Engineering the computational economy

Michael Wooldridge

Department of Computer Science, University of Liverpool, Liverpool L69 7ZF, UK
M.J.Wooldridge@csc.liv.ac.uk

Abstract. If grid and cluster computing is to achieve its potential, then we will require techniques for designing and implementing distributed systems containing potentially billions of semi-autonomous, active software components, designed and built by different organisations and individuals using different hardware and software platforms. This is the domain of software agents, and agent-oriented software engineering. The purpose of this presentation is first to summarise how and why software agents are an appropriate solution for the problem of engineering these systems, and second to sketch out the main scientific and technical challenges that must be overcome if this vision is to be realised.

Agent Systems: What and Why

Software engineers are by now used to conceptualising software systems as collections of interacting but essentially passive objects. The technology of object-oriented programming makes it possible to implement systems directly in terms of these object-based conceptualisations. But the success of the object paradigm does not stem solely from this technology. The object paradigm is so successful at least in part because we find it easy to conceptualise the world in terms of objects. But just as we find it easy to conceptualise and then design many systems in terms of interacting but passive objects, so many other systems can naturally be viewed as consisting of interacting, *active* components, which have become known as *agents*. The field of *multi-agent systems* addresses itself to the issues of how collections of such purposeful, active software components can be designed so that they can work together, typically in order to achieve goals that are beyond the capabilities of any individual agent.

Multi-agent systems differ from "regular" distributed systems in several important ways, but perhaps most important of all is that it is typically assumed that multi-agent systems are *open*, in the sense that:

- The components of the system are not known at design time – system components may enter and leave at run-time, in unpredictable ways;
- It is not possible for any system components to have complete, accurate, up-to-date information about the state of the system – agents have a *partial* view of the system;
- The components of the system were designed and implemented by different individuals and organisations using different hardware and software platforms, potentially with different goals in mind;
- The structure of the system is not predetermined, but emerges as a result of interaction and self-organisation on behalf of the system.

I emphasise that a central component of the multi-agent systems view is that agents are assumed to correspond to *loci of self-interest*. By way of an analogy, consider a regular economy, such as that of an industrialised nation. The economy is generally designed/engineered by the government of the nation in order to achieve certain goals – freedom and prosperity for the individuals that make up the economy. But these individuals themselves are not simply altruistic, benevolent people. They are self-interested, in that they wish to do as well for themselves as they can – given the constraints of the laws and rules imposed by the society. In multi-agent systems, it is assumed that the computational components of the artificial society have exactly such properties: they represent the interests of individuals, organisations, lobby groups, governments, and any other kind of entity that can be regarded as being a locus of self-interest, and they will act in such a way as to maximise their self-interest, insofar as the laws and rules of the artificial society permit.

Grid Computing

In *grid computing*, the idea is to build an infrastructure that will make distributed computational resources available as easily as electric power is through the electricity distribution grid. Part of the original motivation for grid computing came from the problems in processing scientific data, where the use of dedicated supercomputers is expensive and frequently infeasible. Large networks of much cheaper and less powerful processors have long been touted as a natural alternative to such dedicated devices, but there has never been a technology capable of exploiting such distributed computational resources. The aim of grid computing is to provide such technologies: grid computing will enable users to buy, sell, and barter computational resources in the same way that goods and services are bought and sold in the "real-world" economy. Notice that the "plumbing" for grid computing is essentially in place: we already have large-scale networks of distributed computers, connected by a (comparatively) reliable networks using data communication protocols (TCP/IP etc) that are commonly agreed and widely used. The challenges in grid computing therefore lie in developing the software to drive the grid.

Agents for Grid Computing

The problems faced by those wishing to develop grids are exactly those faced by agent community. The goal in both cases is to develop technologies and tools that will enable distributed software components to interact so as to solve problems that are beyond the capabilities of individual software components. Several commentators have therefore suggested that multi-agent systems are a natural paradigm for grid computing. The idea is to create an *agent-based computational economy*, in which software agents buy and sell computational goods and services on behalf of their "owners".

Challenges and Open Problems

In order to make the technology of multi-agent systems work for grid computing, there are several key problems that must be overcome. In the remainder of this paper, I sketch out some of these challenges (note that research has been underway into many of these challenges within the multi-agent systems community for some time):

- *Describing and reasoning about services.*

  A software component that has some computational (or indeed other task) that it wishes to sub-contract to another software component must have some way of describing both the task itself, and the parameters (quality of service etc) within which this task is to be carried out. The simplest way of describing a service is to define it by a method name and a list of parameter types (cf. the "reflection" API in the Java programming language). Another way is to define a computational task by giving an executable representation of it – you give the sub-contractor the program to execute. Still another is to define the task by describing the purpose of the task – setting out a goal to be achieved. All of these approaches have their advantages and disadvantages, but techniques developed to date have limitations. We need standard ontologies for describing services and computational resources with well-defined semantics, and tools for making use of these ontologies.

- *Brokering services.*

  Brokers are services that act as mediators between service providers and service consumers. They put those in need of particular services with certain parameters in touch with those who can provide such services. Broker-style architectures have been used for some time, e.g., in the distributed object world, but we need a better understanding of how computational brokers can be designed to operate in large scale agent systems, where agents require services with complex parameters.

- *Reaching agreements.*

  Agents that are owned by different organisations, with different interests, that must work together in the same society, must be capable of dynamically reaching agreements on matters of common interest. We therefore need algorithms that will enable agents to *negotiate* with one-another in order to reach agreements. Negotiation typically is a process of agents putting forward offers and counter offers, making concessions or constructing alternative offers, until either agreement is reached or else negotiation fails with no agreement being reached. In order for users to delegate negotiation tasks to agents, they will need to *trust* them. That is, they will need to be sure that any agreement to be reached will be fair; that their agent cannot be exploited, that the negotiation process cannot be subverted through some devious means, and so on. In other words, we need protocols for negotiation that can be *proven* to have these desirable properties. In economics, this area is known as *mechanism design*; we need a kind of computational mechanism design for agent systems.

- *Scalability*.

  Perhaps the key challenge facing the agent-based grid community is that of scalability. Scalability has two aspects here. The first is that of engineering protocols and systems that can scale to potentially billions of system components. This is largely an engineering challenge, and we already have an existence proof that such networks are possible: the Internet itself. The second problem, however, relates to the dynamics of computational economies. Human economies are notoriously unpredictable in their behaviour, and there is no reason to believe that computational economies will be any different. Indeed, computational economies may well be *more* chaotic and unpredictable than human economies. For example, in the catastrophic stock market crash of October 1987, it was believed that one factor which greatly increased the severity of the crash was that stock trading programs were designed to sell shares when prices fell; this created a feedback effect, where trading programs sold because prices were low, which in turn pushed prices lower, and so on. In order to prevent such situations arising in agent-based computational grids, we need ways of modelling and understanding large scale computational economies, as well as managing and predicting their behaviour.

For More Information…

For more information about agents and the issues discussed in this short article, see:

`http://www.csc/liv.ac.uk/~mjw/`