Chapter 24
A Two-Layer Approach to Developing Self-Adaptive Multi-Agent Systems in Open Environment

Xinjun Mao
National University of Defense Technology, China

Menggao Dong
National University of Defense Technology, China

Haibin Zhu
Nipissing University, Canada

ABSTRACT

Development of self-adaptive systems situated in open and uncertain environments is a great challenge in the community of software engineering due to the unpredictability of environment changes and the variety of self-adaptation manners. Explicit specification of expected changes and various self-adaptations at design-time, an approach often adopted by developers, seems ineffective. This paper presents an agent-based approach that combines two-layer self-adaptation mechanisms and reinforcement learning together to support the development and running of self-adaptive systems. The approach takes self-adaptive systems as multi-agent organizations and enables the agent itself to make decisions on self-adaptation by learning at run-time and at different levels. The proposed self-adaptation mechanisms that are based on organization metaphors enable self-adaptation at two layers: fine-grain behavior level and coarse-grain organization level. Corresponding reinforcement learning algorithms on self-adaptation are designed and integrated with the two-layer self-adaptation mechanisms. This paper further details developmental technologies, based on the above approach, in establishing self-adaptive systems, including extended software architecture for self-adaptation, an implementation framework, and a development process. A case study and experiment evaluations are conducted to illustrate the effectiveness of the proposed approach.

DOI: 10.4018/978-1-5225-3923-0.ch024

INTRODUCTION

Many software systems require self-adaptation in order to deal with unexpected internal and external changes. This function is intended to make systems dependable, flexible, robust and reliable, etc., in order to satisfy design objectives. Self-adaptive software is normally a closed-loop system that aims to adjust various artifacts or attributes in response to changes in the 'self' and/or within the context that the software is situated. Here, 'self' means the whole body of the software, mostly implemented in several layers, while the context encompasses everything in the operating environment (Salehie & Tahvildari, 2009). Such systems are currently required in many application domains such as the military, enterprise, industry, etc. Challenges such as the issues of abstraction, modeling, mechanism, analysis, design and construction must be addressed during the development of self-adaptive software systems (Northrop et al., 2006; Salehie & Tahvildari, 2009). When systems are ultra large in scale (Northrop et al., 2006) and exist in a coalition of systems-of-systems (Sommerville et al., 2012), these challenges become increasingly complex especially considering decentralized control, autonomous software entity behavior, divergence of requirements, and the need for continuous system evolution. To this end, much research has been conducted within the literature of software engineering on topics such as software architecture (Garlan et al., 2004), self-adaptive rules and description languages (Wang, 2005), reflection technologies (Yoder et al., 2001) and compositional adaptation (McKinley, et al., 2004). These technologies are useful and effective, to some extent, in the development of self-adaptive systems whose boundaries are definite (i.e., the boundaries of the systems can be clearly defined and specified at design-time) and environments are predictable (i.e., the changes of environment can be precisely predicted and described at design-time). Additional issues arise when the environment in which self-adaptive systems are situated is unpredictable and uncertain, when the systems to be developed are complex (e.g., taking form of system-of-systems, ultra-large scale system (Northrop et al., 2006), coalition of systems (Sommerville et al., 2012) and when physical, social and informatics elements must be integrated. First, depending on design objectives, system self-adaptation can occur at different levels such as data, component or architecture, etc.. Questions must be answered as to the nature of required changes as well as levels and types of self-adaptation (Mao & Sun, 2013). Some required changes may result in the low-level and fine-grain adjustment of the data and/or behaviors of the target systems, while others may trigger the high-level and coarse-grain adjustment of the social positions of the agents in target systems. Therefore, mechanisms and abstractions that incorporate multiple self-adaptation techniques should be designed. Second, for many complex systems like ULS (Northrop et al., 2006), to precisely anticipate various expected changes and predefine the complete self-adaptation requirements at design-time is extremely difficult and may become impossible, because new and perhaps un-known system elements may dynamically enter or leave, unexpected events may occur, and system components may be autonomous and/or belong to different organizations. Actually, such systems are often built in an evolving way and their requirements normally are inherently conflicting, unknowable and diverse (Northrop et al., 2006). Therefore, to completely engineer self-adaptation into the system and pre-define the set of self-adaptation events and self-adaptation logic at the design-time by developer is infeasible. For example, developers can not anticipate all possible changes and therefore the self-adaptation logic describing how to respond to changes may need to evolve. Third, the key point in self-adaptive software is that its life-cycle should not be stopped after its development and initial deployment (Salehie & Tahvildari, 2009). Traditional off-line self-adaptation approaches are based on the pre-defined environment changes and well-defined self-adaptation strategies (Garlan et al., 2004; Wang, 2005) provided by developers. When the require-