By Dov Dori, Guest Editor

Conceptual Modeling and System Architecting

Just as engineering drawings express the 3D structure of mechanical and architectural designs, conceptual models are the means by which complex software-intensive systems, as well as systems in the more general sense, are conceived, architected, designed, and built. The resulting models are analogues of the mechanical design blueprint. The difference, however, is that while blueprints are exact representations of physical artifacts with a precise, agreed-upon syntax and long tradition of serving as a means of communication among professionals, conceptual models are just beginning to make headway toward being a complete and unambiguous representation of a system under development. The articles in this special section gently walk you through the abstract world of systems analysis and architecting by means of conceptual modeling and how to evaluate and select models, along with the methods used to construct them.

How to evaluate and select the methods and tools for building conceptual models that faithfully represent the abstract concepts of a software system's architecture.

Illustration by Istvan Orotz

Systems exist in nature as well as in virtually any conceivable area of human activity. We live within and are surrounded by systems. They are the focus of research and development in many fields of human endeavor. In recent years, systems in general and software-intensive systems in particular have grown tremendously in both complexity and number of disciplines involved. These trends have started drawing the attention of researchers and system architects to the growing interdisciplinary domain of systems science and engineering. This emerging "meta-discipline" is founded on the observation that a number of fundamental principles (such as hierarchy, modularity, inheritance, and feedback), acting on a compact set of entities (such as stateful objects and the processes that transform them) are common to all systems, regardless of their domain, discipline, or source. This observation has opened the door to the potential emergence of a simple, powerful lingua franca for systems development.

SYSTEM ARCHITECTING IS A KNOWLEDGE-BASED INTERACTION INVOLVING STAKEHOLDERS AND DEVELOPERS AIMING TO ACHIEVE A SYSTEM'S FUNCTION.

Like biological systems, many contemporary artificial complex social and man-made systems have evolved over years of human history without an explicitly stated, predetermined, well-defined goal. This is especially true for systems with an intensive human component, namely organizations of various kinds. Still, in retrospect, by examining a system's architecture, or its structure-behavior combination, one can usually infer its function, that is, the goal or purpose it serves. This "reverse engineering" process is not required when system architects face the task of architecting a new system.

A major early stage in any artificial system's life cycle is creating its architecture—devising a concept of structure and operation that would best attain the system's goal, such that its stakeholder's intent is satisfied. A key success factor is disambiguating the *what-how* entanglement, clearly separating the function of the system from its behavior. The latter is tightly intertwined with the system's structure; together they account for the system's architecture.

Function is a problem-oriented concept, specifying what goal the system is expected to achieve, while architecture is a solution-oriented concept, specifying how the system's function is to be achieved by a specific architecture. The article by Soderborg et al. emphasizes the what-how interplay and its representation in terms of interconnected objects and processes. It defines the whats and the hows in terms of problem and solution spaces. The authors propose a framework for conceptual representation of systems based on the Object-Process Methodology (OPM). This template-based approach enables a rigorous what-how decomposition that is both theoretically sound and practically applicable as a system-architecting guideline for software-intensive systems, as well as for a variety of other types of complex systems.

Complex systems are hardly ever architected by one person. Rather, architecting is a knowledge-based activity that consumes ample human interaction among stakeholders and developers at various levels. Due to the diverse nature of system architecting processes and their subsequent phases, no single monolithic standard development methodology spanning the entire system life cycle is always suitable for the task at hand. Various methods that suit specific needs of certain types of organizations in certain types of development projects can be used. Effectively combining these method components helps construct a system life cycle support methodology.

This mix-and-match approach is at the heart of the article by Henderson-Sellers. Method engineering can be applied to construct a full methodology from method elements or fragments typically stored in a repository, underpinned by a metamodel.

Regardless of whether the systems development process follows a recognized "software process" approach or is tailored through method engineering, system architects and developers can turn to a collection of modeling tools. Systems and models are therefore intimately related. Modeling is a human activity of generating models, or abstract artifacts representing systems. Models show certain aspects of that reality, including function, structure, and dynamics, as perceived or envisioned by the human modeler or system developer. Historically, some of the earliest models were maps of some geographical region or scaled versions of buildings, ships, and other artifacts to be constructed. As science and technology progressed, model types proliferated to include scientific models that manipulate mathematical and chemical symbols. Early engineering models included such artifacts as mechanical drawings, electrical schemes, and microprocessor layouts. Their common thread was and continues to be the fact that they represent designs of physical systems.

Such models have evolved over centuries. Models of concepts that are themselves informatical and hence more abstract began to emerge only in the 1970s and 1980s with such conceptual modeling schemes as Entity-Relationship Diagrams, Data Flow Diagrams, and Petri Nets and were followed more recently by Statecharts, concept graphs, class diagrams, use cases, Object-Process Diagrams, and Object-Process Language paragraphs. Each diagramming approach has an (implicit or explicit) underlying ontology—a set of semantics-bearing elements (building blocks and connectors) by which the domain's structure and behavior are expressed in the model. One or more of these models is generated such that it constitutes a representation of a system's architecture.

Ontology turns out to be a key factor not only in devising sound conceptual modeling methods but also in their validation and assessment. Indeed, the article by Shanks et al. discusses how information systems and software engineers can apply ontological considerations to validate the system's conceptual model with the focal stakeholders whose worlds they seek to represent. Validating conceptual models is critical to high-quality system development; without such validation, defects in the model that propagate to subsequent implementation activities can be costly. Validating a conceptual model entails ascertaining that it is a faithful representation of the focal domain, implying a guarantee that it is accurate, complete, free of conflict, and consists of attributes that are not redundant.

Assessing the methods or techniques that yield conceptual models requires evaluation at a higher level than just the quality of the resulting conceptual model. Here, constructivist theories of knowledge acquisition may prove useful. Finally, Gemino and Wand address the evaluation of conceptual modeling approaches used in information system analysis. Grammatical and phenomenological approaches are combined to assess the cognitive effect of models. Viewing modeling as a knowledge-construction process, the authors propose a framework for empirical comparisons of modeling techniques based on the learning process.

While drawing from seemingly remote disciplines like philosophy (the origin of ontology) and human cognition (the origin of the constructivist theory), these articles spell out the tangible adverse consequences that improper system architecting and modeling can cause. Software and information systems developers are well served to pay more attention to activities related to modeling and architecting the systems they implement, as the insights they gain are likely to prove beneficial throughout their systems' life cycles. C

Dov Dori (dori@ie.technion.ac.il, dori@mit.edu) is an associate professor of information systems engineering at the Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology, Haifa, Israel, and a research affiliate at MIT Cambridge, MA.