J Internet Serv Appl
DOI 10.1007/s13174-011-0051-x

SI: FOME - THE FUTURE OF MIDDLEWARE

R&D challenges and solutions for highly complex distributed systems: a middleware perspective

Jules White · Brian Dougherty · Richard Schantz · Douglas C. Schmidt · Adam Porter · Angelo Corsaro

Received: 9 November 2011 / Accepted: 15 November 2011

Abstract Highly complex distributed systems (HCDSs) are characterized by a large number of mission-critical, heterogeneous inter-dependent subsystems executing concurrently with diverse—often conflicting—quality-of-service (QoS) requirements. Creating, integrating, and assuring these properties in HCDSs is hard and expecting application developers to perform these activities without significant support is unrealistic. As a result, the computing and communication foundation for HCDSs is increasingly based on middleware. This article examines key R&D challenges that impede the ability of researchers and developers to manage HCDS software complexity. For each challenge that must be addressed to support HCDSs, the article surveys the state-of-the-art middleware solutions to these challenges and describes open issues and promising future research directions.

J. White · B. Dougherty
Virginia Tech, Blacksburg, VA, USA

J. White
e-mail: julesw@vt.edu

B. Dougherty
e-mail: brianpd@vt.edu

R. Schantz
BBN Technologies, Cambridge, MA, USA
e-mail: schantz@bbn.com

D.C. Schmidt (✉)
Vanderbilt University, Nashville, USA
e-mail: dschmidt@dre.vanderbilt.edu

A. Porter
University of Maryland, College Park, MD, USA
e-mail: aporter@cs.umd.edu

A. Corsaro
PrismTech Corp, Paris, France
e-mail: angelo.corsaro@prismtech.com

Keywords Highly complex distributed systems · Middleware · Quality of service

1 Introduction

Highly complex distributed systems (HCDSs) include air traffic control and management, electrical power grid management systems, large-scale *supervisory control and data acquisition* (SCADA), telecommunications, and integrated health care delivery, as well as many other distributed real-time and embedded systems—especially those with cyber-physical world interactions. These types of systems are characterized by a large number of mission-critical, heterogeneous interdependent subsystems executing concurrently with diverse—often conflicting—*quality-of-service* (QoS) requirements.

Developers of HCDSs must manage a number of important—often conflicting—system properties, including reliability, predictability, security, and others, as summarized in Fig. 1 [1].

Creating, integrating, and assuring these properties in HCDSs is hard and expecting application developers to perform these activities without significant support is unrealistic. As a result, the computing and communication foundation for HCDSs is increasingly based on *middleware*. In the context of HCDSs, middleware is infrastructure software residing between applications and the underlying operating systems, networks, and hardware to provide a platform that abstracts away heterogeneity, provides control over key QoS properties, and facilitates the construction and management of HCDSs.

This article examines key R&D challenge areas that impede the ability of researchers and developers to manage HCDS software complexity. Section 2 explores each challenge area and describes how middleware—and associated

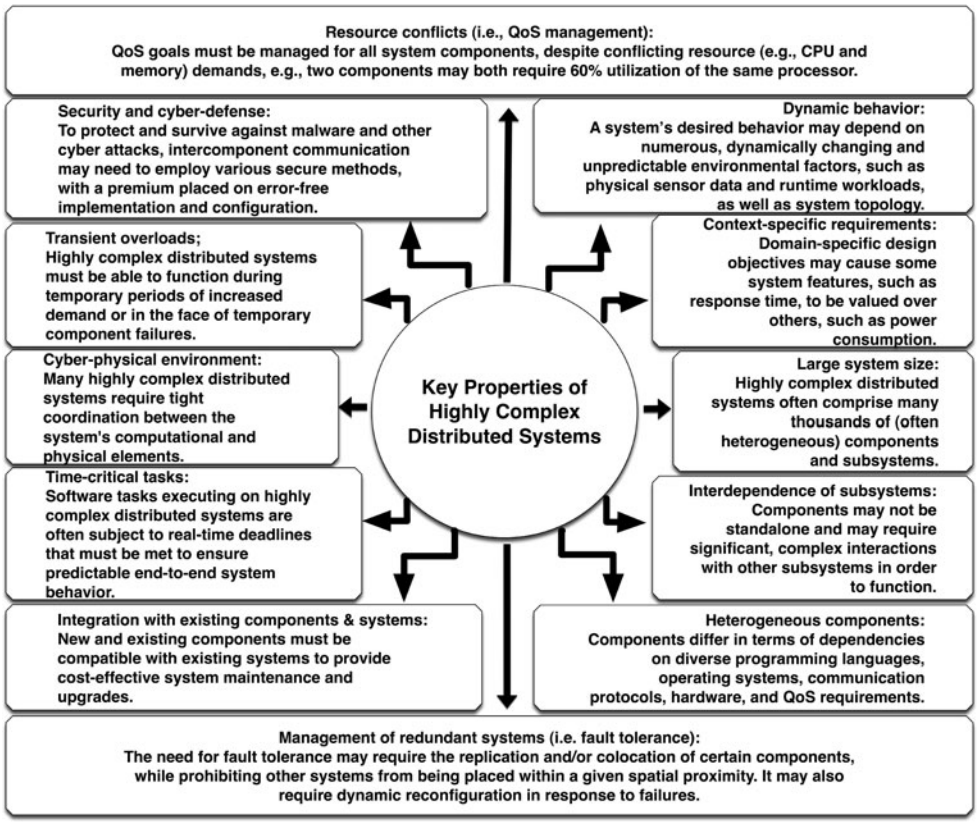
Resource conflicts (i.e., QoS management):
QoS goals must be managed for all system components, despite conflicting resource (e.g., CPU and memory) demands, e.g., two components may both require 60% utilization of the same processor.
Security and cyber-defense:
To protect and survive against malware and other cyber attacks, intercomponent communication may need to employ various secure methods, with a premium placed on error-free implementation and configuration.
Dynamic behavior:
A system's desired behavior may depend on numerous, dynamically changing and unpredictable environmental factors, such as physical sensor data and runtime workloads, as well as system topology.
Transient overloads;
Highly complex distributed systems must be able to function during temporary periods of increased demand or in the face of temporary component failures.
Context-specific requirements:
Domain-specific design objectives may cause some system features, such as response time, to be valued over others, such as power consumption.
Cyber-physical environment:
Many highly complex distributed systems require tight coordination between the system's computational and physical elements.
Key Properties of Highly Complex Distributed Systems
Large system size:
Highly complex distributed systems often comprise many thousands of (often heterogeneous) components and subsystems.
Time-critical tasks:
Software tasks executing on highly complex distributed systems are often subject to real-time deadlines that must be met to ensure predictable end-to-end system behavior.
Interdependence of subsystems:
Components may not be standalone and may require significant, complex interactions with other subsystems in order to function.
Integration with existing components & systems:
New and existing components must be compatible with existing systems to provide cost-effective system maintenance and upgrades.
Heterogeneous components:
Components differ in terms of dependencies on diverse programming languages, operating systems, communication protocols, hardware, and QoS requirements.
Management of redundant systems (i.e. fault tolerance):
The need for fault tolerance may require the replication and/or colocation of certain components, while prohibiting other systems from being placed within a given spatial proximity. It may also require dynamic reconfiguration in response to failures.

Fig. 1 Key properties of highly complex distributed systems

methods and tools—can effectively assure some aspect(s) of the HCDS properties shown in Fig. 1. For each challenge that must be addressed to support HCDSs, we briefly survey the state-of-the-art middleware solutions to these challenges, and outline some open issues and future research directions. Section 3 then focuses on the new research directions needed to manage the fundamental underlying complexity issue: the effective integration and tradeoff management of all of the challenge areas simultaneously for next-generation HCDS requirements.

2 R&D challenges and promising middleware solutions for highly complex distributed systems

Researchers and developers of middleware for current and next-generation HCDSs must address the following challenges to design, implement, and operate these systems, as shown in Fig. 2.

2.1 Challenge 1: encapsulating heterogeneity at scale

Context HCDSs often run on a variety of processor, operating system, and middleware platforms that are interconnected by different types of protocols and networking technologies. Each layer has varying constraints on QoS properties. For example, next-generation SCADA systems (such as those used to control smart grid production and large metro subway infrastructure) need to integrate a wide range of devices, ranging from traditional microcontrollers, to the latest generation of smartphone and tablet devices while dealing with varying protocols network and bus interconnects, software platforms, and processor types and capabilities. Likewise, server infrastructure supporting these systems may use utility computing systems comprised of a large number of virtualized, distributed, compute-intensive, and storage-intensive elements.

State of the art Prior work has explored techniques for developing and configuring HCDSs with thousands of heterogeneous components. Mello Schnorr et al. [2] present visualization techniques for detecting resource usage anomalies between heterogeneous components in large scale distributed systems. Albrecht et al. [3] present a framework for managing distributed applications across a variety of networks and computing environments. These techniques provide management at scale of the myriad of components that comprise HCDSs.

Open problems Deployment decisions, such as which processor to run a software component on, become hard in heterogeneous HCDSs due to the myriad of physical resources. Configuration of software and hardware components is also hard due to the need to simultaneously fine tune the heterogeneous hardware and software resources both locally and

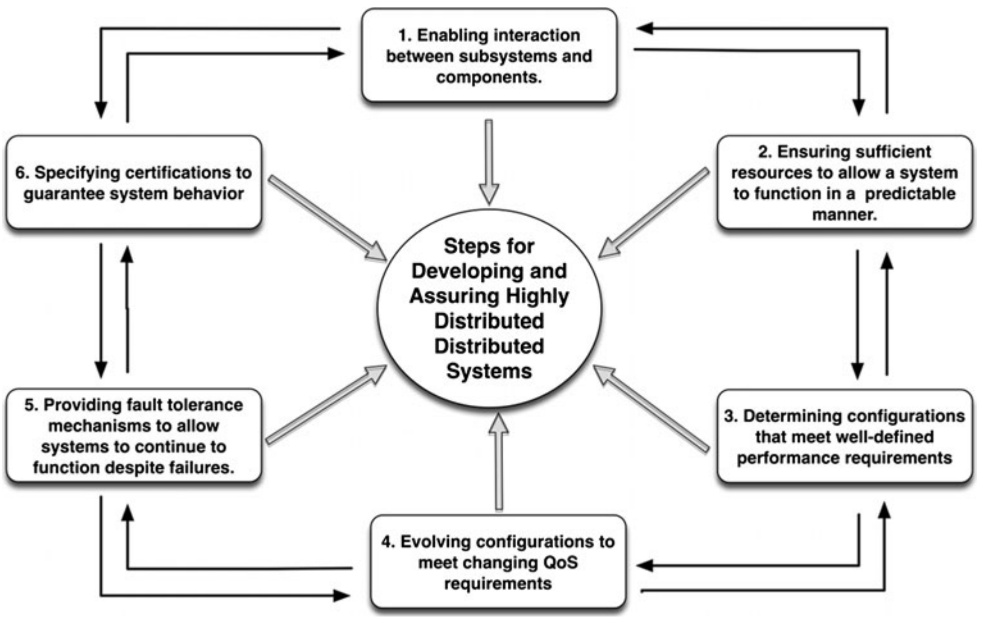
1. Enabling interaction between subsystems and components.
2. Ensuring sufficient resources to allow a system to function in a predictable manner.
3. Determining configurations that meet well-defined performance requirements
4. Evolving configurations to meet changing QoS requirements
5. Providing fault tolerance mechanisms to allow systems to continue to function despite failures.
6. Specifying certifications to guarantee system behavior
Steps for Developing and Assuring Highly Distributed Distributed Systems

Fig. 2 Steps for developing and assuring highly distributed distributed systems

globally across thousands of components. Addressing the challenge of heterogeneity involves focusing both on portability and interoperability. As a result, middleware technologies need to evolve toward providing native interoperability and portability between and across applications. Due to the challenges posed by ultra-large-scale HCDSs [4], interoperability cannot be an afterthought since it would introduce intolerable single points of failure and performance bottlenecks.

Future research directions Next-generation middleware needs integrated configuration approaches, such as those based on *model-driven engineering* (MDE) [5], to assemble heterogeneous system components and optimize the connecting protocols and middleware to meet end-to-end system QoS goals. Manual approaches for crafting these interconnecting elements scale poorly nor lack the sophistication needed to meet HCDS requirements. As a result, automated configuration methods capable of scaling to the size and complexity of emerging HCDSs must be developed.

2.2 Challenge 2: deriving valid, high-performance configurations of highly configurable infrastructure platforms

Context Middleware platforms provide many options to configure HCDSs at compile- and/or run-time. Each set of configuration options can uniquely effect the consumption of available resources, such as memory, CPU cycles, power, system cost, and overall performance. For example, the *Data Distribution Service* [6] (DDS) is a standards-based middleware platform used heavily in many HCDS domains. DDS implementations support dynamic discovery facilities as a means of easily adding additional components as runtime binding, as well as having scores of QoS policies, options, and configuration parameters. Together, DDS's configurability makes the problem of deriving and maintaining appropriate and valid high-performance configurations doable, yet hard to validate.

State of the art Several tools are have shown promise for monitoring component interactions and deriving system configurations that satisfy the myriad of constraints of HCDSs. Albrecht et al. [3] provide an online monitoring infrastructure that allows system engineers to manage, monitor, and integrate heterogeneous components of highly complex distributed systems. Lee et al. [7] present an integrated modeling and specification framework that uses MDE tools to verify heterogeneous components at multiple levels of abstraction.

Open problems While flexibility promotes customization and can enhance QoS, it can also yield a combinatorial number of potential system configurations, each requiring extensive quality assurance. In addition to assuring quality, deployment challenges also arise. As computing platforms migrate away from traditional single-core computational resource models to multicore and massively distributed-core models, where and how resources are allocated to computing tasks plays a prominent role in ensuring end-to-end system QoS and can vastly impact system performance.

For example, distributing frequently communicating tasks across separate computing nodes may overwhelm the interconnection or network links connecting the two, drastically increasing latency. Deploying the same two tasks to the same node, however, may reduce network traffic but overload the processing node, thereby missing real-time deadlines and increasing execution time. Next-generation middleware platforms must optimize these deployment decisions to maximize system performance. Additional restrictions, such as financial and resource constraints, can also invalidate numerous potential configurations for a given usage scenario, which can make it hard to find any valid configuration, much less one that optimizes cost, reliability, and performance simultaneously.

Future research directions While certain configuration techniques, such as applying constraint solvers, are effective for configurations with a limited number of configuration options, the exponential number of potential configurations makes the use of configuration techniques that rely on exhaustive search prohibitively slow for large-scale HCDSs. Configuration techniques using approximation and sampling algorithms [8] have shown promise for determining valid, high-performance configurations in minimal time.

For example, Yilmaz et al. [9] use experimental design theory to select a small, but effective set of configurations to benchmark during performance evaluation. Fouche et al. [10] use a mathematical sampling techniques called covering arrays to select configurations. Westermann and Happe [11] combine model-driven architecture with partially-automated performance measurements to identify highly-performant configurations. Kappler et al. [12] use static analysis to create performance models for individual and integrate them to produce system models.

2.3 Challenge 3: supporting diverse QoS requirements

Context Different uses and application classes within HCDSs require multiple different, and sometimes conflicting QoS requirements. For example, streaming real-time video requires low jitter and predictable end-to-end timeliness properties, which must also scale to large numbers of devices on a network. In other distributed applications, such as air traffic control and management, a high degree of reliability as well as the persistence of some key information is paramount. Even infrequent, minor violations of QoS requirements in these domains can lead to catastrophic results. It is therefore critical that (often diverse) QoS requirements be met, despite disruption, and with appropriate tradeoffs.

State of the art Schantz et al. [13] present several approaches for evolving middleware to handle a wide range of QoS concerns, including real-time constraints, scalability, and security. Otte et al. [14] investigate methods for encapsulating QoS behaviors as components, thereby allowing QoS requirements to be met through component assembly and deployment. Multi-layered dynamic QoS management has been under investigation for some time, and numerous prototypes have been built and studied [15, 16]. This existing work demonstrates feasibility of the managed QoS approach. These approaches, however, have not yet achieved full operational status nor been scaled to open-ended, and widespread unforeseen anomalous conditions that may arise in large-scale HCDSs.

Open problems Next-generation middleware must satisfy the combination of stringent, system-specific QoS while providing high performance and ensuring secure confidential communication at scale. These challenges must be addressed holistically to overcome today's point solutions that address only some dimensions of the overall problem space. Deployment in these scenarios involves cost-effectively selecting resources that will meet the different QoS requirements specific to a given system objective.

For example, deployment solutions in power-constrained environments, such as building automation or disaster recovery, must ensure that resources are released automatically to conserve power as computational demands ease. The configuration problem must deal with determining the right set of parameters to choose for the platforms so that application-specific QoS requirements are met locally and globally.

Future research directions Addressing the challenge of diverse QoS requirements involves creating next-generation middleware that can support a wide range of QoS policies that can be cooperatively enforced at varying levels from the OS to the network to the middleware and to the application. Moreover, mechanisms are needed to reconcile and adjust policies [17] to handle conflicting QoS requirements at the system level, even as they arise at runtime.

2.4 Challenge 4: static configuration and dynamic reconfiguration

Context Due to the dynamic environments in which HCDSs operate, it is often necessary to redeploy and reconfigure these systems depending on availability of resources (some of which may have failed or become overloaded), and on the current demand on the system. In general, configuration decisions are made at multiple time scales, including (1) *compile time*, when the target platform is compiled for the use case in which it will be used, (2) *deployment time*, when applications are deployed on the target platforms, and (3) *run time*, when dynamic reconfiguration and redeployment decisions are made.

State of the art Static analysis and dynamic reconfiguration tools can enhance HCDS performance. Surajbali et al. [18] present an approach that uses aspect-oriented programming to develop dynamically reconfigurable middleware. Otte et al. [19] describe a dynamic reconfiguration technique aimed at increasing the performance of component-based middleware. Atighetchi and Pal [20] use redundant configuration and dynamic reconfiguration techniques to accomplish degrees of survivability under cyber-attack.

Open problems Manually selecting HCDS design and configuration options is hard due to the myriad of QoS constraints and a vast selection of available components offering a range of functionality and differing resource requirements. This problem is exacerbated when extended from static or infrequent configuration to systems needing frequent dynamic reconfiguration, such as systems augmented with cyber defense capabilities. These systems require rapid dynamic reconfiguration in response to the detection or even anticipation of cyber threats, malware, or system failures to remove/replace tainted components and mitigate the effects of cyber attacks.

Future research directions MDE tools can be used to capture the myriad of available configuration options for various system components, such as operating systems and middleware, and to facilitate the deployment and configuration at compile and deployment time. These tools allow designers to track the impact of including a given component in a design configuration on overall system cost, performance, and QoS. Configuration derivation techniques using approximation algorithms and meta-heuristic techniques have shown promise for rapidly determining valid, high performance configurations that meet QoS requirements, which make them ideal when configurations must be determined quickly and dynamically, such as in cyber defense systems [21]. Rapid, real-time response requirements along with low tolerance for false positives exacerbate the challenges in this area.

2.5 Challenge 5: developing robust, extensible and adaptive programming and communication models

Context Many HCDSs are mission- and business-critical, which means that they must meet functional and non-functional requirements despite failures, errors, and attacks. These HCDSs must therefore maintain safety properties while simultaneously supporting and including independent extension and evolution of their subsystems. Doing so will require significantly more advanced hybrid software engineering and development environments keyed to integrating these issues within a consistent overall framework.

State of the art Providing extensibility and supporting system evolution are critical to increasing the longevity and effectiveness of HCDSs. Aguilera et al. [22] present a service that allows heterogeneous components to interact with memory through a common interface in a scalable, consistent manner while hiding complex details, such as accounting for concurrency and failures. Kramer et al. [23] describe the requirements that allow a system to become self-managing, thereby increasing robustness and facilitating system evolution. Environments based upon loose-coupling principles, such as publish and subscribe architectures [24], are an important starting point for dealing with system evolution and change. Environments that focus on the entire information object life-cycle [25] are also be investigated.

Open problems Deployment and configuration issues are also highly influenced by the various programming models used by the systems and supported by the software hosting platforms. Programming models can vary significantly, ranging from sophisticated component and object-based programming abstractions all the way down to simple push messaging models. Moreover, the applications hosted on these platforms may also require different communication characteristics.

For example, the flow of goods within a warehouse or an assembly plant works well in a synchronous data flow model where different elements (such as cranes and conveyer belts) move goods within the warehouse in an orchestrated fashion. Conversely, an intelligent transportation system encompassing automobiles fitted with sensors for monitoring proximity to other automobiles or for sensing road conditions may need an asynchronous, reactive model to operate effectively. Programming and communication models must therefore provide the extensibility needed to tailor them to meet system-specific requirements.

Future research directions To address this challenge, next-generation middleware should be built atop sound type systems that allow users to specify custom types and preserve the semantics of these types end-to-end [26]. These type systems should support type extensibility and evolution, thereby facilitating incremental updates/upgrades, while maintaining system type invariants.

Moreover, since communication models may comprise a mix of synchronous request-response, asynchronous peer-to-peer, or publish/subscribe models in different parts of HCDSs, the extensibility of the middleware infrastructure hosting application and system functionality will need to integrate and adapt according to the model of programming and communication used. Deployment must ensure that the platforms support the different models outlined above.

2.6 Challenge 6: certifying dynamic runtime behavior

Context Many mission-critical HCDSs must undergo formal evaluation prior to being deployed into operational service to ensure the system meets its demand and resource requirements. As the complexity embedded in various aspects of the previous individual challenge areas encroach into system designs, those HCDSs become more dynamic.

Current certification procedures, techniques, and approaches, however, are geared to static systems. As dynamic behaviors become more prevalent these procedures, techniques, and approaches must therefore change to scale up.

State of the art Several projects have focused on large-scale testing of complex, but statically configurable systems. For example, Porter et al. [27] describe the Skoll system that supports the large scale testing of systems, with complex, interacting configuration options, across a large computing grid. Yilmaz et al. [9] applied Skoll to support performance-oriented regression testing of the ACE and TAO middleware. Yoon et al. [28–30] created the Rachet system that supports the large-scale, grid-based, testing of component-based systems with complex configuration and version dependencies. Robinson et al. [31] consider analysis techniques for regression testing of user-configurable software systems.

Open problems Certification processes based on extensive testing and evaluation are generally infeasible for HCDSs systems because the size of the state space of a composed system can be exponential in the number of components. HCDSs typically have a richer set of extensible inputs (including environment conditions and nondeterministic decisions) that affect dynamic system behavior that can be hard to quantify formally, and hence affect certifiability.

Much focus in today's large-scale systems-of-systems is on certification of individual parts operating in isolation. Little attention is focused on certifying the aggregate, integrated packages that share a common base. Likewise, little work addresses minimizing the cost of recertifying a complete package when only a single element (or a few) actually change.

Future research directions Due to the rapid explosion of the size of emerging HCDSs, scalable techniques and tools that can assure such systems will operate as planned and continue to operate properly are a high priority if we are to deploy many of these systems in production environments. There is a need for techniques that isolate key control aspects to keep them amenable to analysis despite scaled size of the overall system, and approximation techniques to selectively prune the enlarging state space.

New deployment and configuration processes, such as those that utilize evolutionary algorithms and swarm techniques [8, 32], show promise for quickly and accurately determining component modifications that will leave system certifications intact. Analysis techniques with real-time constraints for rapid convergence are also needed for to certify dynamic runtime behavior at scale. Likewise, MDE techniques can be used to provide HCDS developers with isolated views of multiple subsystems, allowing easier system modification and control. Current MDE environments support the implementation of custom model interpreters for analyzing and even correcting models to satisfy predetermined design constraints [33]. Future MDE environments should be leveraged to expedite the HCDS design process and give system designers finer granularity of control.

3 The road ahead

Section 2 identified six discrete aspects contributing HCDS complexity, discussed key challenges facing developers and operators of HCDSs that define progress in those areas, and summarize promising research directions needed to address the challenges. More is needed to move forward, however, than simply addressing these challenges individually. In particular, the key remaining challenge for HCDS middleware is to have solutions address *all* these challenges by simultaneously managing the inherent tradeoffs any such undertaking must encompass. For example, if a HCDS meets all configuration requirements and satisfies most QoS constraints, but cannot ensure real-time deadlines are met, the system cannot be considered valid, even though only a small portion of a single challenge was not overcome.

To address this challenge, new design paradigms are needed that not only solve individual challenges, but also deliver them in an integrated, consistent, and compact form. Addressing the complexity inherent in any one challenge is hard; providing a design paradigm that coherently instantiates them all together is even harder. It is even more daunting to do so under the various dimensions of heterogeneity and continuously changing requirements characteristic of many large-scale HCDS domains, such as air traffic control and management, power grid SCADA systems, aerospace, defense, and financial services.

One promising area of research for understanding how to simultaneously deliver a solution that meets multiple individual challenges is *algorithmic mechanism design* [34], which is a theoretical framework for designing algorithms and rewards for a group of interacting distributed participants, such that the overall outcome of their interactions meets a desired goal [35]. For example, algorithmic mechanism design can be used to craft rules for load balancing to ensure that participants providing the computing resources accurately report current loads and service requests fairly. HCDSs will benefit significantly as algorithmic mechanism design theory matures and aids in creating frameworks for integrating competing QoS policies into systems to meet overarching requirement sets. Research advances on algorithmic mechanism design will also help designers build HCDS middleware that can integrate multiple competing objectives.

Nature-inspired computing [36] is another research area relevant to next-generation HCDSs. This approach is well accepted in optimization problems and is gaining acceptance in HCDSs due to the decentralized, loosely coupled, and self-healing properties of many nature-inspired algorithms, such as nonlinear coupled oscillators and ant colony optimization. A successful application of nature-inspired computing to HCDSs appears in [37], where a mathematical model describing fire-fly synchronization is used in to design an internal clock synchronization algorithm that is fully decentralized, robust with respect to relatively high level or churn, and has an error bound that is lower than the Network Time Protocol.

Since individual challenges are intertwined, future design paradigms for HCDSs must manage the various tradeoffs that arise in any usable instantiation. For instance, HCDS developers may determine that the increase in performance provided by adding a state-of-the-art component to a configuration is not offset by the increased cost of the component or the impact of the configuration change on other QoS requirements, such as reliability or security. Moreover, new paradigms must support the trend toward pervasive dynamic behavior and instantaneous reaction to cope with the realities of cyber-physical and ultra-large-scale systems. These paradigms must also account for emerging computing paradigms, such as quantum computing, neuromorphic computing, and nature-inspired computing, with their inevitable new and different measures of complexity. Undoubtedly, next-generation HCDS solutions in these spaces will involve new middleware architectures, tools, and algorithms appropriate to the emerging platforms.

The middleware R&D community has heretofore achieved significant success through "divide and conquer" approaches, which reflect the common sole-PI model of basic research funding. The challenge going forward, however, includes marshaling the various forms of in-depth expertise gained in these individual areas of focus to create common design paradigms and integrated delivery vehicles, with embedded and automated means for varying trade-offs, owing to the differences in particular domains of use. Failure to do so will result in potential solutions collapsing under the weight of their own complexity, becoming outrageously expensive or vastly underperforming, or more likely all of the above.

Advances in the use of metaheuristic algorithms, such as genetic algorithms and particle swarm optimization [38, 39], for HCDS design will help developers understand trade-offs that cannot be managed manually. These algorithms have already been used successfully [8, 40] for design and debugging problems, such as reducing the computational complexity of using model analysis to detect design flaws, generating optimized mappings of software to processors, and balancing competing network and computing QoS values. They have also been used to explore alternative software architectures to optimize multiple performance objectives, such as finding and evaluating alternative architectures for business information systems [41]. As the science of applying these algorithms to system design advances, these algorithms will scale up to handle more key QoS properties of complex HCDCSs and automatically derive designs that meet competing requirement sets.

As the configuration solution space size continues to expand exponentially, constraint solvers, metaheuristic algorithms, and approximation techniques become prohibitively resource intensive. Fortunately, commodity cloud computing environments now provide large amounts of computational resources on-demand [42, 43] and are being used to support powerful federated quality assurance algorithms. For instance, Yoon et al. [28, 30] test and evaluate component-based systems by incrementally building these systems inside virtual machines; sharing copies of the virtual machines across cloud nodes; and then testing, evaluating, and analyzing different system instances in parallel. The computational resources of future environments will aid large-scale design analyses that support more computationally-intense configuration techniques than can run today on in-house hardware infrastructures.

Decades of investment on middleware R&D has firmly established its essential role at the intersection of distributed applications and the underlying host and communication platforms for HCDSs. Next-generation middleware requires significant improvements in our software engineering methods, languages, tools, techniques, and training embodying the middleware-centric solutions to put into practice against problem spaces that by themselves embody considerable complexity. Certain research directions will involve rethinking, reformulating, recombining some core ideas that have carried us this far, while scaling them up, shrinking them down, and making them work correctly over a wider range of conditions. Examples of underlying ideas that likely need refocusing include the following:

– Raising the level of abstraction away from individual host, objects, and connections toward conglomerate behaviors with varying properties sustained over varying mappings and bindings underneath,
– Various forms of multilayer design paradigms appropriate to the higher levels of abstraction, and
– Seeking completeness and consistency in place of today's incomplete abstractions that cause good ideas in theory to turn into bad ideas in practice when confronted with the complexity and scale of next-generation HCDSs.

References

1. Rover D, Waheed A, Mutka M, Bakic A (1998) Software tools for complex distributed systems: toward integrated tool environments. IEEE Concurr 6(2):40–54

2. Schnorr L, Legrand A, Vincent J (2012, to appear) Detection and analysis of resource usage anomalies in large distributed systems through multi-scale visualization. In: Concurrency and computation: practice and experience. Wiley
3. Albrecht J, Braud R, Dao D, Topilski N, Tuttle C, Snoeren A, Vahdat A (2007) Remote control: distributed application configuration, management, and visualization with plush. In: Proceedings of the 21st conference on large installation system administration conference. USENIX Association, Berkeley, p 15
4. Institute SE (2006) Ultra-large-scale systems: software challenge of the future. Tech rep, Carnegie Mellon University, Pittsburgh, PA, USA, June 2006
5. White J, Hill J, Eade S, Schmidt DC (2008) Towards a solution for synchronizing disparate models of ultra-large-scale systems. In: Proceedings of the ULSSIS workshop, Leipzig, Germany, May 2008
6. Corsaro A (2010) The data distribution service for real-time systems. Dr Dobbs J
7. Hatcliff J (2009) An integrated specification and verification environment for component-based architectures of large-scale distributed systems. Tech rep, DTIC Document
8. White J, Dougherty B, Thompson C, Schmidt D (2011) ScatterD: spatial deployment optimization with hybrid heuristic/evolutionary algorithms. ACM Trans Auton Adapt Syst 6(3). Special Issue on Spat Comput
9. Yilmaz C, Porter A, Krishna A, Memon A, Schmidt D, Gokhale A, Natarajan B (2007) Reliable effects screening: a distributed continuous quality assurance process for monitoring performance degradation in evolving software systems. IEEE Trans Softw Eng 124–141
10. Fouché S, Cohen MB, Porter A (2009) Incremental covering array failure characterization in large configuration spaces. In: Proceedings of the eighteenth international symposium on software testing and analysis, ISSTA ’09, pp 177–188
11. Westermann D, Happe J (2010) Towards performance prediction of large enterprise applications based on systematic measurements. In: Proceedings of the 15th international workshop on component-oriented programming (WCOP) 2010, pp 71–78
12. Kappler T, Koziolek H, Krogmann K, Reussner RH (2008) Towards automatic construction of reusable prediction models for component-based performance engineering. Softw Eng 121:140–154
13. Schantz R, Schmidt D (2008) Middleware for distributed systems. In: Wah B (ed) Encyclopedia of computer science and engineering. Wiley, New York
14. Otte W, Gokhale A, Schmidt DC (2011) Predictable deployment in component-based enterprise distributed real-time and embedded systems. In: Proceedings of the 14th international ACM SIGSOFT symposium on component-based software engineering (CBSE), Boulder, CO, USA. ACM, New York
15. Rohloff K, Gabay Y, Ye J, Schantz R (2007) Scalable, distributed, dynamic resource management for the ARMS distributed real-time embedded system. In: Parallel and distributed processing symposium, 2007, IPDPS 2007, IEEE International. IEEE Press, New York, pp 1–7
16. Loyall J, Gillen M, Sinclair A, Carvalho M, Bunch L, Marcon M, Martignoni A (2009) Quality of service in US air force information management systems. In: Military communications conference, 2009. MILCOM 2009. IEEE Press, New York, pp 1–8
17. Loyall J, Gillen M, Paulos A, Bunch L, Carvalho M, Edmondson J, Schmidt D, Martignoni A III, Sinclair A (2011) Dynamic policy-driven quality of service in service-oriented information management systems. In: Software: practice and experience
18. Surajbali B, Grace P, Coulson G (2009) A semantic composition model to preserve (Re)configuration consistency in aspect oriented middleware. In: Proceedings of the 8th international workshop on adaptive and reflective middleware. ACM, New York, pp 1–6
19. Otte W, Schmidt D, Gokhale A (2010) Towards an adaptive deployment and configuration framework for component-based distributed systems. In: Proceedings of the 9th workshop on adaptive and reflective middleware (ARM’10)
20. Atighetchi M, Pal P (2009) From auto-adaptive to survivable and self-regenerative systems successes, challenges, and future. In: 8th IEEE international symposium on network computing and applications, 2009. NCA 2009. IEEE Press, New York, pp 98–101
21. White J, Doughtery B, Schmidt D (2010) Ascent: an algorithmic technique for designing hardware and software in tandem. IEEE Trans Softw Eng 838–851
22. Aguilera M, Merchant A, Shah M, Veitch A, Karamanolis C (2007) Sinfonia: a new paradigm for building scalable distributed systems. In: Proceedings of 21st ACM SIGOPS symposium on operating systems principles. ACM, New York, pp 159–174
23. Kramer J, Magee J (2007) Self-managed systems: an architectural challenge. In: ICSE 2007
24. Grant R, Combs V, Hanna J, Lipa B, Reilly J (2009) Phoenix: SOA based information management services. In: Proceedings of SPIE, vol 7350, p 73500P
25. Cleveland J, Loyall J, Webb J, Hanna J, Clark S (2011) VFILM: a value function driven approach to information lifecycle management. In: Society of photo-optical instrumentation engineers (SPIE) conference series, vol 8062, p 1
26. Group OM (2010) Extensible and dynamic topic types for DDS. Specification version 1.0, Object Management Group, December 2010
27. Porter A, Yilmaz C, Memon A, Schmidt D, Natarajan B (2007) Skoll: a process and infrastructure for distributed continuous quality assurance. IEEE Trans Softw Eng 510–525
28. Yoon I, Sussman A, Memon A, Porter A (2007) Direct-dependency-based software compatibility testing. In: Proceedings of the 22nd IEEE/ACM international conference on automated software engineering. ACM, New York, pp 409–412
29. Yoon I, Sussman A, Memon A, Porter A (2008) Effective and scalable software compatibility testing. In: Proceedings of the 2008 international symposium on software testing and analysis. ACM, New York, pp 63–74
30. Yoon I, Sussman A, Memon A, Porter A (2011) Towards incremental component compatibility testing. In: Proceedings of the 14th international ACM Sigsoft symposium on component based software engineering, CBSE ’11, pp 119–128
31. White L, Jaber K, Robinson B, Rajlich V (2008) Extended firewall for regression testing: an experience report. J Softw Maint Evol, Res Practice 20(6):419–433
32. Martens A, Koziolek H, Becker S, Reussner R (2010) Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In: Proceedings of the 1st joint WOSP/SIPEW international conference on performance engineering. ACM, New York, pp 105–116
33. Amyot D, Farah H, Roy J (2006) Evaluation of development tools for domain-specific modeling languages. Syst Anal Model Lang Profiles 183–197
34. Briest P, Krysta P, Vöcking B (2005) Approximation techniques for utilitarian mechanism design. In: Proceedings of the 37th annual ACM symposium on theory of computing. ACM, New York, pp 39–48
35. Mu’Alem A, Nisan N (2008) Truthful approximation mechanisms for restricted combinatorial auctions. Games Econ Behav 64(2):612–631
36. Liu J, Tsui K (2006) Toward nature-inspired computing. Commun ACM 49:59–64
37. Baldoni R, Corsaro A, Querzoni L, Scipioni S, Piergiovanni ST (2009) Coupling-based internal clock synchronization for large-

scale dynamic distributed systems. IEEE Trans Parallel Distrib Syst 99(RapidPosts):607–619

38. Sivanandam S, Deepa S (2007) Introduction to genetic algorithms. Springer, Berlin
39. Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization. Swarm Intell 1(1):33–57
40. Dougherty B, White J, Balasubramanian J, Thompson C, Schmidt D (2009) Deployment automation with BLITZ. In: 31st international conference on software engineering, companion volume. IEEE Press, New York, pp 271–274
41. Koziolek A, Noorshams Q, Reussner R (2011) Focussing multi-objective software architecture optimization using quality of service bounds. In: Models in software engineering, workshops and symposia at MODELS 2010, Oslo, Norway, October 3–8, 2010. Lecture notes in computer science, vol 6627. Springer, Berlin, pp 384–399. Reports and revised selected papers
42. Buyya R, Yeo C, Venugopal S (2008) Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities. In: The 10th IEEE international conference on high performance computing and communications. IEEE Press, New York, pp 5–13
43. Ostermann S, Iosup A, Yigitbasi N, Prodan R, Fahringer T, Epema D (2010) A performance analysis of EC2 cloud computing services for scientific computing. Cloud Comput 115–131