

A Component-Oriented Programming Framework for Developing Embedded Mobile Robot Software using PECOS Model

Dayang Norhayati Abang Jawawi, Safaai

Deris

*Department of Software Engineering
Faculty of Computer Science & Information System
Universiti Teknologi Malaysia
81310 Johor Bahru, Malaysia
dayang@fksm.utm.my*

Rosbi Mamat

*Department of Mechatronics & Robotics Engineering
Faculty of Electrical Engineering
Universiti Teknologi Malaysia
81310 Johor Bahru, Malaysia*

Abstract

A practical framework for component-based software engineering of embedded real-time systems, particularly for autonomous mobile robot embedded software development using PECOS component model is proposed. The main features of this framework are: (1) use graphical representation for components definition and composition; (2) target C language for optimal code generation with small micro-controller; and (3) does not requires run-time support except for real-time kernel. Real-time implementation indicates that, the PECOS component model together with the proposed framework is suitable for resource constrained embedded systems.

1. Introduction

The promises of component-based software engineering (CBSE) to increase software productivity, improve software quality, and decrease the costs of software development have driven the trend of software development from building software from scratch to component-based software development approach.

In the domain of embedded real-time (ERT) systems, CBSE is envisioned to bring a number of advantages such as faster development times at reasonable costs, ensuring business success of ERT system developers with the ability to develop and port software quickly, and developing more complex software through reuse of existing components.

Industrial component technologies currently available such as OMG's CORBA Component Model (CCM), Microsoft's (D)COM/COM+, .NET, SUN Microsystems' JavaBeans and Enterprise JavaBeans,

are generally complex, require large resources such as memory and computation power, and are platform dependent [1][2]. Furthermore, they do not address the non-functional properties such as how much memory it consumes and timing constraints which are important in ERT systems.

Consequently, a number of component models such as ReFlex [3], PBO [4], Rubus [5], Koala [6] and PErvasive COmponent Systems (PECOS) [7], have been developed to address requirements of ERT software. Evaluation of these component models against the industrial requirements of heavy vehicle sector shows that PECOS is the most complete component technology with good support for industrial requirements [8].

PECOS component model was originally developed for field device systems and some supporting tools such as Component Composition (CoCo) description language for specifying components, code generator for generating Java/C++ code skeletons from CoCo and runtime environment (RTE) which interfaces generated codes to the real-time operating system [9]. However, many of these tools are incomplete and information on the RTE is not publicly accessible as it is a proprietary of the ABB Company [10].

Adopting PECOS for other application and platform requires porting the RTE, which can be difficult without the information. Thus, a simple framework for CBSE of ERT systems, particularly for autonomous mobile robot (AMR) embedded software development using PECOS is proposed here. The main features of this simple framework are: (1) use graphical representation for components definition and composition; (2) target C language for optimal code generation with small micro-controller; and (3) does not requires RTE except for real-time kernel.

The layout of this paper is as follows. Section 2 describes the PECOS component model for which the framework is proposed for. In Section 3 the proposed framework is described and illustrates the implementation of mobile robot software using the framework. Section 4 presents some implementation results of the framework on a real mobile robot. Finally, the conclusion is presented in Section 5.

2. PECOS component model

PECOS component consists of two main parts, i.e. the static structure model which describes the entities included in the model, their features and properties; and the execution model which defines the behavior of the component execution.

2.1. Static structure model

There are three main entities in the PECOS model: *components*, *ports* and *connectors* [11]. Components are used to organize the computation and data into parts that have well-defined semantics and behaviour. A port is a reference to data that can be read and written by a component and enables a component to be connected to another component through a connector. Data passed over the port is specified with name, type, range and direction (in, out or inout). Only compatible ports can be connected with connectors.

Components can be any of three types: active, passive or event component. Active components have their own thread of control; passive components do not have their own thread of control; and an event component is a component that is triggered by event.


In PECOS components can be hierarchically built from other subcomponents. A component which contains subcomponents or children is called a composite component or a parent component, and these subcomponents are not visible outside the composite component. A component without subcomponents is called a leaf component.

A property bundle which describes the nonfunctional aspects such as timing or memory usage is associated with this static structure of component.

The components composition using PECOS static structural model is performed by connecting the compatible ports between components.

Figure 1 illustrates composition of four components in PECOS for a motor speed control application. MotorSpeedControl is a composite component with three subcomponents, i.e. PI, Encoder, and Motor. Two connections in the composition are connection between output port speed to input port actualSpeed and

connection between output port controlSignal and input port signal.

Figure 1 also shows two active components: MotorSpeedControl and PI marked with “” at the upper right corner, and two passive components: Encoder and Motor.

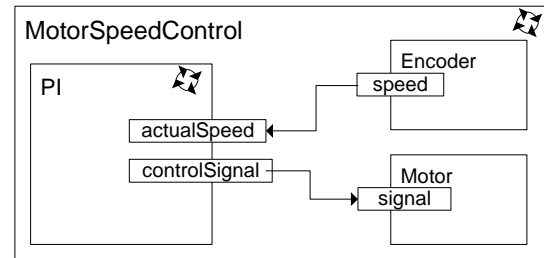


Figure 1. PECOS structure model for a motor speed control application

2.2. Execution model

There are two different behaviors associated with active and event components: *execution behavior* and *synchronization behavior*. Execution behavior determines the action that is performed when the component is executed while the synchronization behavior is responsible for synchronizing the data space of the active or event component with that of the parent [11].

In PECOS data synchronization between two connected active or event components' ports, is required to assure that data cannot get corrupted due to two simultaneous write operations from components in different thread. To solve this data synchronization problem, every active and event component is equipped with its own private data space where the data can be updated independently by the component. At specific intervals, this private data space is then synchronized its parent or composite component.

The execution behavior is based on the following runtime rules [11]:

- The behavior of passive component is executed in the thread of its parent component.
- Synchronization behavior for active and event components is executed in the thread of the parent component.
- Active and event components execute their own execution behavior in their own thread of control.
- Every composite component has to provide a schedule for its children.

2.3. Original PECOS implementation framework in AMR domain

The original framework for implementation of PECOS components has some limitation with respect to our needs for development of ERT software for AMR systems. Some of these limitations include: 1) the use of CoCo language seems to be difficult for the target audience as CoCo is almost a programming language by itself; 2) targeting for Java or C++ may not be appropriate for small microcontrollers with limited resources and Java and C++ supports may not be available for those microcontrollers; and 3) the RTE is difficult to develop since information on the RTE is not publicly accessible and it is a proprietary of the ABB Company.

Some of the results from component-based reuse works are hard to reuse because the solution is not coherent and simple to be used by the mechatronics and robotics engineers which are not from software engineering or computer science background[12]. In order to be widely accepted by robotic community, a solution for coping with systems complexity problem and software reuse for AMR should at least fulfill the following requirements [13]:

- i. **Embeddable and self-contained.** Typically, the AMR software or firmware is embedded in the on-board controller. A self-contained AMR system requires on-board computation and the system is typically constrained by limited processing power and memories.
- ii. **Modular or component-based software.** Component modularity is to provide artifact to be reused that contain domain knowledge from different disciplines.
- iii. **Portable across different platform.** Platform-independent where the component implementation does not depends on hardware, Real-Time Operating System (RTOS) or communications protocol.
- iv. **Predictable real-time performance.** The abilities to predict and analyze timing are key requirements for AMR system.

When adapting existing component models into the AMR software, there are a number of requirements, resource constraints and current implementation practices which must be considered. These considerations include:

- i. Embedded constraints:
 - Limited power – battery operated, 8 or 16-bit processing power.
 - Limited memory – ROM 128K and RAM 64K.

- ii. Real-time requirements
 - Support for multi-tasking is required.
 - Support high-level analysis of timing behavior.
- iii. AMR requirements
 - Support for hybrid deliberate layered architecture of robot software.
 - Variation point of software reuse component for different robot applications.

It is assume that the AMR software developers are not from software engineering field, and have limited background in software engineering and real-time systems.

3. A New framework for PECOS

This new framework was originally intended for development of ERT software for Autonomous Mobile Robot (AMR). The target audience is the mechatronic and robotics researchers with are not from software engineering background and do not have extensive programming experience. However, experience shows that the new framework is generally suitable for developing reactive embedded systems which typically use 8-bit or 16-bit microcontrollers with memory constraints and developed with C language.

To address the above limitations, a new framework which is called Component-Oriented Programming (COP) framework is proposed for implementing the PECOS component model manually in resource constrained embedded systems.

In this framework, instead of using the CoCo language, graphical representation of components is used for components definition and composition. Currently, codes have to be written manually from this graphical representation. The graphical composition environment and code generation tools based on the graphical representation currently are still being developed. The framework describes here is based on manual creation of codes guided with the code templates proposed in the COP framework.

The COP framework is targeted for C language, since, optimized C compilers are available for most micro-controllers, and C is portable across many platforms. Furthermore, C is a familiar language and has been use extensively by the target audience.

The dependency of PECOS model to RTE deployment model will be replaced by two abstraction layers in the proposed COP framework. Figure 2 shows how the component-based software is composed using the modified PECOS model interfaced with the RTOS, micro-controller and robot hardware using

hardware abstraction layer and RTOS abstraction layer.

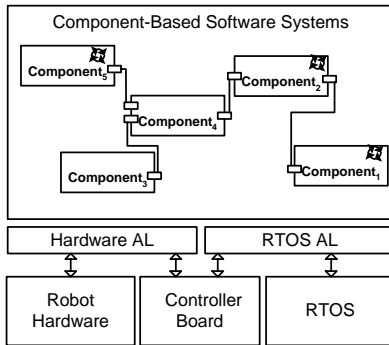


Figure 2. Hardware and RTOS abstraction layers of the proposed COP framework

3.1. Component engineering

Component engineering concerned with the analysis of domains and development of generic and domain-specific reusable components. Based on patterns analysis in domain of AMR systems, common components are identified and the specifications of the components are documented in graphical blocks form, which are further translated manually to C source codes and stored in repository for use in the application engineering stage.

Figure 3 shows a generic component named PID documented in graphical block form. The input ports and output ports, and their associated types and range of values are defined in the block. Each port is numbered accordingly for easy reference. Arrows on the ports indicate the direction or whether they are inports (IP00-IP02) or outports (OP00). Bidirectional ports are not allowed in this framework. The *Period* and *Priority* fields are timing specifications for active component only, usually determined in the application engineering stage. Not shown in the block is the worst case execution time (WCET) of the component.

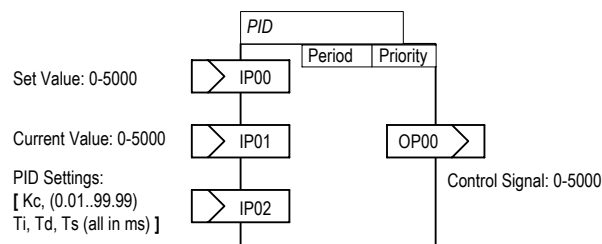


Figure 3. A PID component documented in block form

The generic interface implementation for the PID component in C is as follows:

```
typedef struct
{
    int IP00; // set value(0-5000 )
    int IP01; // current value(0-5000)
    int IP02[4]; // array of PID const
                // [Kc Ti Td Ts]
    int OP00; // control signal(0-5000)
} INTERFPID;
```

The code body for an active component consists of three main parts: initialization part, execution part, and synchronization part. In the passive components, the synchronization part is not required.

The initialization part is responsible for component initialization, and in the case of active component it calls real-time kernel to create the thread or task for the component with the defined *Period* and *Priority*.

The execution part is the main component behavior while the synchronization part updates the data between inner ports and outer ports as defined by PECOS component execution model.

Composite components can be built by hierarchically composing a number of subcomponents and stored in the repository for later use.

Figure 4 shows a generic composite component named *MotorControl* built from three passive subcomponents: *Encoder*, *PID* and *Motor*. The code body for the composite component is still consists of the above three main parts. On top of this, the composite component is responsible for calling the initialization and synchronization parts of its child or sub-components.

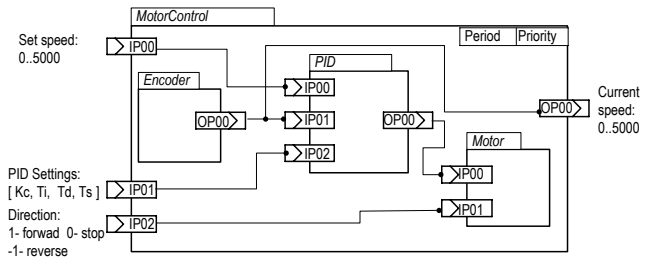


Figure 4. MotorControl composite component

3.2. Application engineering

Application engineering which is also called component-based software development (CBD), involves developing applications using software components previously developed. A design of simple AMR embedded software will be used to illustrate the steps involve in the application engineering using the COP framework.

The major steps involve in application engineering are: identification of required components, composition of components, scheduling analysis and assigning the period and priority of components, and codes writing.

Based on the requirements and patterns analysis, the required components are identified and composed by connecting the compatible ports. Figure 5 shows a block diagram composition of a mobile robot software, which consists of six leaf components and five composite components shown by blocks with shadow.

Sometimes, the concrete components which are application-specific need to be developed from the generic components previously created. For examples in Figure 5, `motorctrl_right` and `motorctrl_left` components are concrete components developed from the `MotorControl` component. In this framework, connections of constants to compatible ports are allowed. This is shown in Figure 5 by the connections

of constants to input port (IP01) of `motorctrl_right` and `motorctrl_left` components.

Once the composition is completed, the next step is to allocate property bundles value such as period and priority for active components using the WCET and some scheduling theory. Rate Monotonic Analysis (RMA) [12] theory is mainly used to allocate priority in this framework.

The result of these steps is the creation of a configuration file `pecos_cfg.h` for inclusion in the main program. This file specifies the components to be included in the project and periods of execution and priorities of active components. The effect of this file is to include only the required components source codes in the final code and to configure periods and priorities of active components for the real-time kernel use. The code as follows shows a fragment of the configuration definitions in the `pecos_cfg.h` file.

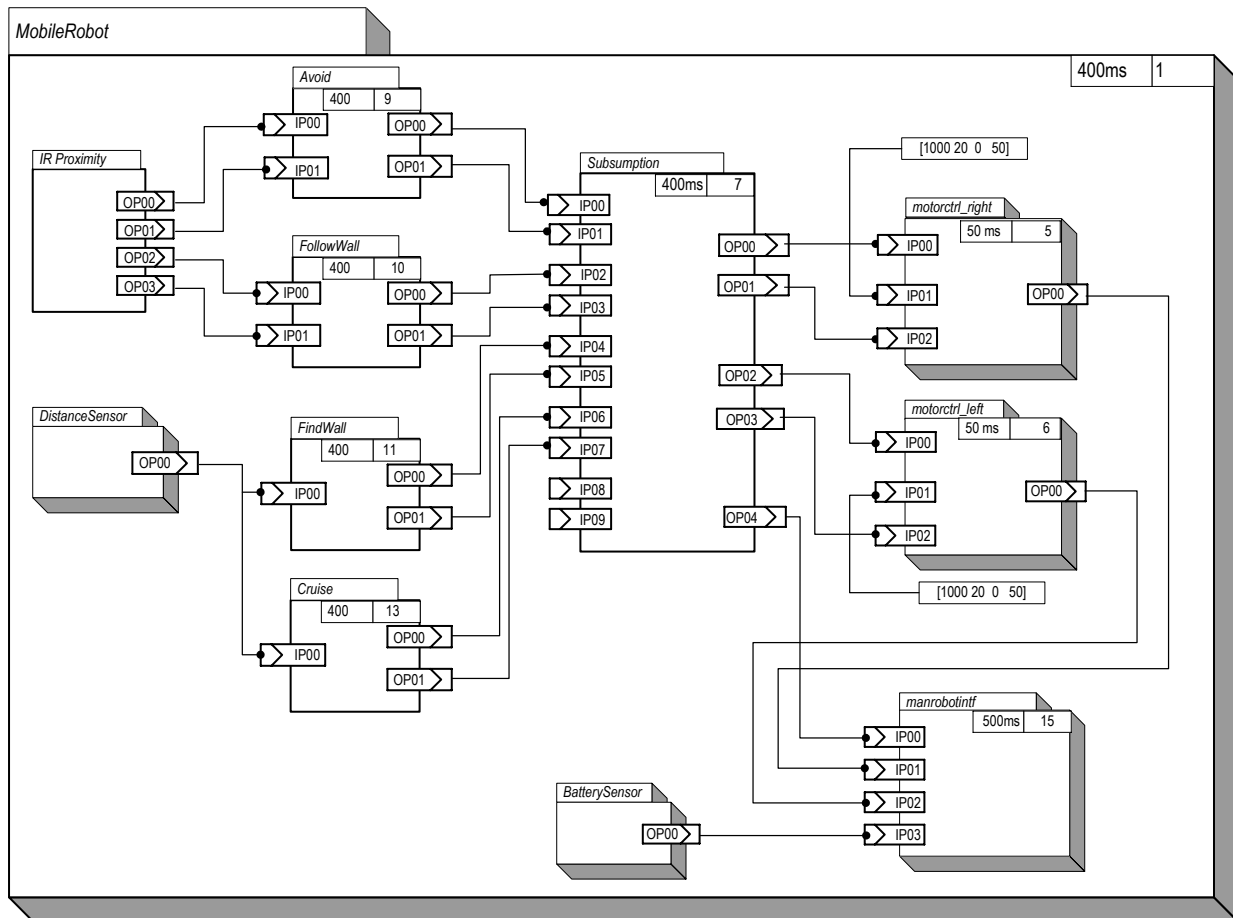


Figure 5. A mobile robot composition

```

// ***** CONFIG PECOS COMPONENTS :
// REQUIRE (1) or NOTREQUIRE (0)

#define COMP_MOTOR_REQ          1
#define COMP_IRPROX_REQ         1
#define COMP_DISTSENSOR_REQ     1
#define COMP_BATTSSENSOR_REQ    1
#define COMP_SUBSUMPTION_REQ    1
#define COMP_BEHAVIOR_BASED_CTRL_REQ 1
#define COMP_MANROBOT_INTRF_REQ 1
#define COMP_MOTOR_CTRL_REQ     1

// ***** CONFIG REAL-TIME PARAMETERS FOR
ACTIVE COMPONENTS *****
//-- MANROBOTINTF
#define MRI_PERIOD              500
#define MRI_PRIO                15

//-- MOTORCTRL_RIGHT
#define MOTORCTRL_RIGHT_PERIOD  50
#define MOTORCTRL_RIGHT_PRIO    5

```

The connections between the compatible input ports and output ports are achieved by assignments of the data defined by the components interface. For example the following code shows how connection between the output ports of Avoid component and the input ports of IRProximity component are made:

```

//-- Connect all inports to outports
Avoid.IP00 = IRProximity.OP00;
Avoid.IP01 = IRProximity.OP01;

```

As shown in Figure 5, the top level component in this composition is the MobileRobot component which has the period of execution of 400 ms and priority of 1, i.e. the highest priority. Thus, the main program for this composition is the execution behavior of MobileRobot component as already discussed in Section 3.1. The template for this main file is shown in Figure 6.

4. Implementation Results

Following the framework described in Section 3 and PECOS component definitions, the autonomous mobile robot software composed as shown in Figure 5 was implemented on a real robot with behavior-based intelligence system. The target board consists of a 16-bit AMD188ES microcontroller with 64Kb ROM and 128Kb RAM. The software tools used for the software development are Paradigm C compiler [14] for generating ROMable code and μ C/OS-II real-time kernel [15] for multitasking support. This C codes is about 5000 lines of codes not including comments and blank lines. The total machine code size for the resulting application is about 21Kb with RAM usage of about 15Kb. This indicates that, the PECOS

component model together with the proposed framework can generate application with minimal memories requirements, suitable for other resource constrained embedded systems. The proposed hardware abstraction layer and the RTOS abstraction layer enable the development of platform-independent components.

```

#include "pecos_cfg.h"
#include "pecos.h"

// TOP LEVEL COMPONENT
void MobileRobot (void* data) {
/***** INITIALIZATION PART *****/
//-- initialize all child components
//-- all ACTIVE tasks will be created
INITmanrobotintf();
INITmotorctrl_right();
INITmotorctrl_left();
:etc
for (;;)
/***** EXECUTION PART *****/
//-- Execute all passive components
//-- Active components already executed
EXECIRProximity();
:etc
/***** SYNCHRONIZATION PART *****/
//-- Synchronous all inports & outports
SYNCmanrobotintf();
:etc
//-- Connect all inports to outports
Avoid.IP00 = IRProximity.OP00;
Avoid.IP01 = IRProximity.OP01;
Subsumption.IP00 = Avoid.OP00;
Subsumption.IP01[0] = Avoid.OP01[0];
:etc
//-- call RTK to create periodic execution
OSTimeDelay (MOBILEROBOT_PERIOD);
}

void main(void) {
//-- initialize hardware dependent parts
//-- initialize Real-time kernel
//-- create MobileRobot task
//-- start the Real-time kernel
while (1) ; // run endlessly
}

```

Figure 6. Code fragments for the main program of the mobile robot composition

5. Conclusions and Future Works

A new COP framework was proposed to better support embedded resource-constrained AMR components integration and composition. The main features of this framework are: (1) use graphical representation for components definition and composition; (2) target C language for optimal code generation with small micro-controller; and (3) does not requires RTE except for real-time kernel.

The component engineering and application engineering aspects of the framework were described and discussed based on a real example of a mobile robot software development. Real-time implementation indicate that, the PECOS component model together with the COP framework can generate application code with minimal memories requirements, suitable for other resource constrained embedded systems.

The proposed COP framework enables the idea in PECOS to be implemented optimally without depending on any support tools and proprietary run-time environment from the original PECOS project. Currently, works are in progress in providing a graphical composition environment and code generation tools based on the PECOS component blocks to automate the process thus ease the CBD for robotics software developer.

6. References

- [1] I. Crnkovic, "Component-based Approach for Embedded Systems", *Ninth International Workshop on Component-oriented Programming*, Session 4 – Application of CBSE, Oslo, June 2004.
- [2] U. Rasthofer, and F. Bellosa, "Component-based Software Engineering for Distributed Embedded Real-time Systems", *IEE Proceeding- Software*, 148(3). 2001, pp. 99-103.
- [3] A. Wall, "Architectural Modeling and Analysis of Complex Real-Time Systems", *Phd Thesis of Mälardalen University*, September 2003.
- [4] D.B. Stewart, R. A. Volpe, and P. K. Khosla, "Design of Dynamically Reconfigurable Real-time Software Using Port-Based Objects", *IEEE Transaction on Software Engineering*. 23(12), 1997, pp. 759 –776.
- [5] Articus Systems at www.articus-systems.com/
- [6] R. Ommering, F. Linden, J. Kramer, and J. Magee, "The Koala Component Model for Consumer Electronics Software", *IEEE Computer*. 33(3), 2000, pp.78 –85.
- [7] Nierstrasz, O. et. al., "A Component Model for Field Devices", *Proceedings First International IFIP/ACM Working Conference on Component Deployment*, Springer-Verlag Heidelberg Volume 2370, Berlin, Germany, 20-21 June 2002, pp. 200-209.
- [8] A. Möller, Åkerholm M., Fredriksson J., Nolin M., "Evaluation of Component Technologies with Respect to Industrial Requirements", *Proceedings of the 30th EUROMICRO Conference on Component-Based Software Engineering Track*, Rennes, France, August 2004, pp. 56-63.
- [9] R. Wuyts, S. Ducasse and O. Nierstrasz. "A Data-centric Approach to Composing Embedded, Real-time Software Components", *The Journal of Systems and Software*, 74, 2005, pp. 25-34.
- [10] B. Bouyssounouse, J. Sifakis, "Embedded Systems Design: The ARTIST Roadmap for Research and Development", *Lecture Notes in Computer Science*, Volume 3436 / 2005, Springer-Verlag GmbH, 2005, pp. 160-194.
- [11] T. Genssler et. al., "PECOS in a Nutshell", *Technical Report*, PECOS Project, September 2002.
- [12] A. Orebäck, "A Component Framework for Autonomous Mobile Robots". *Phd Thesis of KTH Stockholm*, 2004.
- [13] F. Pont, and R. Siegwart, "A Real-Time Software Framework for Indoor Navigation", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Edmonton, Canada, August 2005, pp. 2085 - 2090.
- [14] Paradigm Systems, *Paradigm C++ Reference Manual Version 5.0*, Endwell, 2000.
- [15] Labrosse, J. J.. *MicroC/OS-II The Real-Time Kernel*, 2nd edition, R&D Books, USA , 1999.