Chapter I
Designing Multi-Agent Systems from Logic Specifications: A Case Study

Hong Lin
University of Houston—Downtown, USA

ABSTRACT

In this chapter a program construction method based on \( \gamma \)-Calculus is proposed. The problem to be solved is specified by first-order predicate logic and a semantic verification program is constructed directly from the specification. We exploit this method in synthesizing the architectural specifications of multi-agent systems (MAS) in \( \gamma \)-Calculus based on the logic specifications of the MAS. By enabling the transformation from the logic specifications to operational specifications of MAS, this method allows the design of the MAS to be focused on the architectural definition level. It benefits the development of MAS by enabling logic deduction on behaviors of the MAS, and a design methodology in an incremental fashion. We present this method by a case study of designing a course information management system.

INTRODUCTION

The modeling issue in the abstract computing machine level has been studied in (Banâtre, Fradet, & Radenac, 2004), where the chemical reaction model (CRM) (Banatre & Le Metayer, 1990 & 1993; Banatre, Fradet, & Radenac, 2005a; Le Metayer, 1994) is used to model an autonomic system. Given the dynamic and concurrent nature of multi-agent systems (MAS), we find that the chemical reaction metaphor provides a mechanism for describing the overall architecture of the distributed MAS precisely and concisely, while giving the design of the real system a solid starting point and allowing step-by-step refinement of the system using transformational methods (Lin, 2004; Lin & Yang, 2006).
Although CRM is suitable for modeling MAS, it serves as an operational specification language for MAS and it requires the designers of MAS to have the understanding of chemistry-inspired computational models. As a matter of fact, logic specification of MAS is better suited as a specification method in the current understanding, because logic specification focuses on behavioral properties of the systems without concerns with underlying computational model. We propose a method for generating MAS specifications in \( \gamma \)-Calculus from their logic specifications. We use the “generate-and-test” method to design the re-write process. Generally speaking, this process generates data in the domain of logic specification and creates a verification program in \( \gamma \)-Calculus to verify the logic specification with the generated data.

When applying this method to MASs, there are some problems to solve. The architectural specification of a MAS is different from that of a normal program, because for a MAS, we need to consider a collection of aspects, including distribution, security, performance, etc. This will cause a much more complex synthesizing process. For example, the distribution aspect will cause the communication pattern to be considered in the synthesizing process. In our approach, the communications are defined by the logic specifications of the interfaces of the system in terms of either message passing or shared memory. The practicability of this method is further strengthened by a transformation method we have proposed to implement CRM specifications on realistic computational models (Lin, 2004; Lin & Yang, 2006).

MASs are considered as complex systems whose design issues are difficult to be handled by logic systems. By bridging logic specifications and operational specifications of MASs, our study opens a path to introducing derivative methods in the higher level architectural design of MASs. This work will help formalize the design processes and promote the current research endeavor to end the state of MAS design in case-by-case fashion.

MODELING MULTI-AGENT SYSTEMS BY CHEMICAL REACTION MODELS

Gamma (Banatre & Le Metayer, 1990 & 1993) is a kernel language in which programs are described in terms of multiset transformations. In Gamma programming paradigm, programmers can concentrate on the logic of problem solving based on an abstract machine and are free from considering any particular execution environment. It has seeded follow-up elaborations, such as Chemical Abstract Machine (Cham) (Berry & Boudol, 1992), higher-order Gamma (Le Metayer, 1994), and Structured Gamma (Fradet & Le Metayer, 1998).

While the original Gamma language is a first-order language, higher order extensions have been proposed to enhance the expressiveness of the language. These include higher-order Gamma (Le Metayer, 1994), hmm-calculus (Cohen & Muylaert-Filho, 1996), and others. The recent formalisms, \( \gamma \)-Calculi, of Gamma languages combine reaction rules and the multisets of data and treat reactions as first-class citizens (Banatre, Fradet, & Radenac, 2004, 2005a, & 2005b). Among \( \gamma \)-Calculi, \( \gamma_0 \)-Calculus is a minimal basis for the chemical paradigm; \( \gamma_c \)-Calculus extends \( \gamma_0 \)-Calculus by adding a condition term into \( \gamma \)-abstractions; and \( \gamma_n \)-Calculus extends \( \gamma_0 \)-Calculus by allowing abstractions to atomically capture multiple elements. Finally, \( \gamma_{cn} \)-Calculus combines both \( \gamma_c \)-Calculus and \( \gamma_n \)-Calculus. For notational simplicity, we use \( \gamma \)-Calculus to mean \( \gamma_{cn} \)-Calculus from this point on. Also, we assume that the readers have the basic knowledge about the syntax and semantics of \( \gamma \)-Calculus.

We found that the dynamic nature of distributed agents makes it an ideal object for modeling using the Gamma languages. An agent shows a combination of a number of characteristics, such as