AADL Error Library

featuring Peter Feiler and Sam Procter as Interviewed by Eileen Wrubel

---

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. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

Eileen Wrubel: Hello, my name is Eileen Wrubel, and I am the co-lead of the SEI's Lifecycle Innovation and Automation Directorate. With me today are Peter Feiler and Sam Procter who are going to be talking about their work on the AADL Error Library, which provides a taxonomy for classifying errors in a component-based system.

Peter, you have been a frequent guest on our podcast series talking about your work on AADL, which is also known as the Architecture Analysis and Design Language. While Sam and I have worked together in the past, you're a first-time guest on the podcast series. Welcome to both of you, thank you for joining me.

Peter Feiler: Thanks.

Sam Procter: Sure, you're welcome.

Eileen: Before we dig in and talk about the error library, I was hoping both of you could tell us a little bit about yourselves and the work that you are doing here at the SEI.

Sam: My name is Sam Procter. I recently finished my PhD at Kansas State University in the summer of 2016 where I studied safety-critical software engineering. That work continues here at the SEI. I have, like everybody on this team, an architectural focus. When working with AADL, we look at how systems are architected and then the impacts that has on things like their safety.

Peter Feiler: I am Peter Feiler. I have been at the SEI since almost the beginning, since 1985 so 34 years actually this month. I am an SEI Fellow, and I have been working with the AADL standard for…actually in September, it will be 20 years. We started the committee in September of 1999. Good track record, this thing is still going strong.

Eileen: Indeed. So, can you give us a brief overview of AADL, and then we'll dig into the error library?

Peter: AADL is a notation that lets you model embedded software systems. The idea behind it is to be able to get at issues that we have in these systems because what we've experienced is that, for example in the aircraft industry, 70 percent of all the software-related problems are found post unit-test. As a result of that, the cost of developing systems is getting bigger and bigger, and we see in this many examples whether it's in the auto industry, medical industry, or in the aircraft industry. It's the software that drives things and the software that messes up things.

The standard came out in 2004 originally. Since it is for embedded software systems, we are dealing with time-sensitive systems and safety-critical systems. It is in the context of being a safety-critical system is where we then provide capabilities for dealing with fault modeling and the idea of an error library with a taxonomy.

Eileen: Can you tell us why you felt that this ontology was necessary? Why is it so hard to classify things in a component-based system?

Peter: The first part is the ability to associate fault behavior with a system, so that you can then analyze that automatically by tools rather than by hand. The original work for that was actually done by the same person, Steve Vestal, on MetaH, which is also the basis for the AADL core language. He had added that capability to MetaH in 1989 actually. So, we had that as a first version in the standard and people use it. The reason that is useful is because if you have a model, we can then automatically generate fault trees, for example, and do analysis or do a failure-mode-and-effects analysis—something that in the industry for the longest time has been done manually by filling out spreadsheets.

Often what happens is that those things are only done once. Even Boeing Commercial does it only once every four years—what they call a system safety assessment report. To just illustrate the point with the 737 Max, they had this thing where you as a customer had the option of buying the aircraft working with two sensors versus one sensor, but there were no two safety system assessments as to what the higher risk is if you operate only with one sensor versus two because first of all, they do it early in the process and secondly, it's expensive to produce these things.

With AADL, we push a button and you get a report out what it looks like when you fly with one sensor versus two sensors. That is the value of just being able to do error modeling. Based on the experience of doing error modeling—and there were various research groups that did work with it—they gave us feedback on improving it. In that context, I ran across some research that was done in the 90s where they then looked at characterizing the effects of a system. The kind of effect that you can have, the types of effect is much smaller than the kind of ways something can

fail. That is, for example, if you have say a heater, it can break in a number of different ways whether it is the power supply breaks or the heating rod or whatever it is. But the effect to you is no heat, service omission, or heat when you don't want it. The thing doesn't turn off or too much heat. So, there is only a small number of effects. That is what the taxonomy does, is we standardized what are the effects of…So you still deal with, when you characterize the system your internal faults remain specific, and you deal with whatever it is, but the effects are always the same.

What I found is that there were several groups who had made an attempt at capturing those effects. I then took work basically from three different groups and put it into a common taxonomy that then we made part of the standard. It becomes effectively like a checklist to say, *Have you thought about more than the thing failing*? *Have you thought about it not providing a service or providing a service that you don't have in terms of the rate at which you're getting the data rather than just that it's early or late*? That made it a richer set of things that allows us to then systematically address problems in real systems. We have had a number of cases where we looked at real industry examples and were able to then identify situations that they hadn't thought about because of the taxonomy, because they simply hadn't thought about certain aspects of a system.

Eileen: Sam, can you take us through an example of classifying an error in a component-based system? How would I use the AADL Error Library?

Sam: Yes, absolutely. Like Peter said, you don't want to look at causes; rather you want to look at effects. So, if you have a system with a heater, rather than asking yourself, *Well this heater could break. How could it break? What could make it break?* What you want to say is, *I need heat, so how could that heat be provided wrong*. Well, you could have too much heat, it could be an error of commission. You could have no heat, it can be an error of omission. Or, if it's a digital thing, it could have bad timing or bad value. That is sort of the starting point; you consider that if you are a consumer, whether as a component or another system, if you are consuming the output from something, you think about the ways that that could go wrong. *Any particular failure, what could those effects be?*

Other work in this area stops here and is really pretty abstract. Either input is provided when it shouldn't be or isn't provided when it should be, but the error library actually goes deeper. What we have in the ontology are actually families of errors that take these error types as sort of their root and then expand it from there. Instead of just saying, *Well, you might have an error with the value of a message or the value of an input,* we say, *Well it could be high or low*. It could be high or low within the expected range or it could be entirely out of range. Like if you say that you want a temperature and that temperature reading comes in and it is like 300 degrees when you know it's never going to be hotter than 100 degrees, then you know that that is an error and that

your system may behave differently, given that then some incorrectly high but not entirely out-of-range error. Using these as the root of these families of error types, we then expand them down and refine them into much more specialized error types. Then you can refine them further using your domain knowledge as a system expert to make them specific to your system.

Peter: The basics are when you look at some individual service, given that we are in a software system, you are getting a stream of messages. Now you can talk about sequence behavior, and that means, for example in terms of timing, it now becomes a rate issue: *I'm supposed to send you stuff every 10 seconds, but I only send it to you every 15 seconds, for example, those kinds of mismatches.*

The practical part of it is there is one example of an engine-control system for an aircraft engine that uses what is called a stepper motor. It is a very simple controller that is just one step at a time to open the fuel valve or close it, and it doesn't get immediate feedback, so it doesn't know when it's out of position. We worked with a customer and they said, *We have a problem because the software drives it, and the position is different from what we think it is* because something goes wrong with it somewhere in the software. The first part is, we were able to identify by using a combination of the taxonomy, and then other analysis we have in the AADL latency type of analysis, to point out the fact that every so often the timing of the latency is large enough that it misses sending out a step command, and as a result of that, it's off. As part of the analysis, we then looked at other aspects of it too. So, I looked at rate behavior, and what I found was that if I send it to them at a slightly slower rate—it didn't have to be very much—it incrementally creeps up on missing a step. What they were doing is they were counting on when opening the valve, what's the maximum time it takes to open it, and then the system resets itself. That's why they most of the time didn't see the error and the same with closing. But, sometimes when you do control, you have oscillating behavior. *You open it. You are not quite fully open, and then you're being told to close it. So, if you now continuously oscillate, then it accumulates more.* They would have had additional situations where they were missing steps.

The taxonomy helps you deal with corner cases that you normally wouldn't think about by simply saying, *First of all, there is this class of error.* Then you need to look in detail, *What does this mean?* and then make sure that it doesn't occur. We have that in terms of timing, we have that in terms of value errors. Here is an interesting one too, and that is when I am a controller and what you typically do as a controller, you are giving the controller set points, like, say, *Go this far*, and then once it's reached that, you say, *Go to the next step*. What you really have is, *If I give you a set point that's too far away and they send you new set points every second and you can't get to it, then we are out of sync again.*

So, what we are dealing with here now is not the value itself but the difference of the value cannot exceed a certain maximum. That becomes an error class within the class of what we call

value errors. That is what this taxonomy gives you in each of those categories. First of all take into account the fact that we are dealing with a series in addition to individual values. Secondly, because of that sequencing in terms of time, what are characteristics that are relevant in terms of value?

Eileen: So you are able to capture those unusual cases that are a function of the combination of those kinds of mismatches.

Peter: Exactly. The interesting thing is that all of the classes that we have are based on the real examples. There was one scenario in a DARPA-funded research program where they had autonomous vehicles driving around. There was one situation where a car was driving up to an intersection and stopped there for 10 seconds before it proceeded. At first, they couldn't figure out why. It turned out to be the case that when the car got started up, and it initialized its internal clock, the timer, it had read from the wrong time server. So, its internal clock was 10 seconds off. What had happened is it was getting sensor information of things that happened 10 seconds ago. So, it thought there was a car crossing, and the car had crossed 10 seconds earlier.

Eileen: Kind of like when we get a broadcast delay.

Peter: Exactly, yes. There's a funny story on that one. They were broadcasting Steelers games, and there was a radio broadcaster, Myron Cope, who was very popular. So what people did is they turned the sound off on the TV but had it on the radio. Well, the TV companies were trying to get more commercial time out of it, so they simply dropped one or two frames per second, which sped up the game on the picture, but the sound started being late. That is how people found out that they were cheating. So, this difference in latency is not just in serious systems that affect your health, but it is also in situations like you watching TV and listening to radio at the same time.

Eileen: That's a pretty good story.

Peter: It is one of our classes of errors that we have in our taxonomy.

Eileen: That is a really good example for bringing that, almost some of the abstraction of that concept down to where people who aren't familiar with this can understand the types of...

Peter: Exactly, that's part of what we are trying to do is to say, *Yes, it has its roots in these embedded systems, but any kind of system that is time sensitive, we have interesting error classes*. Whether you are dealing with video, like I was just saying, people dropping frames and frame rates and stuff like that or synchronizing sound with video, like this guy is doing when he is afterwards editing our interview. He faces all these same problems.

Eileen: One of the things we really emphasize on these podcasts is transition. If I'm a software architect or software engineer or program manager, how do I get my hands on the error library, and what kinds of resources are there out there for me using it?

Sam: Yes, absolutely. There are a bunch of answers to that question, and it depends on exactly who you are and what you are looking for. If you are listening to this and you just think, *This is interesting, I want some more information, I want some concrete examples about what is in these error families*, we have a blog post that summarizes a lot of this pretty nicely. It has some examples in there, some figures, stuff like that. If you really want to tear into it or you are coming at this from more of an academic perspective, we have the paper itself that was published, that is linked to in the blog post as well. If you are going to implement this sort of thing, you need to know very precisely what the definitions of all these error types are, there is the standard itself. This is an international standard, so you can have some confidence that it is not going to change. When we make updates to it, those will be published in a future standard, but we are not going to change what's out there right now.

I also really want to mention that if you are a developer, and you don't necessarily care about all of the description stuff, you just want to use this as you are modeling a system, as you are building your system, AADL and the tool environment that we built to support it here at the SEI called OSATE are also publicly available.

What is cool about using the error library in the context of OSATE is that it allows these error types to be specialized or to be operationalized by incorporating that domain knowledge. If you are working, for example, on a TV broadcast system like we were just discussing, these abstract timing errors and rate errors and things like rate jitter may not mean that much to you. You can extend, you can refine these error types to have that knowledge in. So, like, *Ah, if we are dropping frames, that is going to affect the video and audio sync between the TV and the radio*, for example.

You can put your domain knowledge together with the error knowledge and come up with system-specific errors that are going to be meaningful to everybody who looks at your system whether or not they have read this paper or whether or not they are experts on the ontology. You can look at it and say, *Oh I know what this means* because you can bake all that information together using the tooling that we're developing here at the Software Engineering Institute.

Peter: As part of that, just to add a few more pointers to things, we have example system models that are annotated with fault information and have a tutorial. The online help for example gives you some user guidance on how you can systematically go through early on and focus on certain pieces of information that we add on.

The other part that comes in handy too is, you are already creating an AADL model of your system. Instead of you manually having to figure out, say a fault tree or things like that and creating it with some form of tool and putting it in, all you do is annotate your existing library of devices. In some cases, you may already have them around. This fault information attached so all you do is use them. From the usage, the way different parts are combined, our tools then figure out what the propagation paths are for errors, so we can then automatically produce fault-impact analysis reports or fault-tree analyses. If you then put fault probabilities in there, now you get failure rates and all kinds of good stuff out of it. It is all at the push of a button.

Eileen: This is a really rich body of resources and tools for everybody at every level of abstraction or technical depth.

Peter: Exactly, and there are also some tech reports. When we came out with the original standard. There was actually some young woman, a PhD student, who was spending half a year in Pittsburgh. She and I together wrote the first guide on how to do fault modeling with the annex. When we came out with the second version of the standard, we did an updated version that then goes into more depth on the annex as well. So, in addition to some of the papers and the other stuff, there are those things out there as well.

Eileen: OK, great and we will provide links to all of that in the transcript for the podcast, so they will have access as well. So, what's next for your work in this area?

Sam: Like you just mentioned, there is a ton that is baked into this and that is actually by design. One of the things that we say about AADL is that we want your AADL system model to really be the single source of truth. Right now, if you are developing systems without AADL, a lot of times you have maybe a requirements document that is written in Word. You have some diagrams maybe up on your whiteboard or in some other modeling language. You have to as a developer bring all that together when you actually build the system, but with AADL you can bake all that information in. This becomes more valuable then as you can put more types of information in.

Like I mentioned, my background is in safety, and like Peter has mentioned, AADL has been used for safety-critical system development for a long time. Our team is currently looking at how safety and security can be integrated. So, these security concerns, baking those into the error library, is something that we are working on right now. This has been actually pretty promising. Because we take this effects-focused approach rather than causes. Of course there are a million things that a malicious user can do with access to your system. The end results are going to be the same. You are not to get the input from that sensor or the actuator isn't going to respond to the commands the way that it should. That has actually been really promising.

One aspect of security that the error library doesn't cover right now, though, is privacy. In safety there is no concept of, *This person was able to view this message, but they shouldn't have been able to*. Well of course that is a core security concept, so we are looking at extending the error library to include these notions of access that shouldn't have been granted or issues of privilege and access control. That is where we are going right now is baking security in and then looking at the interactions much the same way we have looked at interactions between value errors and timing errors—between these other types of errors and then security issues as well.

Eileen: Peter, do you have any other upcoming projects that you'd like talk about related to this?

Peter: No. I always have something to do, unless you want to hear about my granddaughters.

Eileen: Those are very special projects.

Peter: I spend a lot of time with them.

Eileen: Thank you both so much for joining me today. Sam and Peter have also published a blog post on this topic as Sam mentioned earlier. That is available at insights.sei.cmu.edu. You can search for Procter, P-R-O-C-T-E-R, in the author field, or search for Architecture Analysis and Design Language in the subject area. All the resources that we mentioned today will be included in the transcript to this podcast. Thank you so much for joining us.

Thanks for joining us. This episode is available where you download podcasts, including SoundCloud, Stitcher, TuneIn radio, Google podcasts, and Apple podcasts. It is also available on the SEI website at sei.cmu.edu/podcasts and the SEI's YouTube channel. This copyrighted work is made available through the Software Engineering Institute, a federally funded research and development center sponsored by the U.S. Department of Defense. For more information about the SEI and this work please visit www.sei.cmu.edu. As always if you have any questions, please don't hesitate to email us at info@sei.cmu.edu. Thank you.