UML-Based Service Robot Software Development: A Case Study

Minseong Kim, Suntae Kim, Sooyong Park
Department of Computer Science
Sogang University
Seoul, REP. of KOREA
{minskim,jipsin08,sypark}
@sogang.ac.kr

Mun-Taek Choi, Munsang Kim
Center for Intelligent Robotics
Frontier 21 Program at Korea Institute of Science and Technology
Seoul, REP. of KOREA
{mtchoi,munsang}
@kist.re.kr

Hassan Gomaa
Dept. of Information and Software Engineering
George Mason University
Fairfax, VA 22030, USA
hgomaa@gmu.edu

ABSTRACT
The research field of Intelligent Service Robots, which has become more and more popular over the last years, covers a wide range of applications from climbing machines for cleaning large storefronts to robotic assistance for disabled or elderly people. When developing service robot software, it is a challenging problem to design the robot architecture by carefully considering user needs and requirements, implement robot application components based on the architecture, and integrate these components in a systematic and comprehensive way for maintainability and reusability. Furthermore, it becomes more difficult to communicate among development teams and with others when many engineers from different teams participate in developing the service robot. To solve these problems, we applied the COMET design method, which uses the industry-standard UML notation, to developing the software of an intelligent service robot for the elderly, called T-Rot, under development at Center for Intelligent Robotics (CIR). In this paper, we discuss our experiences with the project in which we successfully addressed these problems and developed the autonomous navigation system of the robot with the COMET/UML method.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *object-oriented design methods*

General Terms
Design

Keywords
Software engineering, object-oriented analysis and design methods, service robot development, UML

1. INTRODUCTION
Robots have been used in several new applications. In recent years, both academic and commercial research has been focusing on the development of a new generation of robots in the emerging field of service robots. Service robots are individually designed to perform tasks in a specific environment for working with or assisting humans and must be able to perform services semi- or fully automatically [1]. Examples of service robots are those used for inspection, maintenance, housekeeping, office automation and aiding senior citizens or physically challenged individuals [2]. A number of commercialized service robots have recently been introduced such as vacuum cleaning robots, home security robots, robots for lawn mowing, entertainment robots, and guide robots [3, 4].

In this context, Public Service Robot (PSR) systems have been developed for indoor service tasks at Korea Institute of Science and Technology (KIST) [5, 6]. The PSR is an intelligent service robot, which has various capabilities such as navigation, manipulation, etc. Up to now, three versions of the PSR systems, that is, *PSR-1*, *PSR-2*, and a guide robot *Jinny* have been built.

The worldwide aging population and health care costs of aged people are rapidly growing and are set to become a major problem in the coming decades. This phenomenon could lead to a huge market for service robots assisting with the care and support of the disabled and elderly in the future [8]. As a result, a new project is under development at Center for Intelligent Robotics (CIR) at KIST, i.e. the intelligent service robot for the elderly, called T-Rot.

In our service robot applications, it is essential to not only consider and develop a well-defined robot software architecture, but also to develop and integrate robot application components in a systematic and comprehensive manner. There are several reasons for this:

- First, service robots interact closely with humans in a wide range of situations for providing services through robot application components such as vision recognition, speech recognition, navigation, etc. Thus, a well-defined robot control architecture is required for coherently and systematically combining these services into an integrated system.
- Second, in robot systems, there are many-to-many relations among software components as well as hardware components. For instance, a local map module requires range data from a laser scanner, ultrasonic sensors, and infrared sensors, as well as prior geometrical descriptions of the environment. On the other hand, the laser scanner should provide its data to a path planner, localizer, and a local map

ICSE'06, May 20-28, 2006, Shanghai, China.

building module. These relationships, as well as interactions among software or hardware modules, must be carefully analyzed and systematically managed from an early stage of development in order to understand the big picture.

- Third, the functional performance of each software and hardware module becomes highly dependent on the architecture, as the number of robot platforms increases [6], and new services are added, or existing services are removed or updated to address changes in user needs.
- Fourth, previously developed software modules like maps, localization, and path planners can be directly reused for new tasks or services by service robot developers. Thus, a robot architecture, as well as systematic processes or methods, are required to support the implementation of the system, to ensure modularity and reusability.

As a consequence, in the previous work [5,6], the Tripodal schematic control architecture was proposed to tackle the problems. Many related research activities have been done. However, it is still a challenging problem to develop the robot architecture by carefully taking into account user needs and requirements, implement robot application components based on the architecture, and integrate these components in a systematic and comprehensive way. The reason is that the developers of service robots generally tend to be immersed in technology specific components, e.g. vision recognizer, localizer and path planner, at an early stage of product development without carefully considering architecture to integrate those components for various services [9]. Moreover, engineers and developers are often grouped into separate teams in accordance with the specific technologies (e.g., speech processing, vision processing), which makes integration of these components more difficult [7, 9]. In such a project like T-Rot, particularly, several engineers and developers (i.e., approximately, more than 150 engineers) from different organizations and teams participate in the implementation of the service robot. Each separate team tends to address the specific technologies such as object recognition, manipulation, and navigation and so on. Engineers who come from different teams are concerned with different characteristics of the system. Thus, a common medium is required to create mutual understanding, form consensus, and communicate with each other for successfully constructing the service robot. Without such a medium or language, it is difficult to sufficiently understand the service robot system and interact between teams to integrate components for services.

Within the domain of software engineering, many approaches have been suggested for a systematic and complete system analysis and design, and for the capture of specifications. The object-oriented paradigm [10,11] is a widely-accepted approach to not only cover the external and declarative view of a system, but also at the same time bridge seamlessly with the internal implementation view of a system [13]. Object-oriented concepts are crucial in software analysis and design because they focus on fundamental issues of adaptation and evolution [14]. Therefore, compared with the traditional structured software development methods, object-oriented methods are a more modular approach for analysis, design, and implementation of complex software systems, which leads to more self-contained and hence modifiable and maintainable systems. More recently, the Unified Modeling Language (UML) [15,16] has captured industry-wide attention for its role as a general-purpose language for modeling software systems, especially for describing object-oriented models. The UML notation is useful to specify the requirements, document the structure, decompose into objects, and define relationships between objects in a software system. Certain notations in the UML have particular importance for modeling embedded systems [17,18], like robot systems. By adopting the UML notation, development teams thus can communicate among themselves and with others using a defined standard [14,17,18]. More importantly, it is essential for the UML notation to be used with a systematic object-oriented analysis and design method in order to be effectively applied [14].

As a result, our aim is to develop the intelligent service robot based on the systematic software engineering method, especially for real-time, embedded and distributed systems with UML. To do so, we applied the COMET method, which is a UML based method for the development of concurrent applications, specifically distributed and real-time applications [14]. By using the COMET method, it is possible to reconcile specific engineering techniques with the industry-standard UML and furthermore to fit such techniques into a fully defined development process towards developing the service robot systems.

In this paper, we describe our experience of applying the COMET /UML method into developing the intelligent service robot for the elderly, called T-Rot, developed at CIR. In particular, we focused on designing an autonomous navigation system for the service robot, which is one of the most challenging issues for the development of service robots.

Section 2 describes the hardware configuration and services of the T-Rot, and discusses the related work. Section 3 illustrates how to apply the COMET method into designing and developing the autonomous navigation system for the service robot, and discusses the results of experiments. The lessons learned from the project are summarized in section 4, and section 5 concludes the paper with some words on further work.

2. BACKGROUD ON T-Rot

2.1 PSR and T-Rot

Fig. 1. KIST service robots

At KIST, intelligent service robots have been developed in large-scale indoor environments since 1998. So far, *PSR-1* and *PSR-2*, which performs delivery, patrol, and floor cleaning jobs, and a guide robot *Jinny*, which provides services like exhibition guide and guidance of the road at a museum, have been built [5,6] (see Fig. 1). The service robot T-Rot is the next model of the PSR system under development for assisting aged persons. Development of T-Rot, in which our role is developing and integrating robot software, started in 2003 by mainly CIR with

more than 10 groups consisting of more than 150 researchers and engineers from academia and industry. This project is based on the needs and requirements of elderly people through the studies and analysis of the commercial health-care market for providing useful services to them. Thus, the aim of this project is to develop the intelligent service robot for the elderly by cooperating and integrating the results of different research groups. This project that is divided into three stages will continue until 2013 and we are now in the first stage for developing the service robot incrementally to provide various services.

2.2 Hardware of T-Rot

The initial version of T-Rot, as shown in Fig. 2, has three single board computer (SBC), that is, mobile Pentium 4 (2.2GHz) and 1GB SDRAM on each SBC. In terms of software environment, Linux Red hat 9.0 and RTAI (Real-Time Application Interface) [12] are used as operating system. Fig. 3 shows hardware configuration as a whole. As mentioned earlier, development of T-Rot is conducted incrementally for various services and thus the platform will be extended with manipulators and robot hands later. In our project, we developed the robot software based on the initial version of the platform. The details of the hardware platform are described in Table 1.

Fig. 2. T-Rot robot hardware platform

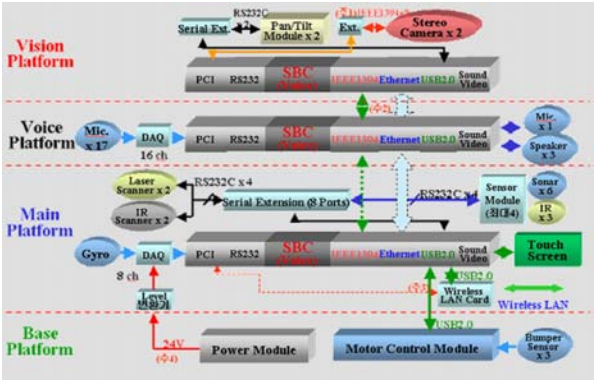

Fig. 3. T-Rot robot hardware platform configuration

Table 1. T-Rot hardware platform devices

SBC	Intel Mobile Pentium 4 (2.2 GHz)
	1GB SDRAM
	30GB Hard Disk
Voice	16 microphones for speaker localization
	1 microphone for speech recognition
	1 speaker for speech generation
Vision	2 stereo vision cameras for recognizing users and objects (1288 H x 1032 V maximum resolution and 7Hz frame rates)
	Pan/Tilt for controlling the vision part
Sensor	2 laser scanners (front and back)
	2 IR scanners (front and back)
	12 Ultrasonic sensors
	1 Gyroscope sensor for measuring balance
Actuator	2 actuators for two drive wheels (right and left)
	2 free wheels (the support wheels)
	2 Servo Motors (100 [w])
	2 encoders (2048 ppr)
	2 bumpers
Interface	1 TFT LCD & Touch (10.4" 1024x768, 26000 colors)
	KVM (Keyboard/Mouse)
	Wireless LAN for communications

2.3 Robot Services

Some of the primary services under-developed that the initial version for T-Rot provides for the elderly are described as below.

- Voice-based Information Services: The robot T-Rot can recognize voice commands from a user (i.e., an aged person) via microphones equipped with the robot and can synthesize voices for services. While a user is watching TV, the user can ask some questions about the specific TV program or request a task to open an Internet homepage by speaking the TV program name.
- Sound Localization and Voice Recognition: A user can call a robot's predefined name, to let the robot recognize the call while the robot knows the direction to move to the user. This service analyzes audio data from 3 microphones on the shoulder for sound localization and 16 mic array on the head for speech recognition to recognize the command from the user.
- Autonomous navigation: A user can command the robot to move to a specific position in the map to perform some task. For instance, the robot can navigate to its destination in the home environment via its sensors, which include laser scanners and ultrasonic sensors. The robot plans a path to the specified position, executes this plan, and modifies it as necessary for avoiding unexpected obstacles. While the robot is moving, it constantly checks sensor data from its sensors every 200 ms.
- An errand service: The robot can carry objects that a user (i.e., an aged person) usually uses, like a plate, books, a cane a cup of tea, beverages, etc according to the user's instructions. For instance, the user can order the robot to bring a cup of tea or beverage by speaking the name of the drink.

Of these T-Rot services, our emphasis was on the autonomous navigation service, which is one of the most challenging issues and is essential in developing service robots, particularly mobile service robots to assist elderly people. It includes hardware integration for various sensors and actuators, and the development of crucial navigation algorithms like maps, path planners, and

localizers as well as software integration of software modules like a path planner, a localizer, and a map building module.

2.4 Control Architecture of PSR

Up to now, there have been many related research activities to develop efficient and well-defined control architectures and system integration strategies for constructing service robots. A recent trend is that many control architectures are converging to a similar structure based on a hybrid approach that integrates reactive control and deliberation [6]. At KIST, for developing service robots, that is *PSR-1*, *PSR-2*, and *Jinny* in the previous work [5,6], the Tripodal schematic control architecture was proposed as the solution to the problem.

One important point of Tripodal schematic design is to integrate robot systems by using a layered functionality diagram. The layered functionality diagram is a conceptual diagram of three layers for arrangement of various hardware and software modules and functions. It also shows the connectivity and the information flow between components. Those layers are composed of deliberate, sequencing, and reactive layers based on the hybrid approach. The purposes of the deliberate layer are to interface with a user and to execute a planning process. The sequencing layer is classified into two groups, that is, the controlling part that executes the process by managing the components in the reactive layer and the information part that extracts highly advanced information from sensor data. The reactive layer controls the real-time command and hardware-related modules for sensors and actuators. The detailed description of whole control architecture of the PSR is introduced in [5].

However, as described earlier, in order to effectively apply this approach and the UML notation to developing service robots, it is essential to use a systematic software engineering process or methods like object-oriented analysis and design methods, especially for real-time and embedded systems. We believe that only a systematic and comprehensive software development process and method will be able to resolve the issues discussed before and will be vital for success in developing service robots.

2.5 The COMET method

COMET [14] is a method for designing real-time and distributed applications, which integrates object-oriented and concurrent processing concepts and uses the UML notation [15,16]. The COMET object- oriented software life cycle model is a highly iterative software development process based around the use case concept. Therefore, in this project, the COMET method with UML was used to develop a system for autonomous navigation by the intelligent service robot, T-Rot. The method separates requirements activities, analysis activities and design activities, and these activities are briefly described as below. The details are described in section 3 with the case study.

- Requirements modeling - A use case model is developed in which the functional requirements of the system are defined in terms of actors and use cases.
- Analysis modeling - Static and dynamic models of the system are developed. The static model defines the structural relationships among problem domain classes. A dynamic model is then developed in which the use cases from the requirements model are refined to show the objects that participate in each use case and how they interact with each other.
- Design modeling – The software architecture of the system is designed, in which the analysis model is mapped to an operational environment. For distributed applications, a component based development approach is taken, in which each subsystem is designed as a distributed self-contained component.

3. APPLYING THE COMET/UML METHOD TO T-ROT

In this section, we explain how to develop robot software for the autonomous navigation system with the COMET/UML method. In our project, the UML notation conforms to UML 1.3 and the Rational Rose tool is used.

3.1 Requirements Modeling

Capturing the functional requirements of the system is the first phase in software development, which defines what the system should do or provide for the user. In our approach, developers can catch the functional requirements or services by using the use case model in terms of use cases and actors (see Fig. 4). To identify and define the requirements of the system more clearly, the system has to be considered like a black box. In the service robot, the actor can be usually a human user as well as external I/O devices and external timer.

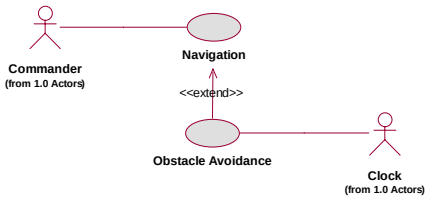

Fig. 4. Use case diagram for Navigation

Table 2 shows a specification for *Navigation* use case. In our navigation system, we identified a *Commander* and a *Clock* as an actor. While the robot is moving, if the robot recognizes obstacles, it should avoid them for continuing to go to the destination. Even when humans or objects suddenly appear, the robot must be able to stop to avoid crashing into those. However, in order to do this, the robot has to check that there are obstacles by using sensor data more often (e.g., every 50 ms) than the normal navigation system does (e.g., every 200 ms). As a result, the *Obstacle Avoidance* use case is extended from the *Navigation* use case. During the *Navigation* use case is executing, if the obstacles are recognized, then the *Obstacle Avoidance* use case is triggered to perform the emergency stop of the robot. If the obstacles disappear, the robot moves again to the destination.

Table 2. Navigation use case

Summary	The Commander enters a destination and the robot system moves to the destination.
Actor	Commander
Precondition	The robot system has the grid map and the current position is known
Description	1. The use case begins when the commander enters a destination. 2. The system calculates an optimal path to the destination. 3. The system commands the wheel actuator to start

	moving to the destination. 4. The wheel actuator notifies the system that it has started moving 5. The system periodically reads sensor data and calculates the current position. 6. The system determines that it arrives at the destination and commands the wheel actuator to stop. 7. The wheel actuator notifies the system that it has stopped moving and the use case is finished.
Alternative	6.1. If the system doesn't arrive at the destination, it keeps moving.
Postcondition	The robot system is at the destination and waiting for the next destination.

3.2 Analysis Modeling

3.2.1 Static Modeling

The objective of static modeling is to understand the interface between the system and the external environment and to describe the static structure of the system under development by developing a system context class diagram. It is specifically important for real-time and embedded systems like robot systems [14]. The system context class diagram can be determined by static modeling of the external classes that connect to the system.

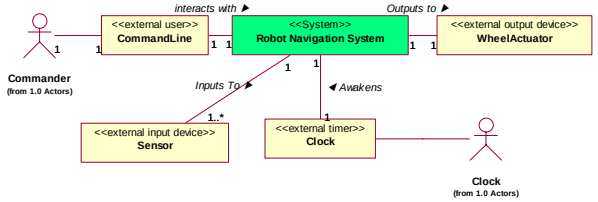

Fig. 5. Robot Navigation System context class diagram

The system context class diagram of the *Robot Navigation System* is shown in Fig. 5, which illustrates the external classes to which the system has to interface. In our navigation system, a commander enters a destination via a command line, to which the robot should move. The system uses sensor data via various sensors such as laser scanners, IR scanners, ultrasonic sensors, etc and it controls the wheels of the robot via the wheel actuator. Therefore, the external classes correspond to the users (i.e., a *Commander* who interacts with the system via a *Command Line*), and I/O devices (i.e., a *Sensor* and *Wheel Actuator*). A *Clock* actor needs an external timer class called *Clock* to provide timer events to the system. This external timer class is needed to periodically check sensor data via those sensors for avoiding obstacles (i.e., doing the emergency stop) while the robot is moving.

Next, to structure the *Robot Navigation System* into objects, object structuring needs to be considered in preparation for dynamic modeling. The objective of the object structuring is to decompose the problem into objects within the system. We identified the internal objects according to the object structuring criteria in COMET (see Fig. 6). In our system, interface objects, i.e. a *Command Line Interface*, *Sensor Interface* and *Wheel Actuator Interface* are identified by identifying the external classes that interface to the system, i.e. the *Command Line*, *Sensor*, and *Wheel Actuator*, respectively. There are four entity objects identified, that is, a *Current Position*, *Destination*, *Navigation Path* and *Navigation Map*, which are usually long-living object that stores information. In addition to those objects, there is a need for control objects, which provide the overall coordination for objects in a use case and may be coordinator, state-dependent control, or timer objects. The *Navigation System* has a state-dependent control object called *Navigation Control* that controls the wheel actuator and sensors. The states of the *Navigation Control* object are shown on a *Navigation Control* statechart (this will be discussed in the dynamic modeling). There are two timer objects, i.e. a *Navigation Timer* and an *Obstacle Avoidance Timer*. The *Obstacle Avoidance Timer* is activated by a timer event from an external timer to periodically check that there is any obstacle around the robot. On the other hand, the *Navigation Timer* is started by the *Navigation Control* object and generates a timer event for navigation. Also, a *Localizer* algorithm object and *Path Planner* algorithm object are identified, which encapsulate an algorithm used in the problem domain, namely the autonomous navigation.

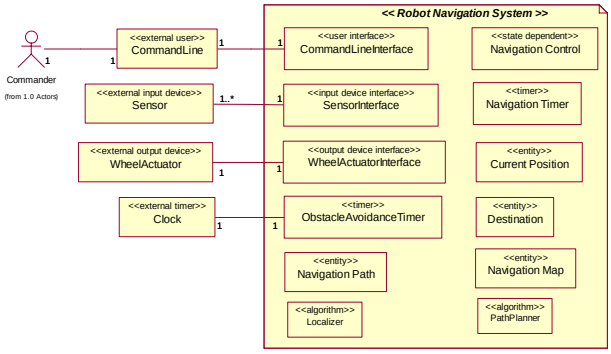

Fig. 6. Object structuring class diagram for Navigation System

3.2.2 Dynamic Modeling

Dynamic modeling emphasizes the dynamic behavior of the system and plays an important role for distributed, concurrent and real-time system analysis. The dynamic model defines the object interactions that correspond to each use case and thus is based on the use cases and the objects identified during object structuring. In our case, collaboration diagrams are developed to show the sequence of object interactions for each use case. Additionally, if the collaboration involves the state-dependent object, which executes a statechart, the event sequence is shown on a statechart.

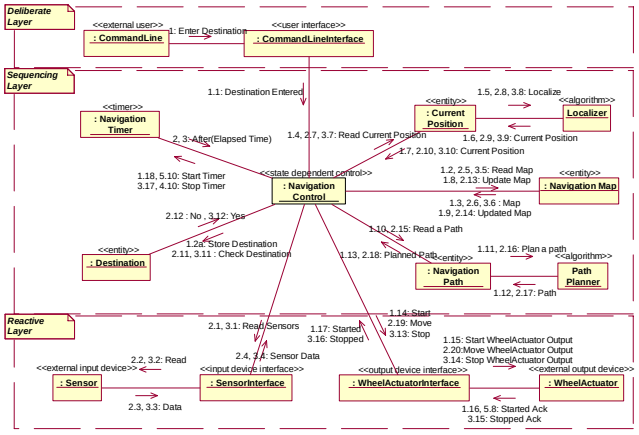

Fig. 7. Collaboration diagram for Navigation use case

In the navigation system, the *Localizer* has the algorithm which can calculate the current position based on sensor data via sensors. So, the role of the *Localizer* is to update the current position of the service robot. In the *Path Planner* object, there is a method for calculating a path to arrive at the destination based on both sensor information and the current position that is calculated at the *Localizer*. The *Navigation Timer* is an internal timer that is controlled by the *Navigation Control*. After the destination is entered from the external user, the *Navigation Control* starts the *Navigation Timer*, then the timer generates a timer event periodically (i.e., every 200ms) until the *Navigation Control* stops the timer.

The *Navigation* use case starts with the commander entering the destination into the navigation system. The message sequence number starts at 1, which is the first external event initiated by the actor. Subsequence numbering in sequence is from 1.1 to 1.18 as shown in Fig. 7. The next message sequence activated by the *Navigation Timer* is numbered 2, followed by the events 2.1, 2.2, and so forth. The following message sequences are illustrated in the collaboration diagram (see Fig. 7).

The collaboration diagram for the *Obstacle Avoidance* use case is shown in Fig. 8. When activated by the *Obstacle Avoidance Timer* every 50 ms, the *Sensor Interface* object reads sensor data via various sensors (Events 4.1, 5.1, 6.1). If an obstacle is recognized, the *Obstacle Avoidance Timer* sends the emergency stop message to the *Wheel Actuator Interface* (Event 4.5). Afterwards, the timer also sends a suspend event to the *Navigation Control*. If the obstacle disappears, the timer sends a restart event to the *Navigation Control* for the robot to move again.

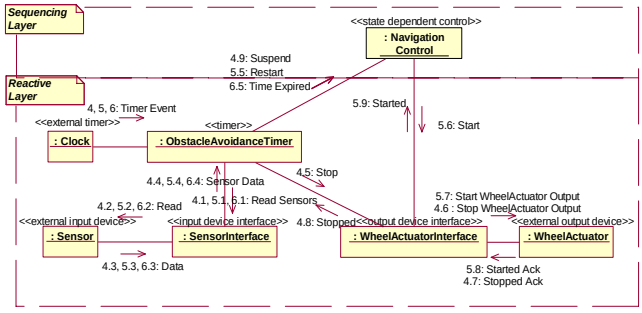

Fig. 8. Collaboration diagram for Obstacle Avoidance use case

With COMET, the software architecture can be based on a software architectural style (pattern) such as client/server or layers of abstraction. In our project, the layered strategy of the Tripodal schematic design described in section 2 is applied for design and modeling of the robot system, which provides a conceptual diagram of three layers (i.e., deliberate, sequencing, and reactive layers) for arrangement of various hardware and software modules and functions. Therefore, in the collaboration diagrams (see Fig. 7 and 8), the *Command Line Interface* is located in the deliberate layer and the *Sensor Interface*, *Wheel Actuator Interface*, and *Obstacle Avoidance Timer* are in the reactive layer. The others are positioned in the sequencing layer.

In our navigation system, after drawing the collaboration diagrams for the *Navigation* and *Obstacle Avoidance* use cases which include the *Navigation Control* state-dependent object, we develop a *Navigation Control* statechart, which is executed by the *Navigation Control* object. The statechart needs to be considered in connection with the collaboration diagram. Specifically, it is required to take into account the messages that are received and sent by the control object, which executes the statechart [14]. An input event (e.g., 1.1: *destination entered*) into the *Navigation Control* object on the collaboration diagram should be consistent with the same event shown on the statechart. The output event, which causes an action, enable or disable activity, like 1.2: *Read Map* (which cases an action) on the statechart must be consistent with the output event depicted on the collaboration diagram.

Because the statechart modeling involves two state-dependent use cases in the navigation system, it is also required to consolidate the two partial statecharts to create a complete statechart. The complete statechart for both the *Navigation* and *Obstacle Avoidance* use cases is shown in Fig. 9.

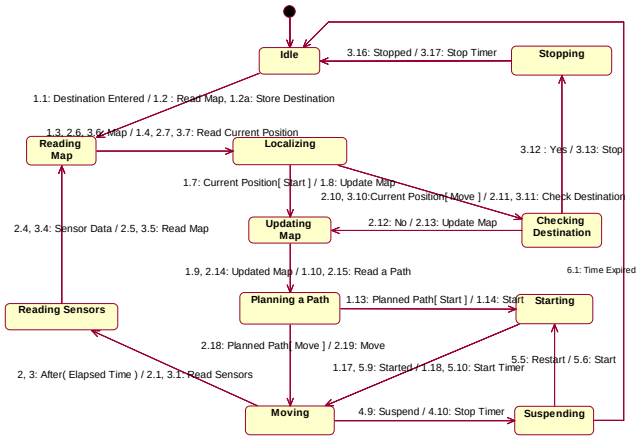

Fig. 9. Statechart for Navigation Control

3.3 Design Modeling

3.3.1 Software Architecture

In this phase, all collaboration diagrams developed for use cases in the analysis model are merged into the consolidated collaboration diagram. The consolidated collaboration diagram is thus intended to be a complete description of all objects and their interactions.

The consolidation of the two collaboration diagrams respectively supporting the two use cases is shown in Fig. 10. Some objects and message interactions appear on more than one collaboration diagram. For instance, the *Navigation Control*, *Navigation Timer*, *Sensor Interface* and *Wheel Actuator Interface* objects participate in both the *Navigation* and *Obstacle Avoidance* use cases. For those objects, their message interactions are only shown once in the consolidated collaboration diagram.

3.3.2 Architectural Design of Distributed Real-time Systems

The robot system is a distributed embedded system and executes on distributed nodes by the communication methods like TCP/IP, CAN (Controller Area Network), and Wire/Wireless LAN. With COMET, a distributed real-time system is structured into distributed subsystems. Tasks in different subsystems may communicate with each other via several types of message communication, such as asynchronous, synchronous with reply, synchronous without reply, and client/server communications, etc.

Hence, we should define distributed nodes and their messages to each node.

The overall distributed software architecture for the robot navigation system is depicted in Fig. 11. In the robot system, objects that are part of the navigation are located in the robot navigation system. The robot navigation system communicates with the external I/O devices via synchronous message without reply communication and with the external timer via asynchronous message communication.

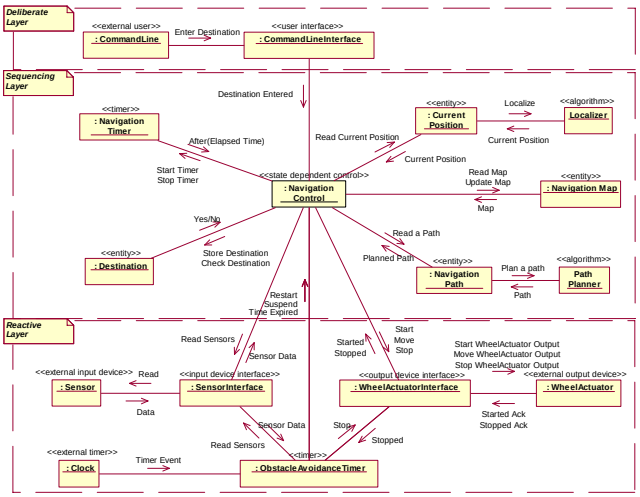

Fig. 10. Consolidated collaboration diagram for Navigation System

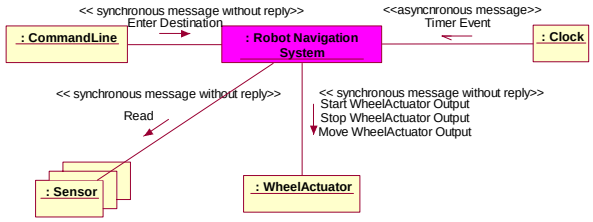

Fig. 11. Distributed software architecture for Navigation System

3.3.3 Task Structuring

During the task structuring phase, a task architecture can be developed in which the system is structured into concurrent tasks, and the task interfaces and interconnections are defined. A task is an active object and has its own thread of control. In this sense, the term "object" will be used to refer to a passive object in this paper. In COMET, task structuring criteria are provided to help in mapping an object-oriented analysis model of the system to a concurrent tasking architecture. At the end of this phase, a task behavior specification (TBS) is developed.

The task architecture for the *Navigation System* is shown in Fig. 12. In order to determine the tasks in the system, it is necessary to understand how the objects in the application interact with each other based on the collaboration diagrams. In the collaboration diagram of Fig. 7, the *Localizer* object reads sensor data and the map from the *Current Position* object, calculates a new current position, and sends the current position to the *Current Position* object for updating it. Thus, the *Localizer* object is structured as an asynchronous algorithm task called *Localizer*. There are two asynchronous algorithms, i.e. *Localizer* and *Path Planner*, which are internal asynchronous tasks. There are four passive entity objects, i.e. *Destination*, *Current Position*, *Navigation Map*, and *Navigation Path*, which do not need a separate thread of control and are further all categorized as data abstraction objects. The *Sensor* and *Wheel Actuator* are a passive input device and a passive output device, respectively because they do not generate an interrupt on completion of the input or output operation.

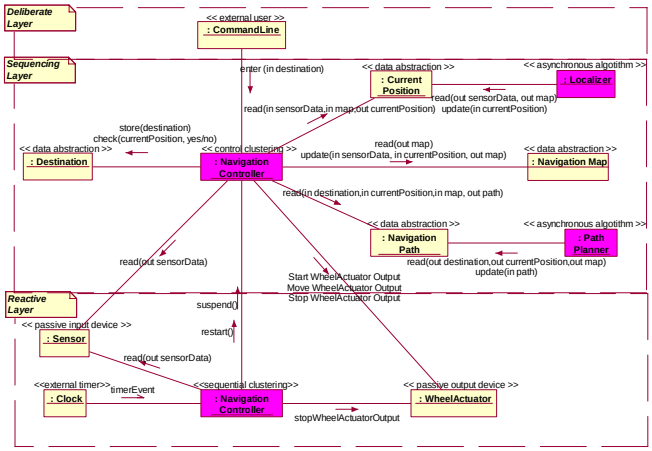

Fig. 12. Task architecture for Navigation System

The *Navigation Control* is a state-dependent control object that executes the *Navigation Control* statechart and is structured as a control task because it needs to have a separate thread of control. The *Navigation Control* object can be combined with the *Command Line Interface*, *Navigation Timer*, *Sensor Interface*, and *Wheel Actuator Interface* objects into one task, Navigation Controller, based on the control clustering task structuring criterion because it is not possible for them to execute concurrently (see the middle of Fig. 12). The *Obstacle Avoidance Timer* object is structured as a periodic task, activated periodically to read sensor data. It can be grouped with the *Sensor Interface* and *Wheel Actuator Interface* into one sequentially clustered task, *Obstacle Avoidance Controller* based on sequential clustering since those are carried out in a sequential order. The design of those composite tasks, the *Navigation Controller* and *Obstacle Avoidance Controller* are considered in the next section (i.e., detailed software design).

After developing the task architecture, a task behavior is described for specifying the characteristics of each task based on COMET. During the task structuring, the TBS focuses on the task inputs and outputs. One part of the TBS, i.e. the task's event sequencing logic is defined in the detailed software design phase.

3.3.4 Detailed Software Design

The internals of composite tasks which have passive objects nested inside them are designed, detailed task synchronization issues are addressed, and each task's internal event sequencing logic is defined in this phase. Before this is done, the information hiding classes (from which the passive objects are instantiated) are designed. In particular, the operations of each class and the design of the class interfaces are determined and specified in a class interface specification (because of space limitation, the detailed TBS and the class interface specification have not been included).

Let us consider the internal design of the *Navigation Controller*, which is a composite task designed as a control clustering task, to show the nested information hiding objects (see Fig. 13). The information hiding objects are the *Navigation Control* state-dependent control object, the *Sensor Interface* and *Wheel Actuator Interface* objects, the *Navigation Timer* object and the user interface object, the *Command Line Interface*. In addition, the *Navigation Controller* contains one coordinator object called *Navigation Coordinator*, which receives incoming messages and coordinates the execution of the other objects. That is, the *Navigation Coordinator* extracts the event from the request and calls *Navigation Control.processEvent (in event, out action)* (see Fig. 13). The *Navigation Control* returns the action to be performed like store, check, start, etc according to the state transition table. Afterwards, the *Navigation Coordinator* initiates the action.

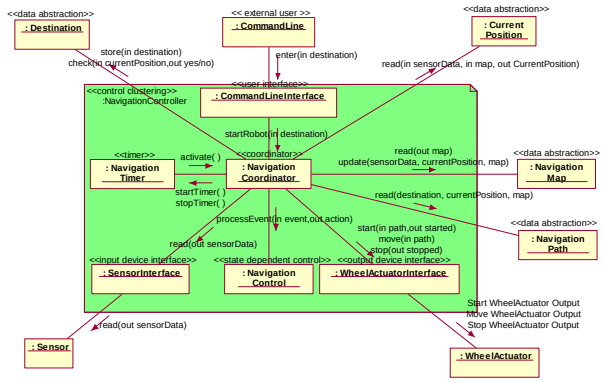

Fig. 13. Detailed software design for Navigation Controller

In our system, communication between tasks such as the *Navigation Controller*, *Localizer*, and *Path Planner* is through data abstraction classes like the *Current Position* and *Navigation Path*. As a result, connector objects [14] are not used for the message communication interface between tasks.

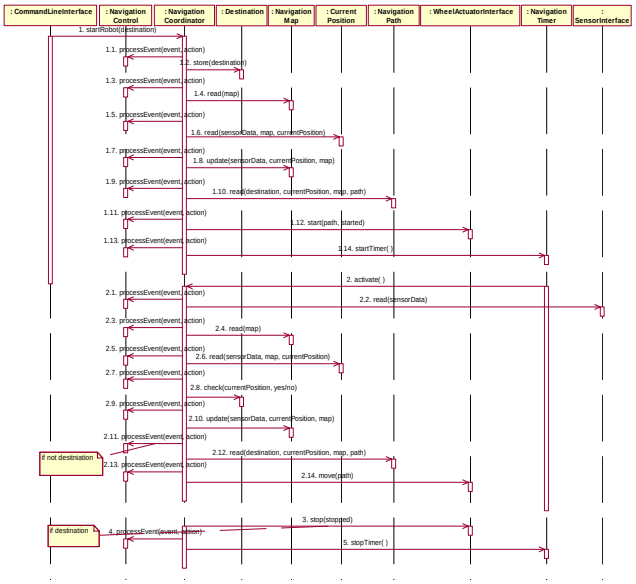

Fig. 14. The task event diagram for Navigation Controller

Lastly, the task's event sequencing logic is specified, which describes how the task responds to each of its message or event inputs. However, instead of using informally Pseudo code in COMET, in this project, task event diagrams are developed for tasks by using the UML sequence diagrams for understanding and readability, which turned out to be very useful when to implement the tasks. Fig. 14 illustrates the task event diagram for the *Navigation Controller*.

4. LESSONS LEARNED

This section summarizes the lessons learned from the project where we successfully applied the object-oriented method with UML to developing the service robot.

4.1 UML for Service Robot Domain

Through the case study, we found that the UML standard was very useful as a notation for specifying the requirements, documenting the structure, decomposing into objects, and defining relationships between objects especially in a service robot system. Certain diagrams and notations were particularly importance for analyzing, designing, and modeling service robot systems as follows.

- Use case diagrams: With the use case model, services or functions (i.e., functional requirements), which a service robot performs or provides, can be defined in terms of actors who are users of the robot system and use cases. Each use case defines the behavior of some aspect of the robot system without revealing its internal structure.
- Class diagrams: The class diagram notation is used to depict the static model, which focuses on the static structure of the robot system. The class diagram shows the classes of objects in the system, their internal structure including attributes, their operations, and their relationships to other classes (such as associations and generalization/inheritance).
- Collaboration diagrams: This diagram shows how objects that participate in each use case interact with each other by sending and receiving messages in the dynamic model. It defines a specific way to use objects in the robot system by showing the possible interactions between them, especially to satisfy the needs described in the use case, namely provide the services. Compared to a sequence diagram, the diagram in particular is useful for synthesizing the collaboration diagrams to create the software architecture of the system as discussed in section 3.3.
- Sequence diagrams: This diagram show objects interaction arranged in time sequence and in particular could be used to describe the task event sequencing logic, which describes how the task responds to each of its message or event inputs. In COMET, the event sequencing logic is usually described informally in Pseudo code. We found that the sequence diagram can help the engineers describe the task event sequencing logic and implement the tasks by showing the order in which messages are passed between tasks and objects.
- State chart diagrams: The service robot system is highly state-dependent like real-time embedded systems. This diagram describes how state-dependent aspects of the system are defined by a finite state machine and can help design and developing highly state-dependent systems. It is

also possible for this diagram to model object behavior over several use cases with the collaboration diagrams.

In addition, by using the UML notation as a defined standard, different research groups and development teams can communicate among themselves and with others to develop and integrate specific components for providing various services.

4.2 Importance of Systematic Process/Method for Service Robot Domain

In order to effectively apply the UML notation and the robot control architecture like the Tripodal schematic control architecture to developing service robots, it is essential to use them with a systematic software engineering process or method, like an object-oriented analysis and design method, especially for real-time and embedded systems. It is not possible to resolve the issues in integrating and developing the service robots discussed before without systematic and comprehensive software development methods, particularly for service robots.

In our case study, we applied COMET/UML method to developing the service robot. The COMET object-oriented software life cycle model is a highly iterative software development process based around the use case concept. In the requirements model, the service or functions (i.e., the function requirements) of the robot system are defined in terms of actors and use cases. In the analysis model, the use case is refined to describe the objects that participate in the use case, and their interactions. In the design model, the robot software architecture is developed, emphasizing issues of distribution, concurrency, and information hiding. This project showed that this was a viable approach because applying the COMET method with UML led to developing an effective service robot architecture by carefully taking into account user needs and requirements, implementing technical components based on the architecture, and integrating these components in a systematic and comprehensive fashion.

4.3 Customizing the COMET Method for Service Robot Domain

Service robots like *PSR-1*, *PSR-2*, and *Jinny* have been built at KIST based on the Tripodal schematic control architecture. The Tripodal schematic design addressed on developing efficient and well-defined control architecture and a system integration strategy for constructing service robots. T-Rot is the next model of the PSR system under development for assisting aged persons. One of our aims is to develop the intelligent service robot for the elderly by cooperating and integrating the results of different research groups in accordance with the Tripodal schematic control architecture that has already been implemented on the PSR and successfully tested. Thus, the layered strategy of the Tripodal schematic design has been applied for design and modeling of the T-Rot. In the collaboration diagrams of the analysis modeling, and the consolidated collaboration diagram and the task architecture of the design modeling, the *Command Line Interface* is located in the deliberate layer for interfacing with a user, while the *Sensor Interface*, *Wheel Actuator Interface*, and *Obstacle Avoidance Timer* are in the reactive layer for controlling and managing the components in the reactive layer. The *Navigation Control*, *Navigation Timer*, *Destination*, *Current Position*, *Navigation Path*, *Navigation Map*, *Localizer*, and *Path Planer* are positioned in the sequencing layer for controlling the robot motion by executing relatively simple computations in real-time. As a result, the Tripodal schematic control architecture was helpful in arranging various hardware and software modules and functions.

Additionally, as stated in section 4.1, in COMET, the event sequencing logic is usually described informally in Pseudo code. We found that the sequence diagram can help the engineers describe the task event sequencing logic and implement the tasks by showing the order in which messages are passed between tasks and objects. Hence, instead of using informal Pseudo code, task event diagrams were developed for tasks by using the UML sequence diagrams to improve understanding and readability. It turned out that these task event diagrams are very useful when implementing these tasks.

4.4 Human Communication

Human communication to understand and develop what is desired of the service robot is likely to be more difficult than expected. In our case study, most engineers who are involved in the project come from the mechanical or robotics engineering field. The different research groups and teams tend to focus on their own technology and components and thus it is not easy to realize how much knowledge they have and how much information will need to be made explicit and communicated to integrate those components for the service robot. Several things can be done to improve the situation. One is for engineers from different teams, especially software engineers and mechanical engineers to work together for analyzing, designing, and developing the robot system during the project. It is very important that all engineers and developers from different groups and teams interact directly. Also, in order to develop a common ground for understanding the domain, technology, process and method, a common medium or language such as UML is critical. In addition to the standard notation like UML, guidelines about what notation to use, when to use it, and how to use the notation comprehensively and systematically are required. This is why the method like COMET is needed. Domain knowledge and experiences in each area will make it much easier to communicate what is desired, e.g. service robot domain, the autonomous robot navigation, vision processing, and so on for software engineers, and object-oriented concepts, software development process, and UML, etc for mechanical engineers. If there is relatively little domain knowledge and experience, to have one day or half-day technical workshop is needed. This has proved useful in a variety of settings in the development of the robot system, such as developing and increasing background knowledge of the domain and technology.

4.5 Necessity of Multi-Aspect Integration Method for Service Robot Domain

A service robot should be able to perform several tasks autonomously to provide various services for human beings in a dynamic and partially unknown environment by applying both technology and knowledge. In order to be able to achieve complex tasks, perform new tasks, and integrate data learned from experience for the robot services, it is required to consider not only the robot's behavior, but also other robot's characteristics such as learning, planning, decision-making, and knowledge representation. It is necessary to allow existing robot behaviors to be used in new ways, to plan for accomplishing more complex tasks, to reuse the knowledge of one task in other tasks, and to

complete tasks more efficiently by learning various action sequences.

In the case study, we focused on designing and modeling the robot's behavioral aspect, which is related to the sequencing and reactive layers in the Tripodal layered design, by applying the COMET/UML method. However, it is clear that planning and learning abilities have to also be considered when designing and developing a service robot, which correspond to the deliberate layer that is responsible for interfacing with a user and executing the planning process. As a consequence, a task manager, which is located in the deliberate layer, has been in charge of these robotic abilities in the project. Because the planning process is knowledge based and not reactive, a different analysis and design approach is needed for the task manager. Hence, we are convinced that methods to model the robot's learning, planning and decision making aspects as well as to incorporate, use and maintain task knowledge are necessary. Furthermore, it is essential to integrate these methods with the COMET method into a multi-aspect integration method for developing service robot software.

5. CONCLUSIONS AND FUTHER WORK

Service robots have been suggested for a growing number of applications. A service robot is a complex system as it includes various technical components (i.e., hardware and software) to be integrated correctly and many different research groups to develop the components. As a result, it is not only essential to develop complex algorithms or technical components, but also to integrate them adequately and correctly to provide the various robot services.

In the paper, we have presented our case study where we developed the autonomous navigation system for the intelligent service robot for the elderly, T-Rot. The object-oriented method for real-time embedded systems, COMET has been applied to the service robot T-Rot with the industry standard UML. It makes it possible to reconcile specific engineering techniques like robot technologies with the UML notation and furthermore to fit such techniques into a fully defined development process towards developing the service robot system. In this way, we contribute to developing service robot software with UML in a systematic manner.

The service robot T-Rot is still under development (at this point, we are at the first stage of total three stages). Thus, the current status of our work is to extend applications that include vision processing, speech processing and manipulation for providing various robot services. Also, we work on designing the knowledge-based task manager for improving the robot's ability.

6. ACKNOWLEDGMENTS

This research (paper) was performed for the Intelligent Robotics Development Program, one of the 21st Century Frontier R&D Programs funded by the Ministry of Commerce, Industry and Energy of Korea.

7. REFERENCES

[1] K. Kawamura and M. Iskarous, Trends in service robots for the disabled and the elderly, Proc. of the 1994 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Vol. 3 (1994) 1674.

[2] R. D. Schraft, "Mechatronics and robotics for service applications," in IEEE Robotics and Automation Magazine, no. 4, pp. 31 - 35, Dec. 1994.

[3] Rofer T., Lankenau A. and Moratz R., Service Robotics-Applications and Safety Issues in an Emerging Market, Workshop W20 proc. ECAI2000, Berlin, 2000.

[4] B. You et al., "Development of a Home Service Robot 'ISSAC'", Proc. of the 1994 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Las Vegas, Nevada, 2003, pp. 2630-2635.

[5] G. Kim, W. Chung, M. Kim, and C. Lee, "Tripodal Schematic Design of the Control Architecture for the Service Robot PSR," in Proc. of the IEEE Conf. on Robotics and Automation, Taipei, Taiwan, pp.2792-2797, 2003.

[6] G. Kim, W. Chung, M. Kim, and C. Lee, "Implementation of Multi-Functional Service Robots Using Tripodal Schematic Control Architecture", in Proc. of IEEE Conf. on Robotics and Automation, New Orleans, LA, USA, 2004

[7] A. C. Dominguez-Brito, D.Hernandez-Sosa, J. Isern-Gonzalez, and J. Cabrera-Games. Integrating robotics software. *IEEE International Conference on Robotics and Automation*, 2004.

[8] Q. Meng and M.H. Lee, "Learning and Control in Assistive Robotics for the Elderly", Proc. Of the 2004 IEEE Conf. on Robotics, Automation and Mechartonics, Singapore, Dec., 2004, pp. 71-76.

[9] M. Kim, J. Lee, K. Kang, Y. Hong, and S. Bang, "Re-engineering Software Architecture of Home Service Robots: A Case Study", Proc. Of 27th Int. Conf. on Software Engineering (ICSE2005), St. Louis, USA, May, 2005, pp.505-513.

[10] G. Booch, Object-Oriented Analysis and Design with Applications, 2nd ed. Redwood City, CA: Benjamin Cummings, 1994.

[11] I. Jacobson, Object-Oriented Software Engineering, Addison Wesley, 1992.

[12] Real-Time Application Interface, 2004. Available at: http://www.rtai.org

[13] Gjalt de Jong, "A UML-Based Design Methodology for Real-Time and Embedded Systems", DATE 2002, March, 2002.

[14] H. Gomaa, Designing Concurrent, Distributed, and Real-Time Application with UML, Addison-Wesley, 2000.

[15] OMG Unified Modeling Language, Version 1.5, March 2003. Available at:http://www.uml.org

[16] M. Fowler and K. Scott, UML Distilled 2nd Edition, Addison Wesley, 2000.

[17] G. Martin, L. Lavagno, and J. Louis-Guerin, "Embedded UML: a merger of real-time UML and codesign", CODES 2001, Copenhagen, April 2001, pp.23-28.

[18] G. Martin, "UML for Embedded Systems Specification and Design: Motivation and Overview", DATE 2002, March, 2002.