A Code Generator for Component Oriented Programming Framework

Mahmood Aghajani Siroos Talab
Department of Software Engineering
Faculty of Computer Science and InformationSystem,
Universiti Teknologi Malaysia
Johor Bahru, Malaysia
E-Mail: ajstmahmood2@live.utm.my

Dayang N. A. Jawawi
Department of Software Engineering
Faculty of Computer Science and InformationSystem,
Universiti Teknologi Malaysia
Johor Bahru, Malaysia
E-Mail: dayang@utm.my

***Abstract*—A component-based Embedded Real Time (ERT) system can improve the system development through enhancing the ability of experts in developing sophisticated ERT software. In order to generate codes, ERT code generator system frameworks have generally two parts: a graphical semi-formal-model representation to define the components, and an optimal code generator that generates codes for a resource limited micro-controller. The Unified Modelling Language(UML) semi-formal-model system does not deal with minute details of the code generation (e.g. IF, THEN, ..). Currently, code generators use a state diagram semi-formal-model to provide optimal codes. However, manual coding has not been quit entirely since current state diagrams are not capable of providing details in code generation. The Component Oriented Programming framework (COP) is an ERT framework which targets on the code generation required in Autonomous Mobile Robot (AMR). Currently, the COP framework can not provide the AMR with code details. This paper presents an implementation of a code generator through exploiting a previously used algorithm to create a state diagram based on information taken from the component composition in the COP framework. Consequently, manual coding in the COP framework can be minimized.**

***Index Terms*—automonous mobile robot, code generation, component oriented programming, embedded realtime system, state machine**

1. Introduction

Currently, embedded systems are used in many applications such as mobile phones, washing machines and robotics [1]. The implementation of an embedded application is complex because of many constrains such as time, hardware (e.g battery) and computing logical results in a minimal code size [2].

In addition, the efforts of dealing with the ERT system complexity have been increasing since the last two decades because of enormous life threatening damages like the Chernobyl nuclear power plant accident in 1986 and the oil spills from the Exxon Valdez in 1993 [3]. Hence, investigating ways to handle the complexity is a crucial demand if a reliable system is to be developed since it can reduce risks and mishaps.

Therefore, recent researchers have introduced some special frameworks to implement ERT systems. However, many designs and methodologies are not well-defined or standardised to build reliable ERT frameworks. Generally, conventional implementation of ERT frameworks is based on manual implementation of an abstract model such as the state transition graph which uses various types of design patterns. Although state machines implemented by design patterns provide reusable source codes, manual coding is still being done [4].

Although optimized compilers improve the code generation in an ERT system, most of them are still unable to perform related modelling language semantics [1]. Therefore, model based development of code generators is one of the reliable solutions due to the capability of the approach in providing a predictable design development. For this reason, state machines are a good alternative to be considered in ERT frameworks instead of directly observing and manually modifying the codes.

A model oriented design provides highly reliable design and development. Therefore, the user is responsible for high level software description and verification. In the next phase, the model oriented design is translated to an executable code by a software tool [5].

Thus, the combination of model oriented design and Component Based Software Engineer (CBSE) is a suitable concept for designing and implementing ERT systems. CBSE improves the software quality besides keeping the cost of the software development low. CBSE provides many advantages such as fast development time with minimum cost, and developing the complex software based on reusability [6].

Descriptions used in analysis, design and implementation of component base ERT software can be categorised into structure and behaviour. An ERT system's graphical composition can represent a structure of ERT system, and its behaviour is implemented by state diagrams. However, the behaviour of tasks description which is achieved by UML modelling cannot provide details about the required code.

Recently, researchers use state machines to model the behaviour of ERT system [7]. Indeed, various types of design patterns and reconfigurable software components are represented by state machines. Although some design patterns provides hierarchical and concurrent state machines by advanced methods such as [4] and [5], but Kraemer [8] framework is related for the implementation of code details in COP framework for the reasons represented in Section II.

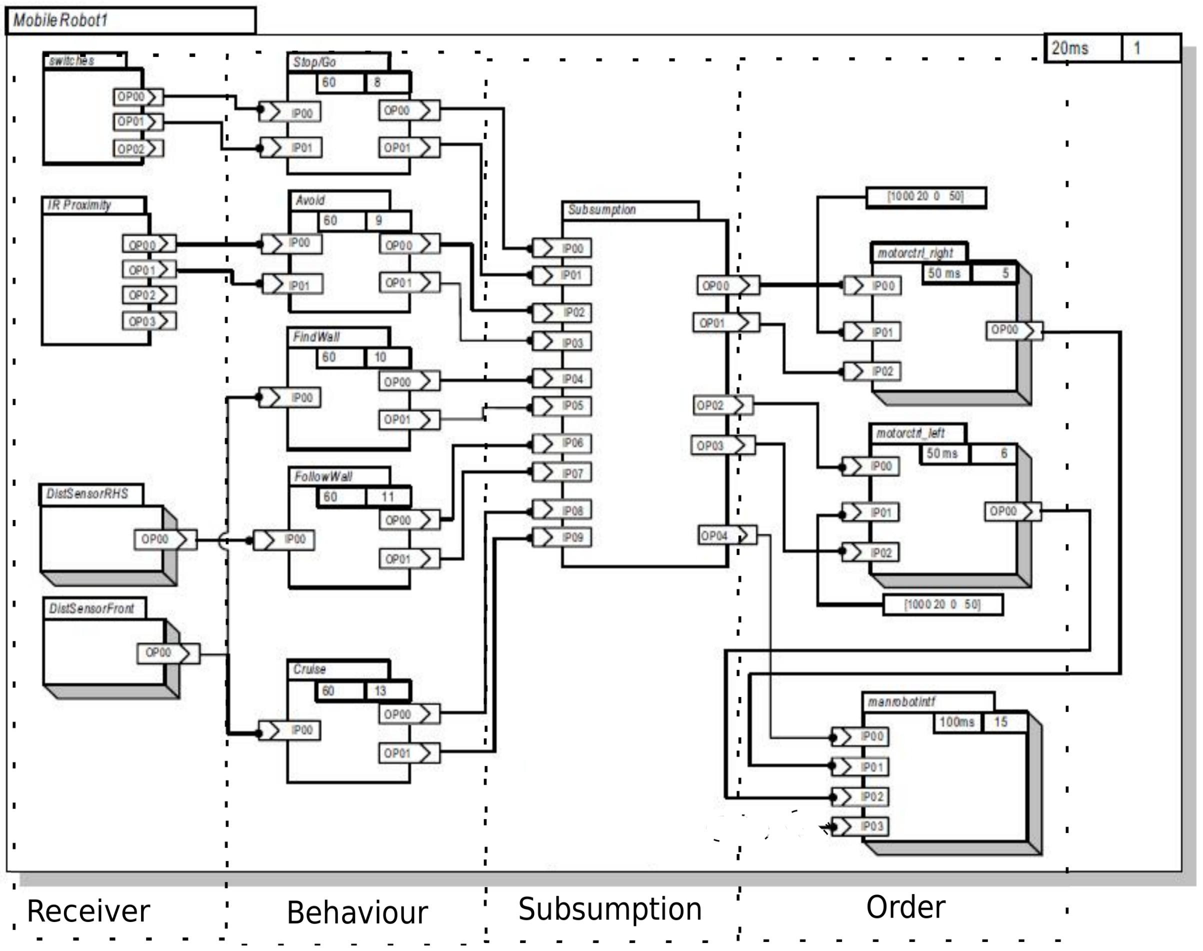

Fig. 1. Component composition of the AMR

Kraemer [8] has provided an algorithm to transform UML activities into state machines. Furthermore, UML activities represent the behaviour of component composition before representing the component behaviour by state machines. However, state machines are generated as an individual component description to generate the code.

COP enables programs to be constructed from pre-built software components, which are reusable and self-contained blocks of computer code [9]. A new COP framework was introduced in [6] for code generation of components and composition in AMR implementation.

This study implements the behaviour of the COP presented in [6] based on the method in [8]. Hence, UML activities will be implemented for AMR. COP may provide a special behaviour because that follows a special AMR architectural model. This behaviour will be made clear in the next section.

The layout of this paper is as follows. Section II describes the COP Framework. Section III describes the integration of COP framework and a model transformation method based on [8]. The generation of activity diagram for AMR is discussed in section IV. State machines' generation and code generation are elaborated in section V. The discussion is made in section VI and finally the work is concluded in Section VIII.

2. A COP Framework based on PECOS Model

The software industry is changed from giant to binary component since maintaining the giant code base is difficult [4]. COP can support greater reusability, extensibility, and maintainability. These software qualities are suitable in the implementation of ERT system by using the CBD [10].

A new framework called COP was proposed in [6]. Visualisation of components is applied for component definition and composition. Currently, the code has to be manually written by translating the graphical visualisation which is shown in Fig. 1.

COP framework originally intends to develop ERT software in AMR. Mechatronic and robotic researchers who do not have an extensive programming experience are the target audience of the COP framework. However, experience shows that the new framework is generally suitable for developing reactive embedded systems that typically use 8 bit or 16 bit microcontrollers with limited memory.

In the following subsection, two CBSE activities, component engineering and application engineering, are discussed to

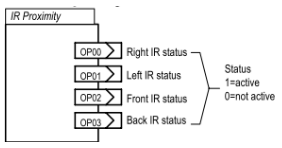

Fig. 2. IRProximity Component

support the component-based software systems based on the COP framework.

A. Component Engineering

Component engineering is concerned with the analysis of domains and development of generic and domain-specific reusable components. Graphical block forms document common components that are identified based on pattern analysis in the domain of AMR software.

Common components are further manually translated to C source codes and stored in the repository for use in the application engineer stage. Generic components, namely, `IRProximity` is documented in graphical block forms shown in Fig. 2.

Each component has two port types, inports and outports. Bi-directional ports are not allowed in this framework. Timing specifications that are determined in the application engineer stage are managed by the Period and Priority fields. The code body of an active component contains three parts: initialisation, execution, and synchronization.

B. Application Engineering

Component Based Development (CBD) application engineering branch is involved with developing applications using previously developed software components. Application engineering steps that use development COP framework illustrate the design of an AMR embedded software.

The important steps involved in the application engineering contain the identification of required components, composition of components, scheduling analysis, assigning the period and priority of components and code writing. Connecting the compatible ports compose and identify the required components based on AMR requirements which is shown in Fig. 1.

Moreover, the embedded software must support the intelligence aspect of the robot in order to response to the conditions in the environment when achieving the goal. The intelligence of AMR is supported by a behavior-based control using subsumption architecture that is shown in Fig. 1. Features of `IRProximity` are illustrated in [11].

This paper categorizes components of AMR in four partitions according to their common behaviour; Receiver,Behaviour,Subsumption and Order which are shown in Fig. 1. Many features of components in each partition are similar. For example, `IRProximity` and `DistanceSensor` components which are shown in Fig. 1. react based on sensors data.

3. Integrating model transformation method in COP framework

Local behaviours of a set of components identify the interaction to fulfil certain tasks. However, the description of the components' behaviour is not enough for code generation in COP. Although some ERT frameworks are introduced with a capability of supporting code generators like Rational Rose. Nevertheless, the state of design and implementation of an application is not easy for users who have no sufficient software engineering knowledge.

Although the COP framework provides proper schema for the required AMR code according to component composition which is shown in Fig. 1., it is unable to implement some required code details or transitions. Recently, code generators have used state diagrams to generate optimal codes. In fact, advanced methods provide design patterns for concurrent and hierarchical state machines, which are reusable at the source-code level [4].

A model transformation proposed in [8] can generate a state diagram by a special algorithm in a form of activities. The most important feature of the work in [8] is that the steps that lead to an efficiently executable code from collaborative service specifications are completely automated. They are elaborated as follows:

1) Systems are decomposed into physically distributed compositions which are modelled separately. The composed behaviour of components specifies the services indirectly. Services are modelled by a number of collaborations as main structuring elements. This is referred to as collaboration-oriented perspective. A collaboration specifies the interactions between the components involved in it. Structural aspects of collaboration and composition are illustrated by the concept of UML 2.0.
2) The description of a component oriented perspective is introduced to represent the behaviour of the individual component which is applied for the execution of the system. In [8], executable UML 2.0 state machines serve as inputs to the code generator.

The algorithm introduced in [8] automatically transforms the collaboration oriented specification into a form of state machines by using an activity diagram. The advantages of this transformation are: (1) it accelerates the development of the services dramatically, (2) state machines are created without any intermediate representation and with efficient memory usage by the algorithm, and (3) a service can be modified by regenerating the state machines.

COP in [6] and Kraemer in [12] are two frameworks that utilize the same component composition approach. COP component composition uses a partitioning concept which is explained in the next section. Similarly, Kraemer's collaboration applied the partitioning concept in the activity diagram as well. Regarding the similarity between COP and Kramer frameworks, the integration of these two frameworks can provide a suitable platform for developing the AMR. Fig. 3. shows the integration of these two frameworks. Component

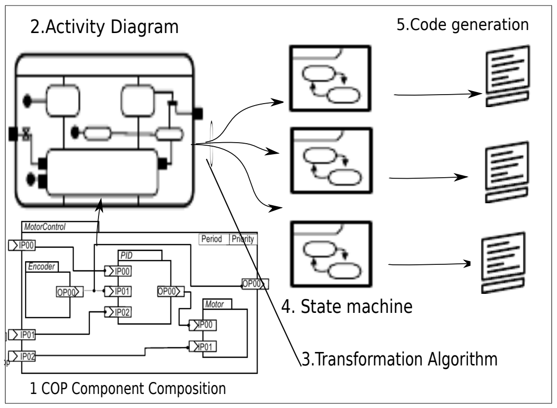

Fig. 3. Integration model transformation method in COP framework

composition of Kraemer [12] can support the general area of ERT system, while COP component composition is applicable just for AMR. This study replaces Kraemer's collaboration [12] with COP component composition, as shown in (Fig. 3. no. 1), and implements activity diagrams (Fig. 3. no. 2), state machines and code generation based on Kraemer's framework (Fig. 3. no. 4 and 5).

COP framework was unable to implement code details before its integration with the state machine. While COP framework can implement the required code details based on Kraemer [12] framework and maintain the structure of COP's schema code.

In addition, Kraemer's collaboration [12] needs a software designer to reconfigure the component composition for new ERT system. While component composition of COP follows a fixed design procedure which is shown in Fig. 1. Hence, the integration of COP and Kraemer [12] frameworks provides two advantages: (1) implementing COP code details (2) decreasing the component composition's design cost.

4. Generating The Activity Diagram For AMR

This study considers the method mentioned in [12] to be implemented in AMR. Hence, the activity diagram is applied in building blocks as a reusable specification. The previous component-based approach behaviour was represented by a direct collaboration in COP framework. In [12], the input and the output of the component composition are identified as activity diagrams that are connected together.

The structural aspect of the composition is represented by the component composition shown in Fig. 1. The behaviour of the component composition aspect is described by an activity diagram. The activity of `IRProximity` , which is shown in Fig. 4., consists of two partitions: (1) a receiver that receives data from the sensor, (2) AMR which reacts suitably based on the received data. Fig. 2. shows the structure composition of the `IRProximity` component which is used for generating the activity diagram. For each component used in Fig. 1., a structure node in the activity diagram that specifies the behaviour contributed by the component composition can be found.

The sensor is a part of the receiver partition that communicates with the AMR partition by sending and receiving signals of action. The receiver receives an ID from the sensors and forwards it to the `IDValidation` in the AMR Partition. Depending on the result received from the ID validation (Valid or Invalid), the decision node will either validate the ID and send it to the output or will reject the corrupted ID and returns back to the listening sensors. These procetures is shown in Fig. 4.

The behaviour of the `Distance` sensor shown in Fig. 1. is similar to that of the`IRProximity` . This means that the Receiver partition should create listening code to receive the data from `Distance` sensor. Despite some similarities in behaviour between these two components, the type of the data sent to the `Behaviour` partition is not compatible with the other component. Hence, the code generator must consider a general data type for the data received by the sensors.

In the next step, if the Avoid component is not busy (`bbcavoidstate==0`), it receives sensor data from the Receiver partition (e.g`IRProximity` component). Fig. 5. shows the activity diagram of `Avoid` Component. `Avoid` component saves the wheels' speed, then the left or right wheels' speed reduces for five seconds according to Sensor signals. Finally, the speed is restored based on the backup.

`Avoid` component states are common among all AMR generations, and they include: (1) receiving the sensor data, (2) saving the speed of the wheels, (3) setting the timer, (4) changing the speed of the left or right wheels based on sensor signals, (5) restore the speed of the wheels.

IRProximity component is represented by a formal activity diagram shown in Fig. 5. The formal activity diagram is improper to represent complex components for users due to a congestion of signals and transactions. So a component shape is introduced for representing the transaction of the input and output signals in the activity diagram. In fact, Sensor component has two input ports as shown in Fig. 5. These ports receive their signals from `IRProximity` . After an evaluation is done for the input signals, `Avoid` component sends a signal for a suitable speed control via `Motorspeed` component.

After making a decision in `Avoid` component, COP framework must manage the priority execution time. It is done by `Subsumption` component which is shown in Fig. 6. Priority of the `Subsumption` is designed in a top to bottom manner, which means that the first input port has the highest priority while the last input port has the lowest. `Subsumption` receives two signals to react in front of the obstacle: (1) reducing the right wheels speed to turn right, (2) restore the right wheels speed after 5 seconds.

All signals sent to `Subsumption` component serve in: (1) stopping right wheels, (2) stopping left wheels, (3) stop all wheels. Hence, the function of `Subsumption` component is common in many AMR generations.

Although the formal activity diagram can represent the `IRProximity` clearly, representing `Subsumption` with formal activity diagram will make it complex and less flexible

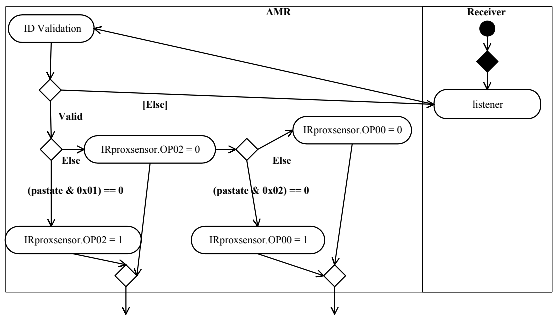

Fig. 4. IRProximity Activity Diagram

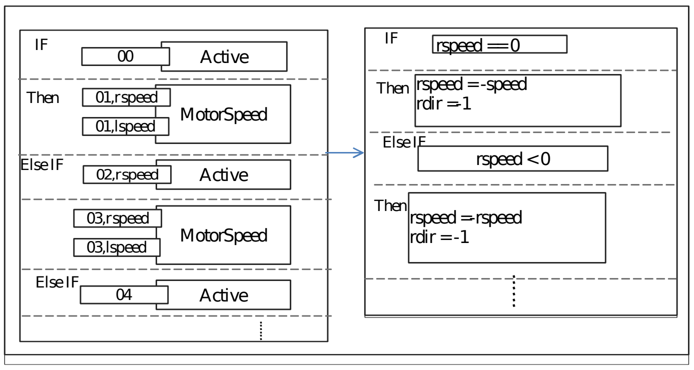

Fig. 6. `Subsumption` Activity Diagram

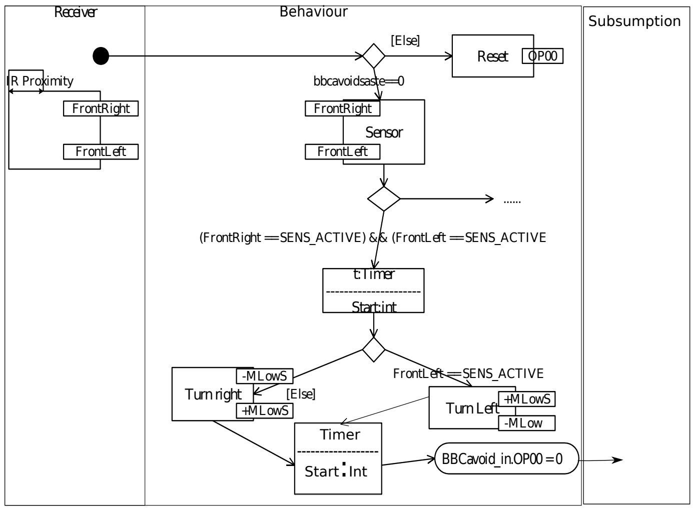

Fig. 5. Activity diagram of the `Avoid` component

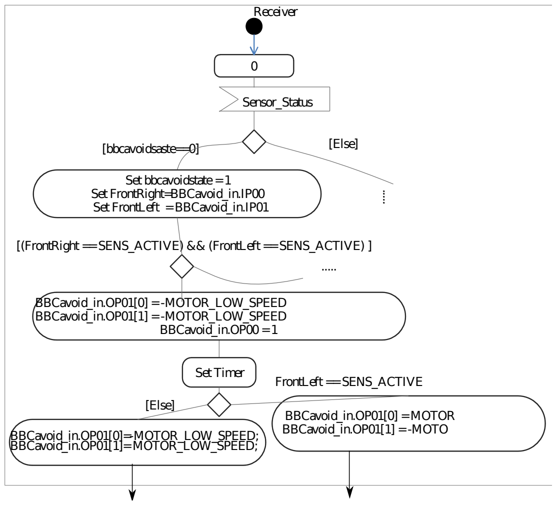
Receiver
0
Sensor_Status
[bbcavoidsaste==0]
[Else]
Set bbcavoidstate = 1
Set FrontRight=BBCavoid_in.IP00
Set FrontLeft = BBCavoid_in.IP01
[(FrontRight == SENS_ACTIVE) && (FrontLeft == SENS_ACTIVE)]
.....
BBCavoid_in.OP01[0] = -MOTOR_LOW_SPEED
BBCavoid_in.OP01[1] = -MOTOR_LOW_SPEED
BBCavoid_in.OP00 = 1
Set Timer
FrontLeft == SENS_ACTIVE
[Else]
BBCavoid_in.OP01[0]=MOTOR_LOW_SPEED;
BBCavoid_in.OP01[1]=MOTOR_LOW_SPEED;
BBCavoid_in.OP01[0] = MOTOR
BBCavoid_in.OP01[1] = -MOTO

Fig. 7. State machine of the `Avoid` component

for modification. Hence, this study proposes OMG UML 2.0 superstructure for implementing an activity diagram. Fig. 6. illustrates `Subsumption` activity diagram based on OMG UML 2.0 which is represents signals by lines. While new simple shapes have been introduced in this study to illustrate signal transactions and conditions at both the input and the output in a manner which is more clear and flexible for modification. In Fig. 6., Active condition tells whether the first input port of `Subsumption` is active or not (`Subsumption.IP00==0`). If the first input port is active, then `Motorspeed` receives two values from the second input port to assign `Rspeed` and `Lspeed` . On the other hand, output ports have the same behaviour as the input ports. As an example, the following code is represented by Fig. 6.

```
public void fireInitial()
if rspeed==0 then
  rspeed =-speed;
  subsumption.out00=rspeed;
end if
```

5. Generating the State machine and Code Generation for AMR

Each activity partition generates one state machine via a transformation algorithm which is presented in [8]. The state diagram shown in Fig. 7. is generated based on `Avoid` activity diagram that is shown in Fig. 5.

COP framework supports the partition concept created by Kraemer's method. Fig. 5. and Fig. 4 show the number of partitions, which are called `Receiver` and `Behaviour` .

Transmitting signals interact with state machines with buffers in event queues. Each state machine consists of an initial state and a number of transitions that are actuated by the signals received. A number of actions are executed by transactions, and they are: sending signals, operation call, and timer control.

In our approach, and the purpose of this work, we use executable UML 2.0 state machines and composite structures, which are suitable inputs for our code generators.

UML state machines present various states and the transition between them. A state machine changes its state from `A` to `B` due to signal `S` . Translation moves to the other state based on the signal received which is managed by transaction tables.

There are three transition methods [13]:

1) Fire initial (Initial transition of state machine is not caused by any trigger)
2) Execute method (caused by signals which can be internal or external to the state machines)
3) Timer execution

The state machine of `IRProximity` starts from the initial state to `S1.StartTimer (x, "timer" this)` creates a timer with x milliseconds delay, and finally the data will be received from the sensors. Transition from `S1` to `S2` is shown in following algorithm:

```
public boolean fire(String signalID, Object data, Object
sender)
 if currentState==s1 then
   send.output("Send me Data");
   scheduer.startTimer(100, "c2PTimer",
   this);
 end if
 if signalID.equals("EVENTSensor") then
   currentState = s2;
   return CONSUMESIGNAL;
 end if
```

6. Discussion

Although some common useful behaviours observed in the partitions introduced in this study, some properties of the receiver are variable. For example, each device demands a different delay to send a signal to the receiver based on hardware features.

In addition, all the states of the state machine which are created in AMR are static. That means each state has just one target. Although some states depend on environment variables such as `Avoid` component, and these states need preconditions [14], the precondition of these states is still predictable. Hence, AMR state machines are under the tester's control.

Although Kraemer [8] introduced a proper method to generate code with generated state machines via a transformation algorithm, it still needs a designer to identify the optimised collaboration composition. Moreover, COP in [6] introduced a powerful structure in component composition, yet, it can not generate code details. Hence, a combination of [8] and [6] can result in a powerful hybrid framework in terms of both component composition and code generations for AMR.

The implementation of state machine in COP framework provides reusability in the generated code, even for other systems. For an instance, the behaviour of `Avoid` component might be applicable in other AMRs that can be used in military, astronomy and etc.

7. Conclusion

A method was proposed to generate COP framework state machines. This method introduced the design and implementation of the AMR behaviour as a part of the COP framework. The current COP framework strictly implements code schema based on component composition and can not implement a detailed code. Therefore, the component composition of COP was integrated with the collaboration oriented perspective of Kraemer's framework since the structure of both frameworks is similar with respect to component composition. In addition, the implementation of the state machine and the required code based on Kraemer's framework was discussed. Generation of both the activity diagram and the state machine was described and discussed based on a real world example of mobile robot software development. This paper proposed an ERT system framework which is created by integration of the Kraemer [8] and COP [6] framework.

In the future, a model validation method will be introduced for AMR besides an investigation into the implementation of different AMRs as well.

8. Acknowledgements

This work is financially supported by Universiti Teknologi Malaysia(UTM) grant under No.00H60 UTM.

References

[1] A. Charfi, C. Mraidha, S. Gérard, and P. Boulet, "Toward optimized code generation through model- based optimization," in *Design, Automation and Testing in Europe Conference and Exhibition*, 2010.

[2] E. Carneiro, P. Maciel, G. Callou, E. Tavares, and B. Nogueira, "Mapping SysML State Machine Diagram to Time Petri Net for Analysis and Verification of Embedded Real-Time Systems with Energy Constraints," *2008 International Conference on Advances in Electronics and Microelectronics*, pp. 1–6, Sep. 2008.

[3] E. Coskun and M. Grabowski, "Software complexity and its impacts in embedded intelligent real-time systems," *Journal of Systems and Software*, vol. 78, pp. 128–145, 2005.

[4] C. Angelov, X. Ke, Y. Guo, and K. Sierszecki, "Reconfigurable State Machine Components for Embedded Applications," pp. 51–58, 2008.

[5] P. R. Omaniuk, "New pattern for Implementation of Hierarchical State Machines in the c Language , Optimized for Minimal execution time on Microcontrollers mixdes 2008," *Computer*, pp. 605–609, 2008.

[6] D. Norhayati, A. Jawawi, and R. Mamat, "A Component-Oriented Programming Framework for Developing Embedded Mobile Robot Software using PECOS Model," in *The Second Malaysian Software Engineering Conference (MySEC'06)*, 2006.

[7] F. A. Kraemer, V. Sl, and P. Herrmann, "Model-Driven Construction of Embedded Applications Based on Reusable Building Blocks – An Example," pp. 1–18, 2009.

[8] F. A. Kraemer and P. Herrmann, "Electronic Communications of the EASST Sixth International Workshop on Graph Transformation and Visual Modeling Techniques Transforming Collaborative Service Specifications into Efficiently Executable State Machines into Efficiently Executable State Machi," in *Sixth InternationalWorkshop on Graph Transformation and Visual Modeling Techniques*, vol. 7, 2007.

[9] C. Szyperski and J. Bosch, "Component-oriented programming," *Object-Oriented Technology ECOOP*, 1999.

[10] J. Hagel, "Net Gain_ Expanding markets through virtual communities," *Journal of Interactive Marketing*, 1999.

[11] D. N. A. Jawawi, R. Mamat, and S. Deris, "A Component-Oriented Programming for Embedded Mobile Robot Software," *Advanced Robotic*, vol. 4, no. 3, pp. 371–380, 2007.

[12] F. Kraemer and P. Herrmann, "Service Specification by Composition of Collaborations–An Example," *2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology Workshops*, pp. 129–133, Dec. 2006.

[13] B. T. Merha, "Code Generation for Executable State Machines on Embedded Java Devices," in *Science And Technology*, 2008.

[14] Y. Yin, B. Liu, and D. Su, "Research on Formal Verification Technique for Aircraft Safety-Critical Software," *Journal of Computers*, vol. 5, no. 8, pp. 1152–1159, Aug. 2010.