6 LOGICS FOR SPECIFYING CONCURRENT INFORMATION SYSTEMS

Hans-Dieter Ehrich,
Carlos Caleiro,
Amilcar Sernadas,
and Grit Denker

Abstract: This chapter concentrates on a challenging problem of information system specification and design, namely how to cope on a high level of abstraction with concurrent behaviour and communication as implied by distribution. Since distributed information systems are reactive and open systems maintaining data bases and applications, it is crucial to develop high-level specification techniques that can cope with data and programs as well as with concurrent workflow and communication issues. Techniques from conceptual modeling, abstract data types, concurrent processes and communication protocols are relevant and have to be combined. In the approach presented here, temporal logic is used for specifying sequential object behaviour, and communication facilities are added for specifying interaction between concurrent objects. We study two distributed temporal logics dealing with communication in two different ways. \( D_0 \) adds basic statements that can only express synchronous "calling" of predicates, while \( D_1 \) adds much richer facilities for making local statements about other objects in their respective local logics. \( D_0 \) is more operational and can be animated or implemented more easily, while \( D_1 \) is intuitively more appealing.
and convenient for modeling and specification. We demonstrate by example how $D_1$ can be effectively reduced to $D_0$ in a sound and complete way.

6.1 INTRODUCTION

In the early phases of information systems development, it is essential to work on a high level of abstraction: careful conceptual modeling and specification techniques help making the right design decisions and adapting the system to changing needs. The objective is to give the developer the ability to prescribe the properties of a system, and to predict and check its behaviour by reasoning, simulation and animation based on specification, and to give a sound reference basis for testing the implementation.

This chapter is about high-level specification techniques for distributed information systems, giving due attention to concurrency and communication among sites. While implementation platforms like CORBA are evolving to facilitate implementation, little is known about how to set up and specify distributed data and behaviour models in a meaningful way.

Information systems are reactive systems with the capability of maintaining and utilizing large amounts of data. A crucial point for specification is to choose the right logic—or family of logics. Our approach combines ideas and concepts from the object-oriented systems view, and from the traditions of conceptual modeling, behaviour modeling, abstract data type theory, specification of reactive systems, and concurrency theory. It is based on experiences with developing the OBLOG family of languages and their semantic foundations that started with [SSE87], in particular the TROLL and GNOME object specification languages. References are given in section 6.7, together with an account of related work.

The outline of the paper is as follows. In section 6.2, we introduce basic concepts and ideas by means of an example. In section 6.3, we give an account of the local propositional logic $L$ used for specifying single objects in isolation. In section 6.4, two distributed propositional logics are introduced that add communication facilities to $L$: $D_0$ adds basic statements that can only express synchronous "calling" of predicates, while $D_1$ adds fancy facilities for making local statements about other objects in their respective local logics. $D_0$ is more operational and can be animated or implemented more easily, while $D_1$ is intuitively more appealing and convenient for modeling and specification. In section 6.5, we demonstrate by example how $D_1$ can be effectively reduced to $D_0$. Thus, $D_1$ does not have more expressive power than $D_0$, and $D_1$ specifications may be automatically translated to $D_0$ descriptions. In section 6.6, we give an extended example drawn from a real application that shows how convenient it