Software Engineering Institute
Carnegie Mellon University

ULTRA-LARGE-SCALE (ULS) SYSTEMS: SOCIO-ADAPTIVE SYSTEMS

Scott Hissam, Mark Klein, Gabriel Moreno, Linda Northrop, Lutz Wrage

December, 2016

Overview

Ultra-large-scale systems are interdependent webs of software-intensive systems, people, policies, cultures, and economics.

The scale and complexity of systems is increasing dramatically. Ultra-large-scale (ULS) systems are systems of unprecedented scale in some of these dimensions:

- lines of code
- amount of data stored, accessed, manipulated, and refined
- number of connections and interdependencies
- number of hardware elements
- number of computational elements
- number of system purposes and user perception of these purposes
- number of routine processes, interactions, and "emergent behaviors"
- number of (overlapping) policy domains and enforceable mechanisms
- number of people involved in some way

How are ULS systems different?

The sheer scale of ULS systems changes everything. ULS systems will necessarily be decentralized in a variety of ways, developed and used by a wide variety of stakeholders with conflicting needs, evolving continuously, and constructed from heterogeneous parts. People will not just be users of a ULS system; they will be elements of the system. Software and hardware failures will be the norm rather than the exception. The acquisition of a ULS system will be simultaneous with its operation and will require new methods for control.

These characteristics may appear in today's systems and systems of systems, but in ULS systems, they will dominate. Consequently, ULS systems will place unprecedented demands on software acquisition, production, deployment, management, documentation, usage, and evolution practices.

The ULS systems notion has inspired us to ask new questions about software-reliant systems:

- What new quality attributes arise due to scale?
- What types of analyses are required to understand and design (at all levels) systems at scale?
- Are new architecture design principles needed?
- What new strategies are needed to control, predict, and bound the behavior of systems at scale?

The SEI strives to answer these questions and to develop principles and technology to understand, control, and bound the behavior of systems that exhibit characteristics of ULS systems.

Research

Challenges of ULS systems

Fundamental gaps in our current understanding of software and software development at the scale of ULS systems present profound impediments to the technically and economically effective achievement of information superiority. These gaps are strategic, not tactical. They are unlikely to be addressed adequately by incremental research within established categories. Rather, we require a broad new conception of both the nature of such systems and new ideas for how to develop them. We will need to look at them differently, not just as systems or systems of systems, but as socio-technical ecosystems. We will face fundamental challenges in the design and evolution, orchestration and control, and monitoring and assessment of ULS systems. These challenges require breakthrough research.

Recommendations

The ULS System Research Agenda presented in *Ultra-Large-Scale Systems: The Software Challenge of the Future* [Feiler et al 2006] provided the starting point for the path ahead. The study proposed a ULS systems research agenda for an interdisciplinary portfolio of research in at least the following areas:

- Human Interaction
- Computational Emergence
- Design
- Computational Engineering
- Adaptive System Infrastructure
- Adaptable and Predictable System Quality
- Policy, Acquisition, and Management

Socio-adaptive systems focus on the effects of the human element in a system. Specifically, socio-adaptive systems refer to large-scale systems comprising distributed system resources such as processors and wireless networks. Humans rely on these resources to carry out missions. Continuous change in the network coupled with changing needs of missions supported by the system pose challenges. Additionally, appropriate responses to changing needs rely on the expressed needs of humans. Therefore, factors such as human self-interest must be accounted for in system design.

Our research in socio-adaptive systems involves

- developing new resource-allocation techniques for mobile ad hoc networks
- applying market mechanisms to ensure efficient resource reallocation in continuously changing tactical settings

Socio-Adaptive Systems

Socio-Adaptive Systems are systems in which human and computational elements interact as peers. The behavior of the system arises from the properties of both types of elements and the nature of how they collectively react to changes in their environment including mission, and the availability of the resources they use. These properties and interactions give rise to new quality attributes that will influence the structure of these systems.

One environment where socio-adaptive systems are needed is that of first responders operating in disaster relief situations. In those settings, first responders rely on wireless networks that are quickly assembled in the field, called ad hoc networks. These networks provide needed information and directions, and they support responders through situational awareness, using messaging-, voice- and video-based applications. The needs of first responders for using such information changes throughout the mission, and the network resources must be used to best satisfy those needs. At the same time, the capacity of the network changes as the network nodes move through the environment, resulting in periods of insufficient capacity. This situation sometimes leads users to overstate their needs in their requests to obtain resources, without considering how this might impair availability for other users. This, in turn, can result in an inefficient use of resources, detrimental to the overall mission.

The SEI created the Socio-Adaptive Systems Project to help enable effective, adaptive, mission-aware use of resources. The goal is to allocate—automatically, continuously, and effectively—scarce network capacity to users based on their own accurately reported needs. The SEI aims to establish a new approach for designing adaptive socio-technical systems in which people, networks, and computer applications can determine locally how to respond when the demand for resources outstrips supply, while guaranteeing the best use of available capacity.

Challenges

This research combines the adaptability of human social institutions, in particular those based on market institutions, with automated network-resource optimization. Highly specialized workers, such as first responders, are trained to act independently and to make complex decisions when confronted with unexpected circumstances. But ever faster operational tempos and the expanded importance of digitally networked resources, combined with a proliferation of new uses for those resources, require substantial automation of resource allocation and optimization procedures. We must address significant challenges to create the desired socio-adaptive combination:

- Overstated needs affect resource allocation.
- Decision makers do not have access to timely, reliable estimates of network capacity.

Innovations

We focus on two interrelated tasks to address the challenges of allocating scarce network resources in environments of uncertainty and change:

- Use computational mechanism design (CMD) to elicit changing mission needs and compute an optimal allocation of resources.

- Use decentralized quality-of-service (QoS) optimization to optimally respond to changes in emergency resource capacity.

Research Approach

We apply microeconomic foundations known as CMD to address the decentralized and dynamic nature of network resource allocation in emergency settings. There is significant literature on using incentive-compatible market mechanisms (i.e., mechanisms that cause participants to truthfully reveal relevant information) for allocating computational resources, but little work has considered the dynamics and uncertainty that typify emergency operations. Our research builds on previous work by generalizing promising mechanisms that model the effects of uncertainty on current and future allocation decisions.

We are developing a distributed version of the QoS resource allocation model (Q-RAM) [Lee et al 1999] to allocate emergency resources to applications requiring them. Distributed Q-RAM (D-Q-RAM) is novel in that it does not require a centralized aggregator of application QoS information, nor does it require explicit knowledge of the capacity of emergency resources. D-Q-RAM provides a vehicle for expressing mission needs incrementally: different levels of QoS are associated with expressions of mission utility or value. CMD will rely on D-Q-RAM to compute an optimal lower level allocation based on responder input.

Unlike previous work, the SEI approach makes explicit and quantifiable the relation between responders' needs as they perceive these needs and the ability of the emergency network to satisfy these needs. Further, this approach provides adaptation mechanisms that allow application performance to degrade gracefully in a way that maximizes mission value. Finally, the results of this work can be applied to other scarce emergency resources, such as emergency vehicles or cloud capacity. This approach is also largely agnostic about the underlying mobile ad hoc network protocol and can therefore be combined with a variety of networking infrastructures.

Objectives

This capability will improve current practice in several ways:

- Incentive-compatible allocation procedures: The best outcome for each responder is obtained when responders report their true needs. Responders will be able to continuously report their changing needs for digital resources without having to speculate on how their reports will influence allocation decisions.
- Effective bandwidth allocation: The proliferation of digital resources, the growing diversity of their uses in emergency networked operations, and the quickened tempo of these operations demand a high degree of automation in managing scarce resources. This capability will reduce reliance on manual intervention to deliver reliable network capabilities where they are most needed, and in time to meet those needs.
- Gracefully degrading QoS when network demand outstrips capacity and automatically recovering when capacity becomes available: Our research allows the network infrastructure and the applications it supports to agree on a range of actions that the application can take to exploit increased network capacity and to accommodate reduced network capacity.

Glossary

abstraction: 1. A process of eliminating, hiding, or ignoring characteristics or aspects of a concept unrelated to a given purpose. 2. A concept or system construct that has been subjected to a process of abstraction.

acceptability-oriented computing: An approach to the construction of systems in which a designer identifies a set of properties that the execution must satisfy to be acceptable to its users. This is in contrast to the traditional approach, which is to construct a system with as few errors as possible. Acceptability-oriented computing was defined by Martin Rinard, Professor of Computer Science at MIT, in the paper, "Acceptability Oriented Computing," presented at the 2003 Object- Oriented Systems, Languages, & Applications Conference (OOPSLA '03).

agile method or agile methodology: A style of software development characterized by its release schedule, attitude toward change, and patterns of communication. The product is developed in iterations, usually one to four weeks long. At the end of each iteration, the product has additional, fully implemented value and is ready to be deployed. The design horizon usually extends only to the end of the current iteration; little code is written in anticipation of future needs. The project is seen by the programmers as a stream of unanticipated requirements. Written natural-language communication is considered a usually inefficient compromise. Face-to-face communication is higher bandwidth (but transient). Executable documentation—code and tests—is permanent, less ambiguous, and self-checking. Agile projects prefer a combination of the latter two over the first.

aleatoric: Pertaining to luck, chance, or randomness.

allopoiesis: A process whereby an organization or network of components produces something other than itself; literally, *other-production*. An example of allopoiesis is an assembly line.

annealing: Any process for increasing the order of a system by first increasing its susceptibility to disorder and then steadily decreasing such susceptibility in the presence of mechanisms or phenomena that tend to create or capture order.

ant-colony optimization: A technique for solving problems that can be reduced to finding short paths in a graph by using a process similar to how an ant colony finds paths to food. In this process, individuals randomly wander the graph, leaving behind a trail that dissipates over time. When a good path is found, the individual returns along the left trail, reinforcing it. Other individuals who find this path are likely to follow it, further reinforcing it. Because trails dissipate over time, longer trails will be less likely to be reinforced than shorter ones.

aspect-oriented programming: A programming paradigm that attempts to aid programmers in the separation of concerns (breaking down a program into distinct parts that overlap in functionality as little as possible). The hallmark of the paradigm is to represent as modules ***crosscutting concerns***, which are distinct design decisions or functionality that are conceptually distinct but whose implementations are usually dispersed throughout the modules of a system. The idea was first presented in the paper, "Aspect-Oriented Programming," by Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda,

Cristina Lopes, Jean-Marc Loingtier, and John Irwin, published in the *Proceedings of the European Conference on Object-Oriented Programming*, 1997.

attribute-specific design rule: A ***design rule*** aimed at maintaining a particular ***quality attribute***.

augmented reality: A view of the real environment augmented with computationally supplied information, providing a composite view of the world that is partly real and partly digital. A military heads-up display is a simple example of augmented reality.

autocatalysis: A chemical reaction whose products include catalysts for that reaction.

autopoiesis: A process whereby an organization or network of components produces itself; literally, *self-production*. An example of autopoiesis is a cell or an organism.

Bayesian technique: Any learning or decisionmaking technique that relies on Bayes' Theorem which, informally, tells how to update or revise beliefs in light of new evidence.

belief logic: Any logical calculus that models belief; for example, as sets of formulae or ***probabilistically***.

black-box abstraction: An abstraction or component whose implementation is hidden and whose functionality is available only through its interface (cf., ***open abstraction***).

black-box testing: *Black-box testing*, *concrete-box testing*, and *functional testing* refer to testing the outputs of a program given knowledge of only its functional specification and not its implementation. Black-box testing is in contrast to ***clear-box testing***.

certification: Declaration via a formal certificate from an accredited body attesting that a particular assurance regarding software, hardware, or a system is true.

cleanroom software engineering: A software-development methodology defined by Harlan Mills and his colleagues, based on formal methods, iterative implementation, and statistical quality control. The objectives of the cleanroom process are to develop software incrementally, produce software that approaches zero defects prior to first execution, and certify software fitness for use. See *Cleanroom Software Engineering: Technology and Process* by S. Prowell, C. Trammell, R. Linger, and J. Poore. Reading, MA: Addison Wesley Longman, 1999.

clear-box testing: *White-box testing*, *clear-box testing*, *glass-box testing*, and *structural testing* refer to testing the outputs of a program given knowledge of how the program is implemented. Clear-box testing is generally done by programmers who try to cover parts of the code and cases that they suspect are prone to coding errors. Clear-box testing is in contrast to ***black-box testing***.

competitive software design: A design process in which competition is intentionally introduced at many levels.

complexity science: A scientific discipline that studies systems of multiple, possibly diverse, interconnected elements that have the capacity to change in response to experience, both external and internal.

composition: An act or result of combining simpler objects into more complex ones. For example, simpler data types can be combined into more complex ones. Composition also refers to the act or result of determining the net effect produced by combining functions; for example a composite function can be determined by applying each given function to the results of the previous function in some order in a cascade.

concurrent: When the execution flow of several computational processes are able to run simultaneously, perhaps while sharing resources. In general, concurrent execution is not expected to save elapsed time over sequential execution (cf., ***parallel***).

context-aware assistive computing or context-aware computing: An approach to the design of a pervasive or ***ubiquitous computing*** system that focuses on the shared understanding between humans and their computational environments, particularly regarding their shared context. Context is any information that can be used to characterize the situation of entities (i.e., people, places, and objects) that is relevant to the interaction between a user and a system, including the user and the system themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects.

contract net: A collection of computational nodes that collaborate to solve a distributed problem by allocating tasks to nodes via a series of contract negotiations using a formal protocol consisting of task announcements, bids, awards, and results reporting. Negotiations involve task descriptions and requirements such as mandatory available resources and time constraints. The concept was first introduced by Reid Smith and Randy Davis in "The Contract Net Protocol: High level Communication and Control in a Ultra-Large-Scale Systems 127 Software Challenge of the Future Distributed Problem Solver," IEEE Transactions on Computers 29, 12 (1980):1104-1113.

CORBA: **C**ommon **O**bject **R**equest **B**roker **A**rchitecture (CORBA) is a standard for software components. It defines application programming interfaces (APIs), communication protocols, and object/service information models to enable heterogeneous applications written in various languages running on various platforms to interoperate. CORBA thus provides platform and location transparency for sharing objects across a distributed computing platform.

crosscutting or crosscutting concerns: A single, coherent design or implementation decision or issue whose implementation typically must be scattered throughout the modules of a system. A crosscutting concern is called an aspect in ***aspect-oriented programming***.

crossover: A genetic operator in ***digital software evolution*** that varies the genetic makeup of a member of the next generation by taking genetic contributions from two members of the current generation. In a ***genetic algorithm***, genetic material is represented linearly, and crossover is via a form of splicing. In genetic programming, genetic material is represented as a tree or as trees, and crossover is via a form of subtree substitution.

cryptographic: Referring to the security of information based on encoding messages in such a way that they cannot be decoded without special, private, and hard-to-acquire information.

cybernetics: The science of systems of control and communications in living organisms and machines.

decentralized system: A distributed system with no central authority for any of its aspects.

Dempster-Shafer theory: A formal theory of evidence (see ***evidence theory***) based on belief functions and plausible reasoning that is used to combine separate pieces of information (evidence) to calculate the probability of an event. The theory was developed by Arthur P. Dempster and Glenn Shafer.

design architecture: A set of decisions that partitions the task of producing the complete design for a system into a set of largely separable subtasks.

design of all levels: An approach to architecture and design that includes as part of the design not only the artifact being constructed but also the organizational, social, and process structure of the design teams, including individuals, firms, and other organizations.

design risk: A design decision made in the absence of certainty of the outcome. In some design contexts, it is not always known whether a decision will result in a satisfactory artifact when the design is completed and implemented; such a decision represents a risk in the design process that must be assessed and managed.

design rule: A decision concerning the architecture of a system that helps establish the degree and nature of the modularity of the system's design. Typically, a design rule minimizes interaction between modules in the design as well as between different designers or design groups; design rules are also typically based on design experience. For example, the decision whether to control the screen of a computer with the CPU or a separate graphics processor is a design rule that structures the design space for a computer.

design space: The set of design parameters along with the range of values for those parameters for a design. A design can be considered an outline for an artifact (its architecture) along with a set of decisions about the nature and details of the outline; the space of possible decisions is the design space.

deterministic: A property of a computation that always has one (and the same) result given the same initial state and inputs.

digital evolution or ***digital software evolution:*** An automated methodology inspired by biological evolution to create software that best performs a specified task. It is a ***machine learning*** technique that uses an evolutionary algorithm (i.e., a ***genetic algorithm*** or ***genetic programming***) to optimize a population of programs according to a ***fitness function*** that measures a program's ability to perform a specified task.

dissipative system: An open system that is operating outside equilibrium within an environment that exchanges energy, matter, or ***entropy***.

distributed cognition: A branch of cognitive science that proposes that human cognition is not confined to the individual, but is distributed by attaching memories, facts, or knowledge to objects, individuals, and tools in the environment.

distributed system: A system in which there are a number of components communicating over a network that are trying to cooperate in some fashion (or which taken together form the system).

distribution: Placing execution flows in different processes or on different computational ***platforms***. Distribution is typically applied to improve fault tolerance or to access remote resources.

domain-specific language: A programming language designed to be useful for a specific set of tasks.

driven system: A system with external energy or information inputs.

econometrics: The application of statistical and mathematical methods in the field of economics to describe the numeric relationships between key economic forces. The main purpose of econometrics is to empirically verify economic theory.

ecosystem: In biology, an ecosystem is a community of plants, animals, and microorganisms that are linked by energy and nutrient flows and that interact with each other and with the physical environment. Rain forests, deserts, coral reefs, grasslands, and a rotting log are all examples of ecosystems. ULS systems can be characterized as ***socio-technical ecosystems***.

embodied interaction: A form of interaction with a computational system that reflects the philosophy that effective communication must take place in physical and social environments, not purely in virtual ones.

embodied virtuality: The manifestation of the results, processes, and mechanisms of computation in the physical world. This term, first defined by Mark Weiser in *The Computer for the 21st Century* (San Francisco, CA: Morgan Kauffman Publishers, 1995) is an alternative to ***ubiquitous computing***, which envisions computers as physical presences in numerous parts of the real world.

Enterprise JavaBeans: A component architecture for building distributed, object-oriented business applications in Java. An Enterprise JavaBean (EJB) encapsulates business logic in a component framework that manages the details of ***security***, transaction, and state management. Low-level details such as multithreading, resource pooling, clustering, distributed naming, automatic persistence, remote invocation, transaction boundary management, and distributed transaction management are handled by the EJB "container."

entropy: Informally, the degree of disorder of a system. Entropy has specific definitions beyond the scope of this glossary in thermodynamics, statistical mechanics, and information theory.

epistemic uncertainty: Uncertainty based on lack of knowledge.

evidence theory: A formal system of reasoning about knowledge based on a belief function (not defined in this glossary) as the representation of degree of belief, and a method of combining evidence and belief. The theory is explained in *A Mathematical Theory of Evidence* by Glenn Shafer (Princeton, NJ: Princeton University Press, 1976).

example-driven design: A variant of ***test-driven design*** in which the tests are written as if they were a series of examples being used to teach someone how to use the code, beginning with simple cases and moving toward the trickier ones.

fitness function: A type of ***objective function*** that quantifies the optimality of a solution in a ***genetic algorithm*** so that a solution can be ranked against all the others.

flow-structure analysis: A method for understanding the compositions of system services implemented in a network to carry out user tasks. See *Flow-Service-Quality (FSQ) Engineering: Foundations for Network System Analysis and Development*, by R. Linger, M. Pleszkoch, G. Walton, and A. Hevner (CMU/SEI-2002-TN-019), Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, June 2002.

fractal: Informally, a shape that appears similar at all scales; formally, a geometric object whose Hausdorff dimension (not defined in this glossary) is greater than its topological dimension (not defined in this glossary).

function extraction (FX): Technology for automated computation of the net functional effect, or behavior, of programs, presented in terms of behavior catalogs for human understanding and analysis. See The Impact of Function Extraction Technology on Next-Generation Software Engineering by A. Hevner, R. Linger, R. Collins, M. Pleszkoch, S. Prowell, and G. Walton (CMU/SEI 2005-TR-015), Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, July 2005.

function-theoretic: Of or referring to ***function-theoretic foundations***.

function-theoretic foundations: An approach to formalizing computation based on a view of programs as implementations of mathematical functions or relations, regardless of their subject matter or language. Function-theoretic foundations provide methods for software specification, design, verification, and analysis, and play a key role in ***cleanroom software engineering***, ***function-theoretic verification***, and ***function extraction***.

fuzzy set: A set with imprecise membership criteria. In classical set theory, a set can be defined by a characteristic function that takes an element of the universe and assigns 1 if the element belongs to the set and 0 otherwise. A fuzzy set is defined by a membership function that assigns a real number in the interval [0,1], where 1 means the element belongs, 0 means it doesn't, and a number in between 0 and 1 indicates the degree of membership. This number is not a probability or likelihood, but an imprecision or vagueness. For example, a person walking through the doorway between the kitchen and dining room whose foot is in the dining room and part of whose body is in the kitchen is partly in the set of people in the kitchen and partly in the set of people in the dining room.

game theory: A branch of applied mathematics that studies strategic situations in which players choose actions in an attempt to maximize their returns.

genetic algorithm: A method of simulating the action of evolution within a computer. A population is evolved by employing ***crossover*** and mutation operators along with a ***fitness function*** that determines how likely individuals are to reproduce. A genetic algorithm usually operates on a population of fixed-length vectors of characteristics.

genetic programming: A form of ***genetic algorithm*** in which members of the evolving population are tree-like representations of computer programs.

glue or ***glue code:*** Code that enables one piece of code or a component to interact with other code or another component.

GOMS model: A description of the knowledge that a person must have to carry out tasks on a device or system. The acronym "GOMS" stands for **G**oals, **O**perators, **M**ethods, and **S**election rules. A GOMS model consists of descriptions of the methods needed to accomplish specified goals. The methods are a series of steps consisting of operators that the person performs. A method may call for sub-goals to be accomplished, so the methods have a hierarchical structure. If there is more than one method to accomplish a goal, then the GOMS model includes selection rules that choose the appropriate method depending on the context. The GOMS framework was proposed by Card, Moran, and Newell in *The Psychology of Human-Computer Interaction* (Mahwah, NJ: Lawrence Erlbaum Associates, 1983).

greedy algorithm: A problem-solving or optimization algorithm that uses the ***metaheuristic*** of making the locally optimum choice at each stage until the entire problem is solved or optimization is complete. It typically produces a good solution quickly but rarely an optimum one.

hybrid systems modeling: The process of and languages for modeling and simulating systems with both continuous and discrete processes. Hybrid systems modeling aims to model systems that can be described by ordinary differential equations, partial differential equations, differential algebraic equations, and ordinary differential equations interfaced with discrete-time algorithms.

hypermutation: A biological process or mechanism to enable rapid adaptation to environmental conditions (including parasites and viruses) by altering (usually accelerating) rates of mutation. For example, the acquired immune system in multicellular animals uses hypermutation.

in situ control and adaptation: A model-based approach for designing adaptive quality of service in ***distributed systems***.

in situ control, reflection, and adaptation: in situ control and adaptation using ***reflection*** as an approach.

institution design: See ***mechanism design***.

Internet storm: A disruptive and sometimes malicious set of time-localized, large-scale Internet infrastructure events. Denial-of-service attacks, Internet worms, and cascading router failures are examples of Internet storms.

jitter: A slight random or irregular variation in an otherwise regular sequence or signal. In telecommunication, an abrupt variation of one or more signal characteristics, such as the interval between successive pulses, the amplitude of successive cycles, or the frequency or phase of successive cycles. In packet-based networking, jitter is the variation in the delay of packets.

latency: A time delay between the moment something is initiated and the moment its first effect begins.

law of large numbers: A statistical law stating that the average of a random sample from a large population is likely to be close to the mean of that population.

machine learning: An approach to certain computations in which, broadly speaking, a computer changes its own structure, program, or data based on inputs or external data in such a manner that its expected future performance improves. More specifically, machine learning refers to a set of techniques

such as automatic rule extraction and application, determining statistical characteristics of a population and using them to make decisions, determining the weights and perhaps topology for a neural net based on positive and negative training examples and using it for classification, etc.

mechanism design: A sub-field of ***game theory*** that studies how to design the rules of a game to achieve a specific outcome by setting up a structure in which each player has an incentive to behave as the designer intends. One branch of mechanism design is the creation of markets such as auctions.

metadata: Data that are about or that describe data. Note that the word "metadata" has been trademarked by The Metadata Company, and the legal status of the generic use of the term has not been settled.

metaheuristics: High-level procedures that coordinate simple heuristics, such as local search, to find solutions that are of better quality than those found by the simple heuristics alone; metaheuristics are typically used in situations where exact algorithms are not feasible. Metaheuristics include ***simulated annealing***, ***genetic algorithms***, tabu search, GRASP, scatter search, ***ant-colony optimization***, variable neighborhood search, and their hybrids.

metastability: The ability of a non-equilibrium state to persist for some period of time. A system in a metastable state is able to pass to a more stable equilibrium when sufficiently disturbed.

microeconomics: The branch of economics that deals with small-scale economic factors, such as the economics of an individual firm, product, or consumer rather than with the aggregate. One of the goals of microeconomics is to analyze market mechanisms that establish relative prices and allocate resources.

middleware: A set of layers and components that provides reusable common services and network programming mechanisms. Middleware resides on top of an operating system and its protocol stacks but below the structure and functionality of any particular application.

model checking: A method to algorithmically verify a design by checking whether a model derived from the design satisfies its formal specification.

model-driven architecture: An approach to model-driven engineering architecture sponsored by the Object Management Group that provides a set of guidelines for structuring specifications expressed as models. System functionality is defined as a platform-independent model (PIM) through an appropriate ***domain-specific language***. Given a platform definition model (PDM) corresponding to ***CORBA***, ***.NET***, etc., the PIM is then translated to one or more platform-specific models (PSMs) for the actual implementation. The translations between the PIM and PSMs are normally performed using automated tools.

monoculture: A system with low diversity.

monotonic: The property of a type of information and a method of reasoning or operating on it for which the addition of new information does not decrease the set of valid inferences or operations.

multi-valued logic: A logical calculus in which there are more than two possible truth values. Examples include fuzzy logic and ***probabilistic*** logic such as ***Dempster-Shafer theory***.

mutator function: As used in a ***metaheuristic*** process, a function that takes a current state and returns a neighbor state, typically in a ***probabilistic*** manner.

.NET: The Microsoft .NET Framework is a component of the Microsoft Windows operating system. It provides a large body of pre-coded solutions to common program requirements and manages the execution of programs written specifically for the framework. Programs written for the .NET framework execute in a software environment that manages the program's runtime requirements. This runtime environment, which is also a part of the .NET framework, is known as the Common Language Runtime (CLR). The CLR provides the appearance of an application virtual machine.

neuro-evolution: The use of ***genetic algorithms*** to create an operational artificial neural network. The term is used ambiguously in the literature to refer both to systems that evolve only the values of the connection weights for a network of pre-specified topology as well as to systems that evolve the topology of the network in addition to the weights.

nondeterministic: A property of a computation that may have more than one result. One way to implement a nondeterministic algorithm is using backtracking; another is to explore (all) possible solutions in parallel.

non-monotonic: The property of a type of information and a method of reasoning or operating on it that the addition of new information may decrease the set of valid inferences or operations; not ***monotonic***.

NP-complete: A decision problem that requires a ***nondeterministic***, polynomial-time (NP) algorithm to solve; an NP-complete problem is among the most difficult-to-solve NP problems. Formally, a decision problem is NP-complete if it is in NP (requires a nondeterministic, polynomialtime algorithm to solve), and every other problem in NP is reducible to it.

n-version programming: A software-development strategy to increase reliability through redundancy. One approach is to create (possibly independently) multiple versions of a component and to combine their results.

objective function: A function to be minimized or maximized in a ***metaheuristic*** optimization process.

off-the-shelf components: Components designed and implemented for specific purposes but with no specific application in mind; such components can be used in a variety of applications, sometimes with external scaffolding. A program library is an example.

open abstraction: An abstraction or component whose implementation can be customized or augmented.

orchestration: The activities needed to make the elements of a system work together in sufficient harmony to ensure continuous satisfaction of a set of specified objectives.

parallel: When the execution flows of several computational processes are able to run simultaneously, typically with little or no resource sharing. In general, parallel execution is expected to save elapsed time over sequential execution (cf., ***concurrent***).

particle swarm optimization: A problem-solving ***metaheuristic*** that mimics the behavior of a flock or swarm. A population of individuals seeking to find the most fit location is placed randomly in a multi-

dimensional search space. Each individual flies through the space with a velocity that is updated according to a linear combination of the current velocity, a vector toward the most fit location that the individual has seen so far, and a vector toward the most fit location seen so far either by nearby individuals (in one variation of the algorithm) or by the entire flock (in another). The latter two vectors are combined with independent random scalars. The process terminates when the location with the best fitness exceeds a given threshold.

pattern: A description of a particular recurring design problem that arises in specific design contexts along with a well-proven solution for that problem. In some cases, the solution is specified by describing its constituent participants, their responsibilities and relationships, and the ways in which they collaborate.

phase transition: A process by which a system changes from one state to another with different properties, sometimes as a result of small changes.

phenotropics: A mechanism for component interaction that uses pattern recognition or artificial cognition in place of function invocation or message passing. The term was coined by Jaron Lanier. See John Brockman, *The Next Fifty Years: Science in the First Half of the Twenty-first Century*, Vintage, 2002.

platform: The combination of hardware and software that provides a virtual machine that executes software and applications. Software platforms include operating systems, libraries, and frameworks.

possibility theory: A mathematical theory for dealing with particular types of uncertainty as an alternative to probability theory. Possibility theory was defined by Lotfi Zadeh in 1978 as an extension to his theory of ***fuzzy sets*** and fuzzy logic.

probabilistic: Pertaining to any method, approach, or form of reasoning that relies on probability or theories of likelihood (cf., ***stochastic***).

proof-carrying code: A technique for safe execution of untrusted code. A code receiver establishes a set of safety rules that guarantee safe behavior of programs, and the code producer creates a formal safety proof. The receiver can use a proof validator to check that the proof is valid.

quality attribute: A property of a system by which its quality will be judged by some stakeholder or stakeholders. Quality-attribute requirements such as those for performance, security, modifiability, reliability, and usability have a significant influence on the architecture of a system.

quality of service: The probability that a system will deliver particular levels of measurable computational and communication properties such as availability, bandwidth, ***latency***, and ***jitter***. Policies and mechanisms typically are designed to control and improve the quality of service of a system.

quasi-stability: The ability of a non-equilibrium state to be long lasting but not perpetual; ***metastability***.

recovery-oriented programming: An approach to software reliability that focuses on recovering from faults quickly and effectively. This approach was devised by David Patterson and his colleagues (Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies, Computer Science Technical Report UCB//CSD-02-1175, University of California at Berkeley, March 15, 2002).

refactor: The process of rewriting software to improve its readability or structure while retaining its meaning or behavior.

reflection: A computational process that is able to reason about itself.

requirements drift: A slow variation in the requirements for a system as conditions change, including as a result of experience with the system or as the set of stakeholders changes.

resiliency: The capability of the ULS system to maintain an acceptable level of service while under stress from adverse environmental conditions such as attacks or cascading failures.

revelation principle: The principle in ***game theory*** or ***mechanism design*** that states that for any equilibrium (stable state) of a game of incomplete information, there corresponds an associated revelation mechanism that has an equilibrium where the players truthfully report their payoff-related, private information.

robust; robustness: The ability of a system to continue to function despite the existence of faults in its component subsystems, parts, or communication mechanisms.

satisfice: To seek a solution that satisfies the minimum requirements necessary.

security: The capability of a system to provide confidentiality, integrity, and availability of data and services, both locally and globally.

self-organizing criticality: A ***driven system*** that radically changes its behavior or structure because of its intrinsic dynamics. The archetype of a selforganized critical system is a sand pile. Sand is slowly dropped onto a surface, forming a pile. As the pile grows, avalanches occur that carry sand from the top to the bottom of the pile.

sequence-based specification: A method of specification distinguished by a process of enumerating all stimulus-response pairs as well as all sequences of stimuli along with their required responses, including whether a sequence is not possible, and the identification of equivalence classes of sequences. The method is described in "Sequence-Based Specification of Deterministic Systems," by S. Prowell and J. Poore, *Software - Practice and Experience 28*, 3 (Mar 1998): 329-344.

service-oriented architecture: A design for linking computational resources (principally, applications and data) on demand using standardized (typically network-based) interfaces and protocols to achieve the desired results for service consumers (which can be end users or other services).

simulated annealing: A problem-solving or optimization ***metaheuristic*** that finds a good approximation to a global optimum by using a process inspired by metallurgical ***annealing***. It operates by repeatedly considering a random nearby solution, and selecting that solution with a probability that depends on the difference between the current fitness and desired fitness, and on a global temperature that decreases according to a schedule. The process terminates after, at most, a fixed number of steps.

socio-technical ecosystem: An ***ecosystem*** whose elements are groups of people together with their computational and physical environments.

speciation: The formation of new and distinct species in a process of natural or ***digital evolution***.

staged computation: Breaking the construction of a program into stages whose outputs are programs. *Partial evaluation* is an example of staged computation.

statistical mechanics: The branch of physics that makes theoretical predictions about the behavior of a macroscopic system on the basis of statistical laws governing its component particles.

stochastic: Involving chance or probability; involving or containing a random variable or variables. For example, a stochastic process is one whose behavior is ***nondeterministic*** in that the next state of the environment is partially but not fully determined by the previous state of the environment.

swarm intelligence: The collective behavior of decentralized, self-organized systems. Swarmintelligence systems are typically made up of a population of simple agents interacting locally with one another and with their environment. Local interactions between such agents often lead to the emergence of global behavior. Examples from nature include ant colonies, bird flocking, animal herding, bacteria molding, and fish schooling.

system health: A measure of the ability of a system to deliver its required functionality at its specified quality levels.

system of systems: A system comprising independent, self-contained systems that, taken as a whole, satisfy a specified need.

system viability: The success or continuing effectiveness of a system.

test-driven design: A style of program design that begins by writing one simple test, then writing just enough code to pass it. Then another simple test is written, and code is added to pass both it and the previous test. The programmer then looks for opportunities to improve the code by generalizing it, removing duplication, restructuring it, or making it more understandable. The test-codeimprove cycle repeats until there are no more tests to be had. It is claimed that a good global design emerges from the need to decouple the code to make tests run fast and the local heuristic rules for code improvement. The tests are retained and run frequently to prevent unintended effects of changes to the design.

trust: The extent to which users of a system can rely on its data and services.

type-safe staged computation: A ***staged computation*** using a language that guarantees that every generated program is type safe.

ubiquitous computing: The integration of computation into the environment; this is in contrast to computers as distinct objects.

ultra-large-scale (ULS) system: A system at least one of whose dimensions is of such a large scale that constructing the system using development processes and techniques prevailing at the start of the 21st century is problematic. ULS systems exhibit the following characteristics: decentralization; conflicting, unknowable, and diverse requirements; continuous evolution and deployment; heterogeneous and changing elements; erosion of the people/system boundary; and normal failures of parts of the system.

universal usability: The characteristic of an information or communications device that it is usable by any person regardless of skill level, knowledge, age, gender, disabilities, disabling conditions (mobility, sunlight, noise), literacy, culture, income, etc., and regardless of the number of (simultaneous) users.

user-centered design: A design philosophy and process in which the needs, wants, and limitations of the end user of an interface or artifact are given extensive attention at each stage of the design process.

validation, software: Confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses and that the requirements implemented through software can be consistently fulfilled.

verification, software: Evidence that a design meets all of its specified requirements.

Von Neumann execution model: A model of computation consisting of a central processing unit (CPU) and a single memory for instructions and data.

wicked problem: An ill-defined design and planning problem having incomplete, contradictory, and changing requirements. Solutions to wicked problems are often difficult to recognize because of complex interdependencies. This term was suggested by H. Rittel and M. Webber in the paper, "Dilemmas in a General Theory of Planning," *Policy Sciences 4*, Elsevier Scientific Publishing Company, Inc., Amsterdam, 1973.

Bibliography

[Feiler et al 2006]

Feiler, Peter; Gabriel, Richard P.; Goodenough, John; Linger, Rick; Longstaff, Tom; Kazman, Rick; Klein, Mark; Northrop, Linda; Schmidt, Douglas; Sullivan, Kevin; Wallnau, Kurt. *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Software Engineering Institute, Carnegie Mellon University, 2006. ISBN: 0-9786956-0-7. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=30519

[Lee et al 1999]

Lee, C., Lehoczky, J., Rajkumar, R., Hansen, J. A Scalable Solution to the Multi-Resource QoS Problem. 315–326. *Proceedings of the 20th IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, 1999.

Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

Phone: 412/268.5800 | 888.201.4479
Web: www.sei.cmu.edu | www.cert.org
Email: info@sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0004468