Zero-Trust Management

Brandon Baylor (Chevron), Eric Daimler (Conexus AI),
Esteban Montero (Chevron), Ryan Wisnesky (Conexus AI)

October 16, 2023

Introduction

Zero-trust management is the name of an emerging community of practice aimed at solving the problem that consensus, as a management method, no longer scales, often leading to disastrous engineering consequences. That is, modern engineered systems are exceeding not only the capacity of the human mind to comprehend them, but also the organizational capacity of enterprises to make correct and trustworthy (rigorously and rationally optimized) decisions about them using consensus-based methods, where by "consensus as a management method" we mean the practice of putting engineers from different disciplines into the same conference room to come to some kind agreement about a decision to be made.

Such consensus-based approaches cannot scale simply because there aren't conference rooms large enough to hold all of the engineers required to make decisions about today's increasingly complex projects. So in this paper we propose a new methodology for scalable management and decision making, that is called *zero trust* because it does not rely on engineers to come to agreement: modern advances in mathematics, AI, computing power, etc make it possible to formally represent viewpoints symbolically and probabilistically on computers and compose them to form nuanced computational truths that are guaranteed to respect the input viewpoints while being free of contradictions. We furthermore describe a curriculum for training in this methodology, where we report on each curriculum component using a case study undertaken by industry experts using various formal methods tools. (In this abstract, we only describe one components, with more described in the upcoming full version of this paper).

Formal Methods for Engineers - An Underdone Science

We claim that *formal methods for engineers* is an underdone science, with the systematic non-production and non-dissemination of such applied knowledge being especially bad in the United States (compared to Europe). Indeed, there is no computational reason why engineering artifacts should be more difficult to apply formal methods to than the software artifacts to which they are now commonly applied; in fact, the field of applied category theory (`https://en.wikipedia.org/wiki/Applied_category_theory`) has arisen in part to apply formal methods, those from an emerging branch of mathematics called category theory, to engineering (applied) domains specifically, and we view this paper as both contributing to those goals and spreading the word that the time is ripe to tackle such challenges. Reasons of space preclude us from speculating on the causes of this science being underdone, but we do have a solution: namely, to train engineers on formal methods. (As opposed to, say, hoping to develop some universal tool or universal methodology, although tools will be required.) We elaborate on this curriculum in the next section.

1 Curriculum: Engineering with Rigor

The goal of the curriculum below (envisioned as a 10 week long course, with one week per item, and a week long intro/outro) is to teach engineers to specify systems in a way that humans and computers can reason about, and to describe the advantages that such reasoning provide, such as automation and assurance. We assume a mathematical background of an engineer in a non IT field, such as oil engineering.

- *Lightweight formal methods.* In this component, the goal is to learn how to rigorously represent engineering artifacts (e.g. data, processes) on a computer so that algorithms can reason about them. The component covers three common lightweight formal methods for doing so.

    - Functional methods (including functional programming for Excel)
    - Equational methods (including Knuth-Bendix completion)
    - Logical methods (including SQL and Datalog programming)

- *Compositionality.* In this component, the goal is to learn how to correctly define and compose (combine) engineering artifacts using a computer so that the combined artifact respects the meaning of the original ones. The component covers three common ways of doing so.
    - Categorical Algebra (including databases)
    - String diagram notation (including non-cartesian operads)
    - Polynomial datatypes (lists, trees, infinite streams, etc)

- *Computational abstractions for engineering.* In this component, the goal is to learn about the formalizations of particular abstractions common in engineering, and how they can be composed using the above techniques.
    - Abstract datatypes (Graphs, Queues, Stacks, etc)
    - Finite State Machines and Stack Machines
    - Concurrent processes (including process calculi and the control theory of feedback loops)

2 Case Study - Spreadsheet Integration

In this section we can only briefly sketch a case study that is written up in full at `https://arxiv.org/abs/2209.14457`; the task described in this case study could be tackled by a graduate of our curriculum. The write-up describes a method for merging multiple spreadsheets into one sheet, and/or exchanging data among the sheets, by expressing each sheet's formulae as an algebraic (equational) theory and each sheet's values as a model of its theory, expressing the overlap between the sheets as theory and model morphisms, and then performing "colimit", "lifting", and "Kan-extension" constructions from category theory to compute a canonically "universal" integrated theory and model, which can then be expressed as a spreadsheet. It describes a detailed example of this methodology on a real-world oil and gas calculation at a major energy company, describing the theories and models that arise when integrating two different casing pressure test (MASP) calculation spreadsheets constructed by two non-interacting engineers. It also describes the automated theorem proving burden associated with both verifying the semantics preservation of the overlap mappings as well as verifying the "conservativity"/"consistency" of the resulting integrated sheet.

3 Conclusion

As industry moves to model-based software for performing day-to-day engineering tasks, it becomes more and more important to ensure the semantic consistency of models composed of related models (integrated models). For example, we don't want errors to propagate from one model to another or to try to integrate models with conflicting requirements (e.g. positive and negative voltage at the same time). Commonly, to ensure semantic consistency of integrated models the human subject matter experts that created the input models communicate informally with each other about their respective understandings of the integrated model and check for consistency of the integrated model with respect to their original models (think NASA mission control with its various 'functions'). Although using groups of human experts to certify/construct integrated models works for small numbers of input models, this methodology is both costly and not scalable, because in principle, all the humans may need to communicate with each other (not everything can pass through mission control), and moreover, people may disagree about the meaning of the input and/or integrated models. Therefore, to construct/certify engineering models in a scalable way, we must eliminate the need to relate each input model to each other, and we must make sure that when models are merged, the original authors of the models need not be involved. It was in looking for ways to solve this problem we realized that "consensus-free" / "zero trust" management was an underdone science, leading us to propose the curriculum above - its graduates are qualified to undertake tasks similar to those described in the case study, and many more described in the upcoming full version of this paper.