Report on the workshop
"Control Software Verification"

held at Carnegie Mellon University (Silicon Valley Campus) on June 28, 2013[1]

This workshop was devoted to the broad range of issues surrounding control software verification and validation. In this workshop, "control software" was understood in a very inclusive sense, and referred to humans, fault detection isolation, reconfiguration systems as well.

Following a keynote presentation by Pierre-Loic Garoche (ONERA/U. Iowa), the audience agreed to divide in three thematic groups:

- One group focusing on "Software verification of control systems: Process for modeling and formal validation, issues and recent developments",
- One group focusing on "Human/System/Environment interactions",
- One group focusing on "Bridging the gap between specification people/control designers and computer scientists".

The opinions expressed by these groups are detailed below.

Software verification of control systems: Process for modeling and formal validation, issues and recent developments

For the process of control systems, and, in general, avionics validation, we need to get information from the **real world**. **Observations** together with **models** of the system usually provide such information. Then the data is processed using software. In this process, statistical models play a key role. We need to know how to assess how good such models represent the real systems. Another important question is to know how to push such statistical models as invariant sets for software implementation and guarantee **correct computations**.

As for generating suitable **models** for verification purposes, there are two main approaches that have been used:

Some models are "**designed by human**", based on **physical** properties of the system. Those models help the engineers (i) **understand** the environment of the problem and the interactions of the system with the environment and (ii) **predict** the behaviors of the system. Such models are also those that have been used to develop software. The main issue with this modeling approach is that the behavior of the model can be complicated, and properties and requirements that have been used to build the model are usually expressed in **natural language.** Another issue is that trying to specify **bounds** on state-space variables may lead to over-conservative results, such as **infinite bounds**. When it

[1] This workshop was supported by the Army Research Office, the National Aeronautics and Space Administration, the US National Science Foundation and the French Agence Nationale de la Recherche.

comes to verification of the software, **methods** are used for the verification of software that uses such models are necessarily **informal**.

A second group of models consists of "**synthesized by mathematical tools**". Usually, the model provided by human beings and discussed above is processed and converted to strongly mathematical models to analyze specific properties. **Stochastic models** and **abstract interpretation** are two approaches towards producing models synthesized by mathematical tools. Some research is also done at NASA to provide build mathematical models for **neural networks** (detailed below). The models that are synthesized by mathematical tools enable the **formal verification** of software. An interesting example of model synthesis by mathematical tools is contained in NASA's approach to capture neural network adaptive controllers. Those controllers provide **online learning neural networks** (OLNNs) in combination with a human pilot for off-nominal flight behavior (such as when a plane has been damaged). NASA's research can provide an idea how to perform verification for executable software (**execution based verification**). Validating these systems is difficult, because the controller is changing during flight in a nonlinear way, and because the pilot and the control system have the potential to co-adapt in adverse ways. Hence a new concept is developed for validating such systems using Bayesian statistics methods.

We face several issues for the development of control software and its verification. The first, and probably the most important issue, is that most physical systems are **continuous-time** in nature. However computer-aided software analysis methods usually deal with discrete-time systems. One reason computer-scientists support the use of discrete-time models is that verification of the **boundedness** of the software variables is reported to be easier to do with discrete-time time models (see, for example, abstract interpretation methods). The second issue faced during the verification of control software is the need for **domain specification**. We need to know exactly what the software is supposed to do and what properties are supposed to be satisfied. Indeed, it is very hard to **generically** verify software except for low-level, nonfunctional properties. And the last issue is dealing with **white boxes and black boxes**. It is well known that the engineering structure of the system can hide properties and leave us with black boxes. Hence we need to reproduce the properties that are necessary for verification. In contrast with black-boxes, we may have some other boxes for which all the details are given. Surprisingly, having all details of a particular element may not always be a good thing, particularly when we need to supervise and manage the system from higher levels. A middle solution, which appears to be very powerful, is using **gray boxes**. In such boxes, only important properties useful for verification are presented.

Solving the aforementioned issues, we can move on to the **validation** and the **verification** of control software. Note that there is a subtle difference between *validation* and *verification.* According to computer scientists, validation is "doing the right thing" whereas verification means "doing things right". In other terms, verification deals with lower level tests in system, such as division by zero, etc. Validation deals with functional properties of the system. We need to "**test**" the software for validation, in which we execute the software. In addition to those traditional tests, we can use formal methods for

verification or validation using models produced to best extract/present specific properties. In this process, we do not execute the software, but we implement particular reasoning provided by formal methods. Correctness of the software operation is concluded in that way.

A recent interesting study is the development of **stochastic approaches** for verification. The word stochastic is used in a different way from what it means in control engineering and refers to a particular method to analyze the software verification problem. The method helps in selecting **abstract domains** and the best possible **verification technology,** if we can precisely define the properties of the problem and the data flow in the corresponding software.

In conclusion, all members of the team agreed on the necessity of working on **continuous-time** control systems particularly to translate properties in that domain for formal verification.

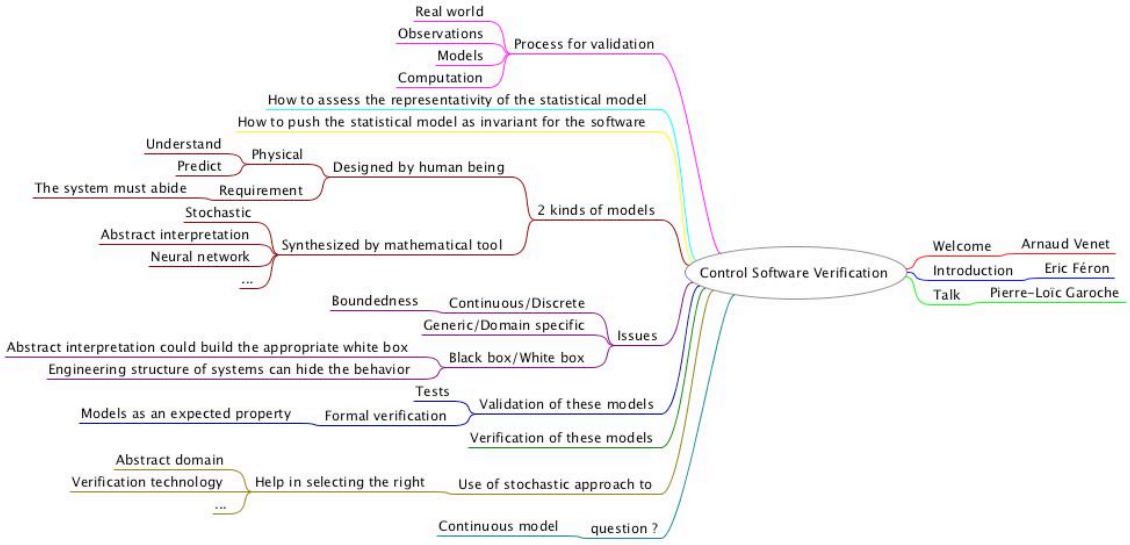

Figure: Verification/Validation group thought development process.

Human/System/Environment interactions

A comprehensive cyber-physical aerospace system validation based on increased formal analyses and decreased flight tests must rely on formal description of human and environment behaviors, in addition to software and hardware behaviors.

System integration issues: Beyond code and people integration, semantic integration

One of the key underpinnings of testing in operational condition (flight testing), or partially operational conditions (hardware-in-the-loop, moving platform simulations) is

the compatibility between the different elements of the system: The human communicates with the control system via proper joystick inputs, the avionics system communicates with the control surfaces and the engines via the proper digital communication network, the avionics systems communicates with the human via a properly designed set of visual, auditory or other interfaces. Integrating the various operational elements of a system represents a considerable challenge, which is, however, surpassed by that of integrating software simulations (operating or denotational semantics) of the same elements. The biggest intellectual challenge, however, is about integrating not only the operating/denotational semantics of these elements, but also their axiomatic semantics, that is, the properties of these elements. Only the latter type of integration will allow researchers to perform strictly formal validations of the system as a whole without systematically resorting to costly and time-consuming flight tests and simulations.

Human behavior representations

Under the impulse of the a variety of agency-supported programs, such as NASA's VVFCS (Verification and Validation of Flight Control Systems), there have been dedicated efforts to extract essential human operating semantics.

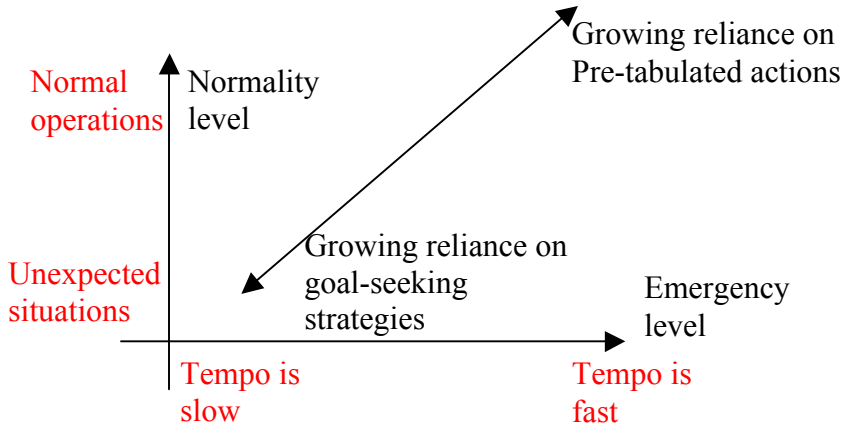

Figure 1: Possible transition map from imperative (procedural) to declarative (goal-seeking) behaviors by human subjects.

These operating (or, rather, denotational, that is, input/output) semantics of humans are still the subject of much discussion, although the necessity to develop them is amply justified by the considerably lower cost of *simulating* human interaction with the rest of the system. *Imperative* models, often mixing continuous dynamics with discrete mode switches that follow standard operating procedures, are placed in competition with *declarative* models where goals are sought (possibly in the presence of competitors) irrespective of what methods are used to seek them. How these models relate to environmental conditions is tentatively sketched in Fig. 1. Conditions under which humans switch between declarative and imperative modes are still unclear.

The denotational semantics of the human subject may be written in high (eg simulink) or low (eg C) codes. These may not need to be deterministic. Denotational semantics of humans form the basic building block allowing them to be simulated in Hardware-In-The-Loop or Software In the Loop environments. These semantics may then form the basis for more compact, property-based axiomatic semantics, which can be established by looking at human behavior alone (open-loop), or as it interacts with other system elements (closed-loop). Many closed-loop properties may be extracted from the composition of the human open-loop axiomatic semantics via composition rules. However, such compositions require unity of modeling grammar and meanings, to describe human behavior and it is an important topic of current and future research.

Semantic models of humans and simulation execution

Human models may not be (and are likely not) deterministic: The complexity of human behavior, its sensitivity to myriads of external events, the presence of fundamentally nonlinear dynamics in the way human schedule and execute tasks all concur to integrate lack of predictability in human models. One of the results is that simulating human behaviors at the component or system levels require extensive computational efforts from the onset, including several repeated simulations to reach statistical significance or "appropriate" confidence intervals. For those reasons, nondeterministic models may better lend themselves to ensemble simulations, such as abstract interpretation, or stochastic simulations involving the partial differential equations driving the underlying probability distributions. However, obtaining axiomatic semantics for complex human work and mental models that go beyond simple regulation tasks is still a long term goal.

Environmental modeling

Not much was discussed there. However, consensus was reached about the necessity to express the environment's denotational and axiomatic semantics in ways that are unambiguous and mathematically rigorous. This task may be particularly arduous when developing semantic models for complex sensor systems, such as video, radar, sonar, or audio sensors, if formal overall system validation is sought.

Bridging the gap between specification people/control designers and computer scientists

Most system design processes usually begin with a requirements analysis phase at very high level where the requirements are expressed in natural languages. A first gap to bridge is then to express those requirements as domain specific languages for example to a formal language but not necessarily. This phase could be achieved either manually or through semi-formalized tools that could support semantics extraction of domain specific requirements from natural language specification.

Building those systems is a living process where both the code and the requirements are updated all along the process. Providing tools that keep this consistency of the

requirements at various levels and could maintain the dependencies among such requirements, seems to be mandatory.

Many topics were considered to be worth further examination:

- From top to bottom: instrumented traceability from design levels to code levels, achieving consistency checking.
- From bottom to top: give the feedback of analysis results.

Since complex systems rely increasingly on software, it is a real issue for engineers and designer to be able to translate domain-specific requirements to the kind of properties that could be addressed at model or code level. What is needed is more domain specific support: focusing for example on specific systems like aircraft controllers, we need to consider the boundary between the two communities and characterize:

- the controllers' interesting properties ;
- the underlying mathematical theories supporting the analyses of these controllers;

and try to

- express those properties in the existing tools/methods at model/code level;
- extend those tools/methods with the required mathematical notions.

All those topics are more or less weakly addressed in the current industrial processes, but they do not handle the complete process and they are not fully instrumented:

- Traceability is usually performed at a domain specific level, eg. DO-178 requires traceability on code but does not require traceability up to the system level requirements. The modification of requirements from upper system level to the lowest level is not automatic at all.
- Few domain specific properties are really analyzable at model/code level. In some cases, even the usual test-based approach is not suitable.
- Feedback, positive or negative, from low level requirement to higher ones is also mainly manual. For example, test oracles are often humans, whereas the specification could be implemented and its evaluation automated.

Full process traceability

A sketch of full process traceability process is given below

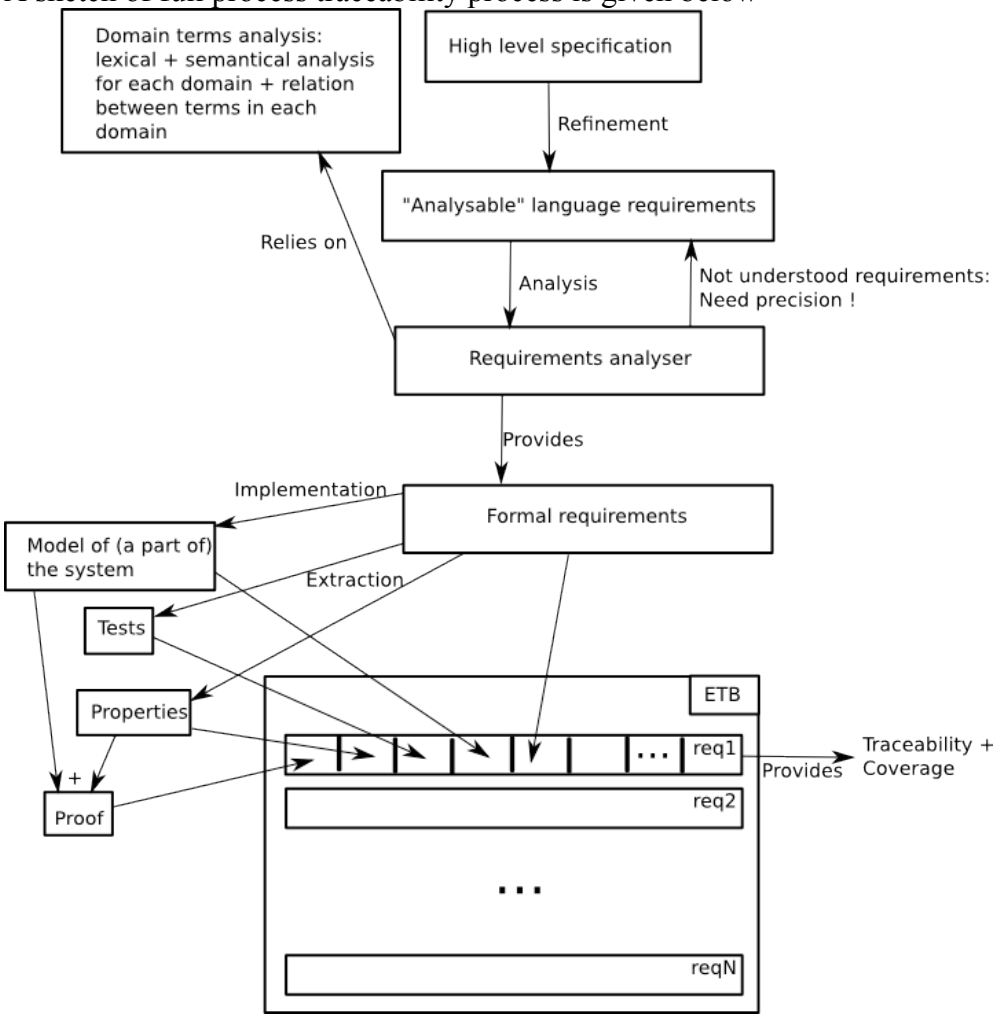

Figure: Full process traceability

Providing means to support the V&V process should also provide feedback either at the software level, at the higher levels of the software design process, or at even higher levels, eg at the control design level, which is the specification level.

Some open questions arise for each kind of analysis: either in case of success (the property is proven), or failure (the property is false, or we failed to prove it is true), we must find a way to provide feedback on the status of the property of interest to the domain level. For example, the current V&V practice relies on the massive use of tests; the challenge is to find how these tests can be made most useful at the specification level? In this context, the usefulness of a test suite can be (i) coverage, or (ii) its ability to properly evaluate a the system's compliance with a specific requirement.

Natively handling domain specific theories

While a growing number of formal methods is applicable to realistic and industry-level code, the set of properties being analyzed, that is, properties supported by tools that scale up to large codes is rather weak. For example, the major success stories at code level essentially focus on run time errors. However, having a system free of run time errors is not a target in itself for the system designer and the developer. Rather, they are interested by the analysis of properties that relate to the initial system specifications.

Addressing those domain specific properties and the underlying mathematical background should really be a key point in the future research project in CPS. They should address properties at the boundaries between the properties supported by domain theories and the validation of those properties the software. One recent example is the development of closed-loop system stability validation procedures at the code level.