Chapter II
Modular Rule-Based Programming in 2APL

Mehdi Dastani
Intelligent Systems Group, Utrecht University, The Netherlands

ABSTRACT

This chapter presents a modular version of a rule-based programming language called 2APL (A Practical Agent Programming Language). This programming language is designed to support the implementation of multi-agent systems where individual agents are specified in terms of cognitive concepts such as beliefs, goals, event, actions, plans, and three types of reasoning rules. The reasoning rules facilitate an effective integration of these concepts and allow generation, repair, and execution of plans based on beliefs, goals, and events. The modules can be used to implement different agent concepts such as roles and agent profiles, or to adopt common programming techniques such as encapsulation and information hiding. The syntax and the informal semantics of the programming constructs of modular 2APL are presented and discussed. A simple example is provided to illustrate how various ingredients of the presented programming language can be used.

INTRODUCTION

Agent-oriented software engineering paradigm is a modern approach for the development of distributed intelligent systems. In this paradigm, software systems (called multi-agent systems) consist of a number of interacting (software) agents, each of which is capable of sensing its environment (including other agents) and deciding to act in order to achieve its design objectives. Examples of such systems are e-commerce applications, auctions, electronic institutions, power management systems or transportation systems.

Multi-agent systems are specified, designed, and implemented in terms of high-level concepts and abstractions such as roles, communication,

beliefs, goals, plans, actions, and events. Different development methodologies (Bergenti et al., 2004), specification languages (e.g., BDI_{CTL} (Rao et al., 1991, Cohen et al., 1990) and KARO (Meyer et al., 1999), and programming languages (Bordini et al., 2007, Winikoff et al., 2005, Pokahr et al., 2005, Hindriks et al. 1999, Kakas et al. 2004, Giacomo et al. 2000) have been proposed to facilitate the development of agent-based systems.

While most agent-oriented development methodologies specify and design system architectures in terms of agent concepts and abstractions, the proposed agent-oriented programming languages and development tools aim at providing programming constructs to facilitate direct and effective implementation of these concepts and abstractions. Moreover, existing agent-oriented programming languages aim at supporting programming techniques such as modularity, reuse, encapsulation and information hiding. The availability and combination of agent-oriented programming constructs and programming techniques characterize and differentiate these programming languages and determine their usefulness and applicability.

Existing agent-oriented programming languages differ as they provide programming constructs for specific, sometimes overlapping, sets of agent concepts and abstractions. They also differ as they are based on different logics and use different technologies. Some of them are rule-based capturing the interaction of agent concepts by means of specific rules, while others are extension of Java programming language. Some capture specific rationality principles that underlie agent concepts in their semantics, while such principles are assumed to be implemented by agent programmers in other programming languages. Finally, they differ in programming techniques that are introduced to support the implementation of multi-agent systems. See (Bordini et al., 2005) for a comparison between some of these agent programming languages.

In this chapter, a modular rule-based agent-oriented programming language is presented that 1) separates multi-agent from single-agent concerns, 2) provides and integrates programming constructs that are expressive enough to implement a variety of agent concepts and abstractions used in the existing agent-oriented methodologies, 3) provides different types of rules to capture the interaction of agent concepts such as beliefs, goals and plans, 4) introduces a specific notion of modules and provides a set of module related operations that allows an agent programmer to determine how and when modules are used, and 5) realizes an effective integration of declarative and imperative programming styles. It is important to emphasize that multi-agent systems can be implemented in any existing programming language. However, we aim at designing an agent-oriented programming language that provides dedicated and expressive programming constructs and techniques to facilitate practical and effective implementation of agent related concepts and abstractions.

The structure of this chapter is as follows. In the next section, we provide a brief discussion on the exiting BDI-based agent-oriented programming languages (BDI stands for Beliefs, Desires, and Intentions). These programming languages are motivated by the BDI logics (Rao, 1996, Rao et al., 1991, Cohen et al., 1990) that are designed to specify agent behavior. The BDI-based programming languages provide dedicated programming constructs to implement individual agents in terms of (cognitive) concepts such as beliefs, desires and intentions. As we will see later in this chapter, these (cognitive) concepts can be used to implement individual agents that can decide to act in order to achieve their objectives. Then, a general description of a BDI-based agent-oriented programming language called 2APL (A Practical Agent Programming Language) is presented and some of its characterizing features are discussed. Subsequently, the complete syntax of 2APL is given and the intuitive meaning of its ingredients