Proceedings of the 2005 IEEE
International Conference on Robotics and Automation
Barcelona, Spain, April 2005

3D Virtual Prototyping of Home Service Robots Using ASADAL/OBJ

Kyo Chul Kang, Moonzoo Kim, Jaejoon Lee, Byungkil Kim
Computer Sciecne and Engineering Department
Pohang, South Korea
{kck,moonzoo,gibman, dayfly}@postech.ac.kr

Youngjin Hong, Hyoungki Lee, Seokwon Bang
Samsung Advanced Institue of Technology
Suwon, South Korea
{bhong,twinclee,banggar.bang}@samsung.com

***Abstract* – Typical robot development requires that hardware be mostly functional before significant software development begins. Utilizing virtual prototype of hardware and its environment can reduce development time and manufacturing cost. Virtual prototyping is, however, a highly challenging task requiring in-depth knowledge in many disciplines. Few of simulation tools developed to alleviate this difficulty fully manage the *complexity* caused by the fact that developer must design and manage *form* (physical shape), *function*, and *behavior* altogether incrementally. Also, such simulation tools often do not support co-development of the target software and its test environment from requirements analysis to implementation.**

In this paper, we present our experience of developing a virtual prototype of Samsung Home Robot (SHR) with Samsung Advanced Institute of Technology (SAIT) using ASADAL/OBJ. Virtual prototyping in ASADAL/OBJ enables *incremental co-development* of both target system and its test environment in an object-oriented way. Furthermore ASADAL/OBJ facilitates *concurrent development* of form, behavior, and function. These features increase productivity as well as confidence through incremental refinement and validation. We give a brief background on the ASADAL/OBJ framework, then illuminate our experience of developing a virtual prototype of SHR.

Index Terms* – *virtual prototyping, 3D simulation, incremental validation, object-oriented development

I. INTRODUCTION

Typical development of a robot consists of two often sequential, but distinct processes – developing hardware components and then software application. Thus, most significant application development is delayed until the hardware is mostly functional. Also, prototype hardware is manufactured in very small quantity because prototype should go through several major revisions until final mass production. The lack of available physical HW leads to lengthening the application development time.

Virtual prototyping of hardware using a software simulator allows much of the application to be developed without physical hardware. This technique gives several benefits [1,2]. First, development time can be significantly reduced. Building a proof of concept virtual prototype takes much less time than building a physical prototype. Thus, software development can start much earlier. Second, the hardware design flaws can be detected through validation of a virtual prototype before manufacturing prototype hardware. Third, debugging/testing an application can be more productive in a virtual environment than in a hardware environment. Without a virtual prototype of a robot, robot application developers and testers should wait for their turn to use available physical prototypes, which may be of very limited quantity. Furthermore, when software and hardware are still in development, available prototypes become further limited because prototypes tend to be damaged or disassembled frequently. Last, the virtual prototype can be utilized as an educational tool for human operators of the machine or as a tele-operation application[3].

Virtual prototyping requires, however, in-depth knowledge in many disciplines such as stereoscopic display, multimodal interaction and processing, computer graphics, dynamics and physical simulation. There exist various simulation tools for alleviating these difficulties. Physics engines including Player/Stage[4], Webots[5], and Open Dynamics Engine [6] provide a virtual reality environment supporting physical laws such as collision and friction in varying degrees [7]. AdeptRapid[8], CM-Labs Vortex[9], and DELMIA IGRIP[10] provide more sophisticated simulation but not real-time performance. All of the above works demonstrate realistic 3D visualization of a robot and its environment, and ease of building 3D models by abstracting away low-level details and providing easy-to-use APIs.

The main difficulty of virtual prototyping lies in the *management of complexity*; developers must design and manage three interrelated facets at the same time - *form* (3D shape of virtual objects and its physical properties), *function* (what virtual objects do), and *behavior* (how individual virtual objects carry out functions) as illustrated in Fig 1.

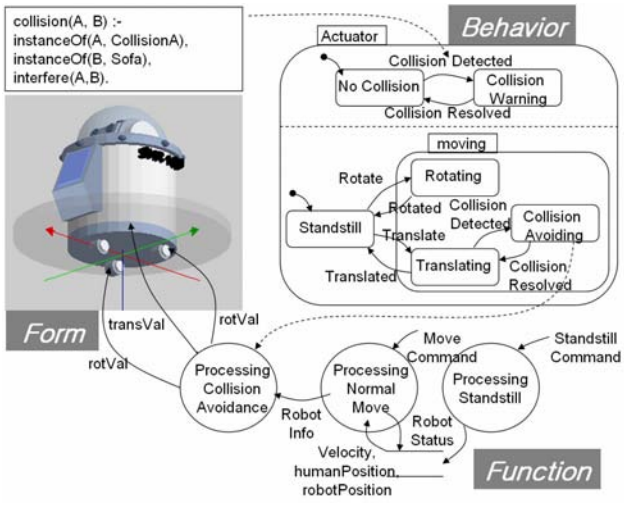

Figure 1. Specification of form, function, and behaviour

Furthermore, construction of a virtual world often requires many revisions. Changing one aspect of a virtual

world will undoubtedly affect other aspects. For instance, different shapes and configurations can result in different dynamic behaviors (e.g. two different robot manipulators differing in size may have different work volumes and capabilities). If virtual prototypes are developed in sequence of form design, then function/behavior programming separately, it easily leads to spaghetti-like code through iterative revisions. As a consequence, information regarding function, behavior, and constraints among them may lose consistency.

In this paper, we present our experience of developing a virtual prototype of Samsung Home Robot (SHR) with Samsung Advanced Institute of Technology (SAIT). We used ASADAL/OBJ[11] to develop a 3D visual model in Form Specification Language (FSL), and behavior specifications in Statechart [12] formalism. We briefly explain the overview of ASADAL/OBJ, and then illustrate our experimental results from the project

II. Overview of ASADAL/OBJ Framework

ASADAL/OBJ is a 3D virtual prototyping extension of ASADAL[13] which is a real-time specification, validation, and verification toolset. With similar goals, SIMAN[14], SIMSCRIPT[15], and SLAM II[16] facilitate accurate description of systems based on Statechart formalism. They, however, are limited to passive visualization of spatial components.

The main modelling philosophy of ASADAL/OBJ is *incremental co-development* of *a target system* and *its environment*. The incremental co-development allows a target system and its environment to be developed, validated, and delivered together in stages, not separately as illustrated in Fig 2. Incremental co-development reduces development risks by breaking a project into series of smaller subprojects. In addition, it increases progress visibility by developing and validating finished operational pieces of a target system against its (testing) environment. Testing environment is also developed incrementally together with the target system, long before the complete target system is operational. The object-oriented nature of ASADAL/OBJ facilitates further this incremental co-development.

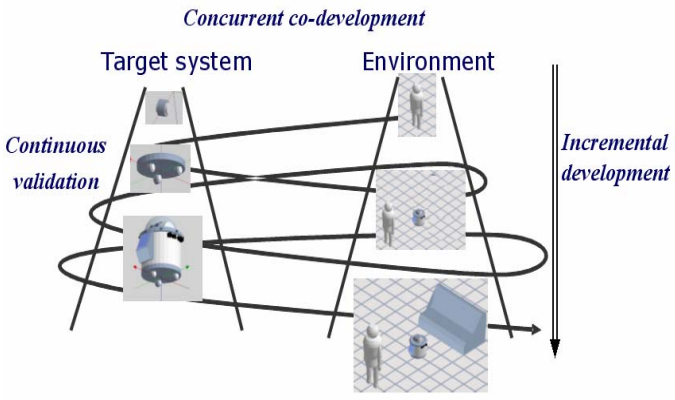

Figure 2. Incremental co-development and continuous validation

Physical properties and configuration of an object (*form*) should evolve incrementally together with its *behavior* and *function*. Form is expressed in FSL describing physical properties/relationships/constraints among physical objects (e.g. a robot body follows its wheels). The behavior and function specification are constructed using Statechart and Data Flow Diagram (DFD) respectively. Spatial movement of an object is coupled with corresponding behavior and function specifications of the object. In other words, a FSL specification may contain spatial properties in terms of control data updated by DFD functions, which are invoked in Statechart behavioral model.

III. Modeling/Validation process of ASADAL/OBJ

A. Form Specification

A physical object of a virtual prototype is represented by an object created by instantiating the corresponding form class. A form class is created by either drawing 3D shape using a CAD-like drawing tool in ASADAL/OBJ or importing CAD or 3D-MAX data directly. There are several meta-classes that abstract various spatial and behavioral characteristics of objects as shown in Fig 3. From these meta-classes, physical characteristics can be simply inherited.

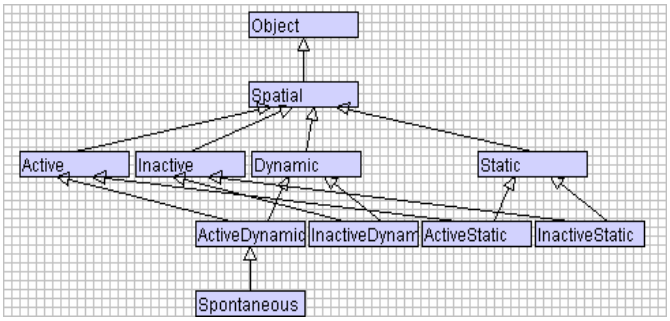

Figure 3. Meta-classes of spatial objects

Every form class inherits `Spatial` class which contains common physical attributes such as volume, position, orientation, and related methods such as `getPosition()` and `getOrientation()`. `Spatial` class has four direct child classes – `Active`, `Inactive`, `Dynamic`, and `Static`. `Active` class represents objects such as motors and joints that have ability to move other objects. `Inactive` class represents objects without such ability. `Dynamic` class means objects which can be moved by other objects. `Static`, on the contrary, stands for objects which can not be moved. `ActiveDynamic`, `InactiveDynamic`, `ActiveStatic`, and `InactiveStatic` classes are created as combinations of these four classes. `Spontaneous` class represents objects which can move autonomously without force outside.

An `Active` object has a list of its passive objects and defines acceleration and velocity applied to its passive children. More specifically, for each moving object, `acc<x>` (acceleration), `vel<x>` (velocity), and `transVal` (moving distance) are defined for linear movement, and `aacc<x>`, `avel<x>`, and `rotVal` (rotating angle) are defined for angular movement where <x> is x, y, or z axis.

Besides movement, physical primitive relations between objects, such as `interfere`, `contact`, `above`, and `below`, can be defined abstractly. For example, `interfere` is established between objects when the

intersection of their volume is nonempty. `Contact` is established when two objects do not interfere with each other but the shortest distance between two objects is smaller than the predefined contact distance. Additional relations can be defined based on the pre-defined primitive relations. When a relation starts to be true or false, a notification event is generated. This event is used by the functional or behavioral specification. For example, when a robot `A` is about to collide with a sofa `B`, a relation `collision (A,B)` defined as below becomes true.

```
collision(A, B) :-
  instanceOf(A,CollisionA*),
  instanceOf(B,Sofa),
  interfere(A,B).
```

Then, an event notifying that `collision(A,B)` becomes true is generated. Behavioral specification of the `actuator` receives this event and invokes a procedure for collision avoidance (`PCA`) defined in function specification (see Fig 1).

B. Behavior and Function Specifications

When we define a form, at the same time, we may specify intended behavior and functional structure using Statechart and DFD. Statechart specifies when processes in DFD are activated as states change. The rounded rectangles in Statechart represent states and dotted lines divide a state into concurrent states. The computations that processes (represented as circles) in DFD carry out are specified using Computation Specification Language (CSL) which is a simplified C-like language.

For example, Fig 1 shows specifications of form, behavior, and function of an actuator component of a home service robot. A composite state `Actuator` located at the top right corner of Fig 1 has two concurrent behaviors. Also, bottom part of Fig 1 describes three functional processes `Collision Avoidance`, `Normal Move` and `Standstill`. These behavioral and functional specifications are responsible for moving a robot safely without colliding into obstacles.

The collision avoidance mechanism in Fig 1 is refined as described in Fig 4 and Fig 5. The sub-state `Collision Avoiding` of Statechart in Fig 1 is refined into three sub-states `Rotating for Avoidance`, `Translating for Avoidance`, and `Adjusting` in Fig 4. Accordingly, the `Collision Avoidance` process of DFD in Fig 1 is refined into two processes `Collision Avoidance Move` and `Processing Adjustment` as depicted in Fig 5.

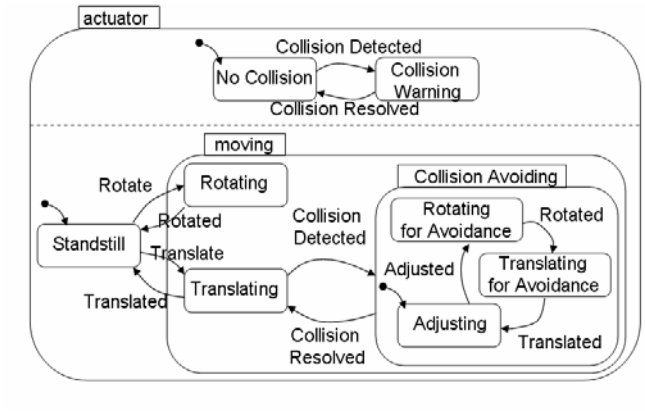

Figure 4. Refined behavioral specification of actuator

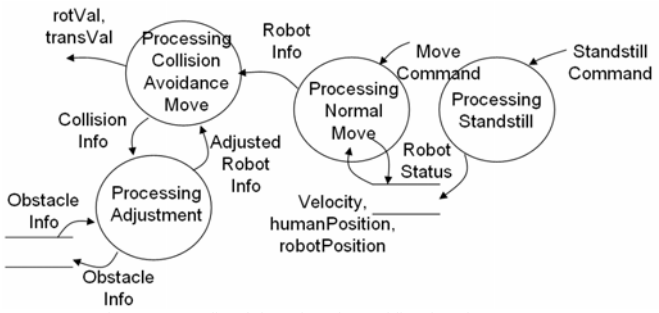

Figure 5. Refined functional specification for actuator

C. Validation Framework

Once form, behavior, and function specifications of objects are developed, logical consistency, correctness, and the timing properties of the specification can be analyzed. The simulator engine of ASADAL/OBJ can visualize operations of a robot based on these form/behavior/function specifications. The simulator, in addition, is capable of performing stochastic data flow analysis, reachability analysis, and non-determinism analysis. The form simulator (implemented using Jun for Java [17]) visualizes transformation of dynamic objects, while updating relations between the objects. Using attribute values of objects (e.g. position or velocity) and primitive spatial relations, the inference engine of ASADAL/OBJ checks constraints on the relations. At the same time, the inference engine logs violations in a simulation log file, or prompts to a user for remedial actions. This simulation driven validation process is continued in an incremental fashion.

IV. Background of SHR100

SHR is a prototype of home service robot for various daily home services such as vacuum cleaning and controlling home appliances, etc.

A. Components of SHR100

SHR100 has a single board computer (Pentium IV with 512MB memory running embedded WindowsXP) controlling peripherals as depicted in Fig 6.

* `CollisionA` is an imaginary object attached to the robot used for collision detection by a structured light sensor (see Sec IV.A and bottom part of Fig 1)

Figure 6. Components of SHR100

- Input peripherals
  - 1 ceiling camera for building a map (640x480 resolution)
  - 1 front camera for recognizing users and remote surveillance (320x240 resolution and 15 frames/s)
  - 8 microphones for speaker localization and speech recognition (8 Khz sampling rate)
  - 1 structured light sensor for obstacle detection
- Output peripherals
  - 1 LCD display for information display
  - 1 speaker for speech generation
  - 2 actuators for right and left wheels
- Input/output peripheral
  - Wireless LAN for communicating to a home server.

B. *Services of SHR100*

SHR100 provides various services including the following primitive services:

1) Call and Come (CC)

This service first analyzes audio data sampled from the eight microphones, which are attached to the surface of the robot, to detect predefined sound patterns (e.g., hand clap or voice command). There are two commands "come" and "stop" Once a "come" command is recognized, the robot tries to detect the direction of sound source by comparing the strength of sound captured by the eight microphones using MUSIC algorithm[18]. Then, the robot rotates to the direction of sound source and tries to recognize a human face by analyzing video data captured by the front camera. If the caller's face is detected, the robot moves forward until it reaches within 1 meter from the caller. A "stop" command makes the robot stop. If any of command recognition, sound source detection, or face recognition fails, CC resets to the initial state. CC is preemptible, i.e., while CC is executed, newly recognized command makes the robot ignore a previous command and follow the new one.

2) User Following (UF)

UF is triggered from the CC service when the robot reaches the user within 1 meter range. The robot constantly checks vision data from the front camera to recognize the color of human skin, and sensor data from the structured light sensor for locating the user and keeps following the user within 1 meter range. If the robot misses the user, the robot notifies the user by speaking "I lost you" and UF turns into CC. Then, the user may make a "come" command to let the robot recognize her and restart UF. UF is a preemptible service.

3) Security Monitoring (SM)

The robot patrols around a house using the map generated by the simultaneous localization and map building (SLAM) module for surveillance. Intrusions or accidents are defined as patterns recognizable from vision and sound data. For example, an intrusion can be detected by watching images and monitoring sound from doors and windows. Once such an event is detected, the robot notifies a user directly via an alarm or indirectly through a home server.

4) Tele-presence (TP)

A remote user can control a robot using a PDA. The robot sends periodically a map of the house generated by the SLAM module to the PDA. The user can command the robot to move to a specific position in the map displayed on the PDA. In addition, the robot can send images obtained from the front camera to the remote PDA for surveillance purpose. While the user is out of house, the robot can communicate with the user through a home server.

V. PROTOTYPING SHR100 WITH ASADAL/OBJ

A. *Development History*

SHR100 is a successor of SHR50 and SHR00. Development of SHR00 started in 2002 by four separate teams consisting of 13 people working on speech recognition, vision recognition, map building, and actuator control. SHR50 as well as SHR00, however, often exhibited unstable behaviors such as missing user commands and stuttered movement although each part had worked successfully when not integrated (this kind of failure is not uncommon, see [19]). As a consequence, SAIT gave up SHR50 and SHR00 and developed both hardware and software of SHR100 from scratch. After ten months into the new development (when hardware of SHR was mostly working), POSTECH joined the project and reviewed the hardware and software requirement specifications and built a virtual prototype of SHR100 for validation of software applications.

B. *Modeling Physical Hardware*

Currently, the overall virtual environment is represented as a room consisting of a floor on which all other objects exist: a SHR100, a human who gives commands to the SHR100, and a sofa which is an obstacle (see Fig 11). We modeled a body and a power switch of the SHR100 first, then added actuators, a front camera, and so on incrementally. The components of SHR100 we modeled are depicted in Fig 8.

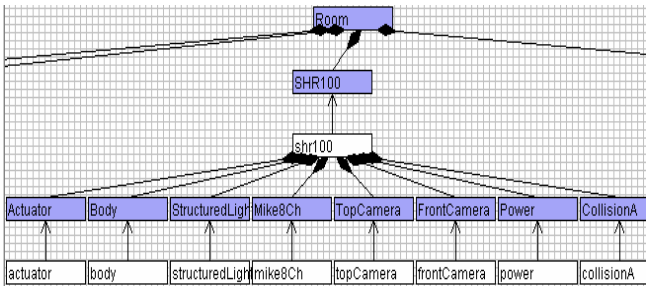

Figure 7. Subcomponents of SHR100

Each component has corresponding behavioral specification as well as functional specification. The assumption we made for modeling hardware components are as follows. We modeled a front camera as an ideal one in that the camera detects human without failure within view angle of 60 degrees and within 4 meter range. Similarly, 8 channel microphones always detect the direction of sound and recognize commands of the user correctly regardless of the distance from the human. Furthermore, the actuator does not fail to rotate a given rotation of angle. Therefore, once a user gives a "stop" command or a "come" command, the robot does not fail to follow the command. These simplified assumptions are beneficial for rapid prototyping of hardware and for validating the software controller quickly. Once we validate the software controller, we can refine these devices with more realistic physical characteristics and then refine software controller accordingly.

C. *Modeling Software Controller*

The overall software architecture of SHR100 is depicted in Fig 8. Each of the service components such as CC and UF controls the computational components such as Vision Manger and Audio Manager. The Mode Manager component defines the system modes (e.g., initialization, termination, and power saving modes) and the interaction policy (e.g., priority, concurrency) between the service components.

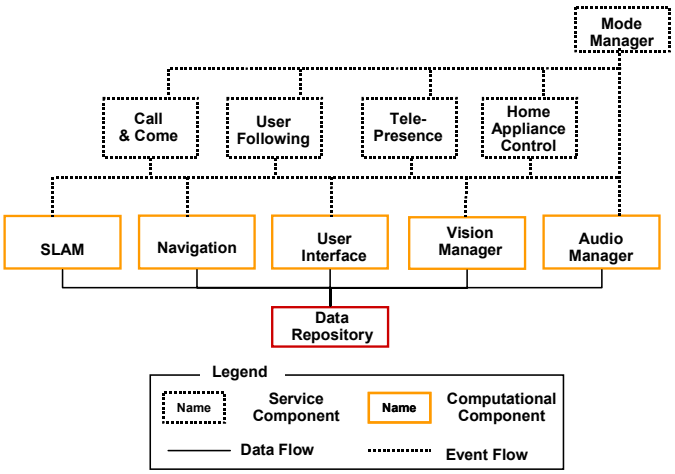

Figure 8. Software architecture of SHR100

The input data from the sensors (e.g., the 8-channel microphones, the front camera, etc) are gathered and processed by the computational components, and the results are stored in the Data Repository. For example, the SLAM component generates map and current position data, and the Navigation component uses the data to determine the next destination of the robot. Also, the computational components send events (e.g., user's voice commands such as "stop" or "come" recognized by Audio Manager) to Mode Manager. The events are processed by Mode Manager to determine the global state of the robot, and then they are delivered to relevant service components. (More details of the software architecture can be found in [20].)

The service components are developed separately but interactions between them are specified in Mode Manager. For instance, Fig 9 shows which events activate the CC or UF services. Note that the active service is switched from UF to CC, when a "User Lost" event is reported from Vision Manager. As a result, the UF service is suspended until the CC service relocates the user and generates a "Resume UF" event (See Fig 10 for the behavior specification of UF).

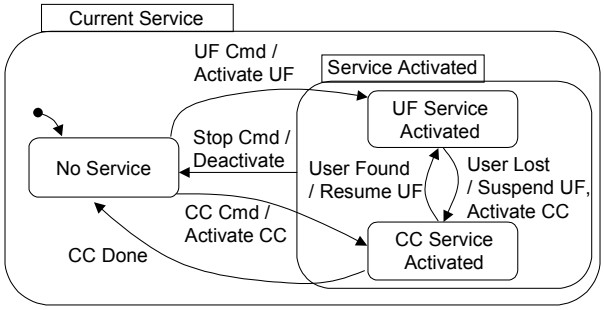

Figure 9. Behavioral specification of Mode Manager

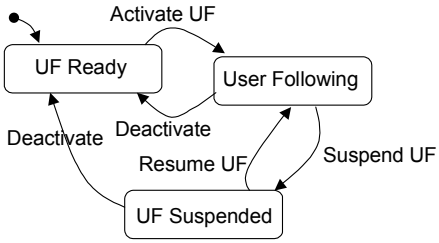

Figure 10. Behavioral specification of UF

D. *Simulation Results*

It took less than a week to develop a virtual prototype of SHR100 and its environment by a graduate student who had no prior experience with ASADAL/OBJ. Fig 11 illustrates a snapshot of the visual simulation of SHR100 using ASADAL/OBJ. In the room, a sofa, SHR100, and a person exist. An ASADAL/OBJ user can move each object via drag-and-drop. Simulation starts by generating an event to power on SHR100. Then, the user moves the person to a specific place such as behind the sofa. Finally, the user generates a command (e.g. "come" or "stop") by clicking an input event from a list of possible input events. As a reaction to the command, SHR100 rotates and moves towards the person by moving around the sofa. While the SHR100 is operational, current states of the components are indicated with red color in the Statechart window as in Fig 11.

We tried two different collision avoidance algorithms. The first one was a simple algorithm – when an obstacle is detected, the robot turns 90 degree in clockwise direction and moves 2 meters forward, then moves towards the destination again. The second one was more sophisticated one: it estimates the size of the obstacle and finds the shortest de-tour path using trigonometric functions. This change was easily incorporated into the robot by simply modifying the function specification for collision avoidance (see Fig 5). We could see that the second algorithm showed better movement.

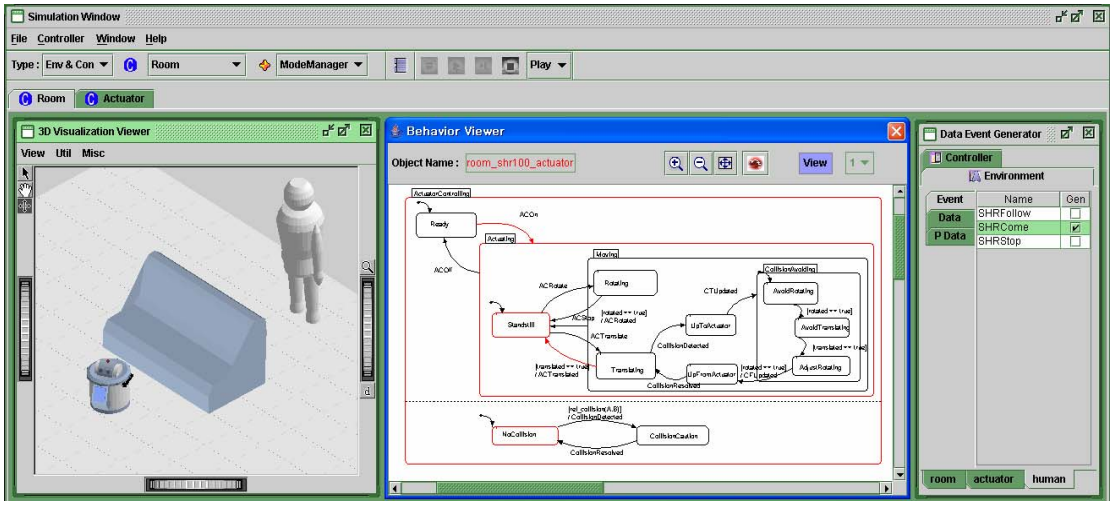

Figure 11. Snapshot of 3D visual simulation

Another notable result during the simulation was that we observed that SHR100 sometimes ignored a "stop" command and did not stop. This problem turned out to be a feature interaction problem [21] between UF and CC. Basically, UF was designed to track a user only with vision data, not with audio data. Therefore, when UF fails to locate the user the robot was following, UF requests CC to locate the user by detecting the direction of the sound source. At this point, the feature interaction occurred when the user gave a "stop" command; the CC service sent the direction of sound source to UF and UF resumed moving the robot to the direction of the user without stopping the robot. After modifying CC, UF, and Mode Manger appropriately, we could solve the problem.

VI. Conclusion

We have described our experience of building a virtual prototype of SHR100 using ASADAL/OBJ. With support for incremental co-development of a target system as well as its environment, developing and validating the virtual prototype was not an intimidating task. We started building the virtual prototype of SHR100 by creating a body. Then, we added components and their behaviors/functions to the body one by one. At the same time, we built a test environment including a sofa and a person so that we could validate the prototype incrementally. As a result, a graduate student without prior experience with ASADAL/OBJ could build and validate the prototype in a week. In addition, through 3D visual simulation of the prototype, we could detect a problem of ignoring "stop" command due to a feature interaction between CC and UF services. Also, we could validate visually the effectiveness of refined collision avoidance algorithm compared to the original one.

As a future work, we plan to add more service features to the prototype and refine the prototype so that we can have a more realistic virtual prototype of SHR100.

References

[1] R.lumia, G.Starr, J.wood, B.Jones, I.M.Shohet, and E.Ledman "An Approach to minimize robotics system development and integration time", International Conference on Robotics and Automation, 1997

[2] S.P.DiMaio, S.E.Salcudean, C.Reboulet, S.Tafazoli, and K.Hashtrudi-Zaad,"AVirtual Excavator for Controller Development and Evacuation", International Conference on Robotics and Automation, 1998

[3] H.R.Nicholls, J.J.Rowland, and M.G.Taylor, "Using Simulation for Plant Monitoring in Real Time", International Conference on Robotics and Automation, 1997

[4] The Player/Stage Project. http://playerstage.sourceforge.net/

[5] Webots. http://www.cyberbotics.com/

[6] Open Dynamics Engine user manual v 0.5 http://ode.org/ode-latest-userguide.html

[7] M. Lewis and J.Jacobson, "Game Engines in Research", Communications of the Association for Computing Machinery (CACM), NY: ACM 45(1), 27-48, 2002.

[8] J.J.Craig "Geometric algorithms in AdeptRAPID" Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics, 1998

[9] Algorithms and Techniques in Vortex by CMlabs http://www.criticalmasslabs.com/products/vortex/algorithms.php

[10]DELMIA web page, http://www.delmia.com/

[11]J. Lee, H.Kim, and K. Kang, "A Real World Object Modeling Method for Creating Simulation Environment of Real-Time Systems", ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, 2000

[12]D. Harel and E. Gery. "Executable Object Modeling with Statecharts" Proceedings of the 18th International Conference on Software Engineering, ACM Press, 246-257, Berlin, Germany, 1996

[13]K.C.Kang, K.W.Lee, J.Y.Lee, and G.J.Kim, "ASADAL/SIM: An Incremental Multi-Level Simulation and analysis tool for real-time software Specifications", Software: Practice and Experience, Vol. 2 8, Issue 4, pp.445-462, 1998

[14]C.D.Pegden, R.E.Shannon and R.P.Sadowski, "Introduction to Simulation Using SIMAN", 2nd Ed., McGraw Hill, New York, 1995

[15]Building Simulation Models with SIMSCRIPT II.5 at ftp://ftp.casiasl.com/pub/simscript/docs/zbuildin.pdf CACI Products Company, 1999

[16]A. Pritsker, "Introduction to Simulation & SLAM II", John Wiley & Sons, Inc., New York, 1995

[17]Jun for Java by Software Research Associate, Inc, http://www.sra.co.jp/people/nisinaka/Jun4Java/index_ja.html

[18]G. Su and M. Morf, "The Signal subspace approach for multiple wide-band emitter location", IEEE Transaction on acoustics, speech, and signal processing, vol. ASSP-31, No 6, Dec 1983

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

[20]M.Kim, J.Lee, K. Kang,Y.Hong, and S.Bang, "Re-engineering Software Architecture of Home Service Robots: A Case Study", submitted to IEEE International Conference on Software Engineering 2005

[21]E.J.Cameron and H.Velthujisen, "Feature interactions in telecommunications systems", IEEE Communications Magazine,31(8):46-51, 1993