The Journal of Systems and Software 83 (2010) 557–573

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Agent-oriented software patterns for rapid and affordable robot programming

Antonio Chella [a], Massimo Cossentino [b,*], Salvatore Gaglio [a,b], Luca Sabatucci [a], Valeria Seidita [a]

[a] Dipartimento di Ingegneria Informatica, University of Palermo, Italy
[b] Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR), Consiglio Nazionale delle Ricerche (CNR), Palermo, Italy

ARTICLE INFO

Article history:
Received 10 October 2008
Received in revised form 22 October 2009
Accepted 22 October 2009
Available online 4 November 2009

Keywords:
Multi-agent systems
Design patterns
Pattern oriented design
Robotics systems

ABSTRACT

Robotic systems are often quite complex to develop; they are huge, heavily constrained from the non-functional point of view and they implement challenging algorithms. The lack of integrated methods with reuse approaches leads robotic developers to reinvent the wheel each time a new project starts. This paper proposes to reuse the experience done when building robotic applications, by catching it into design patterns. These represent a general mean for (i) reusing proved solutions increasing the final quality, (ii) communicating the knowledge about a domain and (iii) reducing the development time and effort. Despite of this generality, the proposed repository of patterns is specific for multi-agent robotic systems. These patterns are documented by a set of design diagrams and the corresponding implementing code is obtained through a series of automatic transformations. Some patterns extracted from an existing and freely available repository are presented. The paper also discusses an experimental set-up based on the construction of a complete robotic application obtained by composing some highly reusable patterns.

1. Introduction

The process of building robotic systems is a complex task principally because these are intricate systems where different categories of problems have to be faced. A robotic system encapsulates algorithms that frequently derive from artificial intelligence and the architecture often includes distributed and heterogeneous components and must cope with real-time efficiency trade-offs. Designing a robotic architecture implies not only modelling the robot hardware and managing its sensors and actuators, but also modelling knowledge about the environment, and the ability to perform intelligent behaviours.

Software development for robotic systems is still today more an art than an engineering discipline. A few system developers have complete control all over the software, and typically write it all by themselves. There is an emerging demand for reuse techniques, with the twofold aim of maintaining software quality factors across projects (Nesnas et al., 2006; Schlegel, 2006; Dominguez-Brito et al., 2004) and of easily communicating and disseminating knowledge about robotic development issues (Schlegel, 2006). The current state of the art in the development of robotic applications suffers from several problems:

- Robotic application variability makes hard to create ad-hoc standards, unified architectures and methods, as well as to profitably import them from other application domains (Nesnas et al., 2001, 2006; Dominguez-Brito et al., 2004).
- Responsibilities and boundaries among applications, frameworks and middleware are not universally defined (Schlegel, 2006). Reuse of code across projects may easily fail if not complemented with design techniques.
- Several frameworks support the reuse of components, but a standard model to create robotic components still lacks because of the difficulty to find out a unified way to represent data and processes (Nesnas et al., 2006; Schlegel, 2006). Moreover, it is not clear the level of granularity to be used for building such components in order to promote the reuse across varying frameworks (Nesnas et al., 2001). Therefore a documentation process would help the component integration process (Nesnas et al., 2006; Cote et al., 2004; Dominguez-Brito et al., 2004).
- Development time and resource limits, typically occurring for experimental robotic systems, demand for environments and tools for fast prototyping of applications and for verifying and testing the system (Nesnas et al., 2006;Cote et al., 2004). These tools would minimize the effort spent for secondary aspects of the system, like component integration or system documentation, and would maximize the effort for research objectives.

By now the agent paradigm seems to be one of the most interesting choices for developing a robotic application by following a rigorous design process (Alami et al., 1998; Dominguez-Brito

* Corresponding author.
E-mail addresses: chella@unipa.it (A. Chella), cossentino@pa.icar.cnr.it (M. Cossentino), gaglio@unipa.it (S. Gaglio), sabatucci@dinfo.unipa.it (L. Sabatucci), seidita@dinfo.unipa.it (V. Seidita).

0164-1212/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2009.10.035

et al., 2004; Chella et al., 1998). Autonomous agents offer powerful instruments for decomposing, abstracting and organizing such complex, distributed and evolving systems. Several works consider robotic software as a collection of agents, where each of them is responsible for a specific functional area of the robot. These agents independently manage robot devices and collaborate in order to exhibit a collective synchronized behaviour, thus achieving a collective goal that is the robot mission.

This structure creates a decoupling between hardware and software, that is a necessary feature of an engineering design in which mission and global requirements, take priority over details about the implementation and deployment platforms.

This paper presents design patterns for agents, defined as a complement of the PASSI (process for agent societies specification and implementation) design process (Cossentino, 2005]) for developing multi-agent systems (MAS).

The contribution of this paper is a pattern-based reuse method supported by a specific tool for automatic code and documentation production. Although other works exist in this field, the specific innovations proposed by this approach are multifold. First of all, they regard the integration between the pattern reuse practice and the PASSI design process; then there is the successful adoption in the development of complex systems like robotic ones. The generation of the system code from pattern reuse is another relevant element; this code is not the common skeleton produced by several design tools nor the behavioural code obtained by applying transformations to dynamic diagrams (like it can be done for state-charts) but it is a complete and fully functional portion of code (skeletons and inner code of methods) reused from a repository and adapted to the specific problem or produced by processing available system specifications. Finally another contribution is in the definition of a repository of patterns that can be widely applied to the design of robotics systems but also in many other developing scenarios.

The paper is organized as follows: Section 2 presents common approaches for building robotic systems underlining the growing need for frameworks and methods for a rapid prototyping of these applications. Section 3 describes a well-known architecture adopted for a robotic case study throughout the paper. Section 4 illustrates the proposed engineering process consisting in a reuse technique based on design pattern composition. In addition, a tool (Agent Factory) is presented for supporting pattern selection, reuse and composition; it also provides automatic code and documentation generation. Section 5 presents some patterns from a repository for agents. They have been identified as a solution to typical and recurring robotic design problems. The section illustrates both pattern features and their usage. Section 6 is focused on the reuse and composition process applied to the proposed robotic application. Section 7 discusses the reusability of this approach and the quality of the produced system. Finally, some conclusions are drawn in Section 8.

2. Robot programming techniques and methodologies

This section explores possible approaches from literature to the development of robotic applications. The analysis starts with specific architectures and frameworks for reusing robotic components. Successively some methodologies for designing multi-agent robotic systems are illustrated and finally design pattern reuse is discussed.

2.1. Component-based frameworks

In the last few years several different platforms have been proposed for robotics programming. These are mainly based on the principle of modularity and make an extensive use of component-based software engineering practices. Several frameworks exist where a set of components specialized for robotic applications can be customized and integrated, thus the process of building a robotic system is made easier.

The CLARAty framework (Volpe et al., 2001) proposes a representation of the system based on two layers: the *Decision Layer* is the strategic level that drives the *Functional Layer*. The upper level provides components for the reasoning engine while the lower level is layered and can represent different levels of system abstractions.

The COOLBOT framework (Dominguez-Brito et al., 2004) explicitly considers software reuse, modular composition and third-party software integration. This framework provides means to design and to build units to be reused and to be composed (hierarchically and dynamically) by using finite state machine diagrams.

The Chimera methodology (Stewart and Khosla, 1996) addresses the design of dynamically reconfigurable real-time systems and robotic applications. Components are specified by describing their interfaces. The result is a software model for objects that can be reused, statically integrated and dynamically reconfigured, it supports real-time applications, and it can be used in a distributed shared memory environment.

2.2. Multi-agent system methodologies

Multi-agent systems represent a means for introducing autonomy, distribution, collaboration, and other advanced features in robotic (and non-robotic) programming. Many design methodologies have been proposed for designing agent systems and most of them can be adopted for the design of robotic applications. Stolzenburg and Arai (2003) propose to specify agent behaviour for robotic applications by using UML state-charts. Model checking techniques are employed to formally analyze behavioural properties of finite state systems and other issues like concurrency.

The organizational-based multiagent systems engineering (MaSE) methodology (DeLoach, 2005) has been conceived for engineering practical multiagent systems. It prescribes a top-down approach where the key concept is the *Goal*, a system-level objective that can be assigned to agents. MaSE has been used to design a team of autonomous, heterogeneous search and rescue robots. Analysis and design models proved to be helpful in the maintenance and modification of the cooperative robotic systems. A tool (agentTool) is provided with the methodology that supports the designer during system development.

The Cassiopeia methodology provides a method to proceed from a collective task global specification to the specification of the local behaviours, which are to be provided to the agents. The methodology has been successfully adopted in order to design and implement the organization of a robot team for the *RoboCup*.

A totally opposed approach is defined in ADELFE (Bernon et al., 2005) that assumes agents totally ignore system goals and the environment where they live. It employs cooperative agents whose design is aimed at avoiding non-cooperative situations descending from incomprehension, ambiguity, incompetence, unproductiveness, concurrency or other conflicts. ADELFE was employed to implement a multi-robot resource transportation system (Picard, 2005).

2.3. Reuse with design patterns

It is commonly recognized that reuse cannot be limited to the development phase (Barnes and Bollinger, 1991; Griss, 1993; Hooper and Chester, 1991. Design patterns are commonly considered the ultimate way for introducing reuse in a design process (Lajoie and Keller, 1995). They also allow for overcoming main limitations of components reuse: (i) libraries of components usually address