A Model-Based Tool for Designing Safety-Critical Systems

featuring Sam Procter and Lutz Wrage as Interviewed by 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. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

Suzanne Miller: Welcome to the SEI Podcast Series. My name is Suzanne Miller, and I am a principal researcher in the SEI Software Solutions Division. Today I am joined by Dr. Sam Procter and Lutz Wrage, two of our researchers also in the SEI Software Solutions Division. Today we are here to talk about their work on a model-based tool to assist in, and most excitedly, partially automate the design of safety-critical systems.

Welcome to you both.

Sam Procter: Thanks.

Lutz Wrage: Thank you.

Suzanne: Let's start by telling our audience about you, what brought you to the SEI, and the work that you do here, especially since I know, Lutz, this is your first one. Lutz, why don't you start us?

Lutz: Yes, what do I do at the SEI? I mostly work with model-based engineering and analyzing architectures, software architectures, system architectures. My interest in that started when I, well, shortly after I came to the SEI, that was almost 20 years ago, and I found myself working in an office next to the office of Peter Feiler. I talked to him and it turned out that he was the technical editor of the AADL [Architecture Analysis and Design Language] standard and one of the driving forces behind it. AADL is a language to describe software and system architectures for embedded and real-time systems. Yes, I got interested.

Together, we started developing an open-source tool called OSATE [Open Source AADL Tool Environment] to build and analyze AADL models. One of the great features of AADL is that it was developed with a focus on creating models that you can actually analyze. Not just build a model and create a nice diagram, put it on a PowerPoint and show it and talk about it. I mean, this is nice for architectures, but in AADL, it's a bit more formal. You have certain components. It is not just boxes and lines, so it is certain components that have certain semantics. You can attach properties to them, and then, you can actually analyze them and can write automated analyses. That has been a big part of my work over the years to develop these analyses, improve OSATE. For the last couple of years, I have led all the development activities around OSATE, with a goal, of course, of making it easier to use, having a broader audience look at it. It is, I would say, fairly stable now, and it is not easy to crash it.

Suzanne: That is always good.

Lutz: It works pretty well.

Suzanne: All right, thanks, Lutz. Sam, why don't you give us a little bit about yourself and what brought you here and what you are doing here?

Sam: Sure. I have always been fascinated by the way software and software tooling can support and reinforce process improvements. I had a friend in college who worked on an improvement to Excel that would highlight a corner of a cell if the formula in the cell was different than the formulas around it. I thought that was fascinating, this idea of a really gentle nudge from a well-designed program that could correct or prevent an error with very minimal user involvement. The user doesn't have to understand anything deeply. They just get a notification that, *Hey, maybe there is something that you need to look into here*. I like this idea of things getting better invisibly over time.

I have always been interested in computers. So the blend of those two is software engineering, the way to build software better. So that is what I studied in graduate school, and while there I was exposed to a lot of pretty formal model-based engineering, including that modeling language that Lutz mentioned, the Architecture Analysis and Design Language, or AADL. AADL was developed by the SEI, and I came here after graduation to continue the work that I started as a student. Now, here at the SEI, I wear a couple of hats. One is I am the lead of the model-based engineering group. I look at how to apply model-based technology to solve DoD and industry challenges, particularly those with safety- and security-critical systems. Then the other hat that I wear is of a researcher, a system architecture researcher. Mostly, I am still doing this sort of hands-on research that drew me to the field in the first place. Most of my work is on safety, but some projects, like this one, are a little bit further afield and a little bit more experimental, and that really keeps it interesting and keeps it exciting.

Suzanne: Lutz knows, I don't know if you know, Sam. I have been following AADL pretty much since its inception as a language. It has some unique aspects, that analyzability and ability to ingest from multiple other modeling approaches is one of the things that is really, really useful in AADL. Now you guys are taking it even farther.

But before we get into what you have done, let's talk about the current state of the art in designing safety-critical systems, because I think it is important that people understand what are the challenges and pressure points for system and software architects that are designing these very complex real-time systems?

Sam: Sure, yes. There were several challenges, but they largely stem from increasing system complexity. Of course, that complexity isn't all bad. Modern systems are pretty undeniably more capable than their predecessors, but that capability comes as a result of more powerful hardware, more sophisticated software, more sensitive and just more types of sensors, things like that. Not only does this make building these systems more complex, but it also makes them harder to certify as well. Because, of course, most safety-critical systems have to be certified for their use before being deployed. You can't just compile and ship code the way you might for a phone app if you are building something like a medical device.

So several techniques have been developed to manage this complexity. A couple notable ones are model-based systems engineering [MBSE] and design-space exploration. Model-based systems engineering, which you will sometimes hear referred to by its acronym MBSE, is a way in which designers can build simple models of their system and then analyze those models to learn something about the system before building the whole airplane, medical device, what have you. You can think of this like building those little architectural models of a building that you sometimes see on display when there is new development proposed or kept and then put in a museum or something. You can learn more about what the full building looks like. You can show it to people and say, *Do you like the columns on this building? Do you want this wing over here painted a different color?* And you can do all of that much more quickly and much more cheaply than actually building the building and then, once it is completely ready to go, asking people if they like it or not. Model-based systems engineering is similar. We build small models of systems. We analyze them, and then if we like it, we build the real system. If we don't, we tweak the model and move on.

Design-space exploration is a different way of tackling complexity, but the idea here is that we input a bunch of information about the components that we can use to build a system or that we might build, and then build our system from these components, and then tell the computer what we are looking for and let the computer search through, potentially, millions or billions of combinations of these components to see what might be a good fit. If we return to our load bearing, or our building example, we might input things like the price and weight-bearing

information of various types of structural components that we could use to build walls of the building. Then we can tell the computer, *Find us the cheapest option,* or *Find us the strongest option, or find us the cheapest option that will guarantee some minimum amount of strength.* Computers are really good at plugging all of these elements together, seeing what satisfies the user's criteria and what doesn't, and then presenting those options to the system designer who can make better-informed choices as a result of all the searching.

Suzanne: Right. The thing that I immediately thought of when I read in the blog post about design by shopping is Amazon and all the really powerful search engines that *I want the blue one of these,* or *I want a red one of these,* or *I want one of these that does this, this, and this*. This is really applying a lot of that same sophistication in algorithms that are now available to do this and other language extensions and things that you have done under the covers to allow us to apply this to that very challenging space where we don't…The constraints…It is not like shopping at Amazon, right? You have got a lot more constraints that you have got to fit into to get your design to work. Let's talk about that. This has been detailed in a recent paper and blog post. Those will be available to everyone in our transcript, but you have prototyped a language extension for AADL. Then you have brought in a different set of tools to work within OSATE. Now we have Guided Architecture Trade Space Explorer, otherwise known as GATSE. It partially automates MBSE so that our engineers can rapidly explore different design options within often very constrained design spaces. Tell me all about it because I am very excited about this.

Lutz: Let me first talk a bit about the tool and AADL in a bit more detail to set the context a bit. We have this OSATE tool to work with the AADL models. That tool itself is based on Eclipse. That is actually something that software developers will likely be familiar with. It gives you the regular IDE [integrated development environment] experience. Of course, we don't do a programming language, but we use it to edit AADL code.

AADL is a bit different than other modeling formalisms in that it supports both textual syntax. In that regard, it's a bit similar to a programming language. Also, you can use a diagram editor to create AADL models. So that is a bit more like UML [Unified Modeling Language] diagrams or SysML [Systems Modeling Language] or whatnot. AADL, as we mentioned before, is good at having all the information to automatically analyze models, and it has its roots…Well, originally it was actually called not Architecture Analysis and Design Language, but I think Avionics Architecture Design Language. So its roots are in avionics systems. But we expanded that out into general modeling of embedded and real-time systems because many things in avionics systems are just applicable across the board. This language comes with built-in notions of processors and threads and the properties that support analysis of schedulability, for example like how often does something execute, how long does it take, and things like that.

In general, we have modeling of the runtime architecture of the software system. There are the threads, and there is the communication between threads, then the execution platform, processors and busses that connect them, and devices. We put in information about the deployment, which software executes on which processor, and we use device components to model various real-world components that we treat as black boxes and just model their properties, so sensors, actuators, maybe also mechanical components. In our paper, we use an example, a wheel-braking system for an airplane. Of course, there you have mechanical components and hydraulics and a computer system to go with it. Yes, that is the part about AADL. Now Sam can talk a bit about the other pieces that we have. What does it stand for, ARL?

Sam: We started with AADL. We had this really strong, model-based tool that we wanted to build on, and we wanted to explore the combination of MBSE with design-space exploration. Looking at tools that support design-space exploration, we found a tool developed at Penn State called the ARL Trade Space Visualizer or ATSV, which is software that supports the exploration of what they call a *trade space* but is equivalent to what we have been calling the design space, which is the full set of possible systems that could be built out of components that either already exist or might be custom created as part of the project.

ATSV was really intended for more physical systems, things like airplane wings. They have a study where they build a satellite antenna with it, rather than the software-based critical systems that we use AADL and OSATE on. It is also designed to take in like a big spreadsheet of all your components and then sort of compute things based on that. But it had the ability to connect to external programs that would describe the set of components and their categories and all their information. What we did was, one of our first tasks was to connect ATSV to OSATE through this programmatic interface, so that they could speak the same language. What we wanted to do, and what we ended up building the system to do, is to make it so that ATSV could select a number of configuration options for a system. Say, *We have this AADL model with a bunch of holes in it.* We call it a skeleton model, *and the skeleton model, we are going to fill in with these pieces. We are going to use this component in our wheel-braking system for the tire. We are going to use that component for the hydraulics*, etc., etc.

ATSV selects all these configuration options. OSATE then takes these choices, assembles the model—it starts with the skeleton and fills in the choices from ATSV—it builds that model, what we call *instantiating the model* in OSATE, and then runs whatever analysis or analyses the user has said that they care about. These might be things like the weight of the built system or its performance according to some metric that the user has specified. Then OSATE will report all of these results back to ATSV. So, it will say, *We built the system,* or *we couldn't build the system, and here is why.* If it could build the system, *We ran some analyses on it. Here are the results.* Like, *Here is the weight of the system. Here is the price.* In our wheel-braking system, we have a

sort of synthetic analysis called braking power. So it might say, *Here is the braking power of this particular candidate wheel-brake system*.

From there, ATSV will use one of a number of algorithms that the user can specify to select a different set of choices. Some of these algorithms are not very sophisticated. They are just sort of a random walk through the entire design space, but some are very sophisticated and are evolutionary algorithms that can infer the relationships between inputs and outputs and actually really hone in on a particular sweet spot that the user has specified. Like, *I want this to be cheap but effective*, or *I want this to be lightweight but very powerful*. You can set these goals in ATSV, and it will use the algorithm that you select to really hone in on that.

Identifying, creating, analyzing, and then reporting back on a given model architecture sounds like a lot of work, and it absolutely is, but it takes less than a second on my laptop. We are talking about a pretty performant piece of software, and we are really pretty proud of that. One of the challenges that you might think about, as you are hearing all of this capability, is how exactly to specify the skeleton model and then the parts of the model that are changeable. For this we had to use one of, I think, researchers' favorite tools, at least in computer science, which is a new domain-specific language. Lutz is the main designer of that. I will let you talk about it, Lutz.

Lutz: We developed this configuration language that essentially lets you specify which place in the skeleton architecture you have that is a hole where something needs to be filled in, and also, the options that you have to actually fill it in. So, for each of these variable elements, we define a set of possible candidates that we can stick in there. Also, it is not just that we have candidates, we also have property values that we can specify there where ATSV can choose among those values and put them in. So this ended up a bit more powerful than the minimum-necessary stuff needed for the GATSE project, because we used this also as a testbed or as a prototype for some capability that might end up in the next version of the AADL language.

Suzanne: Yes, it is a language that evolves. It is not a static language.

Lutz: Yes, it is currently in Version 2 [The language implementation is part of the overall GATSE implementation at https://github.com/osate/osate2-gtse -- specifically, all the packages that have the prefix "org.osate.gtse.config" are language-related.] We just recently made an update to Version 2.3, and well, at some point, it may be a Version 3 that includes a mechanism like we have for this configuration piece that we prototyped for GATSE.

Suzanne: I am seeing all kinds of applications for this from supply-chain management, *I have got these five suppliers, and if one of them goes out, what is the effect on the model? I can model what the effect is instead of bringing examples of all five components into a lab if I can specify the attributes*. Certainly, *Where are the places where we have safety-critical, where we have to*

make a choice. Don't worry so much about the rest, but here's the safety-critical pieces or the security-critical piece. I mean, I am just like, *ahh*, all over the place. How do you envision GATSE helping systems engineers, software architects, certifiers of…What is your ideal use case for people coming in to use GATSE?

Sam: I like the way you phrased that question, because how does it help different populations, different users, I think is the really key part. Because there is a school of thought in design-space exploration that is actually just called optimization where you say, *Here is the goal of the system that I want. Computer, find me the best combination of components, and we are going to build that system*. What people have found is that specifying, a priori, the final system's characteristics can be really tricky, because you don't necessarily know how all the inputs correlate to the outputs.

The basic idea that we had for this project is to incorporate model-based systems engineering and this automated design-space exploration in a way to generate, evaluate, and then present to a designer a number of candidate architectures. Rather than say, *We are going to find the best architecture*, what we say instead is, *We are going to present a number of options to the designer*. From that, the designer is going to be able to learn things about the design space, about the world in which they are building their system. This realizes a paradigm system design that I think you mentioned earlier called *design by shopping*. This term was coined by a professor at Brigham Young named Richard Balling who talks about the importance of presenting options to designers and letting the designers learn from the options that are presented. Ideally, it's not just a random enumeration of possibilities, but you really want to identify and lock in on these relationships between different system-design choices and the characteristics of the built system.

That is sort of abstract. Let's return to that example that you mentioned earlier about buying something on Amazon or just buying like a new shirt, for example. It's not very helpful if you want to buy a new shirt to get the list of every shirt that is available for sale on Earth, even if this would somehow be possible. Instead, you probably go into your search with some characteristics in mind. You want a shirt in a particular size, maybe a certain color. You probably have a maximum price that you are willing to spend, things like that. You might start looking at shirts in a store or online, but as you do this search, very quickly, you are going to learn that there are some relationships between these. Maybe all the shirts from a particular brand are very expensive or all of the shirts made out of some fabric are too warm or too cool for the time and place that you want to get the shirt for. What you are learning here is the relationship between the price and the brand or the fabric, and when the shirt is wearable. As more choices become enumerated, these relationships are what you are really learning. You see the individual shirts, you understand that they are individually available for sale, but what you are really learning is the relationships between the inputs and the outputs.

This is the essence of the design-by-shopping approach that GATSE aims to bring to the world of critical system design or critical system evaluation by certification authorities like you were talking about. A system designer can specify characteristics that they care about. These are automated analyses that run in the OSATE workbench, so things like weight calculations or power consumption or price or what have you. Then given this library of candidate components, restrictions on how they interact in case there is software incompatibilities or maybe an overall power cost or weight budget, the technologies that we have talked about, ATSV and OSATE, moderated by this project, GATSE, will work together to come up with candidate architectures. Then system designers can look at these candidates and discover relationships that may not have been initially obvious, that were not knowable a priori. Of course, they can then refine their search and eventually maybe see some candidates that are particularly promising, and then these can be modeled and explored further using more traditional system-development methodologies.

Suzanne: One of the things I'm getting from this is, which I really like, is that we're not asking the computer to do the job. We're asking the computer to automate so that we can make better decisions by getting better information that is tuned to the problem, and I know, in the systems I work in, this is particularly important because of what you said. There are relationships that are not at all obvious when you start getting these complex systems of systems, and algorithmically, I would not expect to be able to get to an optimal solution. Any optimal solution is probably going to engender some cascading set of effects that I never intended. So I love that idea that it's giving me this candidate set to choose from, and I'm wondering, and we haven't talked about this before, but is there a thought of building a community who contributes analyses of different kinds of safety-related components that people can look at in terms of not just building their own configurations but, *Hey, here's what this jet manufacturer put together for wheel-brake systems, and as of 2022, here were the choices that they had, and here's some of the things they learned from doing these*. Is that part of the vision for this or am I too far down the road?

Sam: I don't want to say that's not part of the vision. That is farther down the road than where we looked in this project, but you're right that once you have these characteristics that you care about, once you have ways of calculating them, there are a number of really, really interesting and cool options for where this can go. Exploring libraries of components is one of those. Putting some of the characteristics that you care about in things like acquisition documents is another potential application of this, that we can start to maybe lift some of the fancy-sounding term, *desiderata*, some of the desired characteristics of the system that we want. If we can formalize those, if we can lift those out of natural language, whether that is in requirements documents, in acquisition specifications, things like that, and move them into things like AADL or the configuration language that we extended it with in this project, that's really a similar arc to the story that we tell with AADL more generally. That if you are able to move a requirement out of a natural-language document and put it into a model of a system, that that has all sorts of benefits,

because there is reduced ambiguity, there is enhanced specificity, things like that. I think what you're identifying there is really an application of the GATSE approach or the design-by-shopping approach, to things that are not modelable in AADL. That is part of the work of the model-based engineering team here at the SEI more generally is formalizing aspects of the system acquisition and development process.

Suzanne: Have you talked about to any of the safety-certification community about how they might use this, because I see where there could be certain certification standards that have to be applied all the time. I will go back to the wheel brakes. Everybody knows what those are. If the certifiers can actually do a little more formalization of what the real boundaries are of these characteristics, where in natural language it may not be as clear, then I can see us getting better designs because I have a better idea of what those boundaries are. Have you had interest from them in looking at this?

Sam: There are actually two versions of the paper that came out, and it is good that you mentioned safety, because this is one of the key differences between them. One of the things that we were able to fit into the journal paper that we did not, in the initial, the earlier conference paper, was an application of GATSE to the safety domain.

One of the things that's tricky about safety, though, is that GATSE requires quantifiable analyses, because it's running these analyses in an automated fashion. It's getting back either a string or a number, whether it's integer or floating-point, and then if it is a number, it is displaying it graphically in one of the many charts that are available through the ATSV software. Safety is a little bit resistant to quantification. There are some traditional safety analyses coming out of the 50s and 60s, things like fault-tree analysis or failure mode and effects analysis [FMEA]. These are well studied and well understood, but there is the increasing recognition in some places that they are less applicable to systems with large software elements.

I say that because software does not fail in the same way as the more hardware-based systems that these traditional safety analyses were designed to use. In the journal paper, we use an example of a cool sort of Bayesian-based safety analysis that has an interesting side effect of being much more computationally intensive than most analyses. We talk about using GATSE in sort of a two-stage process, where you run a lot of cheap analyses early, you identify better candidates, and you focus in on those candidates and run more expensive analyses. That more expensive analysis is the safety analysis. So, we haven't had a ton of engagement with the safety community but just because of, certainly, the background of the group and my own research interests, it is something that we're interested in and that we've looked at a little bit in the journal version of the publication on GATSE.

Suzanne: The thing I know about that community is they just become more and more overwhelmed because as more and more software is the critical functionality, they can't rely on some of the simpler hardware-based safety kinds of ways of looking at the world, FMEA, etc., and there's just more and more and more. Helping them to automate their certification processes, I think, would be something that would make a lot of people sleep better at night. I do want to actually let people go through an example. Can you take us through an example application of GATSE and just generally how it would work?

Lutz: Yes, let me get back to the wheel-braking system as our standard example that we talk about a lot. In the system, we have some mechanical components. There is a wheel and brake assembly, the pilot's brake pedals, for example. Then there is some hydraulic system involved. There are a couple of redundant pumps and various control valves and some pressure vessel and whatnot. Then finally, we have some computer system that controls the whole thing. So, the BSCU, brake system control unit, where, for example, we could have two federated CPUs running monitor software, and we also need a CPU power supply for that.

So now, let's keep this simple, because otherwise, I will lose track while talking about it, and nobody can understand what I am talking about. For simplicity, let's assume we have all components selected, except just some BSCU components and the power supply for the computer. Then next, let's assume then we have all this and we create an AADL model, put all this information in that we have so far, but we use placeholders in the model for the CPU, and the monitor software, and the power supply. These placeholder components have just enough features to describe the external interfaces, so that we can say, *OK, how are they connected?* For example, we have just some placeholder for the power supply that has some wire to the CPU or to the board where the CPU sits on and things like that, but we don't say what exactly these things are. Also, we don't know what the exact properties are, such as the weight or the cost, because that depends on which actual component we then select in the end.

In AADL we have a way to do this that is called *component types* that just describes the outside of a component but not the inside. In the next step, then, we define which specific components we have available to replace or to insert into the holes for the component types. In AADL, these things are called *component implementations*, and we add specific properties to them. So we would, for example, say, *Well, which CPUs do we have? Maybe we have ARM Intel processors and MIPS CPUs. We have various power supplies that we could use. We have software pieces from older, previous projects that we could reuse.* So, we create modeling, of course, AADL model elements for all these things, so that we can then insert them into the overall model.

Again, of course, not all the implementations are a good fit. Based on what we know, which components we have, we can already pre-specify some choices. For example, if we know that an Intel CPU doesn't work here, we will not try this. We won't even try it out, but at least the

known not-to-be-impossible choices, we put them in this configuration file, in the configuration language, and then specify these as lists of candidate components that can be filled in.

In addition to that, we also need to write down any constraints that we have. That is also part of the configuration language. So, for example, one requirement might be that we cannot mix two different kinds of CPU. We want two of the same. We just want to have two of them to have redundant execution to increase the reliability of our control unit. Also, we have requirements on the power supply. It must match the power consumption of the CPUs, obviously.

So, all this, we write that down into the configuration file, and also, which analysis we want to run. We talked about that. For example, we will run the weight analysis. We want to run a power analysis. We want to maybe also run some schedulability analysis to see that the system reacts fast enough and samples fast enough and things like that. We also, as the final component in the configuration, we write down what optimization criteria, if any, we want to have. For example, that the cost should be minimized and also maybe constraints on the costs that we have and things like that.

So, based on all this, we then use ATSV to start the process so that, as Sam described, the ATSV then generates architectures based on the choices that we have in the configuration, or it selects choices. OSATE instantiates this as an architecture, analyzes, and reports back the results to ATSV, and it runs them through a bunch of combinations, and the main thing is to visually inspect the output first to see what results we get. Of course, this is sometimes a bit tricky, because if you have multiple dimensions, you can only put so many into one diagram. You cannot put 20 different dimensions in a diagram. Three or four is about the maximum that you can get. So, that requires then some smarts of the user to evaluate what do these results actually mean, and based on that, select maybe one of the choices that was presented by ATSV or add new components, or add more information into the model to refine the results of the analysis or maybe take some choices away and add more analysis and things like that.

Suzanne: I am thinking about the aspect of, we come up with no viable choices. That is really important information. When we are early in the systems engineering, to be able to say what have designed is not feasible, and we have got to go reassess other elements besides just this component, I find that to be very powerful. It is the kind of thing, when you tell somebody that a design is infeasible, and they want to say, *Well, where is your proof*? I don't know if I call this proof, but it certainly is strong evidence that I am telling you the right thing, because if I try and run these kinds of analyses against multiple configurations, I get no answer that is viable. So that alone, I think some of the systems engineers in our audience are going to be going, *Oh, I want this.*

Let's talk about people that want GATSE. To be fair, I know that AADL has a bit of a steep learning curve, so there's some investment in learning, but what other kinds of challenges can users expect in using this tool so they go in with their eyes open, and how can they mitigate some of those?

Lutz: Yes, I mean, of course, the first thing is this was a research project, so…

Suzanne: It's not a commercial tool, right?

Lutz: Right. It's definitely not a commercial-quality tool. Was anybody else involved? It was essentially a two-person effort. The tool itself is more of a prototype / proof of concept, and not as mature as you would like in the end, but it is at least in a stage where potential users could give it a try with our help, of course, probably. To make this really usable to nonexperts, there would be definitely more work required to make this into something viable.

Suzanne: Sure. Because this starts from an external tool, ATSV, are there other modeling languages like SysML that are commonly used that you would be able to use with this yet, or is that something that would be in the future? Where would that fit, because I know a lot of my engineering friends are SysML people.

Lutz: Yes, we thought about that a bit because you mentioned already that AADL has a learning curve, and a lot of potential users will be familiar with SysML already. There is one thing that makes that a bit tricky to use SysML instead of AADL for this, and that is in SysML you have much more freedom to develop your models. In AADL, it is much more strict and much more formal and has much more constraints, which gives it the advantage that it is easier to write an analysis that is applicable to any AADL model that you throw at it.

With SysML, it is often that you probably, inside a given organization, you probably have some standards that you use for your SysML models, additional constraints that are not part of SysML but that [are] part of how you build your models in your organization. Then the analysis would, of course, need to be specific to this specific style of using SysML. That makes this a bit more challenging to just move to a different language, but that would, of course, broaden the appeal a lot, to be able to support SysML.

Suzanne: Possible future research projects.

Lutz: SysML is also moving into a bit more formal…

Suzanne: Yeah.

Lutz: …with the next version.

Suzanne: I am observing that people, they call them *design rules* or *business rules*, where that idea of *SysML, but you must use these kinds of parameters for this kind of a function*, and where you actually add some of that formalism into the SysML from your local users is a practice that I am seeing more in the engineering communities than I did in the past. So, there might be something there in the future.

I have been leading you into some transition topics, in terms of future uses, and different areas of possible extension. What is the transition strategy for this? Are you looking for partners to evolve the tools? Are you looking more for users to try it and give you feedback on it? If they want to try it, how do they access it? What other resources are available to them in terms of documentation and guidance?

Sam: Absolutely. We are looking for all three of those things: individual users, for organizations to try it out, and then if they want to partner with us to evolve it in a direction that is more amenable to their approach, that is something that we are super open to. We have had some queries from researchers that are doing other research-type things with design-space exploration that are interested in specific elements of the work as well. We are working with them, and we would love to hear from more if you are doing research, and some particular aspect of GATSE interests you and you don't want to use the whole toolchain, we get it. Please reach out.

As a starting point, you can download and install the tool now. There are full set-up instructions on the project's GitHub page, which is just https://github.com/osate/osate2-GTSE. I am sure this is all going to be…

Suzanne: It'll be in the transcript.

Sam: It will be linked in the transcript, too. In short, though, you download ATSV from the ATSV website, OSATE from the OSATE website, and then, install GATSE through OSATE, and you are good to go. I will add my own caveat, just like Lutz said. The SEI produces technology at a range of maturity levels. Most of it is ready to use. Some of it is a little bit more experimental. GATSE is, of course, downloadable and usable, but it is a little bit more experimental. It will definitely work best if you are familiar with AADL and OSATE already. If this is the first you're hearing about those, I would encourage you to check out the training and educational resources that we have. Peter Feiler, who Lutz mentioned earlier, and Dave Gluch, another researcher from the team, put out a textbook that I used to teach myself AADL and OSATE. It's great. We have an e-learning course for more rapid, hands-on learning, and then, of course, all sorts of tech reports and user guides that'll show you the ropes.

As far as documentation for GATSE, the best resource is going to be the journal paper that I mentioned earlier in *Software and Systems Modeling*. We have user-oriented documentation on

the GitHub page. The journal article does a better job of explaining the how and why. If you know exactly what you want to do, the GitHub documentation will help you, but since this is sort of a new area, the journal explanation of why certain things are the way they are will probably be useful. You are also welcome to post any issues you have to the GitHub repository. Email Lutz or [me] with questions about the software, the technology, or pose questions on the OSATE mailing list, which is linked on osate.org, the main website. One last time, [we] really welcome all sort of transition opportunities. So, if you are curious about this, please reach out. We would love to talk.

Suzanne: Well, I want to thank you both for talking about this today. This podcast had to be rescheduled a couple of times. When I first read the blog post, I'm like, *Oh, I'm so excited.* I'm glad that I finally got to talk to both of you about it, because I think this is the direction that model-based engineering can take. You are demonstrating where it can go to where it's not quite so human-intensive for every little piece of it, and that just adds power to the learning that the people get from models. So, I love that.

As we have said multiple times, we are going to include links to all the things we talked about in the transcripts. I want to finally remind our audience that our podcasts are available pretty much everywhere you find your podcasts. My favorite is, of course, the SEI YouTube channel, and if you like what you hear and see today, you are always welcome to give us a thumbs-up. No pressure, but I do want to thank all of our audience for joining us today. I hope that some of you take the plunge and get involved with GATSE and AADL, because I think it's really going in a direction that's going to be helpful, especially to the safety and the quality attributes that get left behind a lot of times in our software architecting. So, thanks to both of you.

Sam: Thank you.

Lutz: Thank you.

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.