Volume 3, Issue 2, February 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

A Reusable High-Quality Software Component for Robotic Applications

Gurusiddesh Sangolli* CS&E Dept DPE College,Pune India

Dhanraj S CS&E Dept EWIT Bangalore India

Appasab Salunke CS&E Dept BSIOTR(W),Pune India

Abstract—Robotics is the branch of technology that deals with the design, construction, operation, structural disposition, manufacture and application of robots and computer systems for their control, sensory feedback, and information processing. A component is a reusable and replaceable software module accessed through its interface. Component-based development is expected to shorten the development period, reduce maintenance costs, and improve program reusability and interoperability of components. All the flexible components using in my project share a common feature: their input and/or output ports are not predefined at compile-time. Instead, they reconfigured at run-time based on the user's needs. The proposed system describes a collection of utility components built over time. These components are created using the programming language embedded C, on AVR studio.

Keywords— Robot programming systems, I/O ports, IR Sensors, Meta function, component-based architectures.

I. INTRODUCTION

A component is a reusable and replaceable software module accessed through its interface. Component-based development is expected to shorten the development period, reduce maintenance costs, and improve program reusability and the interoperability of components. Also, the assumptions and constraints about tasks and operational environments are hidden and hard coded in the software implementation. This increased complexity has lead to increasing demands for modularity, productivity, reusability, integration, and maintenance. Not only do robots have many different types of sensors, actuators, and degrees of freedom, but also their services are becoming more and more sophisticated allowing them to run autonomously, while providing complex services in an unknown or partially known environment. Unfortunately, their services are often not reusable even in slightly different application scenarios because they are tied to specific robotic hardware, processing platforms, and communication infrastructures. In a component-based system, some of the common functionality may be entire components that perform well-known, often-used functions on data streams between the other components of the system. These are called as utility components. This flexibility in the interface, and therefore in the components, is possible because of embedded C's dynamic programming language capabilities.

II. LITERATURE SURVEY

A. RELATED WORK

Choulsoo Jang, Seung-Ik Lee, Seung-Woog Jung, Byoungyoul Song, Rockwon Kim, Sunghoon Kim, and Cheol-Hoon Lee in their paper "OPRoS: A New Component-Based Robot Software Platform[5]" Tells about components as Component technology seems to be an attractive approach to meet the demands in the robotic software field. The main focus of component-based development is concerned with the assembly of pre-existing software components into larger pieces. Unfortunately, their services are often not reusable even in slightly different application scenarios because they are tied to specific robotic hardware, processing platforms, and communication infrastructures. Also, the assumptions and constraints about tasks and operational environments are hidden and hard coded in the software implementation. This increased complexity has led to increasing demands for modularity, productivity, reusability, integration, and maintenance.

Alex Brooks, Tobias Kaupp, Alexei Makarenko and Stefan Williams in their paper "Towards component-based robotics,[3]" gives an overview of Component-Based Software Engineering (CBSE), motivates its application to the field of mobile robotics, and proposes a particular component model. It argues that robotics is particularly well-suited for and in need of component-based ideas and also introduces Orca – an open-source component-based software engineering framework proposed for mobile robotics with an associated repository of free, reusable components for building mobile robotic systems. Distributed Robotics, Robotic Architectures, Component-Based Design, Software Re-use, Standardization Component-Based Software Engineering (CBSE) is an approach that has arisen in the software engineering community in the last decade or so. It aims to shift the emphasis in system-building from traditional programming to composing software systems from a mixture of off-the-shelf and custom-built components.CBSE offers developers the opportunity to source existing plug-in software components, rather than building everything from scratch.

B. Song, S. Jung, C. Jang, and S. Kim, in their paper "An Introduction to Robot Component Model for OPRoS (Open Platform for Robotic Services), [5]" explains, component is a reusable and replaceable software module accessed through its interface. Component-based development is expected to shorten the development period, reduce maintenance costs, and improve program reusability and the interoperability of components. They proposed a new robot software component platform in order to support the entire process of robot software development. It consists of specifications of a component model, component authoring tool, component composer, and component execution engine. The robot device interface platforms such as Player aim at providing interfaces for accessing robot sensors and actuators over the network. ERSP, Urbi, Mobile Robots, and iRobot Aware are robot software architecture platforms that provide layered software architecture. They argued that a good robot software platform needs to offer much more than pure middleware such that it supports the full development lifecycle for robot software. To meet the above mentioned requirements, they proposed a new component technology called open platform for robotic services (OPRoS).

Y. hsin(Oscar) Kuo and B. MacDonald, "A distributed real-time software framework for robotic applications,[11]" explains A distributed real–time robot application framework is designed, to improve the scalability and reusability of software modules. Many robots are equipped with advanced, powerful computing hardware. This enables many new possibilities including distributed robotic applications, since overhead and latency are no longer significant. By deploying a distributed architecture, roboticists can loosen the coupling between software components such as device drivers, algorithms and user interfaces. It provides network-oriented device server along with 2D and3D simulators that are capable of simulating a population of mobile robots. IPC (Inter Process Communication) along with TDL (Task Description Language) are two software packages developed at CMU [Simmons and Apfelbaum, 2004]. TDL extends the standard C++ syntax to provide semantics for task management and it uses IPC for sending messages between servers and clients. CARMEN [Monte Merlo et al., 2002], also based on IPC, provides a navigation toolkit for robotics. SIMOO-RT [Becker and Pereira, 2002] is an integrated development environment for real-time systems consisting of a software development framework and model editing tools.CLARAty is an architecture designed for robot automation systems from the Jet Propulsion Laboratory [Nesnaset al., 2003]. It uses a two-tiered design to separate functional and decision layers of a robot application. The research projects above create custom middleware systems to enable interaction between distributed components. Woo proposed a three-tier software infrastructure [Woo, 2002; Woo et al., 2003] based on CORBA. It is designed around the CORBA Trader Service and relies on CORBA to provide the underlying development environment.OROCOS [OROCOS] is designed to be a software framework for robotic control software. It aims to integrate low-level real-time services provided by operating systems (OSs) and provides a high-level software framework for robotic control software development. Although high-level middleware can greatly simplify software development; its efficiency is a major concern for roboticists. Robotic applications are different from most desktop applications in that they interact with physical objects, so their actions have stricter timing constraints. If the timing of a particular control signals wrong, a robot can cause damage to itself or even to humans. Distributed robotic applications must achieve real-time performance at two different levels, the OS level and the middleware level.

B. Proposed Methodology

Component based software is likely to regularly reuse entire utility components. Such components may perform tasks such as selecting between the outputs of two similar components, logging the data streams between components, and filtering component outputs. These general purpose components are frequently required throughout a component-based system, and so need to be adaptable to a wide variety of use cases. The required adaptability cannot be achieved using fixed interfaces. This leads to significant duplication of effort and wasted time re-implementing utilities for specific needs. Instead, interfaces that are configurable by the user to the specific use case are necessary. Ideally, the user should not need to recompile the component to create the interface they require.

The proposed system describes a collection of utility components built over time. These components are created using the programming language embedded C, on AVR studio. The key point of these components is that they all feature flexible, run-time configurable interfaces. The user specifies the interface they require when starting a component. No further work or time is necessary for the user. This is a major benefit to the reusability of these components; they are written once and used indefinitely. The system describes the functionality of each of these components, with particular focus on their flexibility. Flexible interfaces are, in some cases, very important for increasing both usability and reusability. The components presented here are adaptable to the user's needs. Rather than re-implementing components, even if just wrappers around a provided core, to suit a specific need and specific interface, the user only has to provide options when starting the components.

Embedded C based methodology for a software component is proposed which can be reusable, adaptable and flexible. This method help the robot designer to develop robot prototype very fast and also debugging becomes much easier. Careful design choices are required at interface design time to maximize reusability. No matter how carefully the interface of a general-purpose utility component is designed, it will not be flexible enough, or it may become so large as to be unusable. As a result, utility components tend to be re-implemented over and over again, each time with a different interface matching the new use case.

C. *Software Module of Proposed system*

Embedded C components are reusable and replaceable software modules that do not need recompilation. They are distributed on a network. They run loosely coupled and independently, often representing a robot's devices. A robotic

service is composed of these distributed components in a similar fashion as a robot hardware system is assembled with devices. A communication infrastructure including connection management of the components is provided by the component execution engine that is a runtime environment of the Embedded C components. By this partition of network management from component logic, developers can focus on the logic that they intend to develop without additional concerns about network management. The granularity of a distributed Embedded C component can be at any level. For example, it can be at device level, algorithm level, or coordination level, and so on, and it is up to component developers to decide which one is appropriate. With the distributed components of diverse granularities, a flat or hierarchical composition manner might be used for various robot software architectures.

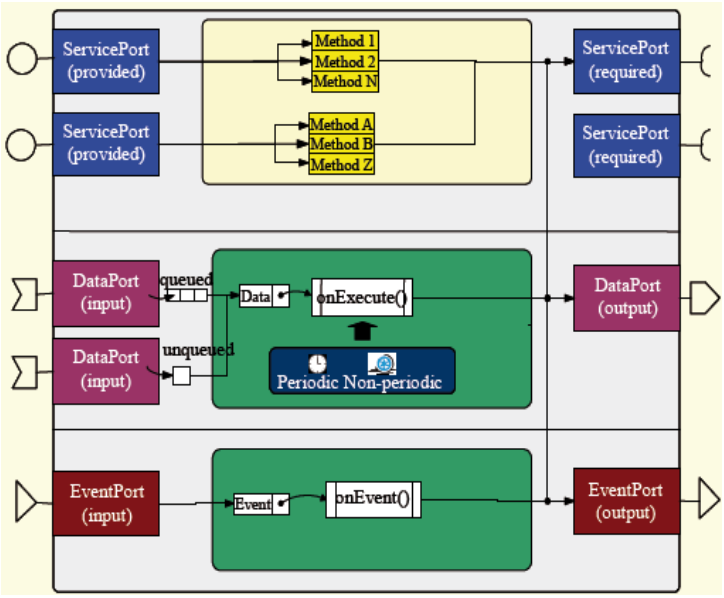

Fig. 1 Software architecture for component based software

III. RESULTS AND DISCUSSION

An individual component is a software package, a Meta function or a_module that encapsulates a set of related functions (or data). All system processes are placed into separate components so that all of the data and functions inside each component are semantically related (just as with the contents of classes). Because of this principle, it is often said that components are modular and cohesive. A careful design of interface makes component software to work efficiently. An embedded C based software method is used to achieve flexibility and adaptability for utility components, this method also help the robot designer to develop robo prototype very fast and also debugging becomes much easier. Another important attribute of components is that they are substitutable, so that a component can replace another (at design time or run-time), if the successor component meets the requirements of the initial component (expressed via the interfaces). Consequently, components can be replaced with either an updated version or an alternative for example, without breaking the system in which the component operates. The hardware components used for this project are interfaced and called by Meta functions for specific tasks to complete. Once the task is completed those components are kept for reusability and so as the software codes. This brings the greatest feature of reusability and thus increases the system efficiency.

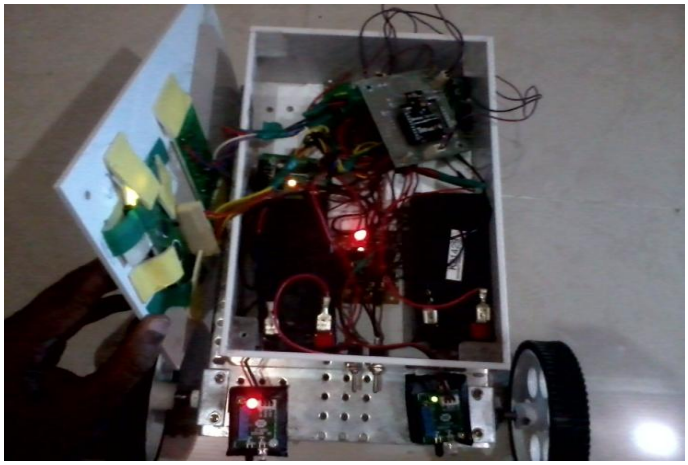

Fig. 2 Free moving robot programmed with component-based software.

IV. CONCLUSIONS AND FUTURE SCOPE

As the complexity of the project increases in robotic system there is a need for special software called component based software. In this project embedded C based software achieves the project objective. Required flexibility of robotic components is achieved by designing the hardware completely for robotic module. With this robot movements can be controlled both manually and remotely. When a component is to be accessed or shared across execution contexts or network links, techniques such as serialization or marshalling are often employed to deliver the component to its destination. With regard to system-wide co-ordination, components communicate with each other via interfaces. This interface can be seen as a signature of the component - the client does not need to know about the inner workings of the component (implementation) in order to make use of it. This principle results in components referred to as encapsulated. However when a component needs to use another component in order to function, it adopts a user interface which specifies the services that it needs.

A. *FUTURE SCOPE*

- Components interaction can further be improvised with well-structured programming skills.
- With advanced programming tools like LAB View more accurate and efficient results can be obtained.

REFERENCES

[1] Choulsoo Jang, Seung-Ik Lee, Seung-Woog Jung, Byoungyoul Song,RockwonKim, Sunghoon Kim, and Cheol-Hoon Lee,"OPRoS: A New Component-BasedRobot Software Platform",ETRI Journal, Volume 32, Number 5, October 2010.

[2] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W.-K. Yoon, "RTmiddleware:distributed component middleware for RT (robot technology),"in Intelligent Robots and Systems, 2005. (IROS 2005). 2005IEEE/RSJ International Conference on, August 2005, pp. 3933–3938.

[3] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Oreback,"Towards component-based robotics," in Intelligent Robots and Systems,2005. (IROS 2005). 2005 IEEE/RSJ International Conference on,Aug. 2005, pp. 163–168.

[4] http://www.ros.org

[5] B. Song, S. Jung, C. Jang, and S. Kim, "An Introduction to Robot ComponentModel for OPRoS (Open Platform for Robotic Services)," inIntl. Conf. on Simulation, Modeling and Programming for AutonomousRobots 2008, Workshop Proceedings of, Nov. 2008, pp. 592–603.

[6] G. Broten, D. Mackay, S. Monckton, and J. Collier, "The roboticsexperience," Robotics & Automation Magazine, IEEE, vol. 16, no. 1,pp. 46–54, March 2009.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns:elements of reusable object-oriented software. Addison–Wesley, 1995.

[8] R. C. Arkin, Behavior-based robotics. MIT Press, 1998.

[9] (2009) 28.12. inspect – Inspect live objects – Python v2.6.2 documentation.[Online]. Available: http://docs.python.org/library/inspect.html

[10] M. Henning and S. Vinoski, Advanced CORBA Programming withC++. Addison-Wesley Professional, 1999, ch. 2.

[11] Y. hsin (Oscar) Kuo and B. MacDonald, "A distributed real-timesoftware framework for robotic applications," in Proc. IEEE Int. Conf.on Robotics and Automation (ICRA'05), Barcelona, 18–22 April 2005,pp. 1976–81.

[12] L. Parker, "Alliance: An architecture for fualt tolerant multi-robot coorperation,". *ORNL TM12920*, Oak Ridge National Laboratory, Oak Ridge, TN, 1995.

[13] G. Pardo-Castellote et.al, "Controlshell: A software architecture for complex electromechanical systems," *Int'l Journal of Robotics Research*, 17(4), 1988.

[14] L.M. Pedersen, et al., "Integrated Demonstration of Instrument Placement, Robust Execution and Contingent Planning," Proc. Int'l Symp on AI, Robotics and Automation for Space, 2003.

[15] P. Schenker, et.al., "Planetary rover developments supporting Mars science, sample return and future human robotic colonization," Autonomous Robots, 2003

