UML Based Design of LEGO Robots

Lorenzo Jones, Janise Fowler, Samuel James, and Yujian Fu
Department of Computer Science, Alabama A&M University, Normal, AL, USA

Abstract—Robots are often used to increase human productivity and decrease human error. It is a key to build a reliable and correct robot that does not contain bugs and errors and demonstrate fault behaviors. In this paper we design and develop an intelligent, multifunctional robot using object oriented software development (OOSD) approach - a UML (Unified Modeling Language) based robotics architecture that uses the UML diagrams and Object Constraint Language (OCL). UML is a standard graphical-based design language that has been widely used in the software-intensive system design. However, there is not enough research work that has been done in the reliability modeling and analysis of robotics. In this paper, the robotics architecture is described by three components - the static structure (class diagram), the dynamic behavior (state machine diagrams) and the property constraints (OCL) on the static and dynamic components. To validate the approach, a robot was built from LEGO Mindstorm NXT tool kit and implemented in the Java platform. The built robot is tested against the OCL properties to validate the required properties. LEGO Mindstorm NXT tool kit is a low cost, multiple platforms, and high integrated setting that mainly used for educational and research robot. Behavior Programming is used in the Java LeJOS platform to sequence the concurrent behaviors in an interleaving way by prioritizing each behavior defined in a class. In conclusion, through our case study robot on the LEGO Mindstorm NXT tool kit, we found that the UML based robotics architecture can be used to successfully design and develop correct and reliable robotics systems.

Keywords: UML model, class diagram, state machine diagram, object oriented design, Java

1. Introduction

In the past several years the interest in robots and autonomous systems has risen. Although robots date back to as early as the 1400’s the actual term "robotics" was not used until 1940 by Isaac Asimov. While introducing this term Asimov developed the Three Robot Laws: (1) A robot may not injure a human being, or, through inaction, allow a humanity to come to harm (2) A robot may not injure humanity, or, through inaction, allow humanity to come to harm, unless this would violate a higher order law, (3) A robot must protect its own existence as long as such protection does not conflict with the First or Second Law [11].

After decades past, these Three Robot Laws are not enough to build a robotics system. More and more rules are added upon these Three Robot Laws. On the other hand, to maintain and obey these three laws, a lot of challenges of robotic development are issued, more techniques are desired to meet more requirements for a dynamic mobile robot to develop a multifunctional, robust, and reliable robot. The goal of this work is to study a new framework on the UML model for the robotics system to investigate the suitability of using the object oriented software development process (OOSD) in the robot development. Unified Modeling Language (UML) is used on the robotics system design and implementation. UML is a standard graphical based language that has been widely used in the software-intensive design and specification. In this paper, we proposed a robotics system modeling framework based on the UML diagrams by structuralizing the system into three components - static structure (represented by class diagram), dynamic behaviors (represented by state machine diagram) and property description (OCL constraints). State machine diagram is associated with the classes defined in the class diagram or the system, and represents the observable state transitions during event stimuli. OCL constraints are a set of formulae based on the set theory and first order logic and used to ensure the correctness and reliable behavior of the robot.

 Autonomous control requires the ability of flexible interaction of system elements autonomously. Therefore, it is very important to handle the dynamics and complexity due to the greater flexibility and autonomy of decision making in the robotic systems. With this study, we show that this process modeling framework for the autonomous control is suitable for the quality assurance of mobile systems.

LEGO Mindstorm NXT tool kit is used to implement the above modeling framework and validate the design requirement specified in OCL. The LEGO Mindstrom NXT robot system is codified in JAVA using Eclipse, which is plugged into the java API’s for LEGO Mindstorm NXT package.

The paper is outlined as follows. Section 2 will give a brief discussion of object oriented software development (OOSD) process, a short introduction of UML model and OCL. Section 3 will present the related works of robotics and complex embedded system design using OOSD and UML. Section 4 presents the UML model of the robot and system constraints that the robot is expected to satisfy after an introduction of primary function of the robot. Two case studies on the
LEGO Mindstorm NXT toolkits are presented in Section 5 with the illustration of UML framework application. Section 6 discusses the implementation of the above robotics system in JAVA with Behavior Programming (BP) and some of the limitations and malfunctions of the system. Section 7 concludes the work and discusses the future work.

2. Preliminaries

This section will introduce the preliminary concepts used in this project - Object Oriented Software Development (OOSD), Unified Modeling Language (UML), and Object Constraint Language (OCL).

2.1 Object Oriented Software Development (OOSD)

The object oriented software development (OOSD) process has five major steps which are requirement gathering and analysis, system design, implementation, integration, operations, and maintenance. OOSD views the system as a collection of objects resulting in a more complex system than other models. The functionalities of the system are defined by the interactions and messages between objects.

Unlike other development processes, like the waterfall and spiral processes, OOSD requires the developer to place more thought and emphasis into the analysis and design phase of the model. This project uses OOSD methodology (UML) to model the robot and realize the system in JAVA, an object oriented programming language (OOPL). Some of the key concepts of object oriented development are the usages of classes, polymorphism, and, inheritance, which are all reflected and implemented in JAVA.

During the design phase the developer is able to plan and analyze different design models and decide which models to implement to satisfy the requirements of their system. When the design models are selected the developer must take into account the complexity and functionality of the system. This process is very selective but it is imperative in order to have a correct and reliable robot. For our project we will be using UML to ensure that our robot is both correct and reliable.

2.2 Unified Modeling Language (UML)

UML is a standard graphical-based design language that has been widely used in the software-intensive design [12]. Although there is few work of modeling robotics systems in UML, many works has been done in the modeling and specification of embedded systems using UML. Several key attributes of UML are important for modeling embedded systems. Several key features are included in the UML such as profile and real time for embedded system design. Supporting for OOSD and appealing to the software community is another key feature. Besides, UML support for state-machine semantics which can be used for modeling and analysis. Finally, UML supports object-based structural decomposition and refinement.

To describe and specify the functionality of the robot and the interactions between the object, we use class diagram to model the static structure of the robot, and state machine diagram to represent the dynamic behavior of the system. Class diagrams represent relationships between classes that are represented by boxes with three sections, the top section indicates the name of the class, the middle section lists the attributes of the class, and the third sections of the diagram lists the methods [3]. State machine diagrams describe the interactions and states of different objects within and outside the system, as well as, with each other. States are graphically represented by a rounded rectangle that represents the elapsed time an object is in a state or waiting. There are three sections to the state machine diagram, from top to bottom, the sections are: name of the state, variables, and triggered operations.

2.3 Benefits of Object Oriented Development and Design

Object oriented development is vastly growing. One of the reasons of this growth are the benefits object oriented development provides. One reason why the OOSD approach is becoming popular is reusability. OOSD allows you to reuse objects and place them in other codes. With this usability it also allows the developer to add on and make changes to a particular object without making any changes to the entire system. Besides reusability OOSD has a number of other benefits, these are simplicity, flexibility, extensibility, and maintainability, and cost.

OOSD models real world objects reducing the complexity of the system but also presenting the program structure in a very simple way that is easily to understand. The real world objects are organized into classes of objects and are able to associate with other objects through behaviors. In order to describe these associations and behaviors OOSD uses class diagram which are a part of UML.

Unlike other developments, OOSD gives you a great and wide range of flexibility. By using objects one can build a new behavior from an existing object. This object can be called and/or created at any time within the system. This characteristic is important especially within our robot because we use several different behaviors in order to perform different functionalities of the robot. OOSD also provides extensibility. If requirements, customer needs, or any other issues occur were a feature must be added OOSD gives you that extensibility to easily add the new feature. During our project there was several times that we had to add new features. However, by adding this new object to satisfy the additional feature the current objects were not affected. Another major advantage of OOSD is maintainability. In OOSD maintaining an object can be done both separately and in different locations, and remotely, making maintainability a lot easier.
2.4 Object Constraint Language (OCL)

To complement the UML diagrammatic notation, the Object Constraint Language (OCL) [10], [6] can be used to express constraints and specify the effect of operations in a declarative way. In each predicate of OCL, the logical statements must be satisfied by all valid instances of the system that represented by constraints.

The OCL [10], [1] is a textual, declarative language based on first-order logic and set theory. In addition to expressing constraints on class diagrams, OCL can also be used to specify the effect of the execution of an operation, using pre and post conditions. A pre condition is an OCL statement that has to evaluate to true before the execution of an operation, while a post condition is a statement that has to evaluate to true when the operation terminates.

3. Related Work

Currently there is few work in the UML design of robotics systems. However, recently there is a trend to implement LEGO Mindstorm NXT using JAVA. JAVA allows the developer to write a wide range of applications, as well as, access to different devices like Bluetooth, cameras, mp3 players. API’s provide JAVA with its wide range of communication resulting in a simple framework. Many works have been done in the modeling of embedded systems using UML model. UML is very popular because it makes for an easy transition from design to implementation.

There are many works in the embedded system design using UML models. Authors in [8] presented an approach for modeling embedded systems using extended UML as well as generating SystemC code from UML class and object diagrams. Damasevicius and Stuikys [4] examined system level design processes that are aimed at designing a hardware system by integrating soft IPs at a high level of abstraction. They combine this concept with object-oriented hardware design using UML and meta-programming paradigm for describing generation of domain code.

However, none of the above approaches use OCL with UML diagram to describe the system constraints for the purpose of verifying the functional correctness of the synthesized system. Our work aims at modeling and validating the system properties to ensure the quality of robotics systems. OCL is used in the framework to specify the constraints of the static structure and dynamic behaviors of robotics UML model.

4. UML Based Object Oriented Robotics Framework

UML consists of a combination of data modeling, business modeling, object modeling, and component modeling [7], [5], [2]. UML diagrams can be categorized as three groups - structure diagrams, behavior diagrams, management package. Structure diagrams represent the organization of the classes in the system in a static view and emphasize relations among classes. Behavior diagrams describe the state changes, communications among objects and instances, which capture the varieties of interaction and instantaneous states in the systems. Management Diagrams include Packages, Components, and Subsystems, focuses on the packaging and setting of diagrams. Behavior diagrams, focuses on the functionality of the system. They describe how the system is going to operate. Structural diagrams are widely used to describe the architecture and organization of a software system. While behavior diagrams are used for the dynamic operations during runtime.

In this paper, we presented a UML based architecture for the robotics system design to ensure the quality of the robot. The architecture is composed of three components (Fig. 1) - static structure (represented by class diagram), dynamic structure (represented by communication diagram or state machine diagram) and system properties (specified by OCL).

The OCL specification can describe all desired constraints on the static structure and dynamic structure. From Fig. 1, we can see that is the OCL properties are not satisfied, we can go back to the model architecture and find out what is the problem. After the properties and constraints are ensured, then the system can be implemented based on the model. The verification of OCL properties can be done by software testing tools or model checking techniques. In the following, we present two case studies using the LEGO NXT toolkit to realize the framework – object detection robot and claw strike robot.

5. Case Studies – LEGO Robots

There are two case studies shown in this paper using the UML Robotic Framework (1). First, we introduce a on-path object detection robot. The sequence of functions of this robot are:

1) Identify the start point and ending point by color.
2) Following path by defined color.
3) Two ways of starting are: started by the button (in NXT brick), and start by clap (voice driven). The default is clap starting.

4) Picking up object on the path.

5) Dropping the object in the destination (after the ending point).

The second robot is a claw strike robot. The robot is able to realize following functions:

1) Identify the object during moving.
2) Stop and back for a short distance if object is found.
3) Rotate the motor to throw the strike and hit the object.

In the following, we discuss the application of UML framework in this two LEGO robot design. Three components are presented in the following – class diagram, state machine diagram and OCL constraints.

5.1 Class Diagram

We first present the class diagram of the path following object detection robot, after that, we will show the class diagram of claw strike robot.

**On-path Object Detection Robot** The class diagram (Fig. 2) for the object detection robot is a graphical description of class relation. It also describes how the light, sound, and ultrasonic sensor will react and communicate with each other and the NXT brick.

![Fig. 2: The class diagram of on-path object detection robot](image)

In the development of the software for the object Detection Robot a total of six classes were used. (Figure 2) Robot is the main class consisting of various objects of the systems, which are objects of classes Stop, Grab, CalibrateColor, FindLine and OnLine. A crucial behavior array was used to represent a series of actual behaviors of the robot. Four behaviors - findLine, online, Grab, and stop - are defined in a corresponding class. The order that the behaviors appear in the array are decided by the priority which can be assigned as an integer number. This priority is used to control the instance of behaviors to be activated at the runtime. Therefore, each time there is one behavior is taken and executed.

**Claw Strike Robot** The claw striker is a robot that detects an object within its perimeters then strikes the object once
detected. The robot not only need to know when the target is in range. Also, it should be able to detect when it is either too close or too far from the target itself. UML the class diagram (3) were initialized to show a visual display of the design for programming the robot. In Fig. 3, a partial design of class diagram is shown to ensure the above behavior. Four classes were implemented the Ultrasonic class, Motor A, Motor B, and Motor C classes respectively. The Ultrasonic class was a public class used to code the Ultrasonic sensor. In this part of the code, the range was an integer data type, two operations detect(), and objectfound() were implemented in this class. Once the object is found, the object needs to be in range; otherwise, the robot will move forward, backward and/or turn around to rotate the motors and move the robot to be in range.

Fig. 4: The state diagram of the on-path object detection

5.2 State Machine Diagram

On-path Object Detection Robot In the on-path object detection robot, three typical states are identified - idle, active, and end states. The robot will be running on the line before reach the object. This period of time, the robot is in the idle state. After detecting the object, the robot will pick up the object and keep following the line before stops. The robot is in the end state when it stops in the specified position. Any actions taken during active state will be represented as the internal states (Figure 4.).

Six states are considered in the active state - GetLight-Values, DetectSounds, FollowPath, GrabObjects, ReleaseObject, and NXTBrickStarts - to illustrate the complexity of the Object Detection Robot. The robot is designed to have two starting ways - sound driven and NXT brick starts. Sound driven is the default one. Before starts, the robot will read the light value of the background (floor), the path value and the end point value into the variables. After reads into those values, the robot can start to follow the path line based on the values.

The state FollowPath is defined as super state with three concurrent states - detectObject, detectPath, follow, which indicates that this super state needs to take care these three states simultaneously. The robot will enter into GrabObject state if the object with the defined color is identified by ultrasonic sensor. The key is when the robot entering into GrabObject state, the robot needs to return back to the FollowPath state.

Claw Strike Robot The state diagram of claw strike is shown in 5. There are four active states are identified except for the initializing state – moving, detecting, adjusting, and throwing. A synchronization bar is used to indicate that the robot will detect during moving status. The condition of object found must be validated before the robot throw the strike to assure that the ultrasonic sensor has the data return positively. Otherwise, the robot will keep moving and detecting.

Fig. 5: The state diagram of the claw strike robot

5.3 System Constraints

To ensure the quality of the object detection LEGO robot, we use OCL to define system constraints. The system is expected to hold all the constraints as well as maintain the functionalities.

On-path Object Detection Robot A system invariant can be expressed as when an object is detected on the path during robot is moving, the robot must pick up the object (Formula i)). Another invariant is once an object is picked up, the robot must take the object to the destination, which is indicated by the end line of yellow color (Formula ii)).

Context: Grab
Inv: If Grab.Grab() then setHasObject() hasObject

Context: Grab
Inv: If Grab.hasObject then getCalibrateLine()=yellow takeControl()=clawopen

In addition, each class has a set of associated OCL constraints that specifies the requirements of the class. the requirements are specified by pre-condition, post-condition and invariant of the class. Each state machine diagram has a
set of associated OCL constraints with pre-condition, post-condition and invariant specified. Due to the space limitation, we only show some of the OCL constraints in the Formulae iii) to Formula viii).

Context:
FindLine
Pre: if FindLine.findLine=true then takeControl()  iii) 
Post: suppressed=true  action() = following line  iv)

Context:
Stop
Pre: CalibrateColor.stopLightValue = yellow  v)
Post: Stop.takeControl() Stop.action() vi)

Context:
Online
Pre: on = online  stopLightValue = 0  vii)
Post: on = online  OnLine.action() viii)

Claw Strike Robot The key class in the claw strike robot is Ultrasonic class. There are two conditions need to be met to ensure the correct behavior: a) the object is found, and b) the object is within the range. This can be specified in the following

Context:
Ultrasonic
Pre: if objectfound=true then checkrange  ix)
  if objectfound=true and range< then throwStrike  x)
Post: motorc.rotate xi)

6. Java Implementation of UML Model in LEGO Toolkit

In this section, we present how to implement LEGO robots based on the UML model shown in above and discuss the issues that were found during experimental.

The on-path object detection robot is a multifunctional and reliable object detection robot. LEGO NXT Mindstorm Tool Kit sensors are used to implement the robots functionality and make the robot more complex and intelligent. LEGO Mindstorm NXT tool kit has been widely used in the classroom for educational and research purposes. The robot will have several main functions: staying on the predefined path with the light sensor, being able to detect objects with the ultra sonic sensor, determine the color of the object with the light sensor, and collect input from user with the sound sensor. The final build of the robot is shown in Fig. 6.

In the claw strike robot, the robot not only need to know when the target is in range. Also, it should be able to detect when it is either too close or too far from the target itself. As shown in the class diagram (Fig. 3), The Ultrasonic class was is associated with three motor classes. In this part of the code, the range was an integer data type, two functions detect(), and objectfound() were initialized in this class. Once the button was pushed on the brick the ultrasonic sensor will use the detect() to start looking for any particular object within a range of 50cm. If the range was below 50 the classes of Motor A,B ,and C will be initialized. Motor A controls the striker which will only strike if the target is at

An implementation feature of this LEGO robot is using Behavior Programming (BP) [9]. Behavior programming (BP) is imperative in the use of LEGO NXT API’s and introduces the theory of priority. The key concept of BP is serialization all behavior by assigned priority: only one behavior can be active and in control of the robot at any time, each behavior has a given value named "priority", the controller can determine if a behavior should take control based on the behavior queue by priority, the active behavior has higher priority than any other behavior that should take control [9]. Three methods – takeControl(), void action(), and void suppress() – need to be overridden in the BP. The method takeControl() is used to indicate if a particular behavior should be active. For our project it would indicate that the robot will perform an action when an object is detected. The method action() performs the actions when takeControl() is true. This action might be to move the motor forward or backwards. The method suppress() is used to terminate the implementation of the action code. Finally, An Arbitrator is an array that regulates when each behavior should become active. The arbitrator regulates the behaviors by using priority with the zero element of the array representing the lowest priority. The Robot class also has the test for the sound sensor by using if statements. If the sound sensor receives a frequency reading frequency>80 then the robot is able to proceed to findLine. If the robot receives a frequency ≤ 80 the sound must loop through again until sound frequency is greater in 80. If this statement is true, the sound sensor is down, user must press orange button on NXT Brick.
a range of equal to 40 cm. If the target is less than or equal to 39 cm, motors B, and C move backward, if the range is between 49 to 41 cm the Motors B and C will move forward. If the ultrasonic sensor detects an object close to it but, not in front of it, motor B, and C will rotate. The problem that occurred was Motor A went forward but, did not proceed backwards once the target was struck by the claw. Also, the claw did not react quicker in striking the target as expected. The solution to these problems was adding a time delay. The time delay was set at 50ms meaning once the robot detects the robot and strikes the claw will autonomously return to its previous position. The final build of the claw strike robot is shown in Fig. 7.

![Claw Strike Robot](image)

**Fig. 7: The claw strike robot**

### 6.1 System Malfunction and Limitations

While we admit that LEGO NXT Mindstorm tool kit is a highly integrated lost cost robot set, some sensors do not have enough precision to maintain some required performance. This caused a lot of time of code revision and validation. If the robot is unable to receive a frequency input from the sound sensor the on-path object detection robot will still be able to operate. Instead the robot will be able to communicate with the user through the LCD screen on the NXT Brick. Although the on-path object detection robot consists of many functions it does have some system limitations. The robot cannot distinguish between different objects it can only determine the color of the objects. Because we are using a light sensor and not a color sensor, colors that have close together in the spectrum light white and yellow are hard for the light the sensor to distinguish between. The sound sensor can only recognize frequencies greater than 80. The robot cannot detect objects less than 3 cm away. This can be improved in the version 2 of LEGO NXT Mindstorm tool kit.

### 7. Conclusions and Future Works

This paper presented a UML based robotics system modeling framework that is composed of three components - static structure, static structure (represented by class diagram), dynamic behaviors (represented by state machine diagram) and property description (OCL constraints). State machine diagram is associated with the classes defined in the class diagram or the system, and represents the observable state transitions during event stimuli. OCL constraints are a set of formulae based on the set theory and first order logic and used to ensure the correctness and reliable behavior of the robot. The purpose of this work is to develop a reliable and correct, intelligent, multifunctional robot using OOSD approach. After the testing and validation of the robot, we can confidently ensure that the robot maintains the required function with the correct behaviors.

Future work for this project falls in two aspects. First, the current framework will be extended with verification component that include model checking technique. In that case, the static structure (class diagram) and dynamic behavior (state machine diagram) will be converted to the programming language that can be read by model checker. We will run this model against the OCL constraints to check if the model meets the constraints specified in OCL. The model checking technique is able to detect the problems in the model with meaningful feedback. Secondly, as a typical real time embedded system, timing concern is a key for the robotics systems. We will extend the current model to describe the timing issue of the robot.

### Acknowledgment

The authors would like to thank all reviewers for the kindly comments and suggestions on this work.

### References


