Chapter II
Concurrent Programming with Multi-Agents and the Chemical Abstract Machine

Wanli Ma
University of Canberra, Australia

Dat Tran
University of Canberra, Australia

Dharmendra Sharma
University of Canberra, Australia

Hong Lin
University of Houston – Downtown, USA

Mary Anderson
University of Canberra, Australia

ABSTRACT

In this chapter, we propose a new concurrent programming approach called MACH (multi-agent extended chemical abstract machine). MACH extends the chemical abstract machine with multiple coexisting agents. This paper focuses on the design, implementation, and verification of MACH. The aim of MACH is to develop a reactive programming language based on an interactive computational model, which we believe is the key to concurrent programming. We present MACH as a simple and efficient programming approach based on a sound theoretical background.

INTRODUCTION

Agent-based software engineering (ABSE) is a new area in software development. The basic idea of ABSE is to use agents as the building blocks for a software system, the same way that objects are the building blocks in object-oriented software engineering. ABSE promises a simplified and enhanced approach to construct complicated software systems. Stemming from the continu-
ous and autonomous nature of agents and also the high level of abstraction and communication amongst the agents, it is especially for concurrent programming (we use concurrent programming to cover multiprogramming in a single computer, parallel programming, and distributed programming; we also use “parallel” and “distributed” interchangeably if there is no confusion). In the last couple of decades, a plethora of concurrent programming languages and models have been proposed; most extend the existing sequential programming languages with thread control facilities. However, parallel programming is still far more difficult than sequential programming. It is intuitive to assume use of multiple control threads for concurrency and that the difference between concurrent and sequential programming is the number of active control threads at a given time. In fact, the difference, realised by the research communities, is not in the single thread nature versus the multi-thread with communications, but in contrast, between a functionality program and a reactive one (Lamport, 1993).

A functionality program is the one that maps an input state (or data) into an output state. Traditionally, we explain a program in this way: It accepts some input data and then, according to the instructions of the program, produces the output data. The instructions are executed in a step-by-step manner. There may be some procedure and function calls, but, at any time instance, only a single control flow, which is also known as a thread, exists. The flow starts from an initial point, normally the first instruction of the main module of that program, and terminates at a stop point. Denotational semantics (Gordon, 1979; Stoy, 1977) is the formal description of this idea, where each program statement transfers the program from the state before its execution the state after it. The behaviour of the whole program is to map the initial state to a final state.

A reactive system emphasizes the interactions among the components of a program: Different parts of a program interact with each other in response to stimuli from the outside world. The intrinsic property of the systems is the interactions among the components rather than the co-existing execution flows (or multi-control-thread), although the latter can also be observed from the outside of the systems.

Multi-agent systems (MAS) stem from the fields of distributed computing and artificial intelligence (Stone & Veloso, 2000). According to Wooldridge and Jennings (Wooldridge & Jennings, 1995), if a piece of software has the characteristics of autonomy, social ability, reactivity, and pro-activity, it is an agent in weak notion. On top of these characteristics, if the software has further characteristics of adaptability, mobility, veracity, and rationality, it becomes an agent in strong notion. A MAS is a society of multiple coexisting agents. Each agent interacts with other agents and the environment. Together, with certain rules to coordinate and regulate individual behavior, they achieve the goal of the system. These agents have a high degree of autonomy, heterogeneity, and parallelism. The properties make them excellent candidates for building blocks of concurrent, including parallel and distributed, programming.

In this chapter, we propose a new programming paradigm called MACH (multi-agent extended chemical abstract machine). It is the further development of T-Cham (Ma, Johnson, & Brent, 1996a, 1996b), where we used transactions to extend the chemical abstract machine. Instead of transactions, MACH extends the chemical abstract machine (Cham) (Berry & Boudol, 1992; Boudol, 1993) with multiple agents (Bradshaw, 1997; Stone & Veloso, 2000; Wooldridge & Jennings, 1995). This allows MACH to conduct computations based on the chemical reaction metaphor where computation proceeds as a succession of chemical reactions in a tuple space, with the agents controlling the coordination of the elements in this tuple space. The combination of Cham and a multi-agent system creates an interactive and modular multi-paradigm programming language.