An Object Oriented Methodology Integrating Design, Analysis, Modelling, & Simulation of Systems of Systems

Judith D. Richardson and Thomas J. Wheeler

US Army CECOM, Space and Terrestrial Communications, Ft. Monmouth, NJ

Abstract

Development of "systems of systems" outstrips the current software systems development strategy. This paper reports on a development strategy which integrates design, analysis, modelling & simulation using a methodology which integrates aspects of modular decomposition, object-oriented design, and distributed system design. It illustrates the methodology on the design of a high bandwidth communications infrastructure for tactical armed forces.

1: Introduction

The recent trend of developing computer systems which are part of a larger system or network is changing development strategy by merging disciplines, integrating development paradigms, and using a collaboration of perspectives and expertise. Integrating the disciplines of design, analysis, modelling, and simulation provides coupling of the insights developed by each discipline and feedback when the scope of design exceeds the designer's intuition. Conceptual integrity of the system and the rational organization of its development fostered by top-down development paradigms along with the evolvability of systems and the flexibility of its development process fostered by bottom-up development paradigms are essential in this class of systems. Effectiveness in the interoperation of the component systems in "systems of systems" is a more important criterion than the functioning of the individual system, thus effective collaboration in their design is essential.

Systems of systems are complex enough that their design outstrips the intuition of their designers; but design is driven by intuition and intuition comes from understanding. We are finding that integration of design, analysis, modelling, and simulation provides a solution to this dilemma. Tighter integration of design and analysis makes the design process iterative, validating design concepts and developing insight while making progress on the design. Integrating modelling facilitates design by synthesis from components; modelling is compositional and allows systems to be envisioned in terms of their subsystems. Integrating simulation, both of performance and of behavior, allows designers and users to develop a feel for aspects of these systems which could not be envisioned intuitively because of the system's structural complexity. Integration of these disciplines enables envisioning of large scale behaviors needed for conceptual integrity, while fostering system and process flexibility needed for system evolution.

Conceptual integrity [7] and evolution of these systems must both be addressed. Conceptual integrity is required to accommodate a large and diverse user community and because the largness of these systems requires a common, understandable conceptual image of the system. The evolution requirement comes from the long lifetimes of these systems and the need to grow from and work with existing systems in organization infrastructures. We address these seemingly opposing requirements by integration of development paradigms, providing the design unity of the top-down style along with the design flexibility of the bottom-up style.

In system of systems development, assessment of a component system's design concepts must be done in terms of the next larger system; likewise, the design of the system of systems must be done by integration. This "one level of indirection" in the thought pattern strains the design cycle, leading to design issues at the level of the system of systems being missed or their impact underestimated. Collaboration of the designers of the individual component systems along with designers of the system of systems is essential, with the integration of design, modelling, simulation and analysis, providing the team with the insight needed to jointly assess design decisions and the organized compositional nature of the methodology structuring this collaboration.

The work reported here describes a methodology, which resulted from experience with each of these issues in ongoing development efforts of two DoD systems of systems. We will first give some background providing a context for the work. We will then outline a rational object-oriented software strategy which addresses the issues by synthesizing the disciplines and paradigms mentioned above. That will be followed by a description of a methodology which implements this strategy. An ongoing system of systems

development will be used as an example of its use. Finally, some conclusions are drawn about the methodology and its use.

2: Background

The Department of Defense is currently enmeshed in the initial stages of the development of integrated "systems of systems", with the aim of providing "seamless" automation support for the operation of the armed forces worldwide. Currently, individual systems are developed independently, with "interoperation" requirements on their development providing the first step in meeting the integration needs of the larger system composed from these systems. This state of affairs is evolving into joint technical efforts to develop the architectural basis for integrating the systems into a seamless infrastructure. This paper results from work on two of these architectural integration efforts, the Army Tactical Command and Control System (ATCCS) software architecture effort and the DoD seamless communications architecture effort called "Global Grid".

The architecture efforts are focusing on the evolution of the existing collection of individual networks of communications and processing systems, moving towards the creation an integrated system of systems providing a seamless communications and processing infrastructure. As a system of systems is an interacting set of systems in which the behavior is ascribed to the interacting set rather than the component systems, the development strategy must start with a defining description of the behavior of that set of systems. This architecture must then be implemented, and evolved, by composition of the component systems, allowing iterative replacement of component systems and interoperation with existing systems. New component systems be cooperatively designed as open systems, with well designed standards based interfaces [7] to cooperating systems, fitting within this architecture framework.

Computer and software systems are currently developed via a functional, top-down strategy [2]. When applied to the development of systems of systems a number of fundamental problems, relating to evolution of systems, have surfaced. Object-orientation has been proposed as a remedy for this situation [3], but it has little support for architecture. Modelling and simulation are used as an auxiliary discipline, justifying decisions after they are made. Four main aspects of this development (and modelling) strategy make it inappropriate for the development of systems of systems:

(1) The strategy is accomplished top-down, making design decisions starting from the user visible decisions and ending with the implementation decisions, with each decision only being constrained by higher level decisions, thus (explicitly) ignoring lower level commonalities, only allowing reuse in an ad-hoc manner and impeding performance engineering via high level structuring.

(2) The strategy focuses on describing the requirements and structure of the units of development, which in a system of systems are the component (sub)systems, with the result that higher level requirements must be met by interoperations requirements on the units of development.

(3) The descriptions in this strategy are functional, defining the behaviors explicitly, leaving the objects implicit, leading to a complicated mapping from those functions to the system's (and its model's) structure, resulting in difficult and expensive system maintenance and model validation.

(4) The models and simulations used in the process are developed and used by a separate, independent process, with different structures than the system, resulting in separate validation of models and systems, limited impact of the information and insight extracted from models and modelling, and the inability of models and components interoperating thereby making it difficult to construct hybrid system-simulation experiments.

We have found that integration of modelling, simulation, and analysis with design, along with synthesis of concept from modular decomposition, which works within the top-down strategy and has strong support for structuring, concepts from object-oriented development, which works bottom up and has strong support for evolution, along with concepts from distributed systems, provides a development strategy which solves all of these problems.

3:Rational Object-oriented Software Strategy

The Rational Object-oriented Software Strategy (ROSS) addresses all four of these deficiencies. It is an object-oriented strategy which combines a rational development process [5] (which is top-down in its documentation of the system, but more flexible in its ordering of work) with bottom-up organization of the commonalities inherent in a system domain and reuse of common concepts. The strategy has a strong influence from distributed systems design [13] and integrates modelling [14], simulation, and analysis into the design process so that insights derived through modelling drive [10] the design. There are five aspects inherent in this strategy:

(1) A system and its operating environment are envisioned and abstractly described using a hierarchical model of cooperative distributed objects, based on the perceived physical structure of the envisioned system in its environment. The model has a one to one structural correspondence with the system and its operating environment and models the system of systems along with the (sub)systems in a unified way.

(2) Commonalities in the system's domain are captured by the model being structured in accordance with a generic architecture [11], which is appropriate for systems

in that system's domain, and being constructed from model components arranged in a classification hierarchy [1] for systems of that domain.

(3) The model provides the framework for both the design of the software objects which interact to produce the system and the simulations which provide insight into the effectiveness of the system's functions and the performance of the system.

(4a) The system is constructed from objects which are developed or reused from the system component library and composed into the system, which only has to have integration testing before being fielded.

(4b) The model components are developed or reused from a model library and composed into models which are validated compositionally, analogous to integration testing, before being used.

(5) The documentation of the products of each phase of this methodology is organized in a top-down dependency structure, reflecting the rational structure of the process, but the methodology does not force a top-down process structuring, as the standard process model does.

Each of the deficiencies in the standard development process are remedied by the rational object-oriented software strategy:

(1) While the system's development is presented top-down in the documentation, it is not constrained to be top-down in its work flow [5]. The work is only constrained in that documents produced by tasks (accomplished in a mixed top-down, bottom-up manner) be properly placed in the documentation structure and that each task (and its document's contents) be dependent on the results of tasks which they are presented to be dependent on. This way of concentrating on a rational documentation structure naturally accommodates reuse of preexisting concepts and components and is a natural way to develop members of system, and component, families [6].

(2) Systems of systems are naturally described as services provided by distributed systems. Distributed services provides a framework for describing the higher level system functions [4], while hiding the distributed nature of their implementation. Distributed functions provides a basis for describing the cooperation and coordination of the component (sub)systems' functions in achieving these services.

(3) The descriptions in this strategy are consistently object-oriented; firstly, in that they naturally follow the physical object structure in the external world and in the (desired) system; and secondly, in that the methodology uses object-oriented concepts in the development of its modelling, design, and implementation products. Because of the consistency of the development style through each of the phases, the mappings from model to design and thence to implementation are simple (close to one to one) thus enhancing the maintenance process and simplifying model validation.

(4) The models and simulations used in this methodology are developed using a process which is unified with the design process, thus have the same structures as the system and have compatible interfaces with the system's components, resulting in a unified validation of the models and the system's design. This fosters the impact of the information and insights extracted from the models on the design process, eases the transition of design ideas into the models, and also provides the ability of constructing hybrid system-simulation experiments.

4: Methodology

We implement this strategy by a methodology which is model centered, using models to capture the system's specification, as well as for analysis and simulation. The methodology defines a design process operating within a descriptional framework and supported by an environment [9]. This section first describes the framework, then describes the design process, and finally outlines the environment needed to support the methodology.

4.1: The Framework

The model and system development are described with respect to a framework that encompasses four domains: the external world (where things exist and interactions happen), the individual mental world of each development participant (where the perception of the real world takes place), the joint modelling and documentation world (where the understanding of the real world is analyzed, developed, and validated), and the implementation world (where the systems and simulations are constructed). Within each domain there is a set of parallel concept[1], which describe different aspects of how we understand a domain (viz. the existence & interaction aspects of things, the occurrence of events, the temporal aspects of the events, the locational aspects of the things and events, and other characteristics of things and their interactions).

The mental model developed by the modeller (in the second domain) from observations of the real world (the first domain) is documented (in the third domain) using an object model in which individual things are represented as objects, which are abstract representations portraying behaviors in abstract interfaces, defining each of the activities which the object participates in. The objects are classified

[1] We have found it necessary to have different terminology for the same aspect in different domains since the object-oriented paradigm, as represented by the phrase "everything is an object", leads to confusion, for instance, whether one is talking about the real world object or its computer representation.

into types and/or classes[2], factoring the similarities among objects and providing high level, protected abstractions which ease valid modelling. The types are organized into hierarchies by means of generalization, capturing similarities in the semantics of types, simplifying the understanding of the objects and fostering reuse. Each object has a number of aspects, important from different points of view, which are provided as different interfaces to that object, and each exists at a number of resolution levels.

The models are conceived and analyzed (in the third domain) as inherently distributable objects providing location transparent services to using objects, and are implemented by distributed objects (in the fourth domain) in a network of systems cooperating to provide the "system of systems" level functions. The distributed interactions take place through abstract interfaces designed to hide the aspects of distribution from the using objects [4]; thus designing (the interfaces of) distributable objects is not different from designing (the interfaces of) local objects. The implementation of distributable objects, on the other hand, requires the knowledge of the distribution of the implementing objects in order to provide the characteristics expected of distributed services, like fault tolerance. Thus there are two complementary aspects of making use of distributed services, the abstraction from distribution provided by the definition of distributed services and the usage of distribution provides by distributed implementation of those services.

Systems of systems are implemented as a collection of cooperating systems developed by a collection of cooperating development projects, each developing and evolving a system which has both a directly usable system interface and an application programming interface (API). The cooperation APIs are specified jointly in a cooperative design/modelling effort which both defines the system of system's behavior which is synthesized form each component system's cooperation interfaces and defines the cooperation interfaces for each of those systems which specifies that part of each system's design.

4.2: The Design Process

The methodology for developing models and systems has activities which take place in the domains described by the framework. A modeller, or designer, observes the real world in which the system exists, or will exist after development, focuses of a part of the real world that is of interest through the perspective of the developing system and develops a mental perception of that part of the real world. The designer interprets various things in the real world by means of two complementary paths. The designer may form analogies of them and relationships among them in terms of familiar concepts which the modeller has a valid intuition for, and develops a mental image of the (interesting part of) real world. The designer may also develop these images by abstracting categories of things and relationships, rather than analogies, but the result is similar. The mental image will have some structure, properties, and behaviors which can be informally compared to the real world and iterating the understanding of the real world embodied in the mental model so that over time the mental model becomes a satisfactory starting point for the more formal, external modelling process. Distribution over time and space is implicit in the mental model. The modeller may make a formal (mathematical) model of the perceived part of the real world to supplement to this process .

Once the modeller has a satisfactory mental model, the next part of the methodology consists of capturing the mental model in an external abstract representation so that that model may be discussed, criticized, analyzed , and organized as suitable to serve as a valid model of the system. The capture and presentation process makes use of object-oriented modelling techniques, representing and classifying aspects of objects in a one-to-one correspondence with the real world objects. The modeller develops a model structure which corresponds to the real world structure by integrating the views of those aspects of the objects and develops model concepts at a number of resolution levels corresponding to the levels of observations which should be made on the models and the system. The modeller does a number of analyses of the system and its environment making use of the concepts at many of the resolution levels, approximately following a coarsest through finest grain resolution level strategy, incrementally increasing her understanding of the system while increasing the resolution of the model.

The modeller makes use of techniques and thought patterns from the modular decomposition strategy and the object-oriented design strategy, and the object-oriented database design strategy. The modular decomposition methodology develops an architecture for the system by interconnecting modules by their interfaces. Within that architecture a model is composed from a set of objects interconnected by their interfaces, as components to make up structures and as related objects to model interacting things. All interactions take place via actions or activities defined on or among object's abstract interfaces. Individual objects and structured models have attributes, actions, and participate in behaviors; each has a number of views (of the properties and actions) through which they participate in these behaviors, with certain views allowing participation in certain behaviors, e.g. an event view allowing participation in performance behaviors while a functional view provides participation in functional behaviors. Each object and model also has a number of resolution levels for each view which allows interactions, and observations of

[2] Type in an intensional concept, providing definitions for objects of that type; class is an extensional concept grouping together objects which are similar in some essential way into sets.

those interactions, to take place at varying temporal, spatial, and functional resolution levels to fit the varying purposes of the model and the varying level of resources appropriate to those purposes. The modeller creates a system structure by decomposition of the system according to a generic architecture, followed by composition of the system by populating that architecture with components from a library of components common to the domain. Using object-oriented techniques, the modeller defines object types which are appropriate to the type of system being modelled, classifies them into categories based on similarities, organizes hierarchies of types both more general than those types and more specialized, thereby introducing new types, which are also generalized and specialized, sketches implementations of all of the types, discovering more new types, and repeats this process until closure is reached and no new types are introduced. In this process, the thinking about the system is implicit, being based on much experience with the development of this type of system, and so, the thinking is mostly confined to understanding what classes of objects exist in this type of system and what kinds interactions these types of object participate in. We describe this kind of thinking as "taking place in the type world" while the kind of thinking that takes place in the module oriented approach is described as "taking place in the component and system world". Object orientation is enhanced here by developing, in parallel with the library development, an architecture framework for systems, organizing the structure of systems while guiding the structure of the class hierarchies.

The system description in our methodology is a model based, rather than functional specification of the system, with the model being described in terms of distributed services and objects. It structures systems and models in a conceptual architecture structured into layers providing location transparency of the system's objects and services on top of distributed functions which are needed to realize those services.

Validation is based on three complimentary techniques. The first is classical validation in which models are implemented and used in experiments in parallel with (the appropriate parts of) the real system, with parallel observations being made and compared, ascertaining that the system and the model (were observed to) behave the same under the same circumstances. The second technique is knowledge based, wherein the modeller develops valid knowledge of the system via scientific understanding of the system's subject matter, implementation technology, structure, and usage, along with a series of varying resolution analyses, which produce consistent results, followed by applying that knowledge in producing the model. The third technique is structural, with the modeller composing the system out of components, using the validity preserving composition technique of composing via the connection of components' interfaces and defining the interface(s) of the system by an abstraction function, in terms of the components' interfaces. Note that only the second and third techniques are possible on systems of systems, as they are too large and expensive to validate classically and their component systems usually exist, and are the units of development, and that the third technique is dependent upon classical or knowledge based validation of (only) the lowest level components.

The valid model is used to design the system or any simulations needed during the development process, with each being structure in a one-to-one correspondence with the model structure, and thus with each other. Simulations, in this methodology are merely implementations of some resolution level of the model, viewed from one of the perspectives. The simulations are placed in a experimentation environment to run the experiments necessary. The structural, performance, and functional insights from the simulation and system developments guide the other development, forming a synergy in the development process. This synergy, along with the synergy of the modelling process with both of these design processes and with the analysis process is the main source of increased effectiveness in the system development process of this methodology.

4.3: The Environment

The environment which supports and makes use of this framework and methodology consists of three parts; the development environment, for developing systems and their components, models and their components, and simulations and their components; the run time environment (kernel) for simulations or systems, and the experimentation environment, for simulations. The requirements for each of these environments have been developed and the architecture of each has been designed; the requirements and architectures are described in a separate report [12]. The development environment architecture has a standard user interface component with a standard programming interface (API), an abstract interface to a structured documentation database which resides in an object database, a standard multi-language communications interface implementing distributed module's interfaces using remote procedure calls (RPCs) using a transport independent messaging protocol. Various commercial and academic components are being investigates, with the aim of implementing this architecture using available parts. Synchronization and consistency of the services is maintained by a run time system which provides language independent and distributed interaction by providing resources to the interfaces which transparently handle interrupts, atomic actions, transactions as well as language interfacing using the same RPC mechanism as used in the development environment. The distributed discrete event kernel provides like facilities to simulation systems and components and is integrated with the run time system when combined real com-

ponent - simulation component systems are run [8]. The experimentation environment interfaces simulations to driving models, described in terms of operational scenarios, and to observation software, which allows queries on the event database generated by the simulation.

5: An Example

To make the structure and process described above more concrete, we will outline development of a system of systems. An example of the use of the methodology is a system of systems that we are currently working on, the military B-ISDN system. We are currently in a multi-organization development of a prototype of the system, using a combination of technical and operational testbeds, along with modelling, simulation and analysis, to develop the system's architecture, functionality, and capabilities, for the purpose of guiding the military communications system's evolution. In this example, we will focus on the broadband system of system's infrastructure spanning the area from the commercial network and the upper tactical echelons, in which the trunks will be optical fiber, down through the lower echelons, where the trunks will be radio based.

Systems of systems development is done by integration, at one or more levels above the level at which systems are procured or developed. For this example, the Military B-ISDN system of systems will be described as consisting of one level above the systems out of which it will be constructed: computer workstations, user-network interfaces, strategic ATM switches, optical fiber based trunks, tactical ATM switches, network management systems, and radio based trunks. The first phase takes a low resolution perspective on the system of systems as a whole. The objects and services visible to the users of the system of systems are defined, modeled and analyzed. This results in the high level architecture which provides information transport services, capable communication of video and multimedia information, between users using fixed rate and variable rate virtual channels; the architecture also provides system management services, but these will not be covered in the example. These services are defined in accordance with the ATM UNI specification.

The second phase takes these services and objects and "zooms in" one level to make visible to the design the objects and services at the system level. The user visible services and objects must be realized in terms of the services and objects provided by the component systems. As these systems are separately developed, this levies a requirement on the design of the system of systems that it be realized by standards and protocols required to be implemented by the systems. At the level of systems, the system of system's communications services are required to be provided by UNIs at each end of the virtual circuit, a sequence of strategic and tactical ATM switches, and both optical fiber and radio based trunks.

The methodology builds upon the standard communications system design paradigm. The upper level (system of systems) functions and objects are allocated to the systems at the next level by a three step strategy: the functions and objects are first allocated to layers in a layered architecture, then the layers are allocated to system components in a generic system architecture, and finally the system components are designed to provide the functions and objects which implement the services allocated to each component. The services of the upper echelon military B-ISDN system of systems to the User Network Interface (UNI) and the strategic ATM switch are mapped onto the ISO-OSI seven layer architecture with, for instance, routing, switching, and end to end delivery allocated to layer three in the strategic UNI.

Finally, the services of the lower echelon military B-ISDN system of systems to the tactical User Network Interface (UNI) and the tactical ATM switch are mapped onto the ISO-OSI seven layer architecture and those layers then mapped onto the generic architecture. It is these mappings where the design of the tactical trunks and the tactical switch differs significantly from their strategic counterparts, with for instance, routing, switching, retransmissions, and end to end delivery allocated to layer three in the tactical switch since error correction and dynamic rerouting protocols are necessary due to noise, distortion, and fading on wide band digital radio based trunks. Thus the tactical switch, tactical trunks, tactical architecture, and some of the network management are "ATM compatible" rather than ATM based, in this design.

6: Conclusions

The framework, methodology and environment presented here facilitate and integrate the design, analysis, modelling and simulation aspects of large communications systems of systems. They integrate all of these disciplines provide high productivity, foster incremental development, and facilitate development and validation of systems from components. The design process synthesizes concepts from the functional modular development methodology, object-oriented development, database development, and model development, enhancing each with concepts from the others, thus utilizing the strengths of each to make up for the weaknesses of others. The analysis process provides an incremental set of analyses, organized by the model structure, building understanding of the system, its characteristics, and its usage while answering questions inherent in the design process. The modelling process develops an understanding of the system and its development issues, structured in synergy with the design process, organizing the functionality and performance issues of the system earlier than is currently possible. The simulation process grows from the modelling and design process, using the same structures that they use, and providing components which can be used to compose the simulations, be re-

used in other needed simulations, and used with real components in combined simulations, to achieve virtual systems for experimentation and training.

References:

[1] Habermann, A. Nico; L. Flon, & L. Cooprider; *Modularizaion and Hierarchy in a Family od Operating Systems*; **Communications of the ACM**; Vol. 19, No. 5; May 1976.

[2] Lamb, Alex; **Software Engineering, Planning for Change**; Prentice Hall; 1988.

[3] Meyer, Bertrand; **Object Oriented System Construction**; Prentice Hall International; 1988.

[4] Morris, Derek and Thomas Wheeler; *Distributed Program Design in Ada*; **Proceedings Ada Applications and Environments Conference**; IEEE; 1986.

[5] Parnas, David and Paul Clements; *A Rational Design Process: How and Why to Fake It*; **IEEE Transactions on Software Engineering**, v.2 (Feb.1986).

[6] Parnas, David; *On the Design and Development of Program Families*; **IEEE Transactions on Software Engineering**; Vol. SE-2, No. 1; March 1976.

[7] Richardson, Judith and Thomas Wheeler; *A Two Layered Interfacing Architecture*; **Journal of Standards & Interfaces**; v.13 (1991) North-Holland.

[8] Richardson, Judith; **An Architecture for Interfacing Systems and Simulations**; CECOM Report; 1993.

[9] Richardson, Judith and Thomas Wheeler; **The Rational Object-Oriented Software Stratigy; A Methodology for Integrating Design, Analysis, Modelling, and Simulation for Systems of Systems**; CECOM Report; 1993.

[10] Ross, Robert G.; private conversations 1991-1993.

[11] Wheeler, Thomas J.; **Software System Development Through the Use of Formal Documentation**, (PhD Dissertation); Stevens Institute of Technology; 1988.

[12] Wheeler, Thomas and Judith Richardson; **The Architecture for the Rational Object-Oriented Software Stratigy (ROSS)**; CECOM Report; 1993.

[13] Army Tactical Command and Control System (ATCCS), **Common ATCCS Support Software (CASS) Software Architecture Specification**; CECOM; 1990.

[14] Zeigler, Bernard; **Theory of Modeling and Simulation**; Krueger; 1986.

Acknowledgements

The authors are deeply indebted to our mentor, Robert G. Ross, for the many hours of brain-stem storming and bagel munching that resulted in much of this work.

Biographies

Judith D. Richardson is a Computer Scientist with the Research Team of the Modelling and Simulation Branch, Communications Engineering Division, Space and Terrestrial Communications Directorate, CECOM, Ft. Monmouth NJ. She received her B.S. in Computer Science for University of Maryland, 1980 and her M.S. in Software Engineering from Monmouth College, 1991. She is currently working on her Ph.D. at Stevens Institute of Technology in Electrical Engineering and Computer Science. She also holds a degree in Geology. Her current areas of research are Interface Technology, Software Architecture, Software Design, Modelling and Simulation, Formal Methods, and Software Engineering Environments.

Dr. Thomas J. Wheeler is an Electronics Engineer with the Research Team of the Modelling and Simulation Branch, Communications Engineering Division, Space and Terrestrial Communications Directorate, CECOM, Ft. Monmouth NJ. He is also a faculty member at Stevens Institute of Technology and Monmouth College. He received the Ph.D. degree in Electrical Engineering and Computer Science from Stevens Institute of Technology in 1988 and has degrees in Physics, Electronic Engineering and Computer Science. His current research interests are Software Engineering Methodologies, Software Design, Formal Techniques, Modelling and Simulation, Programming Languages and Principles, Databases, Distributed Systems, and Environments/Operating Systems