A Vision of Large-scale Software Design

Speech delivered at Ecole Normale Supérieure, Cachan
On the occasion of Doctor Honoris Causa ceremony

Jayadev Misra
April 09, 2010

Mesdames et messieurs: c'est un grand honneur pour moi de recevoir le Doctorat Honoris Causa de cette prestigieuse institution. Et c'est un plaisir et un privilège de prendre la parole devant cet auditoire distinguè.

[Ladies and gentlemen:

It is a great honor for me to receive Doctor Honoris Causa from this prestigious institution. And it is a pleasure and privilege to be speaking to this distinguished audience.]

I have been active in research for nearly 40 years. It is not possible to tell you all my experiences in the next 10 minutes, but I will try! More seriously, let me try to tell you how I stumbled on my latest research problem, what I can see clearly, and where my vision is blurred.

Nearly 15 years ago, I was the chair of my department. We pretend that it is an important position where all kinds of visionary decisions are made. In reality, the job is to make sure that "the trains run on time", that is, certain routine tasks are performed at appropriate moments.

One of the tasks I assigned to a new assistant was to administer the visit of an invited speaker. She ruined it completely. Afterwards, I sat down with her and made a complete list of all the steps she should have taken, as a reminder for the future. Each step was small, but each had to be performed in the appropriate sequence. The assistant contacts the speaker, proposing a set of possible dates for the visit. The speaker responds by choosing one of the dates. The assistant then contacts a hotel and two airlines, to get the cheapest ticket. She sends the hotel and airline information to the speaker who sends an acknowledgment. Only after receiving the acknowledgment, the assistant confirms both the hotel and the airline reservations. Then she reserves a room for the lecture, announces the lecture (by posting it at an appropriate web-site) and requests the audio-visual technician to check the equipment in the room prior to the lecture. There are many other steps

of a similar nature to be undertaken in this case.

I wondered afterwards if the assistant could be replaced by a machine. I pose the question only as a thought experiment, not a prescription for how to build a brutally capitalistic organization. After some thought I decided that it will be a long and complex process to design a computer program to solve this specific problem.

At the same time, I was thinking about the future of the internet. To the man in the street, the internet is a giant communication device; you can send email or you can download some articles to read. Using the internet to do useful computation was novel, and unfortunately, it is still novel today. Consider a typical problem we might try to solve. A client contacts two airlines simultaneously for price quotes. He buys a ticket from either airline if its quoted price is no more than 300 Euros, the cheapest ticket if both quotes are above 300 Euros, and any ticket if the other airline does not provide a timely quote. The client should receive an indication if neither airline provides a timely quote. I felt that we should have simple methods for tackling problems of this kind, but I did not see a way.

Another problem for which I did not see an elegant solution was how a computer would coordinate a disaster recovery operation, say after an earthquake. It may accept inputs from the medical staff, firemen and the police, and direct them by sending commands and information to their hand-held devices. A computer may be better at the task of coordination than a human dispatcher, sending announcements, making reservations in the local hospitals, blocking certain roads to traffic to let emergency vehicles pass, etc.

You may be wondering what these specific problems have to do with research in Computer Science. Don't computer scientists deal in lofty principles that use a lot of mathematical jargon, like category theory and other esoteric algebras? I felt that these problems, though very different, are related in some abstract sense. And a general approach for solving them will yield insight into the design of a large class of complex software problems. In our research, we play a tense game of generality vs. simplicity; we would like to solve as many different types of problems as possible using the same tools, but we would like to have a common simple notion around which to formulate their solutions. If the approach is not general enough, no one would care to use such solutions, and if it is not simple enough no one would care to use such solutions.

Let me talk about design simplicity for a few minutes, and then return to my specific problems.

In their capacity as a tool, computers will be but a ripple on the surface of our

culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind.

– Edsger W. Dijkstra, 1972

While we may debate if computers are merely ripples or tsunamis, let us ask the question, what is one of the main intellectual challenges posed by computers. Designs of large, complex artifacts. Computer systems are among the most complex of human creations. If I imagine that a symbol has one gram of weight, then some modern software systems weigh in thousands of tonnes. But each symbol is different in purpose from any other symbol, thus making such systems dwarf other human creations in complexity.

How do we design large complex software? How does nature design large complex systems? We can draw some inspiration from chemistry and biology where atoms are combined to form molecules, then compounds, then genome, proteins, and living things. There is a hierarchy in the design. We have now been able to understand some of the simpler layers of biological designs, mostly at the chemical level, because it admits of hierarchical explanation. We don't have to understand everything all at once; we can understand the parts one by one, and their interactions.

We can also be inspired by large social systems. We have communities and towns, that form a state, then a nation. Military systems are invariably hierarchical. And, bureaucracies all over the world are brilliant at designing ever more complex hierarchies.

Designs of software systems have adopted this natural principle. A system is designed as a composition of some components, where each component is similarly structured as a set of subcomponents until a component is simple enough to be directly coded or understood. This structure also permits different groups to collaborate in the design, by working on different components when the interactions among them are well understood; for the design to evolve over time, by replacing a component by a better one that corrects some defect, or provides more functionality; and in testing or verifying a system one component at a time.

Now, back to my little problems dealing with visiting speakers and airline quotes. I worked on these problems off and on for about 4 years without much success. I remember when and how I saw a glimmer of a solution. It was at an airport in Austria where I was having a cup of coffee; I did not have a clue before I touched the cup to my lips, and I knew the outline of the solution by the time I had finished my coffee. The inspiration came from Kleene Algebra, an

axiom system that we use quite extensively in certain areas of computer science, far removed from the problems I was tackling. Simplicity is often derived by packaging esoteric algebras within an accessible framework.

I realized that, first, I can, and I should, rely on a large supply of off-the-shelf components, programs written by others, for constructing my solutions. For the airline reservation problem, I can ask the airline database to provide a price quote; I myself don't have to code that program. External world provides many components. Today many hundreds of thousands of services reside on the web. I translated the first part of this speech from English to French using such a service. These services could be written by teams of people, or just a lone programmer; coded in many different languages, and available for free on the web, or in a proprietary form for use within an organization.

My research is about combining the available components. In doing arithmetic or standard algebra, we combine operands using operators such as addition and multiplication. Similarly, I designed operators for combining program components. These operators, or *combinators*, define the essence of the theory. They specify, for instance, the order in which their operands, i.e., the components, are to be executed, one after the other (as in parts of the speaker invitation problem) or simultaneously (as in requesting quotes from both airlines). They also specify how data is communicated among the components. The simplicity and generality of the combinators is a good measure of the effectiveness of research.

Let me give you examples of some of the simplest problems my students and collaborators have solved. A student designed a smart phone application so that you may walk down the street with your phone and it shows the interesting things around you, such as "there is a museum to your right that is having a special exhibit of Monet", or "to your left is a Michelin 3-star restaurant". Another student, who is much interested in live music bands, wrote an application that will automatically fill up his calendar with all the bands playing in my town for the next month, with dates and places; if a particular favorite band will be playing, it emails him and his friends asking if the program should order tickets. A program that runs in the background in my computer checks if I have a trip planned in the next 24 hours; if so, it prepares a weather report (by contacting a weather website), a list of interesting activities in that town (by contacting its chamber of commerce), and travel advisories (by contacting the center for disease control and the state department). My research group plans its meetings by using a program that sends an email to all group members asking for their availability at specific times, reminders if it does not receive timely responses, deciding a time at which most of the group can meet, or an email to me if such a time is impossible to find.

These systems are designed and built in mere weeks, not months.

I describe these problems not because they are hard, but because they capture the essence of many issues that arise in large-scale programming. Our collaborators at ENS Cachan have developed much larger systems for Telecommunications and Supply chain logistics, and are beginning to work on electronic Health applications. Researchers at Inria in Sophia Antipolis are designing novel multimedia applications that can interconnect multiple audio and video devices to be controlled by yet other multiple mobile devices. And, other groups are working on large scale simulations, adaptive work-flows and security of information flow.

Our society has decided to increasingly rely upon large software systems for nearly all facets of our existence. Unlike natural systems that have evolved over ages, we are faced with explicit designs of systems with many interacting components, multiple communication patterns, that are time sensitive, and prone to failure. Science of design is in its infancy. My vision about a grand unification theory of software design is still blurred. Yet, we learn to walk by taking baby steps. My hope is to lay a mathematical foundation, based on logic and algebra, on which others will build great edifices.

I am grateful to Ecole Normale Supérieure, Cachan, for believing in my vision, my collaborators and students for enhancing my understanding, and you, the patient audience, for listening to me. Thank you.