Guest Editors’ Introduction

Dependable Agent Systems

Mark Greaves, DARPA
Victoria Stavridou-Coleman, Intel
Robert Laddaga, MIT Computer Science and Artificial Intelligence Laboratory

It is well known that building dependable software systems for dynamic environments is difficult. It is also well known that building large-scale distributed software systems is difficult. The relatively few attempts to combine these two tasks confirm that successfully building large-scale distributed systems with predictable dependability properties is exceptionally difficult. The articles in this special issue of IEEE Intelligent Systems deal with this issue and discuss an emerging and exciting new approach to building these most challenging kinds of systems.

Three main sources of change have converged to make the topic of dependable distributed systems of great current interest: the growing demand for dependability, a revolutionary change in our models of computation, and the appearance of a potential solution.

The growing demand for dependability

Making large-scale complex systems maximally dependable is not just an academic exercise. These systems are becoming far more prevalent, and we increasingly rely on them. Dependability is a core property that comprises the set of system properties that assures us that the system will behave according to its requirements. When systems are designed to withstand actively malicious behavior, we sometimes call them survivable to emphasize that they must be dependable even when under directed attack.

There is intense customer pull from some of the world’s most sophisticated organizations to successfully field dependable systems. Dependable IT systems are currently deployed in critical application areas such as defense, transportation, energy, process control, and finance, and systems for such areas are carefully engineered to operate in a well-specified envelope. Unfortunately, these systems tend to be logically monolithic and tremendously expensive. Furthermore, they anchor their dependability arguments in fixed assumptions about the relevant organizational structure, workflow rates and quantities, and availability of data and infrastructure resources.

In contrast, both modern business models and transformational military systems draw much of their competitive advantage from a more dynamic vision of global-scale, flexible, adaptive, fully distributed, totally...
integrated IT. This implies that the IT systems must be adaptive, agile, and survivable. Therefore, the systems must dependably support highly dynamic processes and coordination models whose parameters cannot all be known in advance, much less reduced to the precise language of software requirements and testing documents. The military refers to such IT systems as dynamic systems of systems to emphasize their changing and compositional nature.

In concrete terms, the rising demand for these systems means that computer scientists will need to design and implement highly agile constellations containing both new and existing information systems. Additionally, the reliability of these constellations in a shifting and sometimes hostile environment will be critically important. The software world is moving away from individual stovepipe applications (vertically integrated applications with support layers not shared or amortized over a family of applications). In their place, the software world is building custom-tuned networks of applications, and eventually it will achieve full standards-based peer-to-peer computing. Consequently, there will be a steadily increasing demand for distributed software systems with substantial scale and performance requirements, demanding reliability specifications, and critical security prerequisites. Given this clear customer demand, what prevents us from building such systems as a routine matter?

The problem is that a mismatch exists between the distributed, dynamic problem with which we are presented and the techniques that computer science can bring to bear. The traditional methods of building dependable systems rely on rigorous top-down design using conservative principles that allow extensive validation throughout the system life cycle. For example, this method assumes you have unified architectural-design and validation methods, high-confidence operating systems, well-tried and reliable (albeit often slow) hardware, and carefully engineered performance and resource requirements that allow for appropriate design-time trade-offs. It also assumes the rigorous use of conservative programming practices, such as not allocating memory from the heap, to ensure that the computation will stay predictably in the bounds of what the hardware can support.

Each of these techniques (and many others) brings some amount of formal support to the system’s overall dependability case. The dependability case then brings together all the analytical and empirical evidence concerning the system’s future behavior and logically supports the conclusion that the system will exhibit the target dependability properties as long as it operates within the specified environmental envelope. After all, it is not sufficient for a system to be dependable—it also needs an accompanying dependability case so that we are convinced it is dependable.

Unfortunately, the massive installed base of existing software systems means that for dynamic systems of systems, building an adequate dependability case in the usual manner is often impossible. Many distributed systems will be built out of existing hardware and software components of relatively unknown provenance, making the established techniques for assessing and asserting dependability properties difficult or impossible to employ.

So, which techniques should we use in their place to create systems that are maximally dependable given their environment and to build an accompanying dependability case in a way that is convincing and logically sound? How can we produce a web of software that is sufficiently dependable in the dynamic environment in which it must operate, that is largely built from our current systems, and about which we can gather a sufficient amount of structured dependability evidence?

A new model of computation

The second reason for examining new ideas relates to the first. New applications and systems have caused computer scientists to radically revise their views, concerns, and models. For example, nontermination is not an error for an embedded system but the preferred result. A major single example of the kind of application that led to this transformation in world view is the World Wide Web.

The Web has achieved the major precondition for building large-scale, dependable distributed systems—cost-effectively connecting all the computing systems we care about. In turn, this connectivity has stimulated the creation of the first generation of Web-oriented software technologies such as Web Services and service-oriented architectures. The success of these technologies has let an unprecedented number of practicing software architects experiment with dynamically composing real software systems. The Web has made it easier than ever for developers to conceive, market, design, and build fully distributed IT systems that weave together previously independent applications.

More interestingly, this experience of thousands of programmers using the Web to build distributed systems seems to have encouraged a subtle evolution in how we think about computation. A data-driven view now complements the traditional academic models of computation that involve carefully sequenced abstract processes that terminate with a single algorithmic result. In the data-driven view, execution is embedded in a dynamic resource environment, wall-clock performance matters, some percentage of service requests will fail, and the underlying system parameters are at best known probabilistically. Even the core vocabulary we use to describe our discipline has started to expand as we move toward stream and reactive computing:

• Control surface augments algorithm
• Evolutionary stability and lifecycle invariants augment termination conditions
• Partial functions augment total functions
• Stochastic optimization augments determinism

This is an exciting development, but specifying requirements for distributed software systems in control-theoretic terms such as control surface and stability makes it even more difficult to design and build these systems. It also makes it more difficult to understand how to marshal the evidence that the system’s dependability case requires.

Are agents the answer?

Fortunately, an exciting new approach exists for building such systems. We believe that we can use results and techniques from software agent technology to create distributed systems that reason about and dynamically alter their own configurations to maximize their overall dependability.
The software agent community has spent two decades building distributed-agent societies that jointly and flexibly carry out their function in an evolving runtime environment.

Agent societies are typically composed of a set of custom agent-implemented capabilities and a set of agent proxies for existing applications. They employ semantically sophisticated interaction protocols as the flexible control layer that binds the pieces together into a reliable system of systems. Many of agent programming’s main operational characteristics reflect promising computational approaches to building the kinds of agile large-scale distributed dependable systems we seek. Such characteristics include using high-level semantic messaging, goal-oriented negotiation and planning, BDI (belief, desire, intention)-style agency models, task-organized workflow networks, process mobility, and flexible late-binding service invocation methods. Furthermore, the strict attention to semantics that characterizes agent theory promises a straightforward logical linkage to the formal reasoning required in a dependability case.

If we could show that agent-oriented architectures support naturally flexible and survivable design patterns, then we could address the dependability of distributed-systems issues using agent programming’s powerful techniques. First, using software agents to build large-scale distributed systems could help developers describe and implement distributed systems with the relevant dependability properties. Second, a distributed agent system could simultaneously exhibit the desirable agility property of an agent system along with a set of demonstrable dependability properties. Solving two problems would mean that agents could help us build large-scale distributed systems with predictable dependability properties. Most radically, then, dependable systems could be the elusive killer app for software agents.

Problem areas

The case for using agent middleware to achieve dependability properties in distributed adaptive systems is not straightforward, though. For example, the intrinsically nondeterministic and evolving agent computational paradigm might be incompatible with the predictability required to deploy an IT system in critical applications. On the other hand, the adaptive nature of agent societies might supply enough diversity and redundancy to satisfy a set of demanding dependability requirements.

Using agent middleware requires that we address four important, unsolved issues.

Emergent properties

A good deal of rhetoric exists in the agent community about the virtues of emergent properties and their ability to compensate for the inability to write complete software specifications in advance. The problem is that the most common emergent properties in agent systems are race conditions and deadlocks, so the property that emerges is often a system failure.

Controlling a large-scale agent system so that only desirable properties emerge from the nondeterministic interactions of hundreds or thousands of agents involves a thorough understanding of total-system queuing and stability properties—generally not a concern of agent researchers. We do not yet know whether the concept of emergent properties is fundamentally at odds with the requirements for dependable software.

Dynamic workflows with predictable quality of service

The agent community has been extremely active in developing service description languages, matchmaking and proxy algorithms, negotiation frameworks, discovery systems, and the like. The most advanced agent societies manage their task flows using online continuous-planning algorithms that continuously shape workflows to the available resources. In this way, agent societies unify the semantics of agent communication languages, domain tasks, and service components. However, much work remains on predicting and managing quality of service and overall performance guarantees in agent systems.

The evidence for an agent dependability case

Even if we could build a dependable agent system, could we produce a convincing evidence-based argument to that effect? We’re only starting to understand the formal architectural properties of different agent systems, and agent software development methodologies are still in their childhood. The software construction artifacts, testing methodologies, and design principles necessary to support a logically satisfactory dependability case for agent software remain unknown.

The maturity of agent technology

An honest appraisal of agent technology reveals that it remains a risky software application framework. Despite the work of the Foundation for Intelligent Physical Agents (www.fipa.org) and other standards organizations, there remains no community agreement on software frameworks, languages, ontologies, or internal models of agency. Serious agent developer tools do not exist. Semantic peer-to-peer design expertise is specialized and rare. And, in an unsettling echo of AI’s situation during the 1970s and 80s, the marketplace rhetoric surrounding agents is clearly becoming overblown.

In this issue

The articles in this special issue explore the interplay between the promises of flexible, agent-based computation and the desire for dependability guarantees. They address several different aspects of using software agent technology as a foundation to build dependable large-scale distributed software systems.

Two articles, “Model Checking Rational Agents” and “Using Model Checking to Assess the Dependability of Agent-Based Systems,” address issues in how to model and validate different properties of distributed-agent systems. The article “Extending the Limits of DMAS Survivability: The UltraLog Project” discusses a unified approach to building survivable agent systems that encompasses many of the artifacts necessary for building a dependability argument.

Finally, “Survivability of Multiagent-Based Supply Networks: A Topological Perspective” addresses the topological structure of agent interaction networks, shows that certain ones exhibit impressive survivability properties, and, most importantly, provides a tool for predicting their survivability properties.

This is a singular time for dependable distributed systems. Commercially, there is significant and growing demand for these kinds of system and rewards for the organizations that can build them. At the same time, the traditional models we use to understand the relationship between a computational process and its environment are changing from the standard deterministic, input-output-based archetype into ones that are more continuous, embedded, and dynamic—that is, models that are more closely aligned with the practices of reactive systems and agent-oriented computing. Finally, the long adolescence of agent technology and distributed AI has given the intelligent systems community the theoretical framework and tools needed to start building these systems with an understanding of their eventual dependability properties. We hope that the articles in this issue will help to spark your interest in this enormously promising area.

Acknowledgments

The views expressed in this article are strictly those of the authors. They do not represent the positions of DARPA or the US government.

Join the IEEE Computer Society online at computer.org/join/

Complete the online application and

• Take Web-based training courses in technical areas for free
• Receive substantial discounts for our software development professional certification program
• Get immediate online access to Computer
• Subscribe to our new publication, IEEE Security & Privacy, or any of our 22 periodicals at discounted rates
• Attend leading conferences at member prices
• Sign up for a free e-mail alias— you@computer.org

THE WORLD’S COMPUTER SOCIETY