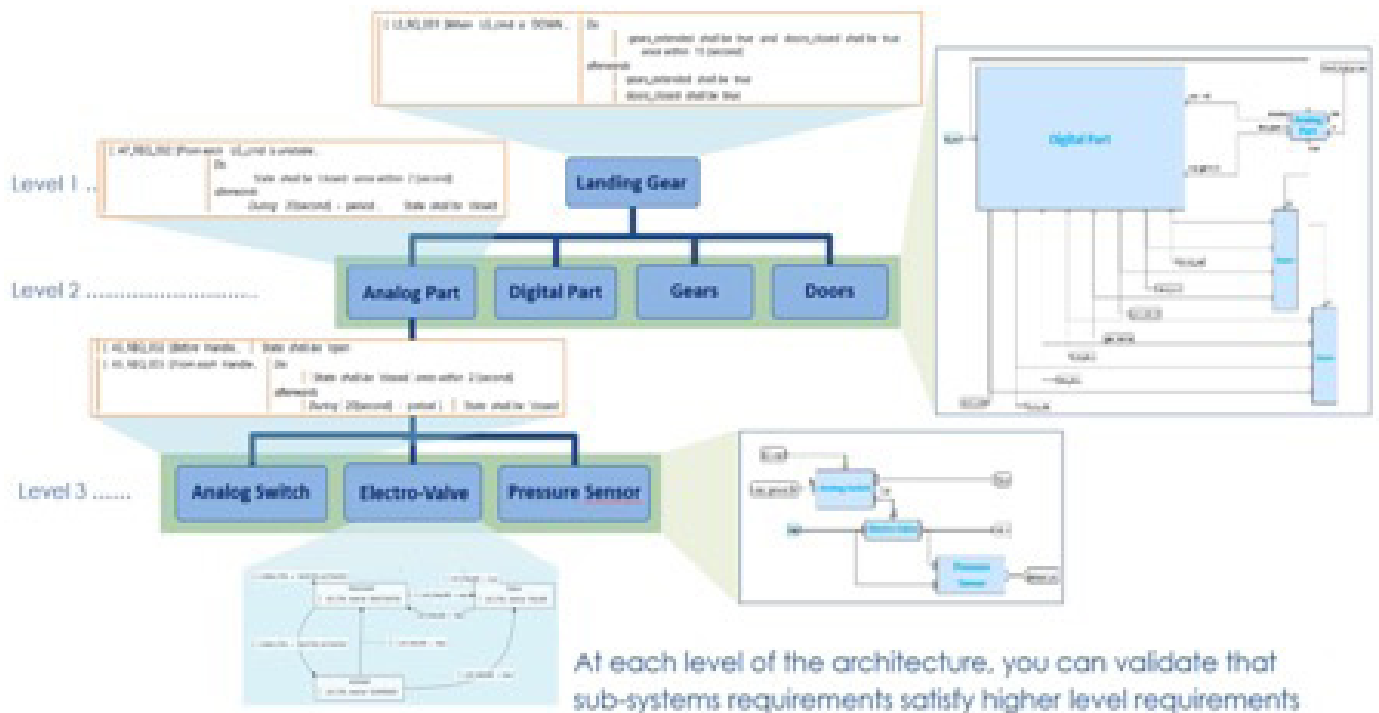


Figure 3: STIMULUS prcess model

We believe that three key points should be taken into account, in designing a better method:

1. The fact that SysML-based specification models ignore the behaviour of the requirements is a severe limitation. The requirements are not part of a model and therefore any attempt to propose a more effective process should ensure that any model includes the requirements.
2. The requirements can no longer be based on free text, or informal templates. This permits the introduction of too many mistakes. However, any solution

should remain very close to natural language, and should also allow customisation so that it is possible for a company to implement company-specific variations according to its particular needs.

3. Simulation is a must. In the world of embedded systems, behaviour matters.

In order to meet these challenges, Argoism proposes a disruptive approach which is supported by a software tool, STIMULUS.

1. As illustrated in Figure 3, the STIMULUS model provides for the description of the system architecture and the interfaces between architecture components with a hierarchy of Block Diagrams. These can be linked to textual requirements and hierarchical state machines, which are used to describe the expected behaviour of these components. Structuring the architecture in this way means that, at each level of the architecture, it is possible to validate that sub-system requirements satisfy requirements at a higher level.

2. As illustrated in Figure

When condition , <BODY>

condition *for more than* duration

x *is less than* y

v1 *shall be* v2



When LightIntensity is less than 60 [percent] *for more than* 2 [second] ,
HeadLight *shall be* 'ON

Figure 4: STIMULUS requirements templates

When switch is 'AUTO' ,
When lightIntensity < 60 % *has been true for more than* 1[second] , headLight shall be 'ON'
When lightIntensity > 70 % *has been true for more than* 1[second] , headLight shall be 'OFF'
When lightIntensity > 60 % and headLight was 'OFF' , headLight shall be 'OFF'
When lightIntensity < 70 % and headLight was 'ON' , headLight shall be 'ON'

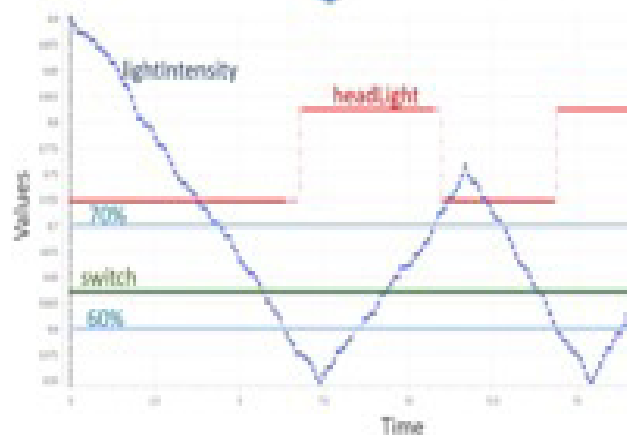


Figure 5: Debugging requirements

- 4, textual requirements are written using a series of predefined templates, called "operators". The library of operators is customisable and extendable by the user.
3. The state machines and textual requirements have executable semantics: simulation is therefore possible. You can now debug a specification in the same way that you would debug a piece of code. More easily, in fact. (Figure 5)



Figure 6: Requirements in the Loop

With these principles, the requirements are not only clear and executable; they are an intrinsic part of the specification model, which is what System Architects have been lacking so far. They are "*In the Loop*", in exactly the same way as other "*In the Loop*" concepts adopted by industry to improve verification processes. Figure 6 illustrates these concepts with reference to a standard V lifecycle. Starting from the right-hand end of the V:

HIL - Hardware in the Loop. The objective is to test the system as close as possible to the production context.

SIL - Software in the Loop. Various execution and emulation environments help in executing and debugging the code.

MIL - Model in the Loop. During the design phase, the models can be activated using simulation, so that

future system behaviour can be verified as early as possible.

That "as early as possible" point has now shifted further up the left-hand side of the V:

RIL - Requirements in the Loop. During the specification phase, the requirements are part of an executable model, which allows the limits of the approaches described above to be overcome.

How does RIL work in STIMULUS?

Following recent research in the field of synchronous languages and verification methods, STIMULUS applies two basic principles:

1. Requirements are considered to be constraints on specified system behaviour.

A constraints solver is therefore able to compute the space of possible system behaviours at a given execution cycle.

2. Use Cases are defined as a set of constraints on and between inputs. The constraints solver defines "acceptable" inputs for a given Use Case, and can generate any number of test vectors, between one and several thousand. This will allow an automatic exploration of the envelope of possible system behaviours. STIMULUS then produces classical simulation plots displaying the evolution of the inputs and outputs during the execution of the model.

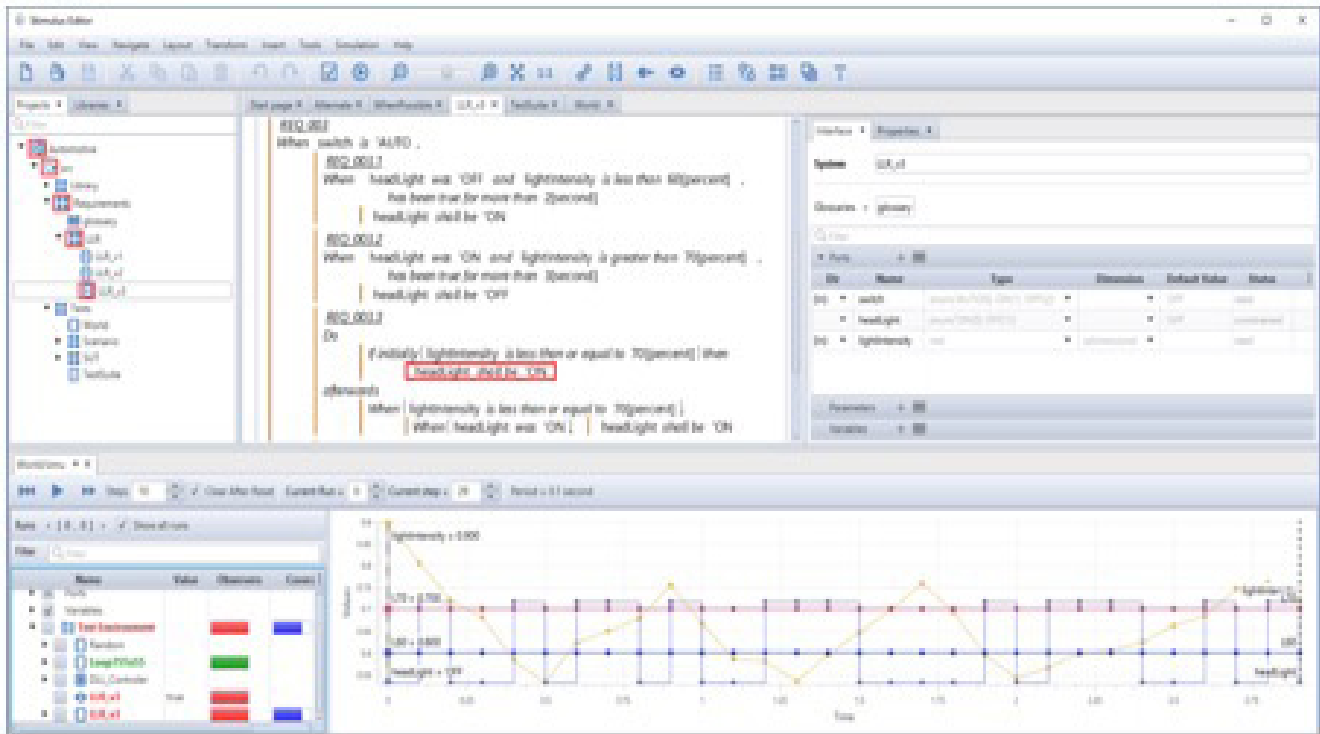


Figure 7: STIMULUS screenshot

The RIL approach has many benefits:

- The requirements elicitation process is supported by a tool. Since one requirement is enough to start simulating, an iterative process can be used to define the requirements incrementally.
- Missing and conflicting requirements are identified automatically.
- You can verify that a specification complies with safety requirements, putting the latter in Observers whilst the specification is simulated by STIMULUS. If the specification violates the safety requirements, the Observer will report it.
- You can verify the

compliance of code with its requirements automatically. Again, using an Observer that contains the specification whilst the code (compiled in a DLL, or an FMU generated by a third-party tool) is executed by STIMULUS. You then have a completely automatic, functional validation environment.

- Different versions of the same specification can be compared. One is executed while the other one is put into an Observer. It becomes possible to verify different hypothesis, variants of the same specification, or the impact of a change.
- The functional coverage is directly measured

by STIMULUS without having to use traceability links, which are always subject to errors. STIMULUS will indicate the requirements that have not been tested during a test campaign so you can design new use cases that will increase the functional coverage.

- The consistency between the requirements of different levels of the system can be verified, which has been extremely difficult to do so far. You can verify if the refinement of the requirements, from the top level down to lower levels, is correct.

In the present industrial context for embedded systems, projects face a stark choice: either explore

new ways of working, or carry on as before and continue to face the consequences of injecting multiple errors into the specification. The consequences of the latter choice are potentially very costly: there is a risk that an organisation will continue to develop the wrong system, and/or incorrect test cases, until realising, towards the end of the project, that project deadlines and budgets cannot be met, because of the lengthy correction loops that are required. The consequences of an error making it through into the final project are potentially very severe. The Requirements in the Loop philosophy, supported by the STIMULUS tool, offers system architects, requirements engineers and test engineers a means of finding errors when they are made, in the specification phase.

For more details of the STIMULUS tool, watch the videos at <https://www.youtube.com/watch?v=aiu5pQRgm7o> and <https://argosim.com/product-videos/>

Ken Nathan graduated from Exeter University with a degree in Physics and Geophysics. He has worked in Sales and Marketing in embedded software and telecommunications for 30 years with such organisations as Vodafone,

Nortel and PolySpace. He has run sales teams across EMEA, US and Australia and is currently the Country Manager for UK and Nordics at Argosim.

Yves G  nevaux graduated from the Ecole des Arts et M  tiers. He has worked in the fields of system and software engineering for embedded and critical systems for more than 25 years which led him to collaborate with key accounts in the aeronautical, railway, automobile, energy, telecommunication and defence industries, both in France and worldwide. After several years working as a development engineer, he developed several start-ups involved in the publishing of development and validation tools (Verilog, PolySpace Technologies) for the French and international markets (Europe, Asia, United-States). He has also worked with large US-based companies such as The Mathworks. He is currently the Sales Director at Argosim.

Bruno JOFRET graduated from EPITA with CSAC major (Cognitive Science and Advanced Computing). He has been working for more than 10 years around numerical simulation. Starting at INRIA, as a software architect, on Scilab Open source numerical computation software, he provided simulation/system expertise through

Scilab Enterprises a company which he co-founded in 2010. He is currently Senior Consultant at Argosim.

Fabien Gaucher graduated from the Grenoble Alpes University and from the Institut National Polytechnique of Grenoble. He holds a PhD in Computer Sciences, carried out at the Verimag laboratory where he studied real-time system validation techniques. He then worked for 10 years at Dassault Syst  mes, where he developed a simulation and code generation workshop for embedded systems. He is currently CEO of Argosim, which he co-founded in 2013 in order to create STIMULUS, the first simulation tool to debug and validate embedded system requirements.

