Coordinating Software Components in a Component-Based Architecture for Robotics

Geoffrey Biggs, Noriaki Ando, and Tetsuo Kotoku

Intelligent Systems Research Institute
National Institute of Advanced Industrial Science and Technology (AIST)
AIST Tsukuba Central 2, Tsukuba, Ibaraki 305-8568, Japan

Abstract. Component-based software is a major design trend in robot software. It brings many benefits to system design, implementation and maintenance. One step in using component-based methods in designing the structure of a robot program is managing the components and the connections between them over time, known as coordination. In this paper we present a framework for coordinating component networks using the OpenRTM-aist software architecture, implemented using the concurrent Erlang language. The framework provides a coordination system that mimics the internal state-change notification system of OpenRTM-aist. Rather than being a fixed-structure coordinator, it allows robot developers to implement a coordinator matching the style of coordination they need. This paper shows that Erlang has potential in robotics.

1 Introduction

Component-based software design and implementation is a current trend in software engineering. Software is divided into individual components, each with a well-defined interface that specifies what functionality that component provides. Multiple software components are combined together into a complete software system [13]. Using this design methodology, robot developers can create complete robot systems from off-the-shelf software components as easily as complete electric circuits can be created from hardware components.

Component-based practices bring many benefits to software design, implementation, maintenance and reuse, including known interfaces that act as “contracts” between components, “separation of concerns” (each component only deals with its individual problem), isolation testing, and rapid development of new systems using existing commoditised software resources.

These benefits also apply to the design, implementation, maintenance and reuse of robot software. As a result, component-based software is a major trend in robotics.

An issue that all robot developers faces is the coordination of behaviours, and so in turn the coordination of the software. Coordination is important to allow the robot’s software to adapt to changes in the robot’s state as it carries out its various tasks. Before the recent rise of flexible component-based architectures,
architectures with fixed structure, often layered, were popular in robotics. In these, a higher layer manages the actions of a lower layer to provide coordination according to some generated plan. Often, the plan itself is generated by an even higher layer, although some architectures, such as CLARAty [7], intentionally do not use this approach.

In this paper, we present a coordination framework for the OpenRTM-aist component-based architecture. It is a framework rather than a complete coordinator because it provides the facilities for programmers to create their own coordination systems. Rather than a fixed coordination style, programmers are free to use whichever style suits their needs. We use the concurrent Erlang language to implement the framework in order to test its applicability in robotics.

The next section discusses coordination methods commonly used in robotics. Section 3 describes OpenRTM-aist, the architecture for which the coordination framework has been designed. The coordination framework itself is described in section 4. Discussion is given in section 5 and conclusions in section 6.

## 2 Coordinating Robot Software

Coordination has a long history in robot software. It has always been necessary to manage the actions of a robot in order to achieve complex goals. It is particularly common to see coordination play a major role in layered architectures. Often, a layered architecture will feature a low-level layer that consists of chunks of functionality, and at a higher level some kind of coordination system controlling the execution of these chunks to meet some goal.

Despite their strong coordination support, no layered architectures have managed to become widely used.

On the other hand, recent years have seen the popularisation of more flexible component-based software architectures for robotics. These architectures allow designers to create component networks. Rather than being layered, the network of components is effectively a single layer of individual programs communicating by one or more methods of transporting data. Such an architecture style is very heterogeneous and adaptable to the needs of the programmer.

Examples of these architectures include OpenRTM-aist [1], ORCA2 [3], ROS [9], ERIC'S [11] and OPRoS [12]. In each case, systems built using the architecture rely on networks of connected components with data flowing amongst them. The behaviour of the robot is represented by what data ultimately arrives at the components responsible for controlling actuators. The shaping of this data, and so determining the robot’s current behaviour, is performed by the components that make up the component network between sensor components and actuator components. The reader may notice that this is similar to coordinating the actions of a lower layer in layered architectures.

Coordination in a component-based system therefore requires changing either the internal behaviour of the individual components, or changing part of or the whole component network.