International Journal of Control and Automation
Vol. 8, No. 1 (2015), pp. 69-82
http://dx.doi.org/10.14257/ijca.2015.8.1.06

Architecture Model to Improve the Development of Robotics Online Reconfiguration

Xiaoan Bao[1], Xiance Sun[1], Ning Gui[1,2], Na Zhang[1], Hui Lin[1] and Shuhan Liu[1]

[1]*Zhejiang Sci-Tech University, 310018, China*
[2]*Distrinet Lab, University of Leuven, 3001, Belgium*
baoxiaoan@zstu.edu.cn

Abstract

With the fast development of robot technologies, robots are increasingly operating in complex and unknown environments. Thus, the robotic software systems need to be online reconfigurable to fast meet users' new demands or environment requirements. In order to improve the online reconfiguration, an innovative way to combine runtime software architecture and robotic development is proposed, and then use model-driven techniques to establish a rigorous and useable abstract architecture meta-model and robotic system meta-model. In order to improve the bidirectional transformation, model difference and merge operations are constructed and proposed. And meanwhile we extending an existing synchronization engine generation tool and makes it adapted to the robot system. Finally, our platform is test with a concrete case study on top of the Microsoft Robotics Developer Studio to further verify its correctness. The result proves this study can effectively improve online reconfiguration of robotic software systems, and architecture model-based online configure makes it can effectively improve the development of different types of robot software architectures, and improves the reusability of the entire development process.

Keywords*: online reconfiguration; runtime software architecture; model-driven; bidirectional transformation; synchronization engine*

1. Introduction

The environments surrounding robots are becoming more and more complex and full of unknown, robotic software systems cannot be brought down for upgrades. Instead, they should adapt themselves to software and hardware failures, changes in their computing and physical environments, and the arrival of upgraded services. Robot must be able to meet users' new demand and adapt to unknown environment at runtime, but it is difficult to meet the support for runtime reconfiguration for most robotic systems now. Sykes *et al.*, [1] and Georgas and Taylor [2] being two representative examples. The observation made by both of the above research groups is that software architecture can serve as the sticking point of robotic software design and adaptation. Both groups have proposed reference architectures for the domain of robotic software that support self-adaptation and different adaptation policies for the resulting systems. These approaches demonstrate the benefits of affording software architecture a central role in constructing self-adaptive and self-managing robotic systems. Moreover, both of these proposed reference architectures, and, more generally, other architectures for self-adaptive robotic systems don't support online reconfiguration very well. Robotic software designed to meet the needs of runtime reconfiguration can be repeatedly modified and prototyping at runtime, robot developers need powerful software designed to support autonomous robotic systems.

In order to solve the effective of robotics online reconfiguration problems, this paper, we innovatively applied runtime software architecture research to the development of robotic software, and proposed a online reconfigurable architecture model development approach,

ISSN: 2005-4297 IJCA

which developers can focus on relatively simple architecture model and use sophisticated architecture-based technology. Then only by simply monitoring and changing the runtime software architecture model, developers can management and control the complex robotic systems.

The rest of the paper is organized in four sections. Section 2 describes the overall design architecture and key technologies needed, then cites simple examples for specific instructions. Section 3 introduces the implementation of our ideas, from the architecture meta-model to the robotic system meta-model, as well as their synchronization mechanisms. Finally on the Microsoft Robotics Developer Studio, we illustrate how to build robot applications and validate the feasibility and correctness of our innovation. Section 4 is the related works done in the robotic community. Section 5 is a conclusion of this paper.

2. Robotic Architecture Meta-model Design

2.1. Runtime Software Architecture

To let runtime software architecture apply to robotics software development, we first need abstract the actual complex robotic systems to be an abstract architecture model, which we use meta-model to present. Runtime Software Architecture system is to be seen as a dynamic runtime architecture model, and it is a dynamic performance of the runtime systems. It also can monitor and adaptive systems in a highly abstract level, having a casual connection with the actual system. Change the actual system will react to changes in runtime architecture, and reconfigure runtime architecture also mapped to the actual runtime system. The whole concept design diagram which applies runtime software architecture to robot development system is shown in Figure 1:

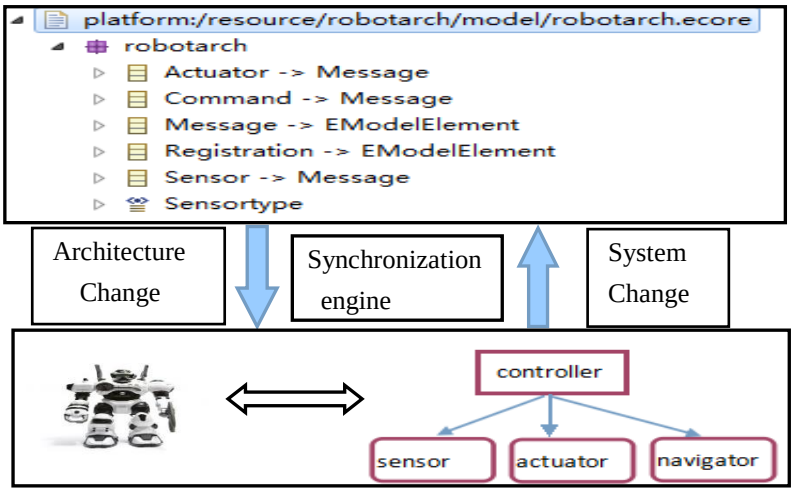

Figure 1. The Overall Design Diagram

As can be seen from the diagram, the upper structure of the abstract model, which contains the elements abstracted from the complex robotic system. Architecture model is an abstract model and base on a different mapping mechanism, it can be mapped to different types of robotic systems. Most importantly it is a loosely coupled relationship with the different underlying robotic systems. The lower layer is the actual robotic system model, it can be seen as the agents of the real robot. To achieve runtime reconfiguration through the upper abstract architecture model to control the actual robotic system, we need the middle runtime synchronization engine to ensure consistency at runtime, and also, we need model difference and merge technology and bidirectional-transformation between models to support our suggestions. In order to ensure the correspondence between the two models, we should ensure

the transition between the model synchronization. Following section is about these key technologies.

2.2. Bidirectional Transformation

In order to ensure consistency between the upper abstract architecture and the lower actual systems model, bidirectional transformation technology is required. Bidirectional transformation [3] uses one relation between two sets of models (such as two meta-models) to derive two directions of transformations between them. Formally speaking, for two meta-models A and S, and the relation can be expressed as $R \subseteq A \times S$, the bidirectional-transformation is constituted of two functions as bellow [4]:

$$\vec{R} : A \times S \rightarrow S$$
$$\overleftarrow{R} : A \times S \rightarrow A$$

$\vec{R}$ looks at a pair of models (a, s) and works out how to modify model s so as to enforce the relation R, and then it returns the modified version of model s. Similarly, $\overleftarrow{R}$ represents changes in the opposite direction. Note that the transformation requires two parameters; they respectively are instances of architecture meta-model and system meta-model.For a ϵ A, there may exist more than one sϵS satisfying(a,s) ϵ R.Detail examples can be found in Czarnecki *et al.*, [5].

2.3. Model Difference and Merge

To control the actual robot through abstract architectural model, it is necessary to reconfigure the architecture model, and mainly for model difference and merge. Differences in the model is the two existing model comparison, by comparing the model elements and attributes, we get a model difference represents by the symbol "−",−: A×A→ΔA,ΔA represents the calculated difference between the two models[6], and is a set of operations for the original model. Model merge can merge the current model with a model of discrepancy, and then produces a new model. Model merge uses the symbol "+" to indicate, +: A×ΔA→A. To more intuitive and clearly describe model difference and merge, the following examples give a simple explanation, Figure 2 shows a simple example of model difference:

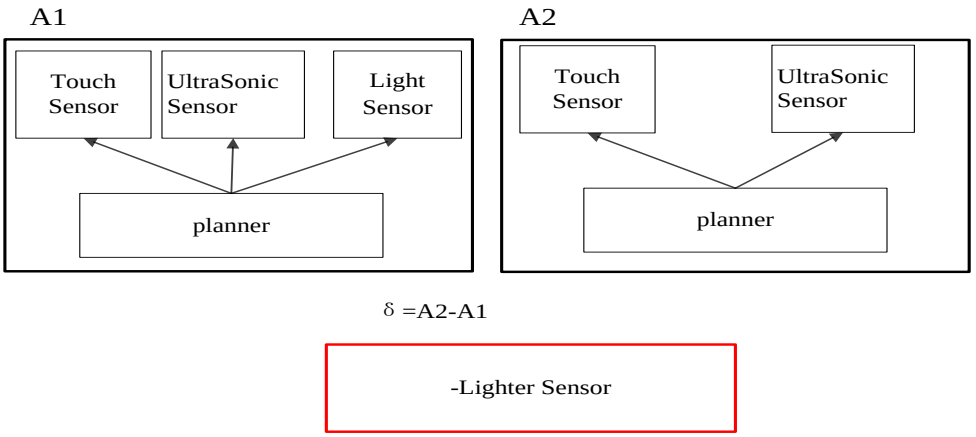

Figure 2. Runtime Architecture Model Difference Instance

As can be seen from the example, by comparing the model A1 and A2 ,we can see that due to the need for some specific scenarios (*e.g.*, low battery, the work situation becomes daytime), the robot smartly shut down Light Sensor, meanwhile the robotic system model

synchronous this transform and then mapped to the runtime architecture. So model A2 misses Light Sensor, and by making model difference with the original runtime architecture model A1 we get the model discrepancy δ, and the Light Sensor in architecture model removed intelligently (expressed as -Light Sensor). Then reflect on the actual robotic runtime system by synchronizing mechanism and socket commands, the sensor will be shut down in the real robotic system. Model merge has the corresponding results as Figure 3 shows:

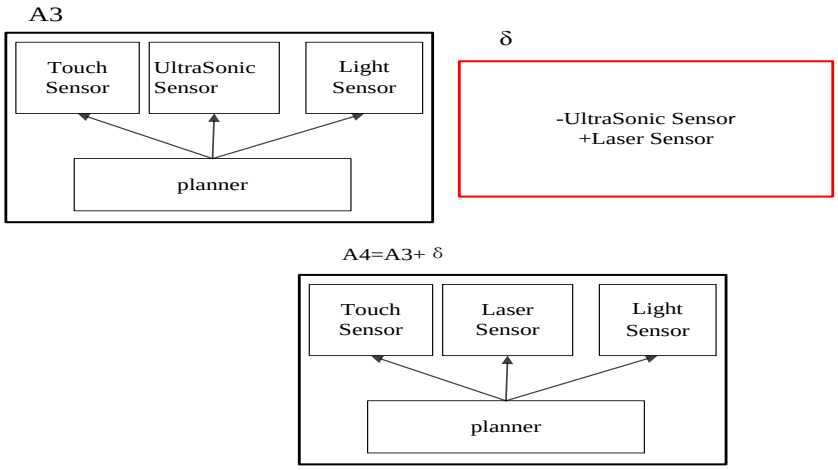

Figure 3. Runtime Architecture Model Merge Instance

The figure above represents the current architecture model merges with the difference model δ, which means the developer will replace ultrasonic sensors with a laser sensor. The model δ is synchronized from the system model change; this transformation on system model can be manifested by the existing model A3 merge with model δ, getting a new runtime model A4. Developers can see the change of the system clearly from the new architecture model A4.

2.4. Synchronization Mechanism

Synchronization is an important safeguard to ensure the operation framework between the model and the actual system. To achieve synchronization mechanism, we must provide the elements required to manage the system model and how to invoke these elements. Synchronization mechanism's importance can be expressed in the overall design of the block diagram in Figure 4:

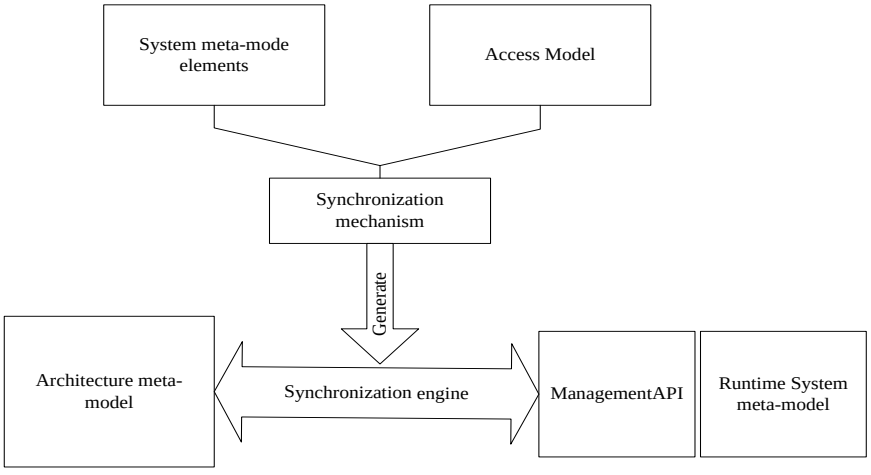

Figure 4. The Synchronization Mechanism Design Diagram

As can be concluded from the design diagram: the inputs include a system model specifying what kinds of elements can be managed and an access model specifying how to use the API to monitor and modify those manageable elements, and then use some kind of synchronization mechanism to generate the synchronization engine based on the inputs. System model elements contained in the inputs are corresponding with the devices in the actual systems, and shows the properties, methods, and collaborative relationships between them. A key question is when the synchronization trigger synchronization engine, synchronization engine is the interaction bridge between architecture model and system model, in order to use the abstract runtime architecture model to control the actual system, and the synchronization engine should touch off before developers read the model instance and after they write the model. For a read operation, the model listener involves and touches off the synchronization engine, and does the read operation until synchronized to the latest model instance; meanwhile, for a write operation the model requires the model listener to trig synchronization engine after the write operation is complete, then synchronized the real-time configuration model instances and reflected in the actual system.

3. Implementation Online Reconfiguration Technology

In order to achieve online reconfiguration through abstract architecture model to control the robotic system, we use Eclipse modeling framework technology to establish the corresponding meta-models, EMF is a powerful framework and code generation tools, and this paper also uses the code generation tools to generate the java runtime environment. The implementation process of our architecture designation is shown in Figure 5:

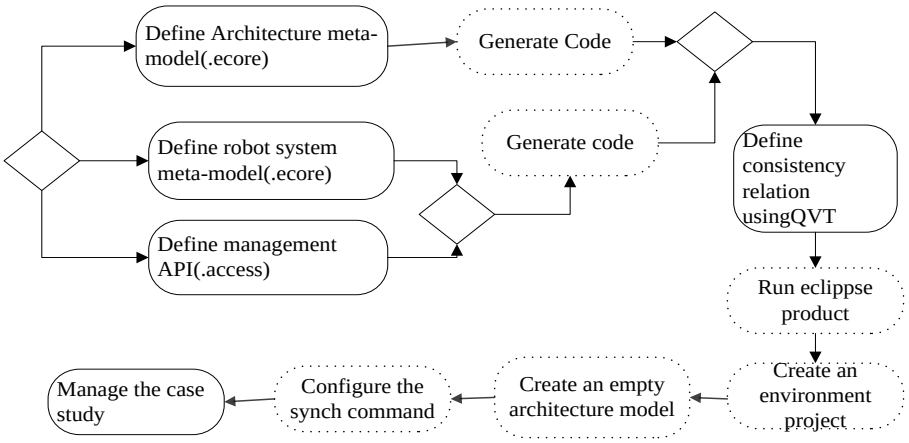

Figure 5. The Flow Diagram of Implement

The dashboard view above shows the whole process for developing a runtime management framework. Solid rounded rectangles represent the activities which require development work, while the dashed rectangles are trivial activities which can be performed with the generation tool. According to the design flowchart, first define architecture meta-model and robotic system meta-model with the selected robot components .Second, define the appropriate management API to invoke and manage these robot components, and then turn the runtime architecture meta-model into code, and converted the defined system model and management API into java code. To further simplify the steps of writing complex code, we use development tools provided by the eclipse platform to automatically generate code based on genmodel created by us. And then to define the conversion relationship between architecture meta-model and robotic systems meta-model by QVT, combines with the generated project code as an eclipse application to generate runtime environment, and then create instances of architecture meta-model and robotic system meta-model. By managing these abstract architecture meta-model examples, we can control the actual instances of the corresponding

elements model of the robot. (Diamond in Figure 5 is for connecting only, and has no real meaning)

3.1. Abstract Architecture Meta-model Constructs

Establish the abstract architecture meta-model is the priority to achieve online reconfiguration through abstract architecture meta-model to control the robotic system. Meta-model can be established in many ways, this paper by using EMF technology and class diagram to create meta-model. Meta-model defines what kinds of elements will appear in the abstract architecture meta-model, how the properties of these elements and relationships between these elements represent. These definitions will be recorded in the EMF meta-model, UML meta-model this paper established represents in Figure 6:

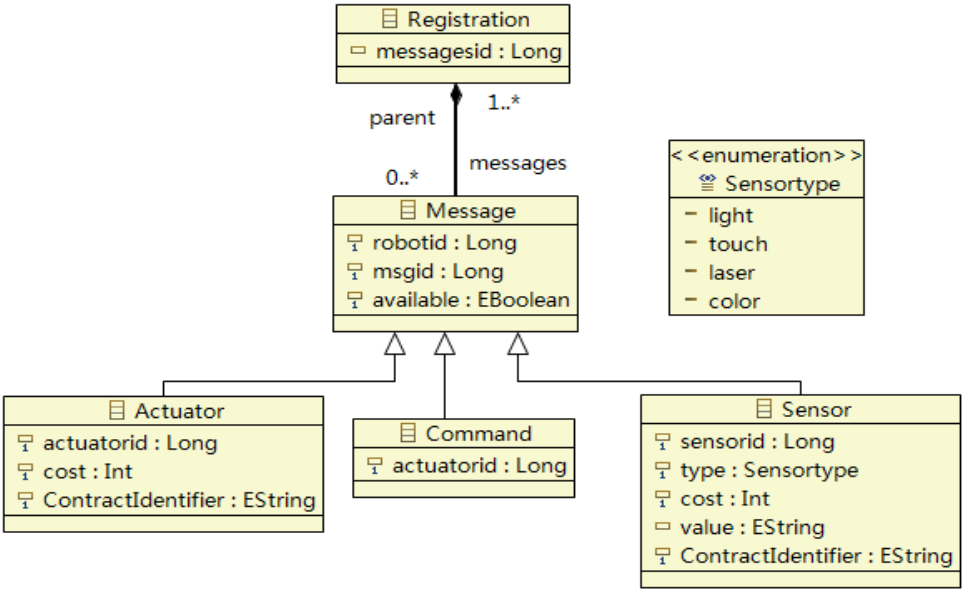

Figure 6. The UML Meta-model of Runtime Architecture

In the class diagram, we abstract five classes to represent robotic architecture meta-model, which Registration class registered the recorded actuators and sensors; Message class is the parent class, it has three attributes: robotid, msgid and available. The attributes' Lower Bound and Upper Bound are "1", representing each instance has the only robotid and msgid value, these two values can be used to distinguish different robot instances. The available attribute indicates whether a device's state is concerned, if you want to detect its value, we can set its value be true, and inverse be false, then the instances' value will not appear in the schema element model. Sensor class contains sensorid, ContractIdentifier, type, value and other attributes, from these attributes we can know how to access these sensors and obtain the value of each sensor, the values type of sensors expressed in String. Command class is the root class, all sent commands must inherit from Command class. Besides, the paper also customizes the Sensortype data types, to distinguish different types of sensors. The above class diagram's ecore model view is shown in Figure 7:

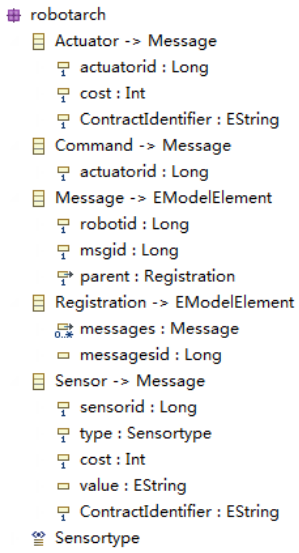

Figure 7. The Meta-model of Runtime Architecture

By configuring the instances of above architecture meta-model, as well as the associated module, we can instead the main components of the robotic system, enabling the actual robotic control. In the adaptive strategy, we can achieve by adding the corresponding annotation attributes, adding comments is a standard extensible way to obtain additional information for architecture meta-model in EMF [7]. Comments can contain semantics and constraints, prototype engine based on these semantics and constraints to automatically modify the architecture meta-model.

3.2. Robotic System Meta-model Constructs

On the basis of the establish of the abstract architecture meta-model in previous section, the establish of the robotic system meta-model's main difference lies in the way elements property represented and the relationship between the elements, the robotic system meta-model model established in this paper as shown in Figure 8:

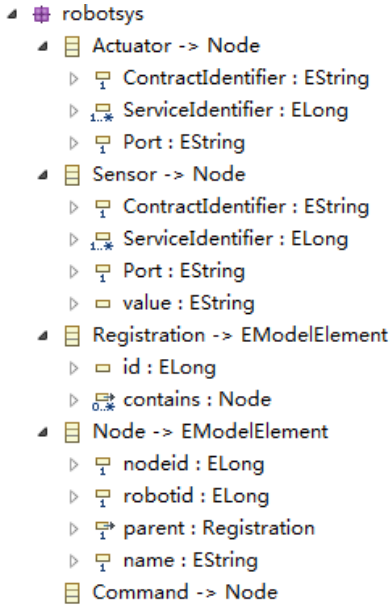

Figure 8. The Meta-model of Robotic Systems

Meta-model of the robotic system is more specific than the runtime architecture meta-model, and the actual properties of the class are corresponding with real robots, so it can be seen as class objects of actual robot components. After the meta-model elements of the robotic system generated class, we can create instances of the corresponding class to represent the component parts of an actual robot, we can form the actual robotic element by creating an instance of the model element. After that, we can use such components, get the component status, and access to collaborative relationships between them through the properties of these instances.

3.3. Implementation Synchronization Mechanism

As a bridge between abstract architecture meta-model and robotic system meta- model, to ensure the causal connection between instances of abstract architecture meta- model and meta-model the following two requirements are required:

1. Correct Introspection. No matter how robotic system changes, the management developers could always get the correct system state through the architecture meta-model.

2. Correct Reconfiguration. Management developers could directly modify the architecture meta-model, and the modifications will dynamically cause the correct robotic system change.

Correctness of the correspondence relationship between the architecture meta-model and the actual system totally dependent on the relation $R \subseteq A \times S$, when a pair of instances $(a, s) \in R$, we say that the abstract architecture meta-model instance and the actual system is consistent, or the architecture meta-model instance is a reflection instance of the actual system instance s. For robotic systems, when the actual robotic system adds a new sensor, then the corresponding architecture meta-model also will add an instance with the same name and sensor properties, and vice versa.

According to the causal connection concepts above, this paper established corresponding genmodel of the above architecture meta-model and robotic systems meta-model, and generate the corresponding java plug-in project based on genmodel combined with eclipse development tool, and then run these projects as the entire runtime operating environment. Second, this paper uses the SM@RT tool to automatically generate the synchronization engine [8], SM@RT tool is a universal tool, if we want to use SM@RT tool, we must provide the elements of the robotic system to be managed and corresponding management API, this paper by expanding SM@RT tool to meet application on the robotic system, elements are transmit to SM@RT by the robotic system meta-model. The robotic management API is presented in java code by creating a DecModel in the EMF project, Figure 9 is a code snap for ButtonPress listening and Handling .Codes in DecModel is for the target robotic system listening and code block to call robot management API, so as to achieve retrieve and update the state of the actual state of the robotic system. Based on the research front, we created a Mapping pool, which implements by hash map, to record the references of runtime architecture meta-model instances and the corresponding robotic system elements. Mapping pool changes with robotic system instances or architecture meta-model instances, dynamic recording elements used by robotic systems. By viewing the contents of Mapping pool, we can know what elements are already registered and available in the registry.


```
String [] commands = { EXIT, QUIT, HELP, QUERIE, DEL, RM, MEM, DEFRAG, LINUX};
public static void main(String [] args) {
KeyListener kl = new CommandLine();System.out.println("NXT 1.0 Connecting...");
try {Keyboard k = new Keyboard();k.addKeyListener(kl); System.out.print(prompt);
      } catch(BluetoothStateException bt) {     System.err.println(bt.getMessage());}
        Button.waitForAnyPress();}
public void keyPressed(KeyEvent e) {
if(e.getKeyCode() == KeyEvent.VK_ENTER) {// Check for command line arguments
String [] args = new String[st.countTokens()];
for(int i=0;i<args.length;i++) {args[i] = st.nextToken();}// Select from list:
            if(command.equalsIgnoreCase(QUERIE)|command.equalsIgnoreCase(HELP)) {
      for(int i=0;i<commands.length;i++) {      System.out.print(commands[i] + " ");}
      } else if(command.equalsIgnoreCase(EXIT)|command.equalsIgnoreCase(QUIT)) {
                        System.exit(0);} ……..}}
```

Figure 9. Manage Code Snap for ButtonPress in DeModel

4. Verification and Analysis of Online Reconfiguration

4.1. Selection and Design of Verification Platform

To further validate the correctness of the proposed ideas, this paper combines the instances of robotic system meta-model with one robotic application platform to verify the feasibility of its real importance. The importance of simulation is to verify the support performance of online reconfiguration, so the robotic applications platform must be able to meet the robot repeatedly reconfigured at runtime. We chose Microsoft robot development studio, due to MRDS [9] is service-oriented, which is different from some component-based robot development platforms, because in component-based robot development platform components' relationship is set in advance and difficult to adjust the architecture at runtime. While MRDS is service-oriented development approach, the relationship of component is defined at runtime, so it can be changed at any runtime to meet the verification for this paper's online reconfiguration request. As we have no real robot to directly download the code generated in Eclipse at present, after selecting the MRDS, we have to implement communication between java code generated by the robotic system meta-model instances and the C# code for writing the actual robot agents. This paper we use the socket serialization order to achieve this goal and have written the communication command between different interfaces. Finally, we successfully implement communication between the simulation robot agents in MRDS and instances of robotic system meta-model. The three-layer structure model we created is shown in Figure10, the upper layer is the abstract architecture meta-model instance we have developed, the middle layer is the corresponding robotic system meta-model instance , the lower layer is the actual robotic agent created with C# code and runs in MRDS. The structure maintains the real-time correspondence relationship between the robotic system meta-model instance and robot agent through socket command.

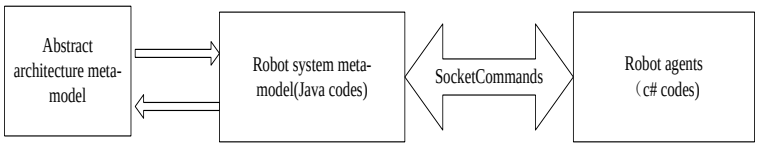

Figure 10. Socket Communication between Meta-models and Agents

4.2. Verification Results and Analysis

The actual result of online reconfiguration on runtime architecture can clearly access from the URL (http://127.0.0.1:50000) provided by MRDS. From there, we can observe all components registered and states of the robot in real-time, so we can judge the actual effect of the online reconfiguration on robotic system. In this paper, we chose the basic and representative Lego NXT robot [10] to finish the case study, NXT can be communicated with PC by Bluetooth or USB, it provides a scalable architecture that can be configured through its services and make their own or third-party device be added at any time, so it is very suitable to meet the needs of reconfiguration. NXT function block has three motor ports and four sensor ports, devices inserted can get which port they are in. To verify online reconfiguration, we can modify instances of the architecture meta-model developed in this paper and achieve switching device registered between different ports. The case study is a simple example, in which we create an architecture and robotic system meta-model that mapping to the simulation robot in MRDS, then we can reconfigure the architecture meta-model and observe the change of the simulation in real-time. Figure 11 is an example of a robotic system meta-model we created, its suffix is **.robotsys**, this example and the simulation we have developed on MRDS is consistent , in the robotic system we create a node, the node contains both a Actuator and two Sensors, their property settings shown in Figure 12 and Figure 13:

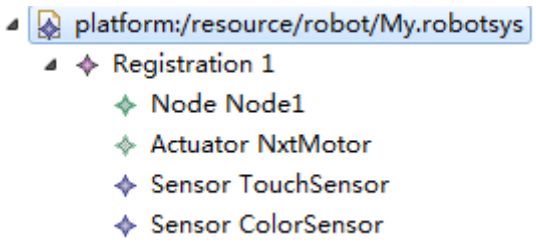

Figure 11. The Instance of Robotic Systems

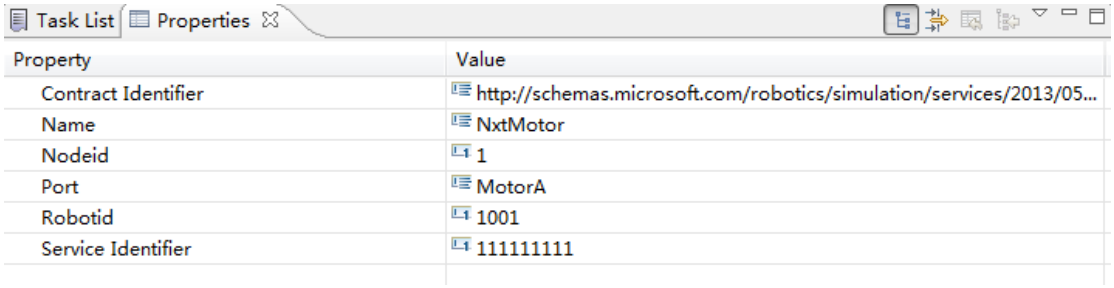

Figure 12. The Property Value of Actuator

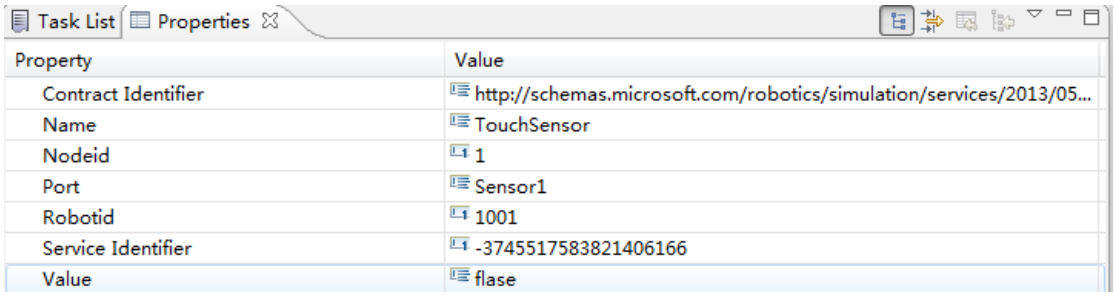

Figure 13. The Property Value of Touch Sensor

Examples of these elements in the property values are exactly the same with our robot written in C # code on the MRDS, as we can see from Figure 10 there are three components

registered in the robot, if we shield away the ColorSensor from the corresponding instance of architecture meta-model, the corresponding ColorSensor registered in the robotic system instance will accordingly disappear after the synchronization engine touches off. Meanwhile, one command to remove the ColorSensor will send to the C # code written robot proxy on MRDS through socket command, so we really canceled the registration of ColorSensor in robot agent. Experimental results in real-time can be seen by visiting MRDS's corresponding web page as mentioned above, and is shown in Figure 14, it is clear that the actual robotic system has been deregistered the ColorSensor, which show that this online reconfiguration has taken effect and intuitively prove the correctness of the above research ideas. In addition to online meet users' different needs and reconfigure properties and collaborative relationships of robot components, we can also monitor the values of each registered sensors from the abstract architecture meta-model, it is convenient for us to get some very important parameters and shield away some disturbance from the underlying complex data.

Port	Type	Model
MotorA	Actuator	NxtMotor
Sensor1	AnalogSensor	TouchSensor

Figure 14. The Actual Registered Elements of Robot Agent

5. Related Work

Along with the surrounding environment of robot more and more complex, in order to meet the adaptive performance, developing robotic software architecture for robotic management is a hot topic in the recent years, researchers' study changes from a simple robotic software architecture to robotic software architecture which has certain characteristics of adaptive, the specific research are:

Kim D *et al.*, [11] developed a robotic framework called SAHGE ,which is composed of six main elements: a situation monitor to identify internal and external conditions of a software system, ontology-based models to describe architecture and components, brokers to find appropriate architectural reconfiguration patterns and components for a situation, a reconfiguration actually change the architecture based on the selected reconfiguration pattern and components, a decision maker/learner to find the optimal solution of reconfiguring software architecture for a situation, and repositories to effectively manage and share architectural reconfiguration patterns, components, and problem solving strategies.

Michael G *et al.*, [12] applies the model-based development and code generation tool EasyLab [13] to the development of robots ,which can be converted hardware model into code ,that is executable for controllers, sensors and actuators of the robot. So it can simplify the robot development, but EasyLab tool has not yet achieve the support for distributed model and can't provide multiple robot components simultaneously in one EasyLab model, also has not yet combine with a service- oriented architecture.

Edwards G *et al.*, [14] divide the software components of the robot into several levels, each of the upper layer can obtain status information of the lower layer, then based on the action-based polices to analysis and determine whether the lower members need for change. If it needs for changing, modifying the underlying components of the organizational structure by the performing component, so as to achieve the purpose of adaptation.

Daniel Sykes *et al.*, [15]describe their combined approach for adaptable software architecture and task synthesis from high-level goals, which is based on a three-layer model. In the uppermost layer, reactive plans are generated from goals expressed in a temporal logic. The middle layer is responsible for plan execution and assembling a configuration of domain-specific software components, which reside in the lowest layer. The implementation demonstrates that the approach enables to handle non-determinism in the environment and unexpected failures in software components

These studies above focused more on the adaptive architecture studies and did not take into account the existing demand for online reconfiguration, but Hui Song *et al.*, [16] used SM@RT tool to automatically generate the synchronization engine, in order to maintain the correspondence between runtime system and architecture model at runtime. They have successfully used SM@RT tool in some actual systems by providing the corresponding elements to be managed and management API, such as JOnAS, JEE server, Jar-UML, Eclipse-GUI; and this fully illustrated SM@RT tool is actual availability and reliability to use in some other systems. In this paper, we innovatively use SM@RT tool to develop a robotic architecture model combines with runtime software architecture to solve this problem.

6. Conclusion

The target of this paper is improving the ability of the robotic software online reconfiguration, and we proposed to lead runtime software architecture into the robotic software development and established a robotic software architecture model, which supports online reconfiguration. By using the sophisticated architecture-based technology, we abstract the underlying complex robotic system into a platform-independent architecture meta-model, and allow developers to focus only relatively simple architecture meta-model instance to reconfigure robotic system at runtime. To support this idea, we designed the runtime architecture model, robotic system meta-model and the transformation rules between them. Then we expand the SM@RT tool to solve the problem of synchronization runtime architecture models and system models for bidirectional transformation. What's more, the elaborated robotic software systems development methods have successfully combined with MRDS, and the experimental results show the feasibility and effectiveness of this paper's study work, it is effectively improve robot software online management and reconfiguration.

Although the examples in this paper use only the NXT Robot, but due to the design of low coupling and the architecture meta-model is platform-independent , runtime architecture meta-model in this paper can also be used to map different types of robots through different synchronous generators, and the corresponding meta-model elements and appropriate management API , so this study can simplify the development of different types of robot development process , and reduce the difficulty of development . Of course, the study of this paper is still in its infancy, architecture meta-model and robotic system meta-model differences and merger now is running under the situation that one stays unchanged; it can't be simultaneously bidirectional transformation. We will further study security issues involving the collision detection and successfully conversion in simultaneously bidirectional transformation.

Acknowledgements

This work was supported by National Natural Science Foundation of China (Grant No. 61202050/61379036), Natural Science Foundation of Zhejiang Province of China (Grant No. Y13F020175), the New-shoot Talent Program of Zhejiang Province(Grant No. 2014R406073), High-Skilled Talent Project of Zhejiang Province(Grant No. 2013R30019), Zhejiang Talent Project (Grant No. 2013R10005), and 521 talent project of ZSTU.

References

[1] R. N. Taylor, N. Medvidovi and E. M. Dashofy, "Software architecture: foundations, theory, and practice", ACM, **(2010)**; NY, USA.
[2] D. Sykes, W. Heaven, J. Magee and J. Kramer, "From goals to components: a combined approach to self-management", Proceedings of SEAMS, **(2008)**; New York, USA.
[3] Y. Xiong, H. Song, Z. J. Hu and M. Takeichi, "Synchronizing concurrent model updates based on bidirectional transformation", Software & Systems Modeling, vol. 1, no. 12, **(2013)**.
[4] P. Stevens, "In Model Driven Engineering Languages and Systems", Bidirectional model transformations in QVT: semantic issues and open questions, Springer Berlin Heidelberg, vol. 9, **(2010),** pp.7-20.
[5] K. Czarnecki, J. N. Foster, Z. J. Hu, R. Lämmel, A. Schürr and J. F. Terwilliger, "Bidirectional Transformations: A Cross-Discipline Perspective", Proceedings of Second International Conference, **(2009)**; Zurich, Switzerland.

[6] A. Marcus and P. Ivan, "A meta-modeling language supporting subset and union properties", Software & Systems Modeling, vol. 1, no. 7, **(2008).**
[7] D. Steinberg, S. Zhan and L. Deng, "Eclipse modeling framework 2.0. Tsinghua University Press", **(2010)**; Beijing.
[8] H. Song, G. Huang, F. Chauvel, Y. F. Xiong, Z. J. Hu, Y.C.Sun and H. Mei, "Supporting runtime software architecture: A bidirectional transformation-based approach", The Journal of Systems & Software, vol. 5, no. 84, **(2011)**.
[9] S. C. Kang，W. T. Chang，K. Y. Gu and H. Chi, "Robot development using Microsoft Robotics Developer Studio,Tayer & Francis Group", **(2011)**; USA.
[10] C. A. Mancipe, J. Velasquez and W. I. Moreno, "Design and implementation of a system for travel in form of convoy of mobile platforms Lego Mindstorms NXT 2.0", Proceedings of Circuits and Systems, **(2012)**; Barranquilla, Colombia**.**
[11] D. Kim, S. Park, Y. Jin, H. Chang, Y. S. Park, I. Y. Ko, K. Lee, Y. C. Park and S. Lee, "SHAGE: a framework for self- managed robot software", Proceedings of the international workshop on Self-adaptation and self-managing systems table of contents,**(2006)**; Shanghai, China**.**
[12] T. K. Ger, F. M. Wahl, M. Geisinger, "A Software Architecture for Model-Based Programming of Robot Systems", In Advances in Robotics Research", Springer Berlin Heidelberg, Berlin **(2009)**, pp.135-146**.**
[13] S. Barner, M. Geisinger and C. Buckl, "EasyLab: Model-Based Development of Software for Mechatronic Systems", Proceedings of Mechtronic and Embedded Systems and Applications, **(2008)**; Beijing, China**.**
[14] G. Edwards, J. Garcia and H. Tajalli, "Architecture-driven self-adaptation and self-management in robotics systems", Proceedings of Software Engineering for Self-Adaptive Systems (SEAMS), **(2009)**; Vancouver, BC.
[15] D. Sykes, W. Heaven and J. Magee, "From goals to components: a combined approach to self-management", Proceedings of the international workshop on Software engineering for adaptive and self-managing systems, **(2008)**; Leipzig, Germany.
[16] H. Song, Y. F. Xiong, C. Franck, H. Gang, Z. J. Hu and H. Mei, "Generating synchronization engines between running systems and their model-based views", In Models in Software Engineering, Springer-Verlag, Berlin**(2009)**,pp.140-154**.**

