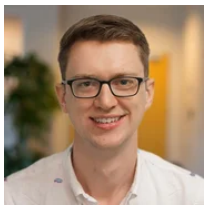
Software Engineering Institute

SEI Blog

A Model-Based Tool to Assist in the Design of Safety-Critical Systems



SAM PROCTER

MARCH 7, 2022

PUBLISHED IN

Software Architecture

CITE

Get Citation

The design of critical systems—those used in aircraft, medical devices, etc.—is becoming increasingly challenging as they increase in sophistication and complexity. A recent research project at the SEI aims to improve the way these systems are designed by allowing engineers to evaluate more design

options in less time than they do now. The state of the art in critical system design is model-based engineering, but it requires engineers to manually construct a model of their system and then analyze it for various performance and cost characteristics. As this post describes, we prototyped a language extension and software tool—collectively referred to as the *Guided Architecture Trade Space Explorer (GATSE)*— that partially automates this process so system engineers can rapidly explore combinations of different design options.

A New Paradigm

We are not the first to look at the integration of automation and system design. At first blush, it may seem like an *optimization* problem, where system designers might simply specify requirements—e.g., "the system shall cost less than $10M" and "the system shall respond to inputs in less than 5ms"—and then, given a supply of components and configuration options, simply find an architecture that satisfies all design constraints. Indeed, this approach has been taken by some researchers in this area. We share the recognition of others, though, that since many of a system's quality attributes are not easily quantifiable, it is better to use automation to augment engineers' efforts rather than partially replace them.

Far more common than *optimization*, however, is the standard *guess-and-check* style of system design, where engineers first select system components and configuration options based on intuition or familiarity and subsequently check their designs using various analyses. This project was designed to explore a newer paradigm, though, called *design by shopping*, where engineers first specify component and configuration options, and then valid system designs are automatically generated and analyzed for performance and cost characteristics. Designers can then "shop for" the system design they want by exploring the space of possible system configurations; since these configurations necessarily entail various tradeoffs between their quality attributes (e.g., a more expensive system might have better performance), this gives rise to the term *trade space*.

Project Tasks

GATSE relies on three modifications to existing technologies to improve the way critical systems are designed.

1. **Modeling Language Extensions**—First, we extended a modeling language that designers use to describe their systems so that their models can be partially specified. In the status quo, system designers must specify each part of their system before analyzing it. In this effort, we modified a system specification language—the Architecture Analysis and Design Language, or AADL—so that designers can fully specify some design options, but only specify the sets of options they're considering for other design options. The system elements that are not completely specified—referred to as *choicepoints*—would instead be specified as a set of valid options, or choices. For example, a system may need a processing unit (the choicepoint), but there might be several different options, each with a different price, computation speed, and required amount of power. Each option is a valid candidate for the processing unit choicepoint.
2. **Connecting to a Trade space Visualizer**—Second, we connected the SEI's model-based engineering workbench, called the Open Source AADL Tool Environment (OSATE), to design-by-shopping software called the ARL Trade Space Visualizer (ATSV). ATSV was developed by researchers at Penn State University (in projects unaffiliated with the SEI) to explore the trade space of physical systems that can be described by mathematical models, such as different options for wing-shapes. We modified OSATE to both be able to receive input from ATSV, and to send analysis results back to the program, instead of directly to the user for manual analysis. This way, ATSV will be able to update its internal (genetic/evolutionary) algorithms with the performance and cost characteristics of the system it chose the design options for. This information can then be displayed graphically, and ATSV allows users to specify their preferences to guide which system configurations are selected and analyzed next. ATSV is designed to run in batches—it takes about a second (on my laptop) to select an option for each choicepoint, build the finalized model, analyze it, and then store the results for display. After the batch is complete, the characteristics of each candidate architecture are displayed graphically so a user can see trends emerge in the system's trade space.
3. **Automating System Configuration and Analysis**—Finally, we modified OSATE so that after it receives input from ATSV, it can use that input to create a valid system model and run the analyses specified by the user. Given a partial system specification (from the user) and a set of component and configuration choices (from ATSV), OSATE will be able to fill in the gaps to create a complete system specification. It then automatically runs the specified performance, cost, and other analyses and reports its output back to ATSV.

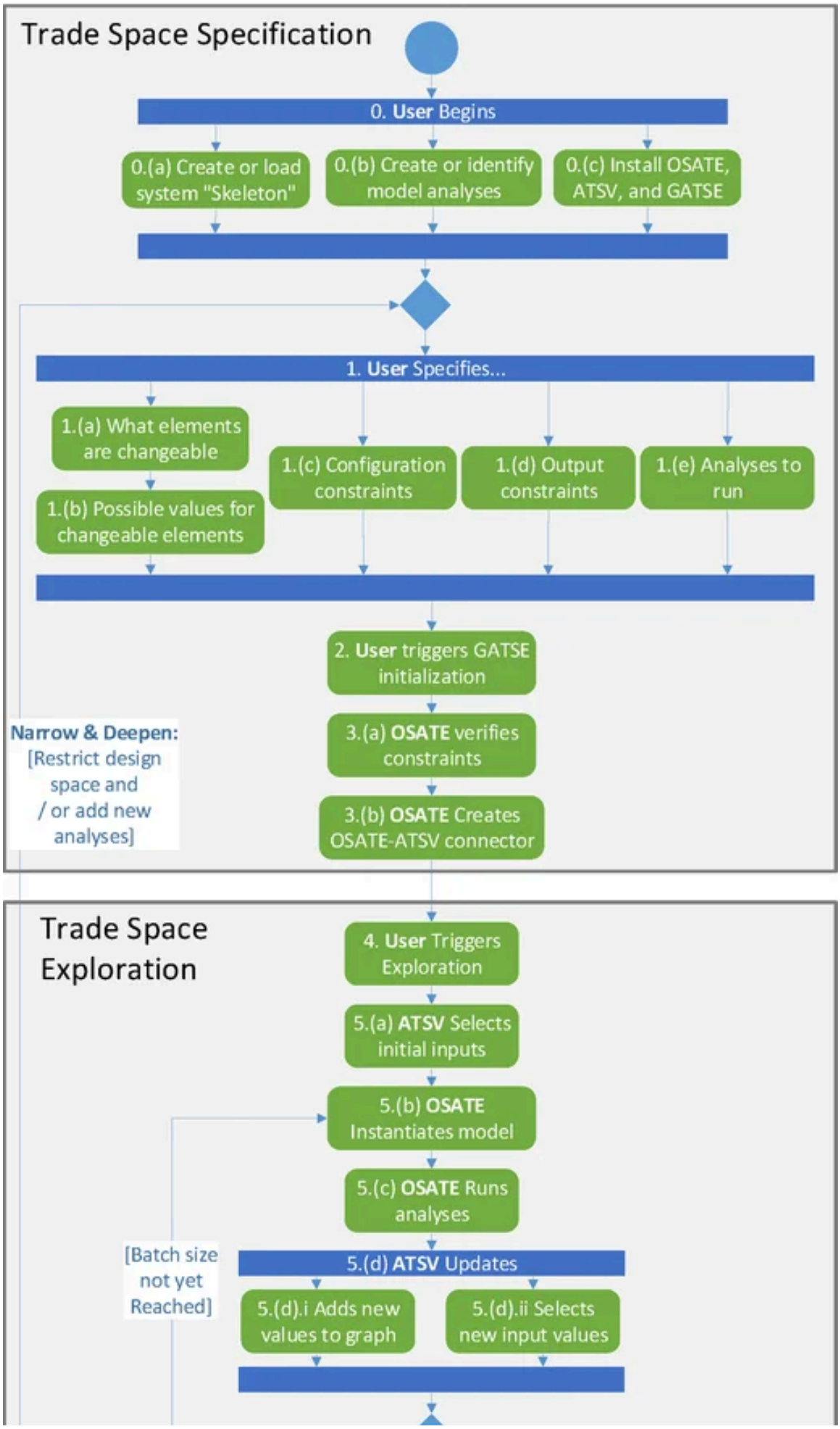

Trade Space Specification
0. User Begins
0.(a) Create or load system "Skeleton"
0.(b) Create or identify model analyses
0.(c) Install OSATE, ATSV, and GATSE
1. User Specifies...
1.(a) What elements are changeable
1.(b) Possible values for changeable elements
1.(c) Configuration constraints
1.(d) Output constraints
1.(e) Analyses to run
2. User triggers GATSE initialization
3.(a) OSATE verifies constraints
3.(b) OSATE Creates OSATE-ATSV connector
Narrow & Deepen: [Restrict design space and / or add new analyses]
Trade Space Exploration
4. User Triggers Exploration
5.(a) ATSV Selects initial inputs
5.(b) OSATE Instantiates model
5.(c) OSATE Runs analyses
5.(d) ATSV Updates
5.(d).i Adds new values to graph
5.(d).ii Selects new input values
[Batch size not yet Reached]

Figure 1 - GATSE Workflow, from “Guided architecture trade space exploration: fusing model-based engineering and design by shopping” by Sam Procter and Lutz Wrage.

Guided trade space exploration using OSATE and ATSV takes place in two main phases: specification and exploration. See Figure 1 for a graphical workflow.

Trade Space Specification—The preliminary task (0, in Figure 1) consists of installing ATSV, OSATE, and the GATSE plugin; specifying an AADL model; and identifying one or more analyses to run. The user must then specify the system’s trade space (1, in Figure 1). This specification is done by indicating what elements in the system can change and the possible values for those changes (e.g., the possible transmission rates for a bus or different models of a sensor), any constraints on those changes (e.g., a particular component might require or forbid the use of another component due to incompatibilities), any output constraints (e.g., maximum allowable power consumption), and which analyses to run. The user can then trigger the GATSE-Initialization (2, in Figure 1) and OSATE will perform several tasks (collectively step 3 in Figure 1):

1. Ensure that the user’s choicepoint constraints are feasible.
2. Create an ATSV engine configuration file. Users should never have to open/modify the engine configuration, but if you’re curious you can see the javadoc/comments in the classes in that package for a deeper explanation of the specific elements of the configuration.
3. Create initial ATSV input/output files. These are very small, simple comma-delimited files named `input.txt` and `output.txt` that contain an entry for each input/output variable mapped to the variable’s default value, which is derived from its type. These files (and those discussed in items 4 and 5 below) are also placed in the user-specified directory, and should never need user-interaction.
4. Generate `request.properties`. This file encodes the user’s choicepoint specifications in a format that is easily used by `connector.jar`.
5. Copy `connector.jar`, `parser.jar`, and `run.sh` to the user-specified directory. These files do not depend on the user’s model, though connector.jar may be updated between GTSE-plugin releases.
    1. `connector.jar` opens a socket and uses it to connect the running version of OSATE to the running instance of ATSV. Its processes are

shown in the middle column of the diagram.
2. `parser.jar` reads the input file and formats it for ATSV.
3. `run.sh` (or `run.bat` on windows systems) is what is executed by ATSV. It calls `connector.jar` with the user-specified port number.

Trade Space Exploration The second phase begins when the user triggers the exploration (4 in Figure 1). This phase is almost entirely automated – ATSV and OSATE (with the GATSE plugin) do most of the work. When the user selects the generated engine configuration and starts the analysis, the following substeps (collectively step 5 in Figure 1) occur repeatedly:

1. ATSV generates possible input values, either randomly or according to an optimization function, depending on if one has been specified. The input values are also consistent with regards to the constraints set previously.
2. The `connector.jar` creates a Request object based on the ATSV input values (both specific choices for choicepoints and analyses to run), serializes it, and sends it to OSATE over its open port.
3. OSATE decodes the request object and uses it to instantiate the specified model using the specified choices.
4. OSATE runs the specified analyses on the newly-created instance model.
5. OSATE creates a Response object with the resulting values (or, if present, the exception that was thrown).
6. The `connector.jar` writes the output to the `output.txt` file and terminates.
7. ATSV reads the output file, uses the new data to pick new input values and —if it's the end of the batch run—updates the display.

At the end of the run, there will be a potentially large number of candidate architectures. Using ATSV, these can be represented in multiple ways, Figure 2 shows a simple two-dimensional graph with a third system aspect represented using color. In it, the user has chosen to use the system candidates' Price as the X axis, Weight as the Y axis, and "Braking Power" – a hypothetical measure – to determine the color of the point. Any quantifiable system aspect could be used for any of these axes, however, and they can be easily changed as the system's trade space is explored.

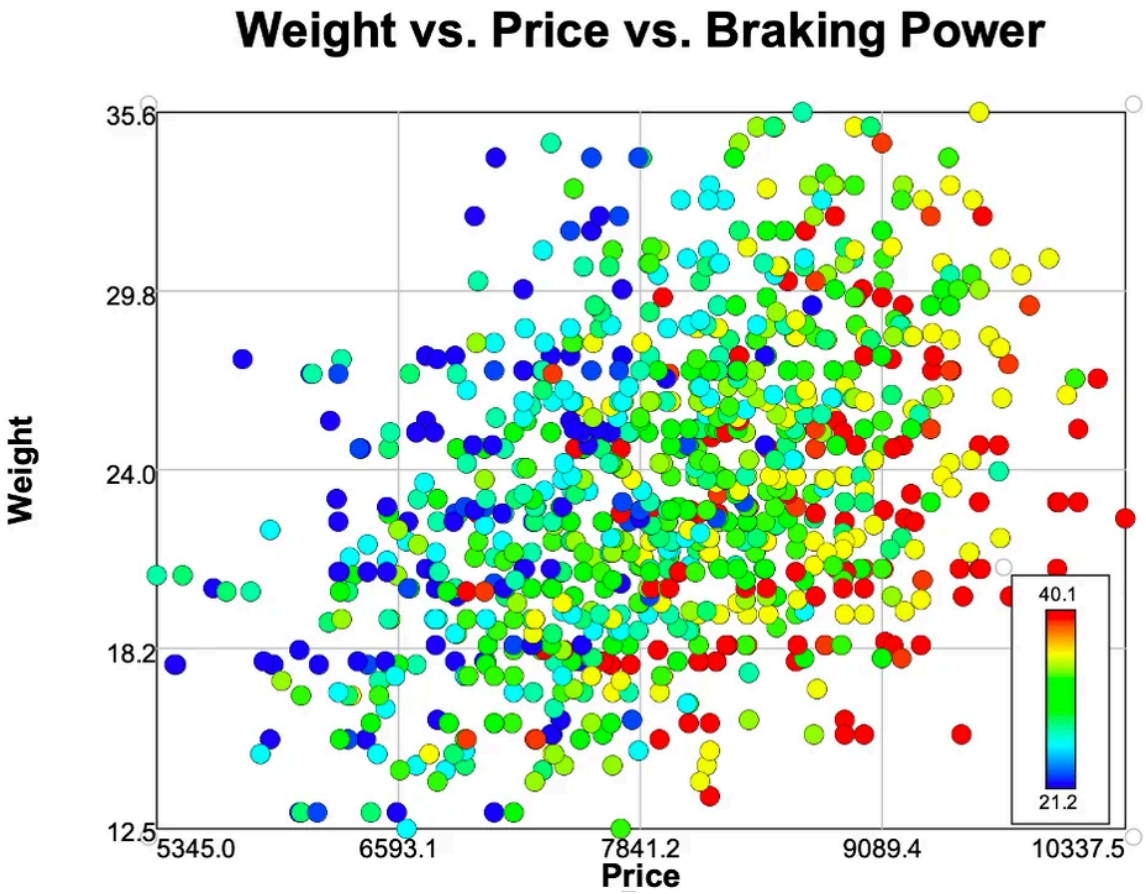


Figure 2: ATSV displaying candidate architectures from GATSE. Each point represents the architecture of a hypothetical aircraft braking subsystem. From “Guided architecture trade space exploration: fusing model-based engineering and design by shopping” by Sam Procter and Lutz Wrage.

If the user is not satisfied with the output values, i.e., none of the architectures generated by the tool will suffice, he or she can further modify the model or input parameters (by returning to step 1 from Figure 1). If one or more of the candidate architectures is satisfactory, though, the user can select them to view additional details, as shown in Figure 3.

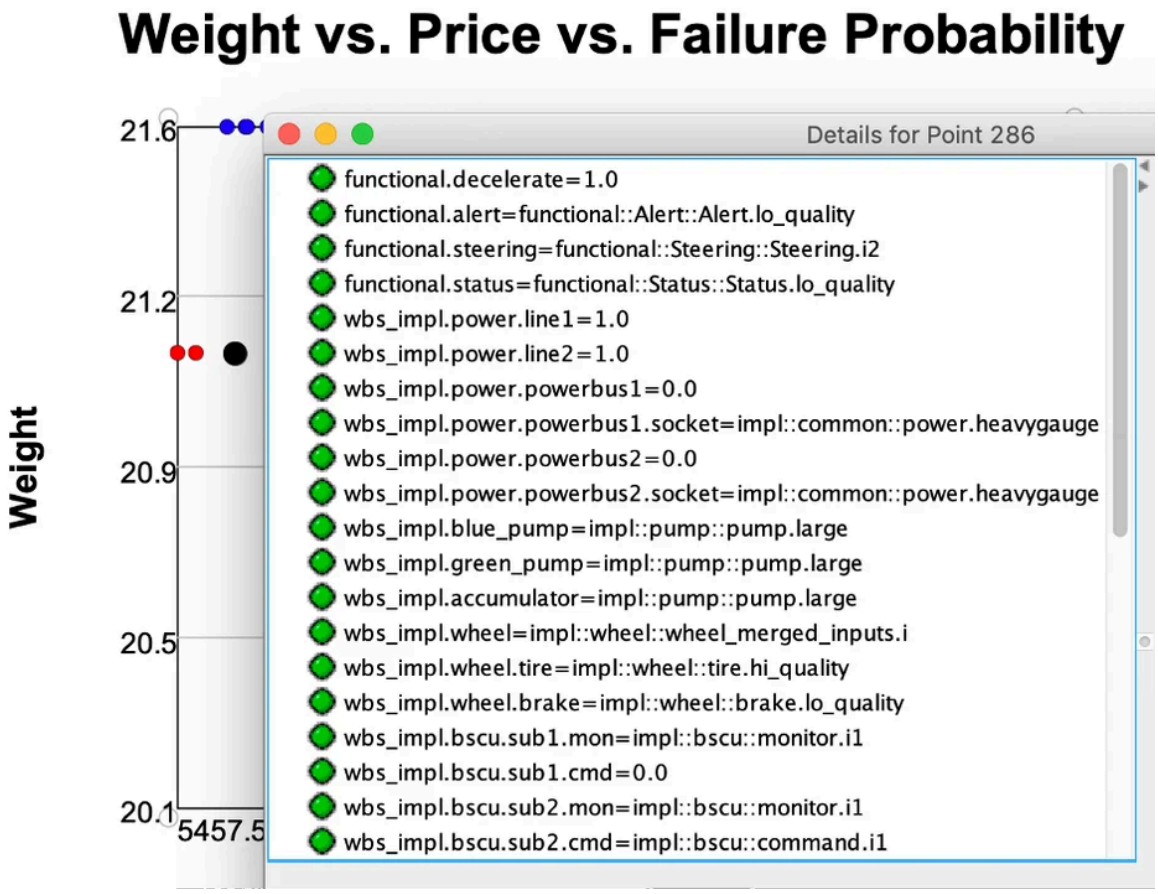


Figure 3: ATSV displaying the details of a candidate architecture from GATSE, including the precise results of system analyses and the specific configuration options necessary to build this particular system architecture. From “Guided architecture trade space exploration: fusing model-based engineering and design by shopping” by Sam Procter and Lutz Wrage.

The Need for Architectural Modeling

All of this modeling and analysis may seem like a lot of extra work for relatively little payoff. If we’ve been designing aircraft and medical devices for years using standard techniques, why should we change something that’s working? It all comes down to the increased complexity of modern, critical systems.

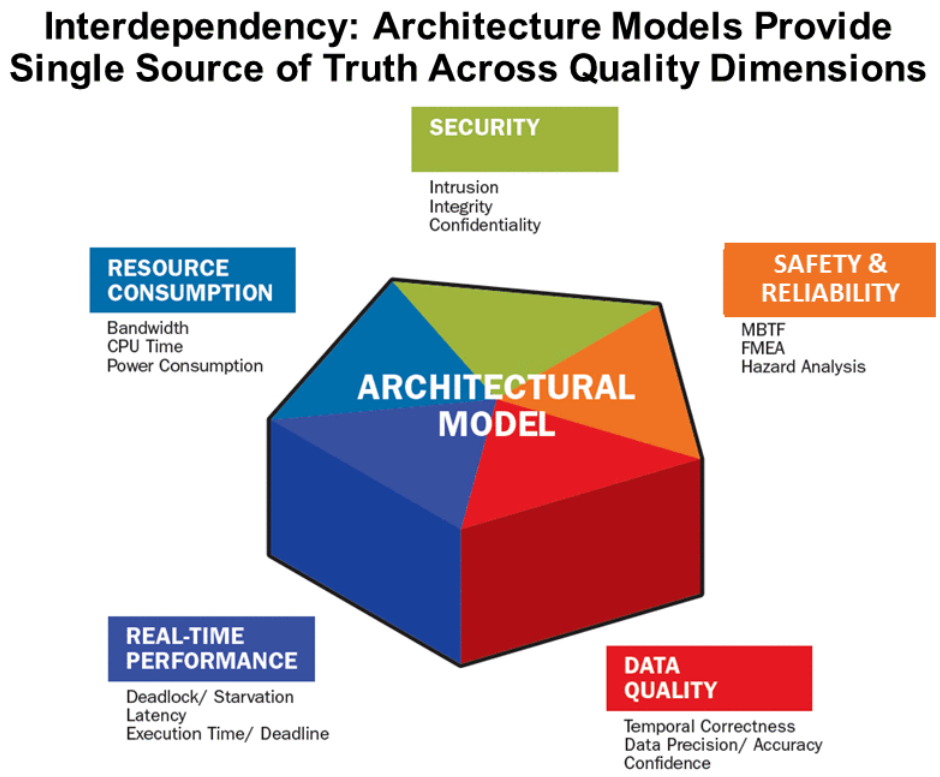


Figure 4: The cascading effects of a system change on different quality dimensions.

Much of this increase stems from the interconnectedness of system attributes. Consider the example shown in Figure 4. If system designers change the key size of the encryption used in a system from 128 to 256 bit, they may only be thinking of the security implications of the change—a larger key size means that decryption of messages will be harder. A larger key size, however, may require more processing power from the CPU, which means that it may take longer and potentially impact system latency. That latency may in turn cause the system to miss its timing requirements, which could potentially lead to a safety hazard. It is this interconnectedness that AADL and OSATE are designed to help with: by using an architectural model as a single source of truth, many aspects of a system can be considered simultaneously.

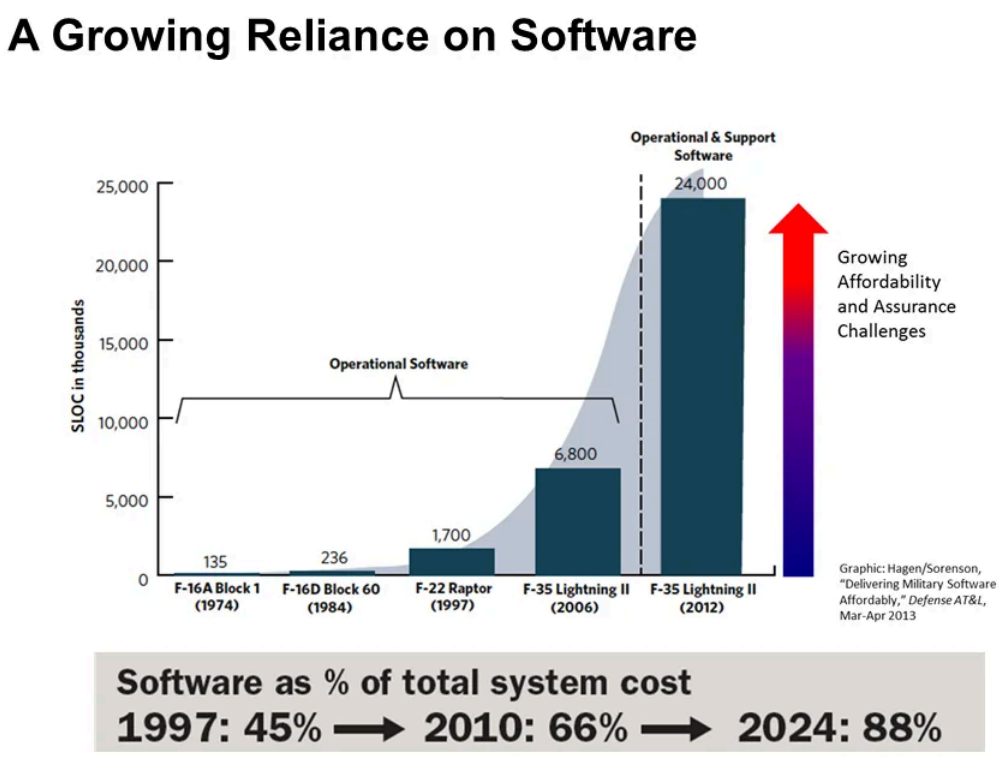


Figure 5: The growth of software size and cost.

Software poses a particular challenge to system design. That's because software's greatest strength—its tremendous flexibility—also poses the biggest risk when it comes time to analyze a system's performance and safety characteristics. Though the design-by-shopping paradigm has been used before, on things such as vehicle design and radios for satellites as well as the previously mentioned wing design, it hasn't been used for software-based system configuration. As Figure 5 shows, software size and costs are growing at an incredible rate—anything we can do in this area should be beneficial to keeping system development costs down.

Challenges Encountered

- **Interacting with ATSV**—While ATSV is feature rich, it was not designed for the use cases we've envisioned in this project. Interfacing it with a significant, standalone system design workbench like OSATE required some careful engineering. A good example of this problem is in how ATSV treats what it calls *configurators*, which are essentially restrictions on the relationships between selected choices. For example, many times

selecting a choice for one part of the system (e.g., the processor architecture) invalidates some options for another part of the system (e.g., software that requires a specific processor architecture to function); it must be possible to specify this relationship between the choicepoints. ATSV assumes that the specified configurators are fairly simple, so it does not validate their consistency. Thus, it's possible to over-constrain a system design so that no viable candidates can be constructed. We addressed this by checking the configurators using some novel theoretical work and a boolean satisfiability checker before passing them to ATSV.

- **Shifting Paradigms**—As we discussed earlier, design by shopping is a new paradigm for critical system design. Finding partially-specified example systems has proven to be nearly impossible, so we've created our own or adapt existing system models. Even if we had a robust, industrial-grade software tool, the gap between the state of the art and practice could be a significant barrier to adoption.
- **Scoping Challenges**—We are fortunate to have high-quality theoretical work in this area to guide development of our tooling and process. That said, the gap between the ideal feature set and what can be built given the time and funding constraints, and understood easily by end-users is significant. We have tried to scope our project carefully, including enough in the initial prototype to demonstrate value, but not tackling the complete set of desired capabilities. We also paid close attention to those features that are hard to intuitively understand or use—their cost-to-benefit ratio will likely be particularly poor in light of the aforementioned paradigm shift.

On to Commercial Use

In a later post, we'll detail how the open and extensible nature of OSATE and AADL dovetail to make GATSE highly adaptable to domain- or product-specific needs. We are interested in evaluating this tool's applicability to commercial or industrial system design. If you know of an opportunity to do this type of evaluation, or you'd like to see how GATSE can help in your system design tests, please reach out!

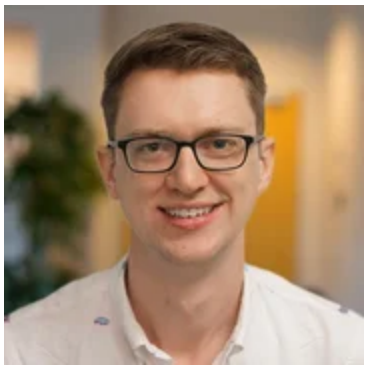
ADDITIONAL RESOURCES

- This work is described in greater detail in the paper "Guided architecture trade space exploration: fusing model-based engineering

and design by shopping," which I coauthored with Lutz Wrage - https://dx.doi.org/10.1007/s10270-021-00889-8

  - That paper is itself an expansion of Lutz and my initial paper on the topic, which shares its title but went to the MODELS conference in 2019: https://doi.org/10.1109/MODELS.2019.000-9.
- Learn more about AADL or OSATE.
- Here is a short video recording of me presenting on this project at the SEI Research Review: https://www.youtube.com/watch?v=_Kkkj8P31OM Note that the project was not complete when this was filmed, so the data and papers from this post contain additional details.
- The best source for understanding ATSV are the papers published by the PSU developers, in particular see Stump et. al's Visual Steering Commands for Trade Space Exploration: User-Guided Sampling with Example.

WRITTEN BY



Sam Procter

DIGITAL LIBRARY PUBLICATIONS

SEND A MESSAGE

MORE BY THE AUTHOR

The OSATE Slicer: Fast Reachability Query Support for Architectural Models

NOVEMBER 13, 2023 • BY SAM PROCTER

Integrating Safety and Security Engineering for Mission-Critical Systems

MAY 10, 2021 • BY SAM PROCTER, SHOLOM G. COHEN

The AADL Error Library: 4 Families of System Errors

MAY 20, 2019 • BY SAM PROCTER

Simultaneous Analysis of Safety and Security of a Critical System

SEPTEMBER 11, 2017 • BY SAM PROCTER

MORE IN SOFTWARE ARCHITECTURE

Building Quality Software: 4 Engineering-Centric Techniques

AUGUST 19, 2024 • BY ALEJANDRO GOMEZ

The OSATE Slicer: Fast Reachability Query Support for Architectural Models

NOVEMBER 13, 2023 • BY SAM PROCTER

How to Use Docker and NS-3 to Create Realistic Network Simulations

MARCH 27, 2023 • BY ALEJANDRO GOMEZ

Software Isolation: Why It Matters to Software Evolution and Why Everybody Puts It Off

MARCH 20, 2023 • BY MARIO BENITEZ PRECIADO

Experiences Documenting and Remediating Enterprise Technical Debt

DECEMBER 19, 2022 • BY STEPHANY BELLOMO