ARS ADVANCED ROBOTIC SYSTEMS INTERNATIONAL

A Component-Oriented Programming for Embedded Mobile Robot Software

[1]Dayang N. A. Jawawi; [2]Rosbi Mamat and [1]Safaai Deris
[1]Department of Software Engineering, Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia Malaysia, 81310 UTM, Skudai, Malaysia
[2]Department of Mechatronics and Robotics Engineering, Faculty of Electrical Engineering, Universiti Teknologi Malaysia Malaysia, 81310 UTM, Skudai, Malaysia
dayang@utm.my

Abstract: *Applying software reuse to many Embedded Real-Time (ERT) systems poses significant challenges to industrial software processes due to the resource-constrained and real-time requirements of the systems. Autonomous Mobile Robot (AMR) system is a class of ERT systems, hence, inherits the challenge of applying software reuse in general ERT systems. Furthermore, software reuse in AMR systems is challenged by the diversities in terms of robot physical size and shape, environmental interaction and implementation platform. Thus, it is envisioned that component-based software engineering will be the suitable way to promote software reuse in AMR systems with consideration to general requirements to be self-contained, platform-independent and real-time predictable. A framework for component-oriented programming for AMR software development using PECOS component model is proposed in this paper. 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 resource-constrained micro-controller; and (3) minimal requirement for run-time support. Real-time implementation indicates that, the PECOS component model together with the proposed framework is suitable for resource constrained embedded AMR systems software development.*
Keywords: *Component-based development, software reuse, autonomous mobile robots*

1. Introduction

An Autonomous Mobile Robot (AMR) is a mechatronics system, which embodies technologies from several engineering disciplines in the domains of mechanical engineering, electronics, automatic control, artificial intelligence, software engineering and computer engineering. Thus, building a complete AMR system requires expertise in those areas. The software aspect of AMR has been recognized as one of the challenging part (Braunl, 2003, Broten et al., 2006) for fully functional and successful AMR.

To tackle the difficulty in developing software for AMR, many researchers in robotics research communities proposed the use of Component Based Software Engineering (CBSE) approach (Messina et al., 1999, Blum, 2001, Schlegel, 2006). A component-based solution can help robotic research groups in the following aspects (Oreback, 2000): 1) exchange of software parts or components between robotics laboratories, allowing specialists to focus on their particular field ; 2) comparison of different solutions would be possible from the available components ; 3) startup in robot research can be accelerated using the available components ; and 4) speed up the transfer of research laboratories works in mobile robot to commercial business application.

There have been some efforts on providing CBSE of robotics software in different approaches. Port-Based Object (PBO) component model was proposed to develop a software framework for CBSE for robotics (Stewart et al., 1997). The PBO is supported by an implementation based on domain-specific real-time operating system (RTOS) mechanisms.

Blum (2001), proposed a component-based system architecture Operating System for the Control of Autonomous Robots (OSCAR) for exploration of indoor environment with an AMR. OSCAR fully relies on the middle-ware standard CORBA 2.3 specified by the Object Management Group consortium. Since, OSCAR is based on CORBA component model; it is not suitable for implementation on resource limited embedded mobile robot systems. This can be seen from the implementation of OSCAR on MARVIN (Blum, 2001) robot which is equipped with four PCs running Linux operating system.

Orca is an open-source CBSE framework designed for mobile robotics (Brooks et al., 2005). An Orca component is a stand-alone process which interacts with other components over as set of well-defined interfaces. Orca framework provides the means for defining and implementing these interfaces. In Orca framework the processing power and memory requirements are large. Thus, Orca framework is only targeted for off-board

processing, i.e. Orca assumes some form of embedded software already present on AMR for communicating with the off-board components. Thus, requirements for self-contained AMR and predictable real-time performance of AMR systems are not considered in Orca framework.

Component-based development of AMR systems has been used in industries. Mobility is a commercially available component-based system produced by Real World Interface Company (iRobot, 2002), to support a certain class of mobile robot platforms which are produced by the company.

Most of the works on CBSE in robotics do not address the issues of ERT software for resource-constrained AMR and platform-independent components. As emphasized by Brega et al. (2000) and Pont and Siegwart (2005), real-world AMR must be self-contained and able to meet timing constraints. When on-board or embedded computation is required AMR software development is typically confronted with limited resources such as computing power and memory. With the limited computing power, a precise observation of the timing constraints of the AMR systems is a necessary to make complex robot systems more reliable. Most important, a predictable real-time performance on a robotic system is a necessary condition for guaranteeing a stable behavior of the robot (Buttazzo, 1996).

To tackle the above issues an implementation paradigm or framework for component-oriented programming (COP) of AMR software is proposed. The COP framework is a programming framework based the PErvasive COmponent Systems (PECOS) component model (Nierstrasz et al., 2002). The proposed framework enables the idea in PECOS to be implemented optimally without requiring any support tools and proprietary run-time environment from the original PECOS project. The deployment model in the proposed framework is based on C language for portability across platforms and compact code generation which is optimal for resource-constrained ERT systems.

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

2. Background

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 (Crnkovic, 2004, Rastofer & Bellosa, 2001). 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 PBO (Stewart et al., 1997), Koala (Ommering et al., 2000), PECOS (Nierstrasz et al., 2002), and ReFlex (Wall, 2003), 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 (Möller et al., 2004).

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 code from the real-time operating system used (Wuyts et al., 2005). 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 (Bouyssounouse and Sifakis, 2005).

Non-functional verification in PECOS is performed using Rate monotonic analysis (RMA) and schedule verification (Wuyts et al., 2005). The RMA verification is to check whether the entire components involved in the composition meet their deadlines while schedule verification is to check the possibility to fit execution and synchronization behavior sequentially in each task.

The original framework for implementation of PECOS components has some limitations 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, furthermore, Java and C++ supports may not be available for those microcontrollers; and 3) the RTE is difficult to develop and information on the RTE is not publicly accessible.

After experimenting with the PECOS component model and implementing it in the AMR applications, it was found that the assembling activities of components at design stage are simple but the implementation of the components is complex without the support from PECOS tools (Jawawi et al., 2006). With the absence of PECOS supporting tools and RTE, there are at least three options in which the PECOS component model can be used effectively in the AMR software development: 1) develop our own supporting tools and RTE; 2) seek alternative from other existing tools such as real-time Unified Modeling Language (UML) tools (Douglass, 2004); or 3) propose an alternative implementation framework for CBSE of AMR embedded software development.

Option three has been selected in this work, where a practical Component-Oriented Programming (COP)

framework is proposed and has been used to implement PECOS component model for CBSE of embedded AMR software.

3. The Component-Oriented Programming Framework

This COP framework was originally intended for development of ERT software for AMR. The target audience is the mechatronics and robotics researchers which are not from software engineering background and do not have extensive programming experience. However, experience shows that the COP 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.

The main features of the COP framework are: (1) use graphical representation for components definition and composition; (2) target the codes in C language for optimal code generation with resource-constrained micro-controller; and (3) depends on minimal requirement for run-time support.

Fig. 1 shows the main elements of the proposed COP framework and layers of dependency of each element. The main software elements of the COP framework consists of the component-based software to be developed, the hardware abstraction layer (HAL), and a real-time operating system (RTOS) with its associated abstraction layer (RTOS AL).

At the bottom layer is the robot mechanical and electronics hardware with the on-board embedded controller. The COP framework does not assume any particular underlying microcontroller unit and RTOS used in the robot hardware. It is up to the HAL and RTOS AL to abstract the robot hardware and its communication with the robot hardware. The HAL provides decoupling between the components and the underlying hardware, while the RTOS AL provides a thin layer of interface between the components and the actual RTOS used. In this way, the COP framework addressed platform independent issue. The dependency of PECOS model to RTE deployment model thus is replaced by the two abstraction layers.

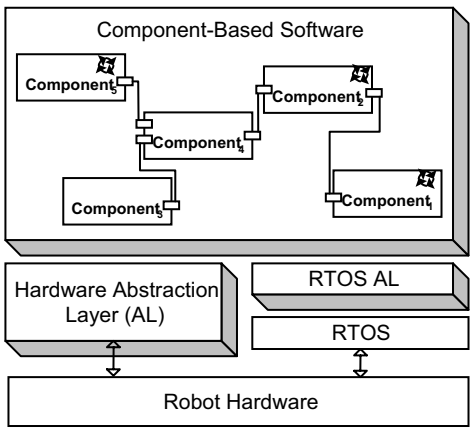

Fig. 1. The main elements of the COP framework

Porting the component-based software to different robot platform, thus, requires only the rewriting of HAL. We have ported an AMR software running on AMD188ES microcontroller to ATMEL MEGA32 microcontroller by just rewriting the abstraction layers only.

On the top of the COP framework is the component-based software to be developed. Those components consist of passive and active components. Active components have their own thread of control while the passive components do not have their own thread of control. The activations of active components are supported by a set of services provided by the RTOS.

To support component-based software development with the COP framework, three infrastructures were proposed based on PECOS component model: PECOS component model specifications and components repository in graphical blocks form, PECOS diagrammatical connection model with two enhancements as reported in (Jawawi et al., 2006), and a deployment model.

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 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.

In the following sections, the application of the COP framework in the two main engineering activities in the development of component-based software, i.e. component engineering and application engineering are explained. The above three infrastructures proposed here, namely, the graphical model specifications, connection model and the deployment model and their use in component-based development of AMR software are highlighted.

3.1. COP in Component Engineering Process

Component engineering process 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. As a result of a pattern mining process, currently ten software components were identified in the analysis pattern for typical AMR software. The components identified are: input-output, actuator, sensor, signal processing, motor control, communication, Human-Robot Interface (HRI), Behavior-Based Control (BBC), coordinator and planner.

Fig. 2 shows a generic component named `PID` documented in graphical block form. The input ports and

output ports, with 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, which has its own execution thread in multitasking environment, usually determined in the application engineering stage. Not shown in the block is the specification of worst case execution time (WCET) of the component.

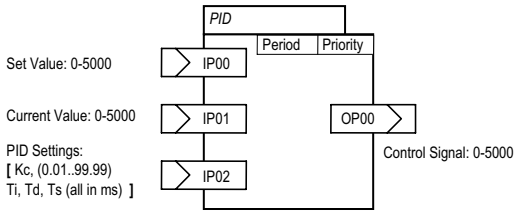

Fig. 2. A PID component documented in block form

The component specifications are further translated manually to C source codes and stored in repository for use in the application engineering stage. Each component is translated into two parts of C codes: code for interface and code body. For example, the generic interface code implementation for the PID component in Fig. 2 can be written in C is as follows:

```
typedef struct
{
     int   IP00;     // set value(0-5000 )
     int   IP01;     // current value(0-5000)
     float 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 RTOS through RTOS AL 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.

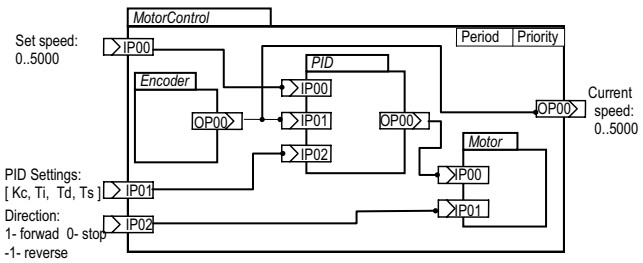

Fig. 3. MotorControl composite component

Composite components can be built by hierarchically composing a number of subcomponents and stored in the repository for later use. Fig. 3 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. In this way complex component can be built by hierarchically composing other composite components, and the user is only consent with the top level component interface and synchronization, while the inner components synchronization are handled by the top level component.

3.2. COP in Application Engineering Process

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

The major steps involve in application engineering of the AMR software 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. Fig. 4 shows a block diagram composition of the AMR software for MobileRobot1, which consists of eight leaf components and seven 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 Fig. 4, 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 Fig. 4 by the connections of constants to input port (IP01) of motorctrl_right and motorctrl_left components. This gives flexibility for testing and debugging of components and also useful for AMR software to reconfigure components for use with specific hardware or application.

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 theories. The Rate Monotonic Analysis (RMA) (Klien et al., 1993) theory is mainly used to allocate priority in this framework.

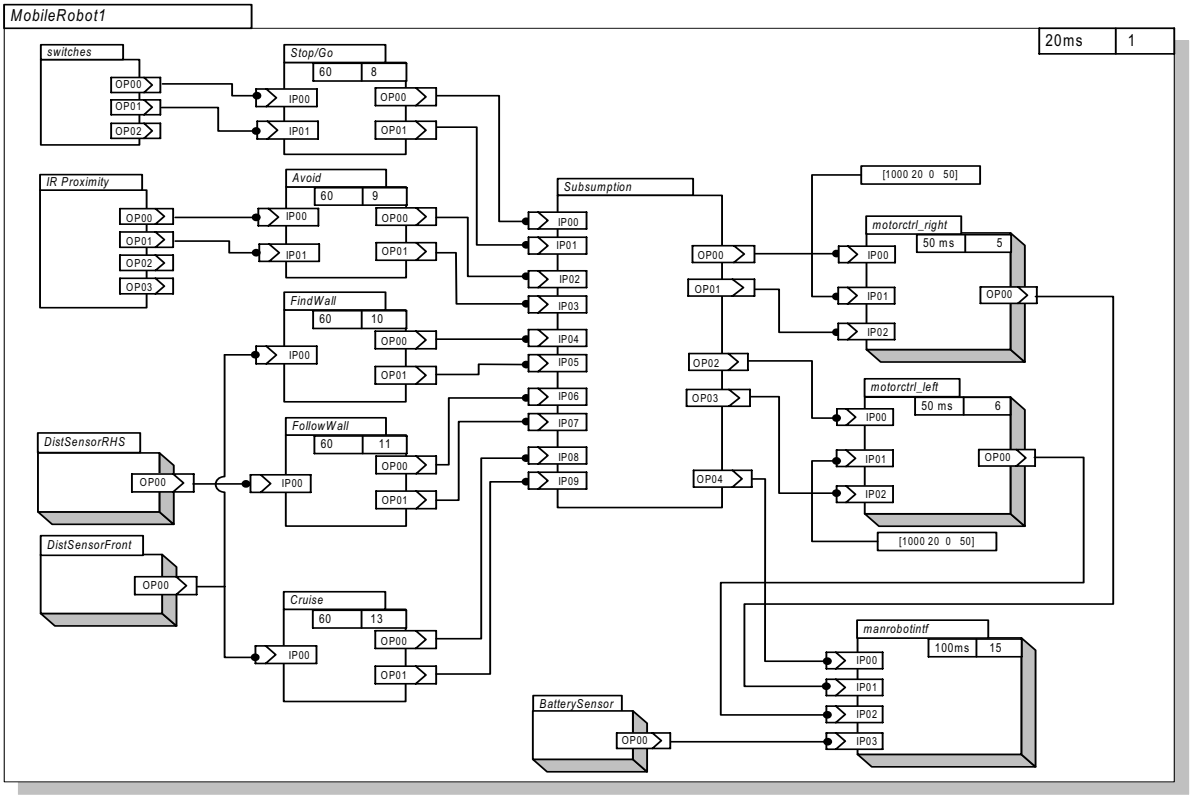

Fig. 4. MobileRobot1 components composition

The `MobileRobot1` components of Fig. 4 are mapped to a task which executes sequentially five passive components i.e. `switches`, `IRProximity`, `DistSensorRHS`, `DistSensorFront` and `BatterySensor`; and nine synchronization parts of its child components, i.e. `Stop/Go`, `Avoid`, `FindWall`, `FollowWall`, `Cruise`, `motorctrl_right`, `motorctrl_left`, `Subsumption`, and `manrobotintf`. The mapping results of all components in `MobileRobot1` composition are tabulated in Table 1. The tasks in the table are ordered from highest to lowest priority. The WCET were measured experimentally in real-time using oscilloscope and a spare bit of output port on the microcontroller, and the tasks periods were obtained from the AMR requirements.

Task	Period (ms)	WCET (ms)	R_i (ms)
`MobileRobot`	20	2	2
`Motorctrl_left`	50	3	5
`Motorctrl_right`	50	3	8
`Subsumption`	50	1	9
`Stop/Go`	60	1	10
`Avoid`	60	1	11
`FindWall`	60	1	12
`FollowWall`	60	1	13
`Cruise`	60	1	14
`Manrobotintf`	100	16	32

Table 1. The timing results for the MobileRobot1

Response time analysis proposed by Audsley et al. (1993) is then used to determine whether the deadline for each task in Table 1 can be met. The goal of response time schedulability analysis is to show that each task invocation finishes before its deadline. For each invocation of a task i, worst-case response time R_i is calculated and compare with deadline D_i. Equation 1 gives the total calculation for R_i of a task i by adding its own WCET C_i plus the interference on how often higher-priority task j invocation can preempt an invocation of task i.

$$R_i^{x+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^x}{T_j} \right\rceil C_j \qquad (1)$$

Since, there are ten tasks derived from the AMR component composition, the analysis were applied for values of i ranging from 1 to 10.

For each i worst case response time R_i was calculated and compare with deadline where $D_i = T_i$, the task period of task i. The calculated R_i for all tasks from the design composition in Fig. 4 are listed in Table 1. This indicates all R_i are within their deadlines, hence the design composition of components in Fig. 4 is predicted to be schedulable according to RMA theory. Fig. 5 shows a time line description of how the `MobileRobot1`'s tasks will be scheduled. The schedule is generated using TimesTool simulator from www.timestool.com (Amnell et al., 2003)

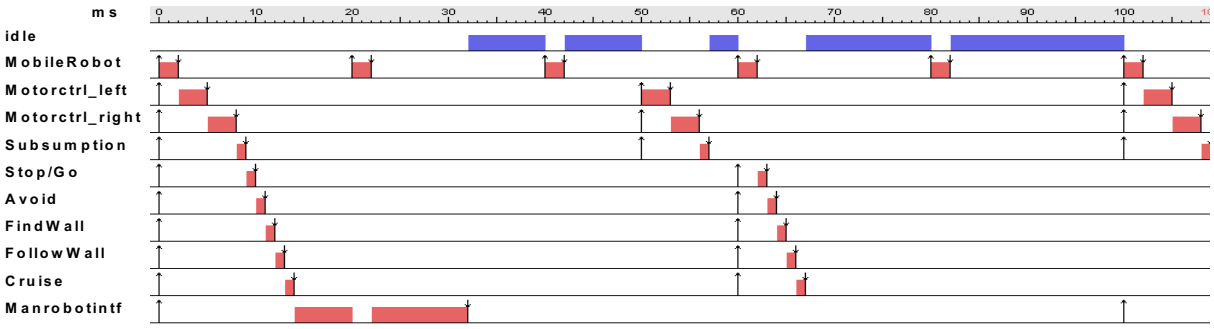

Fig. 5. MobileRobot's task schedule

Once the tasks with the periods and priorities shown in Table 1 were verified to be schedulable, a configuration file **pecos_cfg.h** is created for inclusion in the main program. This configuration file specifies which 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 RTOS use. The following code shows a fragment of the configuration definitions in the **pecos_cfg.h** file.

```
// ******** 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_BATTSENSOR_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              100
#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 assigning the related input and output ports defined by the components interface. For example the following code shows how the connections 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 Fig. 4, the top level component in this composition is the `MobileRobot1` component which has the period of execution of 20 ms and priority of 1, i.e. the highest priority. Thus, the main program for this composition is the execution behavior of `MobileRobot1` component as already discussed in Section 3.1. The template for this main file is shown in Fig. 6.

```
#include "pecos_cfg.h"
#include "includes.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 RTOS to create periodic execution
        OSTimeDelay (MOBILEROBOT_PERIOD);
}

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

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

4. Case-Study Result

Following the framework described in Section 3 and PECOS component definitions, the AMR software composed shown in Fig. 4 was implemented on a real mobile robot with behavior-based intelligence. 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 (Paradigm, 2000) for generating ROMable code and µC/OS-II RTOS (Labrosse, 1999) for multitasking support. The result of implementing the

AMR application is software with about 5000 lines of C codes not including the RTOS source codes. The total binary 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 COP framework can generate application with minimal memories requirements, suitable for other resource constrained ERT systems.

Implementing the software by reusing at functions or libraries level will result in reduction of LOC numbers or size of binary codes since the codes to support component interfaces, synchronization and implementa-tion can be eliminated. However, we do not intend to compare the reuse with functions and reuse with components due to the advantages offered by component-based reuse approach. This is one of the trade-off one has to make in using component-based approach.

4.1. Measuring Amount of Code Reuse

The main concern in the COP framework is the reuse of the components on different platforms and different physical sizes/shapes of AMR. To quantify the benefit of using the COP framework for software reuse, the amount of reuse metrics were used to measure the reuse improvement effort. A common form of reuse metrics is based on lines of code i.e. the amount of reuse percentage is the percentage of lines of reuse code in the software over the total line of code in the software (Frakes and Terry, 1996).

The strategy adopted in measuring amount of reuse was to use two AMR systems with different mechanical and computation platforms and different sizes. The first AMR is called `MobileRobot1` and the embedded software is the implementation of components composition shown in Fig. 4. The second AMR is called `MobileRobot2` and the software is the implementation of components composition shown in Fig. 7. Table 2 tabulates the specifications of platforms and sizes/shapes of the two AMRs. The task of `MobileRobot1` is to find exit passage in an environment surrounded by walls while avoiding obstacles. The mission of `MobileRobot2` is to navigate in a room with obstacles trying to locate fire and automatically extinguishes the fire. To perform these different tasks, the two AMR are equipped with sensors and actuators as shown in Table 3.

From Table 3, it can be seen that two hardware components exist in both systems are motor and distance sensor. The type of distance sensor used in both robots is the same but the type of motor used in the two robots is different. Thus, it expected that reuse of distance sensor component will be high, while the reuse of motor component will be low.

In this measurement strategy, the amount of software reuse involves in developing `MobileRobot2` software from the `MobileRobot1` software is measured. It is assumes that the components from the `MobileRobot1` software can be treated as reusable components for developing the `MobileRobot2` software. This is true in this case because the software components are: i) generic which were derived from AMR analysis pattern and 2) specified using the modified PECOS component model.

To measure the software reuse rate, line of code (LOC) is used because it is a common metric used in measuring development effort and reuse rate (Rothenberger and Hershauer, 1999).

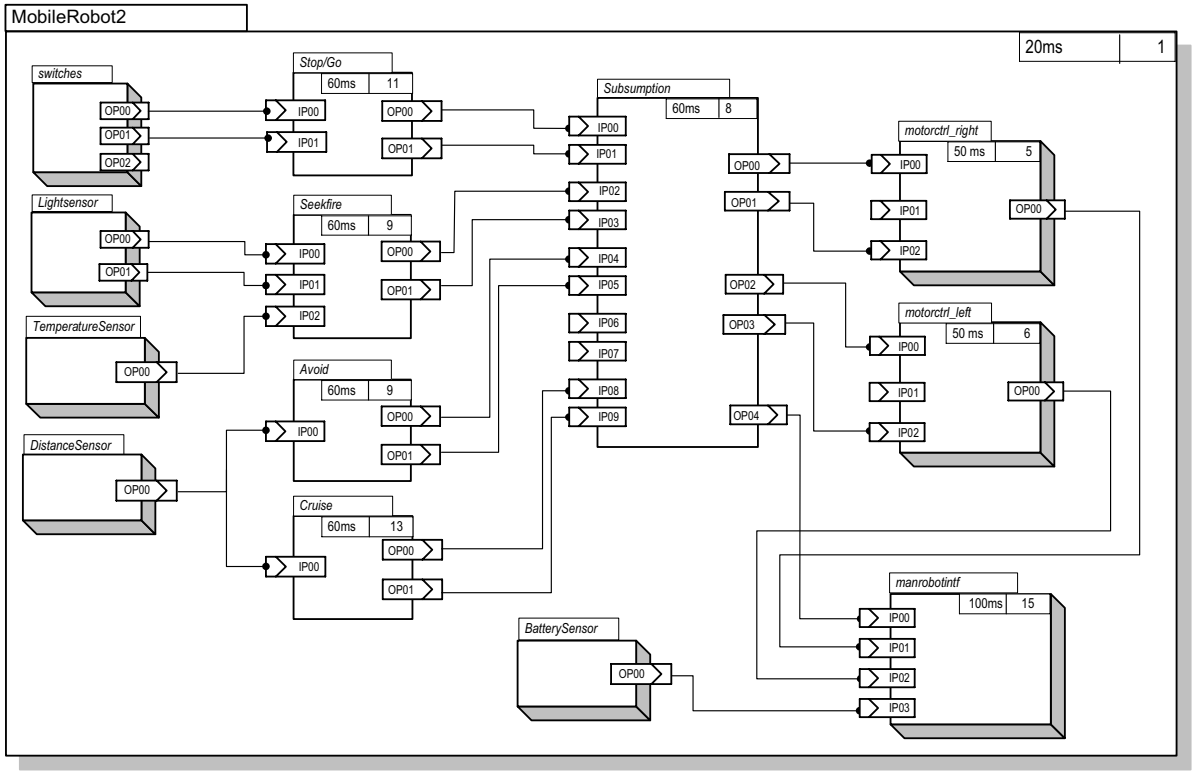

Fig. 7. MobileRobot2 components composition

AMR	MobileRobot1	MobileRobot2
Processor Type	AMD188ES (16 bits)	ATMEL AVR MEGA32 (8 bits)
Size (cm)	40	16
Shape	round	octagonal
EPROM (Kb)	64	32
RAM (Kb)	128	2

Table 2. The AMR systems processor platform and size

AMR	MobileRobot1	MobileRobot2
Motor	2	2
Fan	0	1
Encoder	2	0
Distance Sensor	2	1
Proximity Sensor	2	0
Temperature Sensor	0	1
Light Sensor	0	1

Table 3. The AMR systems sensors and actuators

The two ways for amount of reuse analysis were performed on the components assemblies are:

1. Count the LOC by ignoring the hardware dependence modules which include the HAL and some platform specific parts of μC/OS-II RTOS. Thus, only the platform independence components are considered.
2. Count the LOC considering all the codes.

Without considering the hardware dependent module and RTOS, the amount reuse percentage is calculated as:

$$\frac{Reused\ LOC}{All\ Components\ LOC + pecos\_cfg\ LOC + main()\ LOC} \times 100\% \quad (2)$$

Equation 2 is also used to calculate the new and changed percentage by changing the *Reuse LOC* variable. Fig. 8 shows the composition of reuse, change and new software in developing the `MobileRobot2` software.

Fig. 8 shows that 74% of the software is reused from the `MobileRobot1`'s components, 25% software is new since two new components are required to handle new sensors not previously available on `MobileRobot1` software, i.e. `LightSensor` and `TemperatureSensor`, a new behavior (`BBCSeekFire`) is required for the mission in `MobileRobot2`, and two modules need to be rewritten to configure and integrate the components (pecos_cfg and main function of `MobileRobot2`).

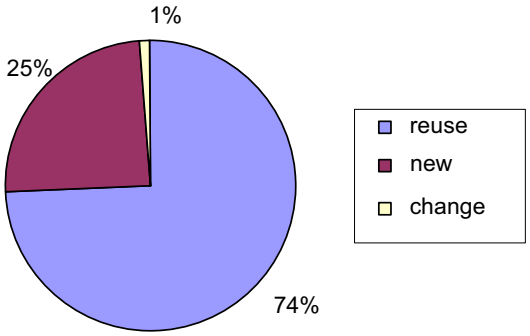

Fig. 8. MobileRobot2 platform-independence software composition

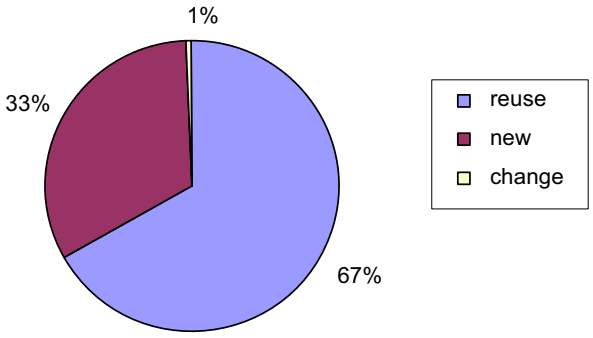

Fig. 9. MobileRobot2 platform independent and platform dependent software composition

Fig. 8 also shows that only 1% changes need to done on the two reuse components `BatterySensor` and `BBCAvoid`. These two components are considered hardware dependence and robot dependence components. The changes in `BatterySensor` component is required to accommodate different battery voltage on `MobileRobot2`. In `BBCAvoid` component, the changes is required because the size and shape of the `MobileRobot2` are different from `MobileRobot1` and this requires some code changes in the `Avoid` behavior. In `MobileRobot2` components composition, five of six hardware dependence components from `MobileRobot1` i.e. `Analogin`, `Motor`, `LCD`, `DistanceSensor` and `Switches` can be reused 100% without change. This is possible with the use of HAL as described in Section 3.

If the hardware-dependence modules and components are included to calculate the reuse percentage, the reuse percentage will reduce to 67% as shown in Fig. 9. The reason for this reduction is that new modules and components need to be written to support:

1. different processor and hardware requirements, RTOS platform (67% of new LOC consists of `pecos_cfg`, HAL and porting components of μC/OS-II RTOS)
2. different mission of AMR (34% of new LOC consists of modules or components `pecos_cfg`, `APIBOTmain`, `BBCSeekFire`, `LightSensor`, and `TemperatureSensor`).

5. Conclusion

A practical COP framework for CBSE of autonomous mobile robot embedded software is proposed to better support embedded resource-constrained AMR components integration and composition. To support this COP framework, three infrastructures were developed based on PECOS component model: PECOS component model specifications and components repository in graphical blocks form, PECOS diagrammatical connection model, and a deployment model.

The proposed COP framework enables the idea in PECOS to be implemented optimally without requiring any support tools and proprietary run-time environment from the original PECOS project. The deployment model in the proposed framework is based on C language. Templates for components and code skeletons for AMR software are provided in C language. As the C language is the common and familiar language for the target audience, i.e. robotics engineers with little training in software engineering, these strategies will speed up the development of AMR application software.

The COP is used to implement embedded software on two real AMR systems with behavior-based intelligence. Results show that, the modified PECOS component model together with the developed COP framework can generate software with minimal memories requirements, suitable for platform-independent resource constrained ERT systems. This is achievable in the COP by removing the dependency on run-time environment with abstraction layers, targeting the codes to standard calls for minimal RTOS, and the use of C language.

The amount of reuse measurement showed that up to 74% reuse rate can be achieved in designing a new component-based robot software from existing software components. This indicates that the use of PECOS based component model enable component engineering products to be created directly from other projects. This measurement also prove that the introduction of HAL and RTOS abstraction layers in the proposed COP framework can increase the reuse capability of the design components in the component-based software development.

6. References

Amnell, T.; E. Fersman; L. Mokrushin; P. Pettersson & Yi, W. (2003). TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems, *Proceedings of the 1st International Workshop on Formal Modeling and Analysis of Timed Systems FORMATS 2003*, pp. 60-72, September 6-7 2003, Marseille, France

Audsley, N.; Burns, A.; Richardson, M.; Tindell, K. & Wellings, A. (1993). Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling, *Software Engineering Journal*, Vol. 8, No. 5, pp. 284-292

Blum, S. (2001). Towards a Component-based System Architecture for Autonomous Mobile Robots, *Proceedings of IASTED International Conference Robotics and Applications (RA'01)*, pp. 220-225, Tampa

Bouyssounouse, B. & Sifakis, J. (2005). Embedded Systems Design: The ARTIST Roadmap for Research and Development, *Lecture Notes in Computer Science*, Vol. 3436 / 2005, Springer-Verlag GmbH, pp. 160-194.

Braunl, T. (2003). Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems, Springer-Verlag

Brega, R.; Tomatis, N. & Arras, K.O. (2000). The Need for Autonomy and Real-Time in Mobile Robotics: A Case Study of XO/2 and Pygmalion, *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)*, Volume 2, pp. 1422 – 1427 Takamatsu, Japan, October 30- November 5 2000

Brooks, A.; Kaupp, T.; Makarenko, A.; Orebäck, A. & Williams, S. (2005) Towards Component-Based Robotics, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)*, August 2-6 2005, Edmonton, Alberta

Broten G.; Monckton, S.; Giesbrecht, J. & Collier, J. (2006). Software Systems for Robotics: An Applied Research Perspective. *International Journal of Advanced Robotic Systems*, Vol. 3, No. 1, (March 2006), pp. 11-16, *ISSN 1729-8806*

Buttazzo, G. C. (1996). Real-time Issues in Advanced Robotics Applications, *Proceedings of the 8th IEEE Euromicro Workshop on Real-time Systems*, pp. 133-138 L'Aquila, Italy, June 1996

Crnkovic, I. (2004). Component-based Approach for Embedded Systems, *Proceedings of 9th International Workshop on Component-oriented Programming*, Session 4 – Application of CBSE, Oslo, June 2004

Douglass, B. P., "Real-Time UML". Addison Wesley. 2004

Frakes, W. & Terry C. (1996). Software Reuse: Metrics and Models. *ACM Computing Surveys (CSUR)*, Vol. 28, No. 2, pp. 415 – 435

Genssler, T. *et al.*, PECOS in a Nutshell, *Technical Report*, PECOS Project, September 2002

iRobot Corporation. (2002). Mobility Robot Integration Software User's Guide

Jawawi, D.N.A; Deris, S. & Mamat, R. (2006). "Enhancements of PECOS Embedded Real-Time Component Model for Autonomous Mobile Robot Application". *Proceeding of The 4th ACS/IEEE International Conference on Computer Systems and Applications*, pp. 882 – 889. Dubai/Sharjah, March 8-11 2006

Klien, M.; Ralya, T.; Pollak, B. & Obenza R. *A Practitioner's Handbook for Real-time Analysis*, Kluwer Academic Publisher, 1993

Labrosse, J. J. (1999). MicroC/OS-II The Real-Time Kernel, 2nd edition, R&D Books, USA.

Messina, E.; Horst, J.; Kramer, T.; Huang, H. & Michaloski, J. (1999). Component Specifications for Robotics Integration, *Autonomous Robots*, Vol. 6, pp.

Möller, A.; Åkerholm, M.; Fredriksson, J. & Nolin, M. (2004). Evaluation of Component Technologies with Respect to Industrial Requirements, *Proceedings of the 30th EUROMICRO Conference on Component-Based Software Engineering Track*, pp. 56-63, Rennes, France, August 2004

Nierstrasz, O. *et al.* (2002). A Component Model for Field Devices, *Proceedings of 1st International IFIP/ACM Working Conference on Component Deployment*, Springer-Verlag Heidelberg, Vol. 2370, pp. 200-209, Berlin, Germany, June 2002

Ommering, R.; Linden, F.; Kramer, J.; & Magee, J. (2000) The Koala Component Model for Consumer Electronics Software, *IEEE Computer*, Vol. 33, No. 3, pp. 78 –85

Oreback, A. (2000). Components in Intelligent Robotics, *MDH-MRTC-15/2000-1-SE Component Based Software Engineering - State of the Art*, Mälardalen Real-Time Research Centre Mälardalen University, 233-243. ISSN 1404-3041

Paradigm Systems (2000). Paradigm C++ Reference Manual Version 5.0, Endwell

Pont, F. & Siegwart, R. (2005). A Real-Time Software Framework for Indoor Navigation, *Proceedings of the IEEE/RSJ Intenational Conference on Intelligent Robots and Systems (IROS)*, pp. 2085 – 2090, Edmonton, Canada, August 2-6 2005

Rastofer, U. & Bellosa, F. (2001). Component-based Software Engineering for Distributed Embedded Real-time Systems, *IEE Proceeding- Software*, Vol. 148, No. 3, pp. 99-103

Rothenberger, M. A. & Hershauer, J. C. (1999). A Software Reuse Measure: Monitoring an Enterprise-Level Model Driven Development Process. *Journal of Information & Management*, Vol. 35, No. 5, pp. 283-293

Schlegel, C. (2006). Communication Patterns as Key Towards Component-Based Robotics Task. *International Journal of Advanced Robotic Systems*, Vol. 3, No. 1, (March 2006), pp. 49-54, *ISSN 1729-8806*

Stewart, D. B.. Volpe, R. A & Khosla, P. K. (1997). Design of Dynamically Reconfigurable Real-time Software Using Port-Based Objects, *IEEE Transaction on Software Engineering.* Vol. 23, No. 12, pp. 759 –776

Wall, A. (2003) Architectural Modeling and Analysis of Complex Real-Time Systems, *Mälardalen University*, Phd Thesis

Wuyts, R.; Ducasse, S. & Nierstrasz, O. (2005). A Data-centric Approach to Composing Embedded, Real-time Software Components, *The Journal of Systems and Software*, Vol. 74, pp. 25-34