Safety and Behavior Specification using the Architecture Analysis and Design Language

featuring Julien Delange interviewed by Suzanne Miller

Suzanne Miller: Welcome to the SEI Podcast Series, a production of the Carnegie Mellon University Software Engineering Institute. The SEI is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

My name is Suzanne Miller. I am a principal researcher here at the SEI. Today, I am very pleased to introduce you to Julien Delange, who is a researcher here at the SEI since November of 2012. Julien's work focuses on the Architecture Analysis and Design Language, also called AADL. Before joining the SEI, he worked as a software engineer at the European Space Agency where he led and contributed to several research projects related to software and system architectures. He also has a Ph.D. from Telecom ParisTech in France and developed the real-time operating system called POK for building safe and secure systems, something that's very important for both space and many other areas of our work.

In today's podcast, we're going to be talking about two new developments with the Architecture Analysis and Design Language: the behavior annex and the error model annex. Welcome, Julien.

Julien Delange: Thanks for having me, Suzanne.

Suzanne: Absolutely. Julien, let's start off by having you give us a little background on Architecture Analysis and Design Language. I know we've interviewed you and Peter Feiler before for this work, but please explain a little bit about it for those who are not familiar with AADL, as it is commonly known.

Julien: Well, AADL is a language, and we started [work on this] many years ago. The first version was out in 2003.We made a new version in 2008. The main goal of AADL is to capture software and system concerns. So, for example, the different interfaces between the system, the data; also, the different type of components, if this is a device, a software command like a task, a process.

The non-functional properties, for example, if we have a periodic task, a sporadic task, the timing aspect, how fast a device will send the data, and so on.

Suzanne: The thing that I know about AADL that I think makes it very important to many of our listeners is that, unlike other languages that work at system level, you could have multiple levels of abstraction of the components that you are putting into a model that is based on AADL. So, if I only know a little bit about timing, I can put in what I know, and if I know a lot about particular internal interfaces, I can put that detail in there, too. Having those different levels of abstraction available and allowing you to increase the data as you go to improve the model, that's a unique aspect of AADL that I think our listeners would be very interested in.

Julien: That's correct. That is [what] we call the incremental modeling aspect, meaning that you start with a really high-level description, and then—when you go through the development process—you refine the component description. You enrich the specification and add more detail and properties. In that aspect, AADL is a unique language. Also, we use the unique capability of AADL in different projects. For example, we have a project that's called SAVI.

Suzanne: What does SAVI stand for?

Julien: SAVI is a System Architecture Virtual Integration. The goal of SAVI is to make use of different modeling languages to design avionics architecture. At the SEI, we are taking part of SAVI and we are applying AADL techniques to design and develop avionics architecture. So far, we have used AADL to design avionics architecture and analyze different aspects and what you call quality attributes, like performance, latency, safety, and so on.

Suzanne: So, with these most recent additions to AADL, the behavior annex and the error model annex, you are extending the capabilities for incremental modeling and for interacting with other kinds of modeling languages, so that you can get a better picture of what's going on with the system you are trying to model, yes?

Julien: That's correct. So far we have many different annexes in the language, and you can define your own annex. At the SEI last year, we developed the error model annex. We also designed new tools to analyze the system from a safety perspective. So, for example, when you start to develop your architecture, you can define the error propagation: what kind of error you have in your system. For example, a device can send invalid data; a bad value can send a value too late or too early. In fact, you want to capture this kind of issue to see the error impact across the architecture.

Suzanne: These kinds of things were difficult to model in other languages in the context of the larger system. So, one of the things that is really important about AADL is that it is helping to bridge the gap between the software engineering architecture and the system's architecture.

Julien: That's correct. Also, something that is very unique is, we have all these specifications in the same language, the same model. We don't have different models. So, we can capture the different inconsistencies in this single model. So, yes, we have different annexes. In the last year, we developed the error model annex at SEI. The goal of this annex is to define the safety concerns of your system.

You can start with these high-level components and say what kind of different faults you have. Having all the different aspects of your system in a single location also brings you the ability to check inconsistencies between different aspects of your system. For example, if you have a late value, this is an error, but this error can be triggered by a bad behavior specification. In other words, if you take too much time to compute a data, it can result in a late value. So, you have this error propagation, and you can see the impact. This is really important, because, in fact, most of the issues of your system are in the specification, when you design the system in the architecture. So, almost 70 to 80 percent of the faults or errors of the system are found in the design. But, the thing is, you will test your system, and you will find exactly where the error happens in the testing.

Suzanne: Which is much later in the lifecycle.

Julien: Exactly.

Suzanne: So, what this is doing is it's allowing us to bring that error detection much earlier in system design and allow you to make design choices that prevent certain kinds of errors that you really couldn't do before.

Julien: Exactly. I will try to describe one problem in a project. For example, in the project Ariane 5, (this is a European project), they have a rocket, and they launched the rocket. They reuse the software from Ariane 4. In fact, the problem is that when they reused the software, they didn't expect to have a change in the value in the interface. Because of that, the software failed and the rocket exploded. So, because of that, you have to check when you integrate the system, you have to check that your system can work. Most of the time it happens that you discover such errors during the integration testing campaign.

Suzanne: Right. That's very expensive as well, as if you have a catastrophic failure. It really puts the whole schedule and the project in jeopardy if that happens. So, if you can model those behaviors—what you said about this all being in a single model, I don't know. If you are not in system design, you may not really understand that what typically happens is there are multiple models that deal with different specialties related to the system. The integration of those models into something that can be used to do end-to-end simulation is almost a whole project in and of itself. So, this move towards being able to not only model the basic behavior, the intended behavior, but also to model potential errors, is really, in my mind, a huge breakthrough in system modeling.

Julien: This is something we are working on right now. For different models, for different domains of engineering—meaning that, for example, many people will use Simulink because Simulink is really convenient.

Suzanne: Yes, Simulink is everywhere.

Julien: Yes, exactly. You also want to make sure that your architectural model is consistent with your Simulink model. What we are working on right now is to check that your AADL model, your architecture model, is compliant with the Simulink model you can develop early on.

Also, most of the time, when you want to make a rapid prototyping of your system, you make a Simulink model. You test and everything; it works. But after that, you want to move on and start the architectural model. So, what we are doing right now is—we describe that process in a blog post recently— you take your Simulink model and you can start an AADL skeleton model by importing the Simulink model. Then you can refine it and explain what is the execution platform, what are the different processors and buses and all this kind of thing, and, later on, add the timing specification of your system. Then you can start to make a latency analysis. Then you start to add also some error specification, and you can see the full propagation and so on. It is really an incremental process. You can start early on by having a high-level description, import that into an AADL model, start to make some analyses and so on. With the model, you can really find errors from the specification. Before all implementation efforts, you can discover a lot, a lot of issues.

We are also working on a new project that will help us to define the architecture issues from the early architecture. In other words, when you import a Simulink model, you have an AADL model, and it will highlight some architecture issues. For example, if you have nested state machines, in that case, you might need to share some global variables. For example, global variable can be a major issue when you start to validate your system because this is really difficult to track.

Suzanne: Where is it? Where is it in the system at any point in time, yes?

Julien: Who modifies the data, and so on. So, you want to avoid this kind of thing.

Suzanne: That's a security issue as much as anything else.

Julien: Exactly.

Suzanne: So, there are lots of implications from some of these architectural decisions.

Julien: Exactly. So you can, in fact, detect this kind of architecture problem and then propose some alternative design. The thing is, it's really important when you modify your system and update your system because early on, 20 years ago, global variable was a common paradigm for performance because you know, it was really convenient. But right now, it's a problem because performance is no longer such a big issue.

Suzanne: Right. Our hardware has really accelerated, so we're not looking for the software to give us all the performance advantages that we did 20 years ago.

Julien: But, we are really careful because the size of software [has grown] so much that it's so difficult to check who accessed the data or not. So, using some practice like encapsulation, controlling who is accessing the data, and so on, it's really better to do that. So, when you change the software and update the software, you can adapt the architecture and use some common practices to encapsulate the data and so on. So, there is a different architecture pattern. You can detect good and bad architecture patterns, and you can try to…

Suzanne: The nice synergy here is that the SEI has been working with architectural patterns and architectural issues for many, many years. So, there's a lot of expertise inside the SEI that can help with identifying some of the issues that need to be incorporated into this kind of analysis. That's really very synergistic.

Julien: Yes, and we have a lot of people interested in this approach, especially in the safety-critical domain. If you think about all the military systems, all the aerospace systems, they have to be updated periodically. Also, when they make a new version of the system updating the software, they have to maintain it for 20, 30, 40 years.

Suzanne: Yes, and we know of some that are that old, yes, and still flying.

Julien: Yes. We have a lot of interest from the avionics community. Right now, many people from the avionics community were interested in AADL because they have a certification process…

Suzanne: Air worthiness certification, yes.

Julien: Exactly. But right now, the automotive domain starts to really have interest in this technology, but also the FDA: we have a lot of interest from other domains as well.

Suzanne: Yes, and our listeners may not realize that AADL is a language of the Society of Automotive Engineers. So, it is actually an international standard, and it is governed by a committee that the SEI's very active on. So, it is very appropriate that the automotive domain starts paying attention to it, I think.

Speaking of the ADDL committee, you just recently returned from the meeting with the AADL committee. Is this new project what it has been decided that the committee is going to be working on? Or, are there other things you want to tell us about what's going on with the standards aspect of the language?

Julien: Yes, so there are three things I'm really excited about. First of all, AADL needs a graphical editor, the tool itself.

Suzanne: It needs better graphical tools. I agree with that.

Julien: So, we have two notations in AADL: a graphical and a textual. So far, we use mostly the textual notation. Textual is really convenient when you know the language, but for…

Suzanne: For learning the language, it's very challenging.

Julien: Right now we have a new project for a graphical editor that works really well, and integrate that into the stable release of the AADL tool, OSATE that we developed at the SEI. We are really excited about it because it works pretty well. It's already used by many users. We will develop new features. It is really good because many new people can learn AADL really easily.

Another project that we are really excited about is to capture different network architectures. For example, all the networks in the automotive domain like CAN or also all the networks in the avionics domain. Finally, the last project I'm really excited about is the code generation from AADL.

Suzanne: Code generation, here we go again. Everybody wants to do code generation.

Julien: No. OK, if you think a little bit about it, so I worked on that topic when I did my thesis so, so long ago. The really nice aspect of it is that we go from a really high-level description to go down to the code. If you think about it, this is what we do with C. We abstract the assembly language in C, and then with Ada, what we did is that we abstract more, for example, the tasking issue, and then you can make additional checking. For example, if you apply what we call Ravenscar profile on Ada, for example, you have to use data protection with locking protocols to provide many safety issues.

Right now, we abstract the software with a model, and we generate the code. We abstract all the big concerns system. We can make sure, by evaluating the model early on, that we avoid any security or safety issues. Then, we make the software protection better. In fact, if you think about it, what we do with programming languages, we abstract the main concerns, and try to validate them to make sure that you have no problem, no indeterminism and so on. Then, you generate the code. I think this is a way to go to generate more safe and secure systems.

Suzanne: So, the reason earlier that I said "here we go again" is we've seen at the SEI multiple attempts to do code generation. Some of them in specific domains have worked quite well, but it also has its own set of difficulties in terms of bloat and other kinds of issues that come up when you're trying to be much more deterministic without knowing the whole context. I am going to be very interested to see how AADL deals with some of those kinds of issues as this project progresses.

Julien: I think, just for safety-critical systems, it is really doable because the semantics and the code patterns are really common most of the time.

Suzanne: Well, yes. You've got restrictions on them. You've got enough constraints. This may be a domain where you actually have enough constraints that it's more feasible to do that.

This is very exciting. I think this is some of my favorite work at the SEI. Those of you that have listened to me before have heard me say that. I really want to thank you, Julien, for joining us today and catching us up on what's been going on.

Those of you that have not used AADL before, I do encourage you to go to the wiki site, http://www.aadl.info/. That's where you can get information about OSATE, which is the implementation toolset that the SEI has developed, as well as reading about the standard itself. Go try it. If you are in safety critical systems, I hope you're already using it.

Julien: Also, we have the graphical editor right now, so it's more convenient.

Suzanne: Yes, that's right. The graphical interface, so you're finally going to get to go beyond text.

To read the series of blog posts that Julien referred to that he authored along with Peter Feiler, please visit blog.sei.cmu.edu, and click on the Architecture Analysis and Design Language tab.

This podcast is available on the SEI website at sei.cmu.edu/podcasts, and on Carnegie Mellon University's iTunes U site. As always, if you have any questions, please don't hesitate to e-mail us at info@sei.cmu.edu. Thank you for listening.