Engineering Software Intensive Systems

Joint EU/NSF Strategic Research Workshop

Vladimiro Sassone
University of Sussex
vs@susx.ac.uk
Tel: +44 1273 873828
Fax: +44 1273 671320

Short CV (10 lines):
My research activity concerns the semantics of concurrency and mobility. A central theme thereof regards methods for safe resource management in mobile, distributed systems, aimed at laying the foundations of robust, high-level programming paradigms for *global ubiquitous computing*. Recently my interests expanded to include trust management and reputation systems.

I am the Coordinator of the EU funded project ``MyThS: Models and Types for Security in Distributed Systems'' (2002--2005), the Principal Investigator of the EPSRC ``Third-Party Resource Usage for Pervasive Computing'' (2003--2006), and the Director of the EU Marie Curie Research Training Centre ``DisCo: Foundations of Distributed Computation'' (2002--2005). After completing my PhD in 1994, I held a Marie Curie Mobility Fellowship. I was since employed in Aarhus, Pisa, London, Catania, Copenhagen and Sussex.

Key words (no more than 5): GLOBAL COMPUTING, SECURITY, TRUST & REPUTATION, RESOURCE USAGE, CONTEXT-AWARENESS

Abstract (20 Lines):

ENGINEERING TRUST-BASED SOFTWARE INTENSIVE SYSTEMS

where I advocate a trust-based approach to context-awareness

Our society increasingly depends on open-ended, global computing infrastructures consisting of millions of individual components. Third-party computation, whereby migrating software and devices execute on networks owned and operated by others, lies at the very heart of the model, and will soon be the norm. Countless `pervasive' devices equipped with limited resources and computational power will roam the network, and support end-to-end applications which far exceed the devices' own capabilities. In such a scenario, the notion of `third-party resource usage' will rise to unprecedented centrality.

From the point of view of their owners, resources will need to be advertised, made available, managed, protected, and priced, while from the user's viewpoint, they will have to be discovered, explored, acquired, used according to rules, and paid for. The complexity of `digital communities' of such kind is rapidly growing beyond that of other man-made artifacts, and will reach that of natural social systems. The software-intensive mechanisms involved exceed by far our ability to design, comprehend and control, and are by and large the limiting factor to building and deploying innovative applications. Although we are able to analyse and explain the behaviour of each single component and each single component's interface, we are essentially clueless when willing to analyse – and less than ever predict! -- the system as a whole. This is not dissimilar from trying to make a prediction on, say, house prices growth, on the basis of our perfect understanding of the mechanisms at the root of market economies.

As we grow accustomed to rely on such systems for things as precious as human lives, their lack of robustness and vulnerability become unacceptable. We need to design and develop for safety, as we currently design for efficiency and for reuse. The impact of these demands on engineering software intensive systems – and actually

on Computer Science as a discipline – is dramatic. Scalability is of course a very present issue, intrinsically accompanied to our specific `globality' hypotheses, but by no means the only one. More generally, ubiquitous software systems are exposed throughout their lifetimes to the great variability of their operating environments, of which they have a very partial knowledge to start with. Change in scale is only one particular case of variations arising from a potentially unbounded number of different sources: new agents' arrivals or service deployment, connections and systems failures, new resource distribution and pricing schemata, and so on. For the purpose of this discussion, we will comprehend them all by the term *context-awareness.* Software systems will need ways to assess the contexts surrounding them, and adapt to their changes.

Recent approaches to context-awareness rely on powerful middleware, from which the software can `read out' all sort of interesting information about the environment. This appears unrealistic in general, and likely to presume too much of middleware's flexibility. In our intended scenario, we need to stage substantially faster and more feasible reactions to unforeseen events than getting back to the middleware design table. In other terms, middleware mustn't barely support context-awareness; rather, it must support *design and programming* for context-awareness. I discuss below an approach based on trust.

The idea of third-party resource points towards a model based on negotiation and protection of resource bounds, whereby owners and users dynamically agree on transient resource allocations. This latter is to be realised against resource pricing policies, whereby users agree to pay for their resource usage. As perfect knowledge is a rare commodity in global networks, such allocations will rely on risk assessment based on resource values and on the clients' trustworthiness. A server with a degree of trust in a client may be willing to lease it a resource at a certain price. Lower trust levels may mean higher prices, or refusal to interact altogether. Dually, a client must trust a server before entering negotiations with it, or using potentially harming resources received from it. Risk assessment may be based on a novel concept – yet to be discovered – of `program reputation,' that is a measure of code's past `good behaviour.' Reputation will change in time as a result of interactions as observed by the digital community, and so will systems' trust assessment and, thus, decisions. This gives rise to a number of unresolved issues:

- How do resource owners specify their resource usage policies?
- How do host and client code interact to determine resource requirements and bounds?
- What safety and robustness provisions can be made for both hosts and clients?
- How can the host trust the guest code not to abuse the resources made available?
- What is an appropriate pricing model for resource usage and how does cost negotiation take place?

None of these questions are, as yet, resolved.

I advocate an approach to context-awareness based on trust and reputation management. Systems will explore contexts as resource providers, by relying on a notion of context reputation. Reacting to contexts' changes, upon trust evaluation, systems will carry out self-inflicted actions dealing with configuration, healing, maintence, runtime extension. Such `autonomic' behaviour addresses scalability in a novel way, as a risk-assessed reaction to context variations.

These challenges make the tasks of software design, implementation, validation and maintenance even more troublesome than usual. They require the acquisition of a suitably general conceptual understanding of infrastructures, computational models, and language mechanisms, and breakthroughs are needed at all levels. There should be core computational models which identify flexible and robust notions of context and trust. Foremost, we need to develop a model of trust suitable for global systems based on third-party computation, as those envisaged here. We also need a computationally feasible notion of reputation. For instance, the reputation of entity E could be (an approximation of) the average trust in E over a local fragment of the global network. We then need to identify abstractions suitable to design programming languages, together with development tools to reduce the frequency of bugs, and tools for both qualitative and quantitative analysis. Solid theoretical foundations should ensure safety and security properties. Infrastructure and virtual machine technologies which support resource usage monitoring, global trust evaluation, pricing and negotiation need to be built. Compilation techniques targeted at resource conscious architectures should be investigated. Novel static analysis techniques to alleviate the execution cost of runtime monitoring should be employed.