

Can't Be Built

Richard P. Gabriel
IBM Research
rpg@us.ibm.com

Abstract

“An ultra-large-scale (ULS) system is one that is impossible to build (today) because it exceeds some critical limit of today’s software engineering technology.” This means that only ULS systems of the past can be built. This logical problem provides the key for how to proceed.

1. Introduction

Ponder this:

An ultra-large-scale (ULS) system is one that is impossible to build (today) because it exceeds some critical limit of today’s software engineering technology.

This was the first working definition of ultra-large-scale systems used by the authors of the SEI ULS study group. Looking at it logically—that is, naïvely—we can conclude it’s impossible to ever say “look, I’ve just built a ULS system” because, by definition, to have built it means that it does not exceed some critical limit of engineering (anymore). Perhaps it once did. One can argue that most of the apps we routinely write today are ULS systems of the past. In fact, my own erstwhile company, Lucid, was frequently lambasted because its primary product (a Lisp implementation) was absurdly big—3 megabytes. I just read this on a Java™ discussion thread:

By default the JVM size is 64 mb.

And applications can be 3gb or more. However, none of the ULS report authors took seriously the logical impossibility of building a ULS system. Yet, much can be learned from this definition and conundrum.

2. Not Built

The obvious answer to the paradox is that an existing ULS system was not built. Not in the usually understood sense of the word “build.” For example, we don’t exactly build gardens, parks, or even cities. These grow up through organic and only partially planned acts. In many details there is no plan or design aside from what just happens or what is appropriate at the right time and place.

Making things this way doesn’t require absence of rhyme or reason, just that the rhyme and reason come as side-effects of other facts and actions. And though not built in the conventional sense, it’s still clear such artifacts are, indeed, artificial. That is, made by the hands of people. This of course is a conceit—when a gardener chooses seeds, tills the soil forming rows, and plants and nurtures those seeds into tomatoes, beans, and squash, we attribute the successful garden to the gardener, but the seeds themselves and the plants played a vital role, and they were “designed” (largely) outside the influence of people.

Even parts of systems today are not built in the old-fashioned sense. Consider a highly optimizing / parallelizing compiler as a system constructor. A programmer indeed specified many of the details of the program that is actually executed after compilation, but most such programmers would be in for a surprise to see what his/her code turned into. On several occasions, I’ve written the equivalent of compilers in order to construct code my intellect was unable to grasp well enough to do by itself. Other examples abound: modern operating systems are put together by a build process that is locally understandable but globally incomprehensible.

ULS systems will therefore not be built but will emerge from processes perhaps only dimly understood. For such systems to make sense requires understanding how order emerges from locally understood and motivated actions. Such work is underway, but I believe attitude and philosophy lag reality. Many software

developers and process wonks don't like to think that software just happens—and it doesn't, but it will feel more like that for ULS systems than ever before.

3. Where Will Ideas Come From?

The key ideas are as follows:

- if things like APIs will play a role, their wiring will need to be done by software and not by people
- if people are to “wire things up,” the wiring will need to be self-creating or at least self-detailing
- accumulation, self-assembly, and self-generation will be the primary mechanisms for system creation

There are two basic sources of ideas for how to piece together systems without building them:

- look at earlier efforts to create systems without building them in the traditional sense
- look at other disciplines and sciences for metaphors and approaches

Earlier / existing efforts include urban design, mechanism design, game theory, and the older models of artificial intelligence systems (where the bulk of research was on programming languages and system building). Disciplines to look at include economics, biology, chemistry, physics, and sociology. The idea is to look at how systems come into existence and sustain their own existence. This is the creation of new foundations for computing—at least for the purpose of creating ultra-large-scale systems; perhaps the fundamentals of small-scale software will remain the same.

4. Visibility

One expects that among the issues to be explored is visibility within modules (to use an overworked term). The exploration of aspects highlights the problems with visibility. Older concepts of modularity (Parnas, for example [1] [2]) tout encapsulation and near-total information hiding. Any attempts to create modules that are not directly invoked through narrow communication pipes run afoul of these draconian information-hiding edicts because any communication between modules outside the narrow interfaces require some sort of description or visibility into them.

To define an aspect requires describing what other module it modifies and how it accomplishes that modification. The point of defining an aspect is to make construction of systems easier by keeping like con-

cerns together. However, the stated point of diamond-like modules is also to make construction of systems easier by keeping things not related *implementationally* separate. By definition almost, semantically unrelated facets of the same basic activity must be related implementationally, and so old-style modularity forces them together, while the concerns of ultra-large-scale systems want to force them apart. What has to give is the notion that activities of a program are not visible to the surrounding context.

An example—a stupid one—will illustrate what I mean. Suppose you know a forgetful old man who is still able to take care of himself mostly. You want to help him out by hiring an assistant who will go around with him, helping and keeping him out of trouble. Suppose one problem your old friend has is locking his keys in the car. The assistant tagging along can easily notice when they are about to get out of the car and can remind your friend to take his keys with him. This doesn't require some sort of Vulcan-inspired mind-meld to see into the inner workings of the old man; it's clear from what he is doing in plain sight what is happening and that the advice is appropriate. The old man doesn't require direct manipulation of his mind or body to be able to remember to take the key—an act of interpretation is all that is required.

We need to construct modules and systems so that there are things and actions taking place in plain sight, and modules need to be able to interpret advice and instructions.

5. Non-Human Design

Some aspects of design are too detailed or difficult for people to do reliably. In creating ultra-large-scale systems, we must be able to harness automated techniques to do this work. These techniques can be either algorithms (like optimizations), expert systems or other AI-based systems, or aleatoric systems like genetic algorithms, generate and test, or randomized search.

Systems put together this way will be understandable in only the ways we understand any naturally occurring system—through observation and essentially scientific methods. This kind of understanding will give us disquiet for a while, but we will need to get used to it.

6. Sustainability

A system put together without as much human intervention as we're used to will need to be “kept alive” using techniques we are not as comfortable with (at the moment). Many fault-tolerant techniques require

noticing errors and failures based on knowing what the component or components are supposed to be doing. What is noticed is deviation from normal activity. To date, the primary ways of noticing this require knowing beforehand what normal means.

It may be that to keep a system alive over the long haul, the mechanisms that accomplish that will need to be designed and constructed in the same non-built way that the ULS system itself is. This can be considered a coëvolution, and it might be even more dramatically subject to aleatoric mechanisms.

We imagine immune-system-like mechanisms, the use of populations of small components, a notion of proximity (in a dimensional space) to represent collaboration, and other, perhaps exotic techniques to add something akin to geometric structure to augment the primarily topological structures that we are used to in systems.

7. Conclusions

The conclusion I would like you, the reader, to reach is that we need to rethink some aspects of how

we put together systems. Not because the techniques we use today are wrong, but because they won't scale up well. We can always, I think, muddle along and get things done (including building ULS systems), but programming today is too difficult if we demand our programmers to command vast piles of indistinguishable details.

The realization we cannot build ULS systems should be seen as an opportunity to explore, not a reason to lament.

8. References

- [1] Parnas, D. L., "Information distribution aspects of design methodology," *Information Processing 71, Proceedings of the IFIP Congress 1 (North-Holland, 1972)* 339–344.
- [2] Parnas, D. L., "On the criteria to be used in decomposing systems into modules," *Communications of the ACM 15:12 (1972)* 1053–1058.