Software Engineering Challenges for Adaptive Robotic Ecologies

Mauro Dragone(*), David Swords, G.M.P. O'Hare
University College Dublin
(*)mauro.dragone@ucd.ie, david.swords@ucdconnect.ie, gregory.ohare@ucd.ie

Abstract. Adaptive Robotic ecologies are networks of heterogeneous robotic devices (sensors, actuators, automated appliances) pervasively embedded in everyday environments, where they cooperate to the delivery of of useful services and adapt to evolving operational and user needs. Adaptive robotic ecologies offer a disruptive new paradigm by which to introduce cognitive robots into the everyday life of people, by distributing sensing, cognition and actuation within the smart space. However, their effective realization necessitates novel software solutions to facilitate their specification, programming and execution. In this extended abstract we briefly illustrate the characteristic software requirements dictated by adaptive robotic ecologies and provide an overview of the specific solutions developed as part of the EU project RUBICON.

Introduction

While their inherent flexibility makes robotic ecologies increasingly popular, significant work is needed in order to improve their autonomy and their ability to cope with the ever changing user needs and evolving contexts. In the EU FP7 project RUBICON (Robotic UBIquitous COgnitive Network) [1] we addresses the core cognitive problems of how the participants of the ecology and their capabilities should be coordinated to analyze the situation of the environment, decide what they need to do in order to satisfy applications objectives, and use their past experience to drive their autonomous adaptation. We tackle these issues by endowing robotic ecologies with planning and learning capabilities which enable them to autonomously adapt to evolving environments and achieve useful services which are not merely restricted to those a priori situations envisioned by their designer. This can greatly simplify design, customization and adaptation, reduce the need of costly and long pre-programming phases, and ultimately enhance the user technology acceptance.

An important pre-requisite of such an endeavour, which is discussed in this paper, is the necessary software infrastructure underpinning the specification, integration, and the distributed management of robotic ecologies. Current software solutions in robotics are often of a component-based software engineering genre and provide a number of mechanisms and methodologies that can be used for the design, development and the execution of modular system architectures in terms of loosely coupled components [2][3]. However, such solutions are generally concerned with maximizing the opportunities for component re-use across differing robotic systems and how to address the challenges posed by the fast innovation rates of both the AI methods employed in robotics and the underlying technology. Consequently, robotic software systems are becoming increasingly better equipped for the support of the application composition phase where components are initialized and bound to different robotic platforms or re-used for different applications. Generally, they also account for re-configuration and modification of collaboration patterns among system components [4]. However, the same mechanisms have relative importance at runtime, as once finalized such applications will generally run in a stable and capable computational environment. On the contrary, robotic ecologies need to accommodate the complexity brought about by the interaction of heterogeneous robotic devices embodied in dynamic and open execution environments, with associated varying computational and communication constraints and evolving physical settings. They should be able to dynamically re-configure components' collaboration path-ways so as to implement different tasks or in response to changing circumstances, as typified by components joining or leaving the ecology, as result of system maintenance, component failure, network disruptions and/or mobility. They also require a consistent approach to increase their opportunities for learning, adaptation and dynamic self-configuration, but also to generally reduce the complexity of their programming and facilitate software re-use. Besides the obvious economic advantages associated with re-use of existing and well-tested components (e.g. for path-planning, safe navigation, localization and activity

recognition) robotic ecologies require functional primitives to avoid the need for online learning of complex sensing-acting strategies. Rather, a robotic ecology needs pre-existing modules to provide a base-line behaviour, to constrain their online exploration, but also to guarantee that the system will not behave too erratically during its initial learning stage. Finally, robotic ecologies need to be able to monitor their own execution and assess their own performance in carrying out their services. This must happen not only when the robotic ecology has completed some service, or in response to failure, but also at runtime, for instance, in order to give interim status feedback to high-level cognitive and learning functionalities and give them the opportunity to re-assess the goals of the ecology. Instrumental to supporting the AI advancements developed within the project RUBICON, we are developing a software suite to complement mainstream software approaches in robotics and to handle the characteristic requirements posed by robotic ecologies. Our solution, briefly illustrated in the remaining sections of this extended abstract, builds on both middleware for robotic ecologies and agent & component based solutions for self-adaptive software systems.

RUBICON Software Suite

In order to provide the functional and adaptive coordination of a robotic ecology, we build on (i) plan-based solutions [5][6] to decide the collaborative strategies with which the devices of the ecology must interact with the physical world while also exchanging information in the process, and (ii) a distributed learning network [7] to improve these strategies over time. However, applying a consistent approach to coordination and learning contrasts with the many requirements that must be upheld during strategy synthesis, execution and adaptation. One of the problem we face in the practical development of a robotic ecology, is that we need to leverage its highly heterogeneous resources, encompassing, for instance, both wireless sensor and actuator networks (WSANs) and mobile robots, and even devices with no (customizable) computational capability at all, such as Radio Frequency Identifications (RFIDs). To these end, we employ the PEIS middleware [8], previously developed as part of the Ecologies of Physically Embedded Intelligent Systems project [9]. The PEIS kernel is written in pure C (with binding for Java and other languages) and with as few library and RAM/processing dependencies as possible in order to fit on a wide range of computationally constrained devices. PEIS includes a decentralized and shared tuple space blackboard that allows for zero configuration, automatic discovery, high-level communication and collaboration through subscription based connections.

It also includes a program called PEIS-Init that acts as a proxy and access point to each device in the ecology. In this manner, the system's planner is able to start, stop and monitor the execution of any functional component, independently from its location, implementation, and computational capabilities. The other technical challenge tackled by our software suite is the lack of a component model shared across all the heterogeneous functional components harnessed by the robotic ecology. We address this issue by building on *Self*-OSGi [10], a modular and lightweight agent & component-based framework based on the Open Service Gateway Initiative (OSGi) [11]. OSGi defines a standardised component model and a lightweight container framework, which is used as a shared platform for network-provisioned services and components specified through Java interfaces and Java classes. OSGI offers container and life cycle operations to install, start, stop and remove components together with a declarative model for automatically publishing, finding and binding their required/provided services based on XML component definitions. *Self*-OSGi addresses the lack of common adaptation mechanisms in OSGi and in other component-like frameworks by injecting agent-based autonomic features to each component. Specifically, with *Self*-OSGi, each component's service requirement is treated as a goal of the agent system, which then manages the automatic and context-sensitive search for components able to provide each service requirement, as well as the automatic recovery from their failure. In addition, *Self*-OSGi adopts a proxy design pattern to manage external components, that is, components that do not live within the OSGi container, and that communicate among each other through legacy communication mechanisms. Within RUBICON, these features are used to define an agent-based interface layer toward heterogeneous software modules, including gateway components used to access wireless sensors and actuators, and robotic nodes based on the Robotic Operating System (ROS) [12]. Rather than actually implementing low-level functionalities and directly supporting their collaborations, each agent is used to manage the actual components running on the robotic devices, by re-using the communication and configuration mechanisms employed in the underlying managed system, such as topics, services and name-spaces in ROS, to monitor their progress toward their objectives and define their connectivity (wiring) with the other components in the system. For example, whenever a *GoalNavigation* service is first posted to a

robot managed by *Self*-OSGi, this will, respectively: (i) activate a *NavigationMonitor* agent, (ii) instantiate any of the the navigation components able to lead the robot toward its intended destination, and (iii) configure the system it to make sure that the selected component will be updated with the necessary data, such as the location of the robot and the information concerning the obstacles in its surroundings. Finally, the *NavigationMonitor* agent will then subscribe to the ROS message carrying out the information about the location of the robot. In the case that the navigator component fails, e.g. due either to an hardware, software or physical failure, the monitor will notice either a lack or a problem with the progress feedback received the robot system. At that point, *Self*-OSGi will trigger the re-wiring of the system, resulting in the activation of the second navigation component. In this manner, monitor/proxy agents help grounding the semantic of the functional components (which is already described by their XML component representations) by linking them with the operative description of what these components must achieve once activated. The result is that we are able to "project" a uniform component model upon an heterogeneous functional basis, and offer a homogeneous view to the higher-level control and learning mechanisms of the robotic ecology.

Conclusions and Future Work

While the software described in this paper is successfully supporting our cognitive architecture, our current approach relies on a number of ad-hoc interventions to the functional components in the robotic ecology. We are developing integration mechanisms that make this job as easy and automatic as possible, by using semantic, declarative descriptions of each component [13], and by implementing a library of monitor agents to decouple control & learning mechanisms from the functional layer of the robotic ecology. We plan to consolidate our experience by formalizing a component model for adaptive robotic ecologies.

References

[1] G. Amato, M. Broxvall, S. Chessa, M. Dragone, C. Gennaro, R. Lopez, L. Maguire, T. M. McGinnity, A. Micheli, A. Renteria, G. M.P. O.Hare, F. Pecora, Robotic UBIquitous COgnitive Network, In: Ambient Intelligence - Software and Applications - 3rd International Symposium on Ambient Intelligence (ISAmI), Salamanca, Spain, 28-30th March, 2012, Series: Advances in Intelligent and SoftComputing, Vol. 153, pp. 191-195. Springer. DOI: 10.1007/978-3-642-28783-1_23.

[2] D. Brugali and L. Gherardi. Component-Based Robotics Models and Systems. In *SDIR Tutorial on Component-Based Robotics Engineering, IEEE/RAS International Conference on Robotics and Automation (ICRA 2010*), May 2010, Anchorage, USA.

[3] N. Hochgeschwender and A. Shakhimardanov: Component-Based Robotics Middleware. In*SDIR Tutorial on Component-Based Robotics Engineering, IEEE/RAS International Conference on Robotics and Automation (ICRA 2010)*, May 2010, Anchorage, USA.

[4] Andreas Steck and Christian Schlegel, "Towards Quality of Service and Resource Aware Robotic Systems through Model-Driven Software Development.". In First Int. Workshop on Domain-Specific Languages and models for ROBotic systems (IROS - DSLRob), Taipei, Taiwan, 2010.

[5] R. Lundh, L. Karlsson, A.S.: Plan-based configuration of an ecology of robots. In: ICRA, Rome, Italy (2007)

[6] Lundh, R.: Plan-based configuration of a group of robots. Licentiate Thesis. University of Orebro, Sweden (October 2006)

[7] Davide Bacciu, Stefano Chessa, Claudio Gallicchio, Alessandro Lenzi, Alessio Micheli and Susanna Pelagatti, "A General Purpose Distributed Learning Model for Robotic Ecologies", International IFAC Symposium on Robotic Control (SYROCO 2012), session on Adaptive Robotic Ecologies, Dubrovnik, Croatia, 5-7 September 2012. In Robot Control 10(1), pp. 435-440. DOI: 10.3182/20120905-3-HR-2030.00178.

[8] M. Broxvall, B.S. Seo, W.K.: The peis kernel: A middleware for ubiquitous robotics. In: Proc. of the IROS-07 Workshop on Ubiquitous Robotic Space Design and Applications, San Diego, California (2007)

[9] Mathias Broxvall, Marco Gritti, Alessandro Saffiotti, BeomSu Seo, Young-Jo Cho: PEIS Ecology: Integrating Robots into Smart Environments. ICRA 2006: 212-218

[10] M. Dragone, A BDI model for component & service-based systems: Self -OSGi, In: 10th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS), Salamanca, Spain, 28-30th March, 2012.

[11] Open service gateway initiative. Web site http://www.osgi.org/Main/HomePage. [Accessed 15th March 2013].

[12] Quigley, Morgan., Conley, Ken., Gerkey, Brian P.., Faust, Josh., Foote, Tully., Leibs, Jeremy., Wheeler, Rob., and Ng, Andrew Y. ROS: an open-source Robot Operating System, ICRA Workshop on Open Source Software, 2009.

[13] M. Dragone, D. Swords, S. Abdel Naby, G.M.P. O'Hare and M. Broxvall. A Programming Framework for Multi Agent Coordination of Robotic Ecologies, Tenth International Workshop on Programming Multi-Agent Systems, ProMAS 2012.