Coding with AADL (Architecture Analysis & 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 at Carnegie Mellon University in Pittsburgh, Pennsylvania. 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, an SEI researcher since November 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 Paris Tech in France and developed the real-time operating system called POK for building safe and secure systems, something that is very important for both space and many other areas of the work of the SEI.

In today's podcast we're going to talk about coding with AADL. AADL is a modeling language, and so coding is not necessarily part of the base language. So, let us talk about what it means to code with AADL. Welcome, Julien.

Julien Delange: Thanks for having me, Suzanne.

Suzanne: So, for those that don't know about the Architecture, Analysis and Design Language (AADL)—we have interviewed you before about this work—but please just explain just a little bit for those that aren't familiar with AADL, what it is meant to do and how it does it.

Julien: So, AADL is a language to capture your software architecture and also explains deployment of the architecture and different execution components, like processors, bus, and so on. By using AADL, you can specify what are the different constraints and the requirements of your system, how you execute different sub-program functions of the different tasks, of different tasks with their different periods.

There are aspects such as timing, safety, security requirement. You can use the language to support different aspects of your software development process such as specification. So, you specify your system is a language, but also variation, or you can validate your requirement using the AADL language. And for that we have different plug-ins in the OSATE toolset. OSATE is a reference tool to use AADL and to process AADL models.

Suzanne: That's an Eclipse-based tool, which is an open-source environment for doing process kinds of work?

Julien: That's correct. Also, the OSATE toolset, that is an Eclipse plug-in, is an open source. Anybody can download it. We have several analysis plug-ins. For people that are interested in a presentation and overview, I did a webinar two months ago at the SEI, which is publicly available. Feel free to research in the SEI library, and you will find it. You can have a good introduction about the language, and we have the model available publicly to use it.

Suzanne: So, one of the things that [have] always excited me about AADL as a language is that, unlike other kinds of modeling environments, it allows you to say what you know at the time about, say, security. You may not know everything right now about security, so you can say what you know. Let's say, for component one, you know a little bit, and, for component two, you know a lot; both of those can coexist in the language at the same time, which is very different from other modeling languages. So, I am really curious about how does that play when you start talking about generating code from the model? How do you go about dealing with that, and what is it that AADL's code can do that may not be possible in other settings?

Julien: So, first of all, we have to say that AADL is a language that describes system architecture and not really the behavior of the system itself. In other words, right now what is happening is we have many, many languages that specify a behavior such as Simulink, SCADE, and all that. If you look at it, Airbus just certified the new airplane, A350. They used a modeling language called SCADE, but it is to specify a behavior. Right now AADL will be used to specify how you plug these components, the behavior of the system, the function, on the run-time architecture; how you associate the different components into the different processors; or, how does it communicate with a bus, all these kinds of things? After that, AADL can be used to generate all the configuration files for the operating system to configure the bus and other run-time architecture, create the tasks, and so on.

Suzanne: So, the code that is being generated is not behavioral code, but it is the code that allows you to configure the run-time architecture so that that behavioral code will perform as it is expected?

Julien: That is correct. This is exactly what we are looking for. In other words, you can--with the automatic code generation--you can assess different architecture alternatives. If, for example, you are looking to use a single processor with different tasks, that can be a good thing, because you lower the cost because you are just using one processor. Sometimes you want to distribute the system two processors and use redundancy patterns.

Suzanne: You may have a performance requirement that you have modeled in AADL that leads you to look at a multi-processor solution. You can then see what the effect on that behavior code is of doing that different configuration.

Julien: Exactly. So, you keep exactly the same code. What you change is just the deployment policy. With AADL, you can automatically generate this deployment policy and configure the different operating system. So, which comes, which invite me to present the benefit as a co-generation. This is a huge benefit because if you have a product line with different variations in terms of deployment, you can make sure that you don't change the functional code. What you are going to change is just the way you deploy the code. You automatically generate the new code, and you don't have to maintain a bunch of source code for each viability factor, for each variability point. So, you just change the model. It is with the same behavior code or behavior model in fact. SCADE, Simulink, whatever you name it, whatever model you are using right now. You just deploy the functional code on top of AADL and assess the new generated code.

A huge benefit as well is— as you said in AADL—you can early on define the component and then refine it later and describe all the implementation aspect of each component. For example, if you have a product line, you can have different implementation of the same component. Then, we can have the library of components and refine the different component.

One good aspect is, you can define the library of components and then have different implementation and choose what components are going to be used for each product line, for each project.

Suzanne: I hadn't even really thought about the use of AADL in the product-line environment, but that is very powerful. One of the challenges in product line deployment and maintenance is managing the variation and making sure you understand whether you've deployed the correct configuration to each individual instance. So, AADL can be the home of the performance parameters, the different configuration settings that are related to each of the different environmental instances. Oh, that is very cool.

Julien: That's really cool. There is something else that people have to keep in mind is, for example, if a new domain, a domain where they start to write a requirement for safety, security, and stuff like this, they are starting to develop some new APIs [application programming

interface]. So, you have to adapt your code. So, if you just code by hand, what are you going to do when the standard changes? You are going to date your source code manually. With a code generator, you just change your code-generation pattern.

Suzanne: Right.

Julien: In other words, you change one pattern, you have the same model, and you can change your code. And, you will be compliant to the new API requirement whatever the new…

Suzanne: Whatever the new standards are that you have to comply with.

Julien: Exactly.

Suzanne: In the safety world, those are fairly volatile. As we learn new ways to fail, we add new things into standards. So, that is an ever-evolving environment.

Suzanne: Exactly. So, after that day, the challenge is that when you rely intensively on the code-generation pattern. In other words, what you generate has to be right. This is a big challenge because, after that, how are you going to prove that what you generated is right?

So, there are different ways to look at it. One way is to certify as a code generator. But, it is going to cost you a lot and especially if you change the code generator it is going to cost a lot.

Suzanne: And, any change to the code generator, you basically have to go through a certification process over again…

Julien: That is correct.

Suzanne: …because that is the basis of your safety assertion.

Julien: That is correct. Another way to look at it is to automatically generate a code with some assertion, pre- and post-condition, to connect some validation tool, also to make use of formal analysis tools, also to generate an automatic test case and stuff like this. In other words, generate the code, but you automatically generate the certification materials.

Suzanne: Nice.

Julien: So, you come with the code, all the proofs, and you can go to the certification authorities.

Suzanne: You are probably still going to have to have somebody evaluate whether those certification materials are valid.

Julien: That is correct.

Suzanne: You are never going to get away from that, but you can at least remove one layer of manual generation of certification elements.

Julien: Yes. What we are trying to do is to remove, as much as possible, all manual activities and try to automate everything, as much as possible, and just not rely on manual labor. By definition, we are human. We make mistakes.

Suzanne: Well, yes. The human-error factor is the thing that, especially in safety and security arenas, you have got both [intentional and unintentional errors]. In safety, it is usually not intentional, but unintentional is bad enough. In security, you can have either intentional or unintentional kinds of faults inserted into manually based codes. So, these are ways of reducing the possibility of that.

Have you worked on any projects where you have actually been able to apply the co-generation and look at some of these analyses that you are talking about?

Julien: Five years ago, when I was young and working at the European Space Agency, I worked on a project called TASTE [The Assert Set of Tools for Engineering] and we used AADL as a source language for the code generators. So, we generated space applications from AADL. It was for European projects. Europe is pretty active.

Suzanne: Europe is very much involved in model-based engineering in general, much more so than the U.S currently is. So, lots of projects to choose from over there.

Julien: So, but it was still a research project. In the meantime, other agencies in other countries abroad have started to work on the topic and use AADL on operational projects, not to generate the whole code but at least to generate the OS configuration.

Suzanne: And, to do analysis like you are talking about. So, you are doing analysis. *What happens if I change this aspect of configuration or that aspect with this known behavior packet?*

Julien: Which is a big advantage of AADL, to generate the code. We have one model, one consistent representation of the system, nothing else. You can't have several specification[s] in different files. So, you can't make a mistake.

You have--a strong semantics always it's the same model just consistent across all the different activities--validation, coordination, verification, and so on. So, coming back as a project right now in other countries that develop tools to generate the OS configuration is why we are now working with Europe and partners to import this functionality of code generation. Something we want to do is demonstrate is the use of code generation for commercial operating system. Specifically, COTS operating systems for the avionics community, like ARINC653. We have different operating systems that are in compliance with this standard.

Suzanne: ARINC653 is a standard related to safety?

Julien: Yes. We want to demonstrate the code generation capability for commercial operating system and show that we do not have so much overhead; that the cost of co-generation is really light and that you can afford it.

Suzanne: That is unusual because a lot of the co-generation systems of the past, one of the complaint was that they had very high overhead. They created bulky, kind of awkward run-time code bases that were difficult to well…The co-generation itself was hard to duplicate.

Julien: Yes, because you have a lot of glue code and all that.

Suzanne: Right.

Julien: Here, the main benefit with AADL is we have a really strong semantics. There is a project in France where they generate code from AADL, and they show that overhead, in time of processing, is less than five percent compared to hand-written code.

Suzanne: Five percent is excellent.

Julien: It is really good Yes.

Suzanne: Yes. It is very good.

Julien: So, in fact we are going to import that into the tool we have at SEI and also demonstrate the process for the commercial operating system [OS].

Suzanne: And these are operating systems that generally would be things that would end up in airplanes or automobiles? They are things that are for embedded systems not just IT systems, correct?

Julien: So, the operating systems we are targeting right now is for ARINC-653, which is an avionics standard. So we're going to generate…

Suzanne: Avionics kinds of applications.

Julien: Exactly. That is the point, because avionics is really focused on safety and they have a lot of standards about safety and security.

Suzanne: That is an area that has really—you mentioned the Airbus earlier—it is one of the domains that is really picking up on model-based engineering as a way to both reduce safety concerns and also to prove that they have met many of the standards that are related to safety.

Julien: That is correct. So, safety is a big concern for this community. We have new communities that start to work on the standard, start to work on their own approach and how to address security and safety issues. They are taking ideas from the avionics community. So, they start to adopt more model-based engineering as well.

Suzanne: So, AADL is a standard. It is a standard language of the Society of Automotive Engineering.

Julien: That is correct.

Suzanne: Automotive is not just wheels on the ground, it is also movement in the air. Avionics is part of that. Is there work being done to do things to update the standard related to code generation? Do you have enough knowledge about how this works yet or is that something that is a little further down the road?

Julien: We have several activities right now. The first one is to improve the ability of the tool to make it more user-friendly with a graphical interface and so on. Have a different views, for example.

Suzanne: So, allowing different visualizations of the model that you are creating?

Julien: That's correct. For example, if you look at an AADL model, we can capture power requirements, also timing requirements or safety requirements. What we want to do sometime is to isolate a view. For example, I am a safety engineer, I want to focus on the safety aspects.

Suzanne: *Don't show me timing, I don't care about performance. Just show me the safety, fail-safe kinds of things.*

Julien: Exactly. In this view, I just want to have the the safety-analysis tool, and no performance view. So, we are working on improving the user experience. Something else we have been working on is to improve the safety-analysis tool and try to be compliant as much as possible with the safety standards.

So, having the safety community that tells us, *OK, What you did guys is really great and can be useful.* So, we have some feedback and improve the tools. So, we need that during the last couple of weeks. I think the next OSATE release will be a good success on that front.

Suzanne: Excellent. So, you are working both on the fronts of experimenting with how you can generate code using some different tools and how you can leverage the code generation capability out of AADL. Then, you are also looking at *How do you update the standard to incorporate these capabilities?* I think you'll be busy for a little while.

Julien: Probably.

Suzanne: Well, I thank you very much Julien for joining us today.

As you know, AADL is one of my favorite technologies at the SEI, and I love seeing the different things that you and Peter [Feiler] and the rest of the team are doing with this. Thank you for joining us today.

For more information about AADL and the work of the standards committee, please visit the AADL Wiki site at www.aadl.info/.

To read the series of blog posts that Julien and Peter Feiler have written on AADL and model-based engineering, please visit blog.sei.cmu.edu and click on 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 email us at info@sei.cmu.edu. Thank you.