Software Directions for Network Centric Distributed Computing Systems

Dr. Richard E. Schantz
Dr. Joseph P. Loyall
{Schantz, Jloyall}@bbn.com

BBN Technologies
Cambridge, Massachusetts

October 31, 2001

Abstract: We are moving rapidly toward a world in which the dominant computer architecture is that of network centric systems of systems, with widely distributed embedded and non-embedded components. We envision a simultaneous evolution of software environments and techniques over the next decade to better support the development of these network centric systems. This evolution will be from the current process of programming relatively static systems with centralized control to the construction of flexible systems from off-the-shelf parts – systems with decentralized control, awareness of the environments in which they are embedded, and the ability to automatically adapt and make the resource management tradeoffs appropriate to changes in those environments. These systems will be longer-lived and more robust than those enabled by today's program from scratch, ad hoc distributed systems. The dominant technical barrier is the development of a new computational model incorporating resource aware and managed tradeoff behavior in a highly networked environment constructed through composition techniques.

Submitted to the Workshop on New Visions for Software Design and Productivity: Research and Applications

Software Directions for Network Centric Distributed Computing Systems

1. Background and Overview

We've achieved remarkable technical progress over the last 20 years, but in reality we've only set the stage for what needs to come next. Advances in computing and communication and the software engines that drive both of these, as well as their integration, have gotten us to the point where we can embed computing power into just about anything and everything. Increasingly, everything is being connected through one medium or another to almost every other thing, and systems encompass more and more of these interconnected computational devices interacting with intelligent (computational) behavior within the physical world we populate. Such is the background for network centric distributed systems and the software which organizes, controls, integrates and provides useful abstractions for programming the increasing larger scale, miniaturization, and increasing interconnectivity of the systems that we as a society need to construct over the coming decades. Our ability to conceive such systems far outstrips our ability to construct such systems, due in large part to the limitations of the software base from which we start.

Over the past 10 years we have evolved our "standard" architectural frameworks for constructing network centric systems to include middleware components whose main purpose is to address the issues faced in multi-machine use and integration. The challenge before us is extending the middleware concept to the implications of new physical contexts and sizes of modern and emerging environments for system building. Key challenging characteristics of these environments include the following:

1. Recognizing change and volatility in the environment as the rule, not the exception
2. Scaling way up, without the need to reprogram the participating pieces
3. Scaling way down, without the need to reinvent or reengineer the contexts for developing the small scale pieces
4. Environments which simultaneously include and integrate elements with both significantly more and significantly less computational and communications capability, and a wider spread between the extremes
5. Larger software systems, made from smaller components, with higher user expectations for a more controllable and consistent "user" experience
6. Systems which require a much higher degree of "always on, always available" behavior, owing to their use in critical and high risk/high liability situations, and as a much more routine everyday part of everyone's life experience.

Projecting forward 10 years, there will still be a need to "program" these systems, in one form or another, to do what is required of them. The age of automatic programming in these contexts based on high level requirements is still likely beyond our grasp for the next generation. However, that "programming" must result in systems that are more sensitive to the environments in which they operate, be no more complex to write than current programs of roughly comparable functionality, and be of generally a much higher quality. To do this will require much more capable and higher level programming techniques and environments which "take care of" the complexities of the volatile, physical environment, as well as methodologies which are oriented toward composing large systems, not programming them as a whole.

2. Taking Stock of Current Software Systems

With progress to date, mechanisms for connecting distributed parts together, despite heterogeneity, is greatly simplified and largely off-the-shelf. What is not so easy is building end-to-end systems with predictable and manageable properties from these parts. Considerable effort has gone into formalized methodologies for certain types of interactions, while others remain in their infancy. In particular, connectivity involving 1-to-1 and 1-to-many interactions are well understood and part of what is being fielded today. On the other end of the spectrum, concepts for systems organized around many-to-1 are less well developed, while organization involving many-to-many interactions are largely ad hoc or non-existent. Similarly, despite physical and geographic separation, almost all of our system activities eventually gravitate toward centralized control schemes because they are easier to build and understand and are routinely deployed. This leads to additional limitations, such as scalability and security, because of the single control bottleneck and the single point of failure. In order to take full advantage of the distributed nature of many problems, we need to get equally facile in understanding and easily deploying fully distributed control strategies as well.

To date, we've been content with constructing relatively static systems, i.e., systems whose behavior is determined largely at design time and remains so for its lifetime. The proliferation of operating contexts and the individuality of user preferences forces us to consider much more dynamic system structures. Instead of one-size fits all, we need to be enabling the flexibility to permit systems to automatically change and adapt to conditions, instead of placing that burden for adaptation exclusively on the users and maintainers of the system, as it is currently. In order to do this, systems will need to be able to access and understand their own characteristics, and tailor their behavior to the precise situation prevailing at that time. These adaptive and reflective architectures hold the promise to move the locus of responsibility for orchestrating correct behavior under varying networked conditions from outside the system (the "user") to within the system (the "developer"), greatly simplifying the user experience. They also hold the promise of being able to adapt much more rapidly and in a much greater system-wide coordinated manner, providing more stable behavior under a variety of conditions, a more consistent user experience, and more robust, longer-lived systems.

3. **Moving Forward**

In order to go down these paths, there needs to be significant change in the way we currently develop software. Our concepts, methodologies and strategies for developing network centric systems have evolved directly from the methods for constructing non-network centric systems. But aspects of the operating context for network centric systems are much more dynamic, with larger scale at one end, and smaller scale at the other, and serve to integrate these extremes and all the points in between under a single system umbrella. These mandate sweeping changes in the way we develop software for the next generation. We are headed for an era in which network centric methodologies become THE standard way to develop software and are practiced by all software developers. Because in the end, all software developed will eventually find itself part of an integrated distributed environment, whether by design or by legacy wrapping to include it, and incurring the necessity to have the context in which it is embedded partially dictate the proper behavior of the software component. We are moving away from a current computer-centric culture of “best effort” resource management, in which we formally link parts and then fix shortfalls in derived system behavior, and toward a human-centric and unattended embedding culture, in which systems themselves are responsible for adapting to provide a consistent user experience over whatever resources are available for the task. In this model, we reverse the traditional order, by dealing with system constraints first, with the parts filled in accordingly and varying with time.

In order to do this in an organized, reproducible and trainable way, we need new computational models that provide directly for resource conscious programming for attaining properties such as real-time interactions, dependability, safety, energy consumption and footprint, to name a few. Also, because each new environment and the prevailing conditions under which they operate are constantly changing, this computational model needs to routinely reevaluate the engineering tradeoffs that must be made to make systems work properly, effectively, and efficiently. This is the true promise of “write once, run anywhere,” provided these sorts of tradeoffs are managed as part of the development process and runtime support.

To complement these new computational models, the very nature of the network centric viewpoint encourages a system of systems approach, where composition of flexibly developed components and subsystems are the rule, not the exception. In order to do this we need a complete overhaul of our ideas about how components are glued together and the influence of the system context on the behavior of the components. Furthermore, our concepts about system scale and lifetime will be extended beyond our current models of build it once and run it for a fixed duration. Systems will continue to grow and grow, with the expectation that for some, they will be turned on and never actually completely shutdown. In essence, they run forever, with “live” update and expansion alongside continued productive response, and adaptive changes to the appearance of new subsystems coming on line and to subsequent changes in environmental conditions.

Conclusion

We are moving rapidly toward a world in which the dominant computer architecture is that of network centric systems of systems, with widely distributed embedded and non-embedded components. We envision a simultaneous evolution of software environments and techniques over the next decade to better support the development of these network centric systems. This evolution will be from the current process of programming relatively static systems with centralized control to the construction of flexible systems from off-the-shelf parts – systems with decentralized control, awareness of the environments in which they are embedded, and the ability to automatically adapt and make the resource management tradeoffs appropriate to changes in those environments. These systems will be longer-lived and more robust than those enabled by today's program from scratch, ad hoc distributed systems. The dominant technical barrier is the development of a new computational model incorporating resource aware and managed tradeoff behavior in a highly networked environment constructed through composition techniques.