Chapter 16
The Formal Design Models of a Set of Abstract Data Types (ADTs)

Yingxu Wang
University of Calgary, Canada

Xinming Tan
Wuhan University of Technology, China

Cyprian F. Ngolah
Sentinel Trending & Diagnostics Ltd., Canada

Phillip C.-Y. Sheu
University of California, Irvine, USA

ABSTRACT

Type theories are fundamental for underpinning data object modeling and system architectural design in computing and software engineering. Abstract Data Types (ADTs) are a set of highly generic and rigorously modeled data structures in type theory. ADTs also play a key role in Object-Oriented (OO) technologies for software system design and implementation. This paper presents a formal modeling methodology for ADTs using the Real-Time Process Algebra (RTPA), which allows both architectural and behavioral models of ADTs and complex data objects. Formal architectures, static behaviors, and dynamic behaviors of a set of ADTs are comparatively studied. The architectural models of the ADTs are created using RTPA architectural modeling methodologies known as the Unified Data Models (UDMs). The static behaviors of the ADTs are specified and refined by a set of Unified Process Models (UPMs) of RTPA. The dynamic behaviors of the ADTs are modeled by process dispatching technologies of RTPA. This work has been applied in a number of real-time and non-real-time system designs such as a Real-Time Operating System (RTOS+), a Cognitive Learning Engine (CLE), and the automatic code generator based on RTPA.

DOI: 10.4018/978-1-4666-0264-9.ch016

INTRODUCTION

Computational operations can be classified into the categories of *data object, behavior,* and *resource* modeling and manipulations. Based on this view, programs are perceived as a coordination of the data objects and behaviors in computing. *Data object modeling* is a process to creatively extract and abstractly represent a real-world problem by data models based on the constraints of given computing resources.

Using types to model real-world entities can be traced back to the mathematical thought of Bertrand Russell (Russell, 1903) and Georg Cantor in 1932 **(**Lipschutz & Lipson, 1997). A type is a category of variables that share a common property such as the kind of data, domain, and allowable operations. Types are an important logical property shared by data objects in programming (Cardelli & Wegner, 1985; Mitchell, 1990). Although data in their most primitive form is a string of bits, types are found expressively convenient for data representation at the logical level in programming. Type theory can be used to prevent computational operations on incompatible operands, to help software engineers to avoid obvious and not so obvious pitfalls, and to improve regularity and orthogonality in programming language design.

Definition 1. A *data type*, shortly a *type*, is a set in which all member data objects share a common logical property or attribute.

The mathematical foundation of types is set theory. The maximum range of values that a variable can assume is a type, and a type is associated with a set of predefined or allowable operations. Methodologies of types and their properties have been defined in Real-Time Process Algebra (RTPA) (Wang, 2002, 2008a, 2008b, 2008c), where 17 primitive types in computing and software engineering have been elicited (Wang, 2007). A type can be classified as either *primitive* or *derived* (complex) types. The former is the most elementary types that cannot further be divided into more simple ones; the latter is a compound form of multiple primitive types based on certain type rules. Most primitive types are provided by programming languages; while most user defined types are derived ones.

A *type system* specifies data object modeling and manipulation rules of a programming language, as that of a grammar system that specifies program syntaxes and composing rules of the language. Therefore, the generic complex types can be modeled by abstract data types (Guttag, 1977; Broy et al., 1984), which are a logical model of a complex and/or user defined data type with a set of predefined operations.

Definition 2. An *Abstract Data Type* (ADT) is an abstract model of data objects with a formal encapsulation of the logical architecture and valid operations of the data object.

An ADT encapsulates a data structure and presents the user with an interface through which data can be accessed. It exports a type, a set of valid operations, and any axioms and preconditions that define the application domain of the ADT. ADTs extend type construction techniques by encapsulating both data structures and functional behaviors. The interface and implementation of an ADT can be separated in design and implementation. Based on the models of ADTs as generic data structures, concrete data objects can be derived in computing.

A number of ADTs have been identified in computing and system modeling such as *stack, queue, sequence, record, array, list, tree, file,* and *graph* (Wang, 2007). A summary of the ten typical ADTs is provided in Table 1 where the structures and behaviors of the ADTs are described. ADTs possess the following properties: (i) An extension of type constructions by integrating both data structures and functional behaviors; (ii) A hybrid data object modeling technique that *encapsulates* both user defined data structures (types) and allowable operations on them; (iii) The interface