Chapter XV

An Interactive Viewpoint on the Role of UML

Dina Goldin and David Keil
University of Massachusetts, Boston, USA

Peter Wegner
Brown University, USA

The role of the Unified Modeling Language (UML) is to model interactive systems, whose behaviors emerge from the interaction of their components with each other and with the environment. Unlike traditional (algorithmic) computation, interactive computation involves infinite and dynamic (late binding) input/output streams. Tools and models limited to an algorithmic paradigm do not suffice to express and manage the behavior of today’s interactive systems, which are capable of self-reconfiguring and adapting to their environment.

Whereas procedural languages may express precise designs of closed processes, UML’s objective is to provide support for the analysis and specification of increasingly complex and inherently open systems. Interactive systems require dynamic models where interaction has first-class status, and where the environment is modeled explicitly, as a set of actors whose roles constrain the input patterns through use cases.

UML’s interaction-based approach to system modeling fits well with the encapsulation-based, object-oriented approach to implementation. By coupling these approaches, the software engineering process can promise to provide a more complete solution to system design and implementation, leading the way for widespread adoption of networked and embedded intelligent agent technology. A theoretical framework for modeling interactive computing can strengthen the foundations of UML and guide its evolution.
INTRODUCTION

The Unified Modeling Language (UML) emerged in response to a need for a notation (a visual language) that can express the behaviors of today’s interactive computing systems and that can guide in constructing them. In the UML framework, software design entails building an object-oriented representation of a system, as well as of its environment, e.g. its users (modeled as actors).

Interactive systems of the kind modeled with UML represent a new paradigm in computation that inherently cannot be modeled using traditional, or algorithmic, tools. At the heart of the new computing paradigm is the notion that a system’s job is not to transform a single static input to an output, but rather to provide an ongoing service (Wegner, 1997). The service-providing nature of present-day information systems was specifically noted by the Object Management Group in defining the UML standard (OMG, 2000).

When a system is viewed as a service provider, the interaction between the system and its environment becomes an integral part of the computing process. UML presents a uniform domain-independent framework for modeling the different interactions present in today’s systems: those among objects or software components, those between users and applications, those over networks (including the Internet) and those among embedded devices.

Programs that work non-interactively, transforming a given input to an output by a series of steps, represent the traditional, or algorithmic, paradigm of computation (Figure 1). Theoretical tools for modeling algorithmic computation include Turing Machines (Hopcroft & Ullman, 1979), recursive function theory and the lambda calculus, which all define the same set of computable functions (Turing, 1936). Algorithmic computation is present throughout interactive systems modeled by UML, at the low implementation level, but the entire system is more than merely such computation. Models of interactive computation are more recent; they include the Calculus of Communicating Systems (CCS) and the $\pi$-calculus (Milner, 1990, 1999), input/output automata (Lynch et al., 1994) and Interaction Machines (Wegner & Goldin, 1999b, 1999c) that maintain persistent state information between interaction steps.

Figure 1. Algorithmic Computation

A software system modeled by UML is a computing entity. System components and objects are also computing entities:

Computing entity: a finitely specifiable software system, component or object that is being modeled by UML. Entities may contain sub-entities, or be a part of a larger entity.

Once a system is implemented, its external behavior emerges out of its interactions with its environment:

Environment of an entity: the producer of inputs for the computing entity, and the consumer of its outputs; it is outside the entity, interacting with it via inputs and outputs.
Actor: an active role player in a given entity’s environment; for a system, actors are usually system users, but they may be other systems, software or hardware.
Related Content

Visualising COBOL Legacy Systems with UML: An Experimental Report
www.irma-international.org/chapter/visualising-cobol-legacy-systems-uml/4937/

Sharing Ontologies and Rules Using Model Transformations
www.irma-international.org/chapter/sharing-ontologies-rules-using-model/35871/

Modular Rule-Based Programming in 2APL
www.irma-international.org/chapter/modular-rule-based-programming-2apl/35853/

On Quality Assessment of Learning Technology Specifications
www.irma-international.org/chapter/quality-assessment-learning-technology-specifications/73174/

Transforming UML Class Diagrams into Relational Data Models
www.irma-international.org/chapter/transforming-uml-class-diagrams-into/30543/