Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Applying the model driven generative domain engineering method to develop self-organizing architectural solutions for mobile robot

LIANG Hai-hua†, ZHU Miao-liang

(*School of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China*)

†E-mail: lhhwanmit@hotmail.com

Received Sept. 26, 2005; revision accepted May 22, 2006

Abstract: Model driven generative domain engineering (MDGDE) is a domain engineering method aiming to develop optimized, reusable architectures, components and aspects for application engineering. Agents are regarded in MDGDE as special objects having more autonomy, and taking more initiative. Design of the agent involves three levels of activities: logical analysis and design, physical analysis, physical design. This classification corresponds to domain analysis and design, application analysis, and application design. Agent is an important analysis and design tool for MDGDE because it facilitates development of complex distributed system—the mobile robot. According to MDGDE, we designed a distributed communication middleware and a set of event-driven agents, which enables the robot to initiate actions adaptively to the dynamical changes in the environment. This paper describes our approach as well as its motivations and our practice.

Key words: Domain engineering, Agent oriented software engineering, Mobile robot
doi:10.1631/jzus.2006.A1652 **Document code:** A **CLC number:** TP311.5

INTRODUCTION

Model driven generative domain engineering (MDGDE) is improved on the basis of generative programming (Czarnecki and Ulrich, 2000) for the purpose of benefiting from the new emerging software engineering methods, such as agent oriented software engineering (Wooldridge, 1997), model driven architecture (MDA) (Kleppe *et al*., 2003) and aspect-oriented software development (AOSD) (Filman *et al*., 2004), etc. Agents are regarded as some special objects having more autonomy, and taking more initiative than standard objects. The foundation of generative programming (GP) is domain engineering. GP is a software engineering paradigm based on modeling software families such that, given a particular requirements specification, a highly customized and optimized intermediate or end-product can be automatically manufactured on demand from elementary, reusable implementation components by means of configuration knowledge (Czarnecki and Ulrich, 2000). MDA is a framework for software development defined by the object management group (OMG), key to MDA is the importance of models in the software development process (Kleppe *et al*., 2003; Arlow and Ila, 2003). AOSD is a new technology for separation of concerns (SOC) in software development. It is widely recognized that Object-Oriented techniques (OO) supports well software reuse. We found that GP, MDA, AOSD and OO can benefit from each other. If application engineering and domain engineering are combined, they can improve and complement each other, support reuses for the whole software life recycle, and thus accelerate the development, obtaining higher quality at the same time.

Multi agent system (MAS) provides a new viewpoint for software engineering. Adopting an agent-oriented approach to software engineering means decomposing the problem into multiple, interacting, autonomous components (agents) that have particular objectives to achieve. Agents are suitable

for analyzing, designing and implementing complex, distributed systems. Although they have the potential to significantly improve current practice in software engineering and to extend the range of applications that can be feasibly tackled, although there are more and more attempts to cast agent system as a software engineering paradigm, the many agent programming languages and platforms that have appeared (Bordini *et al*., 2005) are not mature enough compared to OO technology. The key abstractions of agents are interactions and organizations. Wooldridge (1997) defined: "An agent is an encapsulated computer system that is situated in some environment and capable of flexible, autonomous action in that environment in order to meet its design objectives."

Autonomous mobile robot is a machine able to do environmental navigation on its own without direct human manipulation. The robot must be able to perceive its surroundings through different kinds of sensors and initiate appropriate actions in that environment through actuators to achieve its designed goals. The robot must also deal with the uncertainness of incomplete knowledge about its environment and about the effects of its own actions and has the ability to respond to potentially dangerous situation in real-time while maintaining enough safety and performance. Autonomous mobile robot can be viewed as a multi-agent system.

The robot architecture is a key issue in the design of a mobile robot. Architecture defines the principles involving organizing hardware and software function modules, integration methods and supporting tools (Alami *et al*., 2000). In this paper, we describe the MDGDE method, and its application to implement domain specific optimized architecture and reusable assets for mobile robots.

This paper is organized as follows. Section 2 discusses several agent modeling techniques, and what is the point of MDGDE. Section 3 presents the MDGDE method and its application. Finally, Section 4 concludes this paper.

RELATED WORKS

MaSE (DeLoach *et al*., 2001) is a general purpose multi-agent method. The point is that MaSE inherits much from object-oriented technology, supporting tracing and adapting to the change. The core concepts in MaSE are: Target, the abstract representation for the function and non-function requirements of system; Role, the actor which acknowledges some function; Task, a set of acts required for agent to do for its target; Agents, the agents class; Dialog, communication occurring between agents. MaSE makes use of following modeling activity: system target modeling, use case modeling, sequence diagram modeling, role modeling, concurrent task modeling, agent class modeling, dialog modeling, agents architecture modeling and system design modeling. Gaia, developed by Wooldridge (1997), is an agent-oriented method, which understands, describes and realizes multi-agent system from a society and organization viewpoint. The core modeling concepts in Gaia are system, role, responsibility, activity, and protocol, agent type, service and familiar. The modeling activity includes role modeling, interaction modeling, agent modeling, service modeling and familiar modeling.

Kinny *et al*.(1996) developed the first methodologies for the development of BDI agents based on OO technologies. The agent methodology distinguishes between the external viewpoint and the internal viewpoint. External view is characterized by two models: Agent Model and Interaction Model. Internally view comprises beliefs, desires and intentions, which represent respectively their informational, motivational and deliberative states. MESSAGE (methodology for engineering systems of software agents) (Caire *et al*., 2001) is a methodology built upon best practice methods in current software engineering such as for instance UML for the analysis and design of agent-based systems. MESSAGE presents five analysis models: Organization Model, Goal/Task Model, Agent/Role Model, Domain (Information) Model, and Interaction Model. Tropos (Bresciani *et al*., 2004) is another good example of an agent-oriented software development methodology based on object-oriented techniques. In particular, Tropos bases all the requirements analysis and part of the architectural design on agent and goal based notions and diagrams. The Tropos specification uses Actor and Dependency Model, Goal and Plan Models, Capability Diagram, and Agent Interaction Diagrams.

Compared with other agent-oriented methods, our MDGDE method emphasizes feature driven domain analysis, and recognizes the agents, aspects and objects as the analysis and design elements.

MDGDE AND ITS APPLICATION TO MOBILE ROBOT

Overview

Agent as a kind of special object in MDGDE owns more autonomy, and takes more initiative. At the logical level, agent is equipped with mental states, involving concepts like knowledge, belief, desires and goals, in order to display autonomy and in particular pro-active behavior. At the physical level, agent as subsystem is equipped with common features and variation features. Agents are defined with objects, aspects and functions. So the definition of agents in MDGDE involves three levels: logical (corresponding to Domain Specific Language, meta-domain modeling), physical high level (corresponding to domain modeling) and physical low level (corresponding to application modeling).

Software is the knowledge storage medium, while the software development process can be viewed as knowledge acquisition activity (Philip, 2003). The process of the MDGDE is the process of knowledge capturing and reusing. The development process of MDGDE is divided into 3 phases: domain analysis, domain design and domain implementation, as shown in Fig.1. Each of the phases can be iterated.

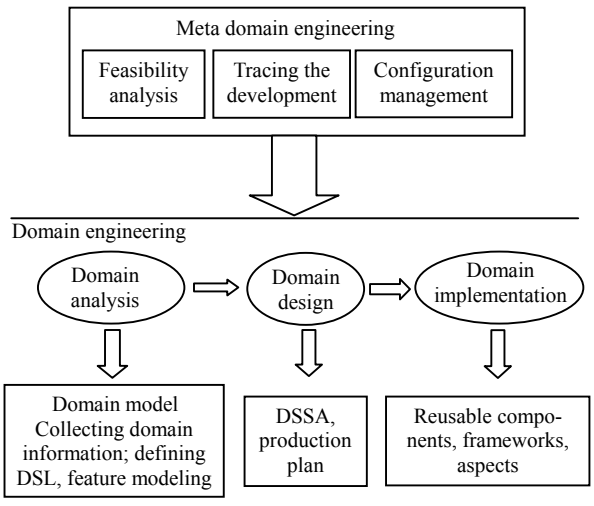

Fig.1 The main phase and product of the MDGDE development procedure

Domain analysis

The purpose of domain analysis in MDGDE is: choosing and scoping the domain under study; collecting domain information, then unifying them into a consistent domain model. The sources of domain information include existing systems, domain experts, user manuals, prototypes, and other requirements sources. Domain analysis in MDGDE comprises two main steps: domain scoping and domain modeling. The main jobs of domain modeling are analyzing the commonality, variability and dependency, consequently making the definition and description of problem space.

Domain metamodel and domain model are the core products of domain analysis. We define the domain metamodel as the reference model of one set of applications which share similar functions, activities and structures; domain metamodel is composed of the language used to define the domain model. So the domain metamodel makes constraints to domain model; the module, process and subsystem in domain model can be mapped to the module, process and subsystem in domain design.

Domain scoping defines the domain of interest, stakeholder and the target of stakeholders. Domain scoping may be affected by many factors such as maturity of domain, usable resource, etc.

Domain scope can be defined using existing applications or counterexamples in the domain. For example, this domain includes the software solution for a self-organized robot, not for a group of robots.

Building domain metamodel involves six steps:

(1) Identify domain's stakeholders, and their viewpoints and requirements;

(2) Make list of features and high level requirements; building the feature model;

(3) Identify the system objective and main business process and activity;

(4) Identify the context of system, divide the system into subsystem and module; identify the shared features between modules; the shared features would be modeled as aspects;

(5) Make object-oriented analysis on the results; identify use-case and class;

(6) Design DSL; describe metamodel using DSL.

Steps (1) and (2) belong to domain scoping, while the other steps belong to domain modeling. UML is usually not sufficient for describing a domain. So a set of UML profiles are defined as extension to UML. We use the UML profile as the DSL to describe problem space.

Type schema and method schema represent a collection of types and functions, which is used primarily to specify the scope of advices, which are inspired by Yao *et al*.(2005).

Also, there are many constraints for agents which need monitoring in autonomous mobile robot. So we need many interceptors to monitor the constraints. Aspect is a good choice for implementing these interceptors.

Steps (1) and (2) belong to behavior modeling; while Steps (3) and (4) belong to functional modeling. The agents are regarded as stakeholders in mobile Robot domain.

1. Agent modeling

Agent can be regarded as a unique problem solving unit which has its own knowledge model and input/output capacity. Agent can perceive its environment via sensors and act upon its environment through actuators. Each agent comprises 3 parts: kernel, IO ports and switch. The kernel contains specific function library or knowledge base. Architecture of the agent is shown in Fig.2. IO ports enable agent to communicate with other agents. Switch enables agents to make decision based on system state, choose function and IO ports, evaluate the input data, and output the result as event to Bulletinboard. In our multi-agent based architecture, the robot carries out its missions through the internal agents' communication and cooperation.

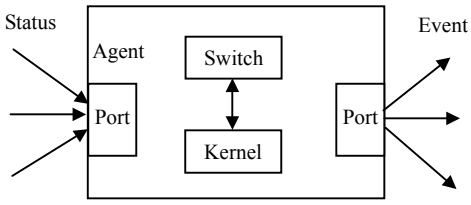

Fig.2 Architecture of the agent

An agent has three states: running, sleeping, and waiting. When an agent is running, it gets system status and processing data information, reports events; when it is sleeping, it only receives system status and awakes on demand; when agent is waiting, it receives status and reports events, but the data outputs are suppressed, so it acts as monitor. For example, when system state is straight road, the Fusion agent will sleep.

(1) Agents definition and domain meta-modeling

According to their responsibility, agents are classified as follows:

Sensory agents get the environment information as input, and output description of the environment to the Fusion agent. They include:

RFCCD (road following CCD): uses both long and short-range CCD cameras to detect the white line of structured road and edges of semi-structured road. Then it provides the extracted data to the Fusion agent.

STCCD (stereo CCD): uses stereo CCD cameras to detect obstacles on road and impassable area off-road. Then it provides the circumscribed polygon of the obstacle or impassable area to the Fusion agent.

RADAR: uses RADAR to detect positive obstacles on road or off-road. Then it provides the circumscribed polygon and height of the obstacle to the Fusion agent.

Actuator agents take action to achieve the robot's objective:

Drive: generates locomotion and steering commands to the robot based on the path given by LPL.

Outputs of the Sensory agents may be mapped to actuator directly through the neural network.

Processing agents process the information from other agents:

Fusion: integrates information from the Sensory agents. Then it gives a unique description of the environment and provides this information to LPL agent.

GPL (global path planning): based on a priori known low-resolution map database, plans a global path. If the current planned path is found infeasible, the global path may need re-planning.

LPL (local path planning): according to the global path (from GPL) and the current description of environment (from Fusion, or directly from Sensory agents when the environment is simple), plans a smoothing trajectory for the robot to follow.

Bulletinboard: based on the reasoning rules and events from the Sensory agents, Actuator agents and Processing agents, deduces the current state of the environment and the mobile robot.

Localization: using GPS/INS and odometer to provide real-time high accuracy position (x, y, z) and stance (roll, pitch, yaw), velocity and acceleration of the robot.

Human-machine interface and assistant agents

provide some extra control capability:

Supervisor: initializes a mission, monitors current environment obtained by sensors in real-time, the position and internal status of the robot. It can stop the robot or awake another agent under emergent situation that the robot is dangerous or situation that the robot will get stuck.

TeleControl: takes control of the robot seamlessly from the robot's on-board computers. When the robot leaves the area beyond its on-board system's capabilities, TeleControl will give the control back to the on-board system seamlessly.

Recorder: logs the operating data of each agent that are valuable for off-line analysis into a database.

Mapper: maintains and updates the priori global maps used by the GPL and Fusion agents.

(2) Bulletinboard agent in detail

The Bulletinboard agent is the core of mobile robot solution. The main functions of the Bulletinboard agent are: (i) to receive events from agents and evaluates; (ii) to deduce the system status; (iii) to broadcast the status to agents through ROBIX in order to control and organize the agents. The internal agents are dynamically changing their working status and links to other agents based on the system status obtained from the Bulletinboard agent, so that the mobile robot achieves self-organized control (Zhu *et al*., 2000). Thus, the robot architecture is dynamically reorganized to adapt to the dynamic environments.

As a high level monitor, Bulletinboard collects events and evaluates the current system status. The current system status is the key for activity and cooperation between agents. So the evolutions must be accurate and fast enough. The internal structure of the Bulletinboard agent is composed of a real-time reasoning system based on DES (discrete event system) model and events Monitor, as shown in Fig.3. The reasoning system includes a Rules base, a Facts base, and an Inference machine based on improved Rete algorithm. The Inference machine is fast enough. The events Monitor agent monitors the events from the Sensory agents, Processing agents, and Actuator agents, etc. and puts them into the Facts base. Once new facts are monitored, the Inference machine will be activated to deduce the new system status based on the rules in the Rule base. As soon as the new system status is available, events Monitor broadcasts it to all the agents in the system. After the agents receive this status, they decide what to do, based on the internal looking-up table or FSM. Thus, agents will be reorganized; robot can make correct actions to adapt to the changes of the environment.

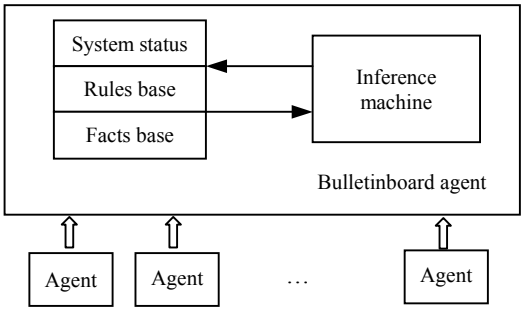

Fig.3 The Bulletinboard agent

2. Feature modeling

We define three kinds of features in MDGDE:

Aspect feature: this kind of feature captures crosscutting concerns, we implement aspect features using AspectC++ (Lohmann *et al*., 2004; Gal *et al*., 2001). Log is an example of aspect feature.

Concrete feature: this kind of feature will be mapped to an implement component. For example, algorithm feature of Fusion agent will be mapped to some kind of concrete function library.

Abstract feature: this kind of feature has no direct implementation. It will be implemented by combination of components and aspects. Reliability is an abstract feature.

3. Domain model

We design the domain model based on the agent modeling, as shown in Fig.4. This model organizes

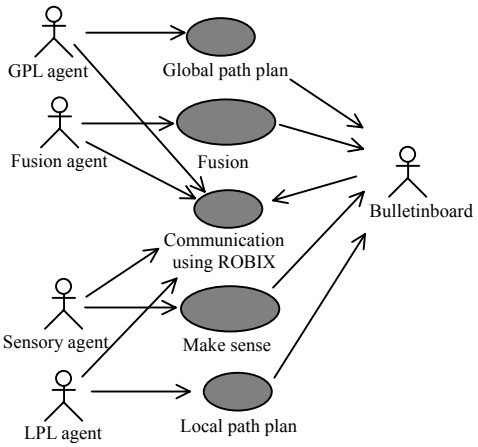

Fig.4 Part of the Domain model of mobile robot

domain knowledge and provides a language for the team. Each object in the model has a clear meaning (please refer to the former sections).

Domain design

The purpose of domain design in MDGDE is to develop one or one set of domain specific architecture for applications in the domain, including the guideline for how to apply the architectures; develop a production plan, describe the interfaces, process of the components development and how to deal with the changing requirements, and how to configure the development; and map from PIM to PSM, design components to implement common feature and variable feature for integrating AOSD, OOD and component-oriented design, as shown in Fig.5.

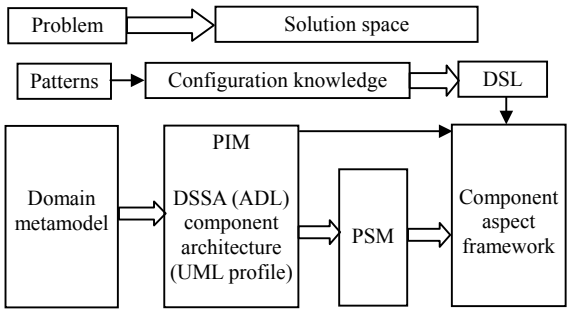

Fig.5 Solutions space and problem space in MDGDE

The purpose of designing domain specific software architecture (DSSA) is to design one or one set of architecture satisfying the requirement of all stakeholders (Evans, 2003), involving dividing system into components and interaction between components (Duffy, 2004). Also the rules on how to apply or select architecture are defined.

The input of designing DSSA includes domain metamodel (module, process, aspect and non-functional requirement) and the forecast on future changing. The design of architecture may use references on the design patterns (Gamma *et al*., 1995). Output product is the system architecture which is represented as UML class diagram and usecase diagram.

Designing DSSA involves 3 steps:

(1) Identify modules. Modules include functional modules, aspect modules and adapter modules. Functional modules encapsulate separated function. Aspect modules define features crosscutting several modules (Filman *et al*., 2004), such as log, synchronized, and message passing features in mobile robot system. Adapter modules which encapsulated interaction protocols are used to connect functional modules seamlessly.

(2) Define the relations between modules. There are three types of relations: is-part-of, depends-on and is.

(3) Choose architecture style. The definition of architecture style includes: A set of module types; Relations between the modules, represented as topological layout; A set of semantics constraints; A set of interaction mechanisms (such as subroutine calling, blackboard) which determine how the modules go with each other.

Architecture style enables the perfect idea on architecture design to be preserved and communicated. We can find many architecture style definitions and the rules on how to choose an appropriate style in (Bass *et al*., 2000).

Design of common features includes design of universal component, class, and aspect. Because common features may evolve too, emphases of common features design are hierarchy and interfaces of components which should satisfy the Demeter and open-closed principle. We found mixin (Smaragdakis and Batory, 2002) very useful in our practice of designing hierarchy of classes. In our case, the agent communication class is designed utilizing mixin. The benefit is that we do not need to define a communication class for each agent type and reduce the cost of dynamic binding.

The main jobs of design of variable features are designing binding sites and binding pattern. Binding sites determines where, when, who can bind a feature. Binding pattern comprises static binding which occurs in compile time and dynamic binding which occurs during run time.

The PIM (platform independent model) in MDGDE comprises architecture and platform independent part of component design. Mapping from PIM to PSM can be accomplished manually, or through some tools and some rules. Eclipse+BasicMTL of INRIA are on of such choices.

1. The domain architecture of the mobile robot

The domain architecture defines the domain specific state, the domain specific action and the

domain specific structure. There exist three main classes of control architectures developed for mobile robots. One is based on functional decomposition; the other is based on behavioral decomposition (Rosenblatt and Payton, 1989; Sowmya, 1992; Watanabe *et al*., 1992); and the third is the hybrid architecture that combines the former two (Pons *et al*., 1993; Ollero *et al*., 1994; Low *et al*., 2002).

For the mobile robot, we defined a layered architecture, as shown in Fig.6 (Evans, 2003; Duffy, 2004). The lower layer provides service to the upper ones. ROBIX as the fundamental communication infrastructure will be discussed in detail later.

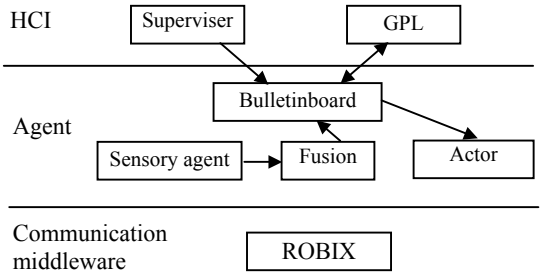

Fig.6 The layered architecture adopted in the mobile robot

2. The self-organized architecture of the mobile robot

Based on the result of domain analysis and domain model, we defined a self-organized architecture. A robot reorganizes different sets of agents to achieve its goal. For example, when robot state is RS{road straight}, the robot will organize seven agents to achieve the task: the system state Bulletinboard agent gets is "road straight"; RFCCD, RADAR, STCCD, LPL, Drive and Location agents will be organized to accomplish the move of the robot. Whenever RADAR agent reports there are suspected obstacles again to Bulletinboard; Bulletinboard agent deduces the new system status RO{road driving with obstacle}, then broadcasts this new status; some agents get awakened, such as Fusion, it requests outputs from RFCCD and RADAR to make a unique description of the environments and sends processing results to LPL; some agent will change their interior function, such as LPL will change to compute the path of the steering obstacle; thus relatively safe driving is achieved. These two processes are depicted in Fig.7a and Fig.7b, respectively.

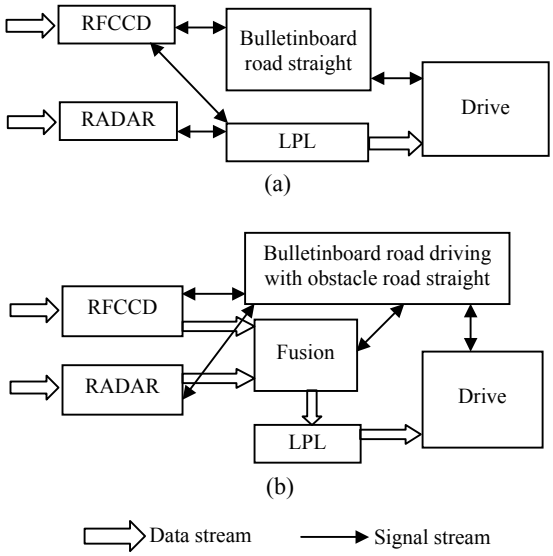

Fig.7 Architectures of status RS (a) and RO (b)

All messages in the system are classified into 3 types. One type of message is the description information on the images, maps, paths, etc., which is transferred between agents, and successively processed by agents. The other is event from the evaluation of an agent to its processing results, which is coded and reported to the Bulletinboard agent. The third is the current system status, which is evaluated by Bulletinboard and broadcasted to all other agents.

There exist two different information/control streams in the system, one is data stream, and another is signal stream. The data steam consists of object data, forming a loop of sense-plan-act, while the signal stream consists of event and current status, forming state-decision-action circuit. The latter is a supervising circuit to the former including activation and organization. Robots will organize agents into different topological structure to adapt to different environments. The adaptive capacity of the robot is improved notably by separating data stream and signal stream.

3. ROBIX—the communication middleware

Agents in our mobile robot are distributed on different computers running different operating systems, from Microsoft Windows 2000, Red Hat Linux 9.0 and VxWorks. So a communication middleware is needed to enable agents to make transparent communications. We have designed a real-time communication middleware named ROBIX to achieve this

goal. ROBIX provides a unified communication environment enabling agents running on different operating systems to communicate and share resource independent of location, hardware and OS in a united way (Fig.8).

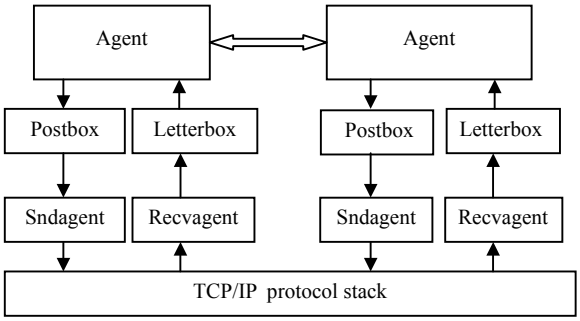

Fig.8 The agents communicates through ROBIX

ROBIX is based on a postoffice model, comprised of two processes running on two separate computers. One process is Sndagent which sends messages; another process is Recvagent which receives messages. Letterbox and Postbox are applications of message queuing pattern and implemented with sharememory. We use a Mutex object to maintain the access to the Letterbox or Postbox (Fig.9). Agents which want to send message do not need to know the location of the target agent, they just put their messages into the Postbox and create an event named "Letter2SndXXXX", where "XXXX" is the name of the agent sending message.

When "Letter2SndXXXX" is monitored, Sndagent collects the message from the "XXXX" agent's postbox and packet the message. If the message type is broadcast message, it will broadcast this message to all agents; otherwise, it checks the name of the destination agents, and then fills in the IP address of the target agent. IP address and socket information of agents are stored in the process table (Table 1), and the host table (Table 2), respectively. Sndagent first searches its own cache for these two tables, if not found, it will request a copy from Supervisor agent. When ready, if the target agent is running on the same host, Sndagent will copy the message into the target agent's Letterbox directly, and create an event named "LetterRcvXXXX", where "XXXX" is the name of the target agent. Otherwise, Sndagent sends the message to the receiver agent through the socket. Recvagent will pick up the message and copy it into the Letterbox of the target agent. If the message type is urgent, Recvagent can interrupt the receiver agent to ask him to handle the message immediately.

Table 1 Process table

Indicative bit	Process name	Host IP	PID
0	Fusion	192.168.0.9	1288
1	…	…	…

Table 2 Host table

Indicative bit	Host name	Host IP	Socket	Local
1	AMR0	192.168.0.3	4	1
2	…	…		…

Domain implementation and test result

The main jobs of domain implementation include designing the detail of the components, frameworks and aspects, implementing the architectures, component and aspect as what the production plan prescribed and developing configuration DSL (Consel *et al*., 2005). Those reusable architectures, components, frameworks and aspects form the solution space. In our case, the configuration DSL is implemented using C++ template.

The process of domain engineering is a domain too. We define the meta domain engineering in MDGDE as follows: analyze the feasibility, plan domain engineering, trace the domain engineering process, implement process improvement and domain optimizing, software configuration management and collect/analyze the data on domain engineering process (Fig.1). Again, this phase may also be iterated.

We have tested the architecture solution in this paper through simulation and making outdoor field experiments for both on road and off road navigation under different weather conditions. Experimental results showed that the self-organized architecture solution enables the mobile robot to move on structured roads with clear marks, semi-structured roads paved with gravels and moderate undulation cross-country roads with sparse vegetation. The system architecture is dynamically reorganized according to the changes in the environment.

The experimental results also showed that the self-organized architecture solution could make dy-

namic reorganization according to the dynamically changing environment, and inherited the merit of both behavior-based architecture and functional decomposition based architecture. The solution satisfied our expectation about intelligence and real time response.

Through MDGDE methods, we developed reusable assets that accelerated our development process and improved the reliability of the system. We list the duration of each phase in MDGDE and classic development process in Table 3.

Table 3 Duration of each phase in MDGDE and the classic development processes

	Duration (d)	
	MDGDE	Classic
Analysis	10	7
Design	5	3
Implementation	10	20
Test	1	15

DISCUSSION AND CONCLUSION

MDGDE is based on GP, UML profile, Agent-oriented technology and AOSD, combined with design patterns and OO methods, provide good support to software reuse. And in this paper, we developed a self-organized multi-agent based architecture solution for outdoor mobile robot and obtained satisfactory results. Domain engineering provided reusable DSSA, component, framework and aspects, also including other products, such as document, design principle. That is a kind of knowledge reuse.

We found MDGDE to be improved in several aspects. MDGDE should provide more support for knowledge capture and management. This is meaningful for development. Some developers abandoned the development; so we added new members now and then. The new members bring pressure of communication. If we have good knowledge of the sharing mechanism, this kind of pressure will be minimized. On the other hand, MDGDE should support DSL development better. We have an idea on this point. We conceive a virtual machine which could run the DSL code. The product resulted from MDGDE will be the DSL program. The program could be run on the virtual machine. This will advance the development efficiency dramatically.

References

Alami, R., Herrb, M., Morisset, B., Chatila, R., Ingrand, F., Moutarlier, P., Fleury, S., Khatib, M., Simeon, T., 2000. Around the Lab in 40 Days [Indoor Robot Navigation]. Proceedings of IEEE International Conference on Robotics and Automation. San Francisco, CA, **3**:88-94.

Arlow, J., Ila, N., 2003. Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML. Addison-Wesley, p.25-30.

Bass, L., Paul, C., Rick, K., 2000. Software Architecture in Practice (2nd Ed.). Addison-Wesley, p.17-20.

Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A., 2005. Multi-agent Programming: Languages, Platforms and Applications (1st Ed.). Springer, p.89-102.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J., 2004. TROPOS: an agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, **8**(3):203-236. [doi:10.1023/B:AGNT.0000018806.20944.ef]

Caire, G., Coulier, W., Garijo, F.J., Gomez, J., Pavón, J., Leal, F., Chainho, P., Kearney, P.E., Stark, J., Evans, R., *et al*., 2001. Agent Oriented Analysis Using MESSAGE/UML. Proceedings AOSE 2001. Springer, p.119-135.

Consel, C., Latry, F., Réveillère, L., Cointe, P., 2005. A Generative Programming Approach to Developing DSL Compilers. Proceedings of the Generative Programming and Component Engineering 2005, p.29-46.

Czarnecki, K., Ulrich, W.E., 2000. Generative Programming: Methods, Tools, and Applplication. Addison-Wesley, p.44-78.

DeLoach, S.A., Wood, M.F., Sparkman, C.H., 2001. Multiagent systems engineering. *The International Journal of Software Engineering and Knowledge Engineering*, **11**(3):231-258. [doi:10.1142/S0218194001000542]

Duffy, D.J., 2004. Domain Architectures: Models and Architectures for UML Applications. John Wiley & Sons, p.120-135.

Evans, E., 2003. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, p.77-102.

Filman, R., Elrad, T., Clarke, S., Aksit, M., 2004. Aspect-oriented Software Development. Addison Wesley Professional, p.133-135.

Gal, A., Wolfgang, S.P., Spinczyk, O., 2001. AspectC++: Language Proposal and Prototype Implementation. OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-oriented Systems. Tampa, Florida, p.20-36.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley, p.136-139.

Kinny, D., Georgeff, M., Rao, A., 1996. A Methodology and Modeling Technique for Systems of BDI Agents. Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-agent World (MAAMAW 96). LNAI 1038, Springer.

Kleppe, A., Warmer, J., Bast, W., 2003. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison Wesley, p.33-45.

Lohmann, D., Blaschke, G., Spinczyk, O., 2004. Generic Advice: On the Combination of AOP with Generative Programming in AspectC++. Proceedings of GPCE'04, p.55-74.

Low, K.H., Leow, W.K., Ang, Jr M.H., 2002. A Hybrid Mobile Robot Architecture with Integrated Planning and Control. Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems. Bologna, Italy, p.219-226. [doi:10.1145/544741.544797]

Ollero, A., Mandow, A., Munoz, V., Gomez de Gabriel, J., 1994. Control architecture for mobile robot operation and navigation. *Robotics and Computer-Integrated Manufacturing*, **11**(4):259-269. [doi:10.1016/0736-5845(95)00032-1]

Philip, G., 2003. The Laws of Software Process: A New Model for the Production and Management of Software. Auerbach Publications, p.136-148.

Pons, N., Delaplace, S., Rabit, J., 1993. Mobile Robot Architecture Dedicated to Asynchronous Events Management. Proceedings of the 8th International Conference on Applications of Artificial Intelligence in Engineering. Toulouse, France, **2**:547-560.

Rosenblatt, J.K., Payton, D.W., 1989. Fine-grained Alternative to the Subsumption Architecture for Mobile Robot Control. Proceedings of IJCNN International Joint Conference on Neural Networks. Washington DC, p.317-323.

Smaragdakis, Y., Batory, D., 2002. Mixin layers: an object-oriented implementation technique for refinements and collaboration-based designs. *ACM Transactions on Software Engineering and Methodology*. **11**(2):215-255. [doi:10.1145/505145.505148]

Sowmya, A., 1992. Real-time Reactive Model for Mobile Robot Architecture. Proceedings of SPIE Conference on Applications of Artificial Intelligence X: Machine Vision and Robotics. Orlando, FL, **1708**:713-721.

Watanabe, M., Onoguchi, K., Kweon, I., Kuno, Y., 1992. Architecture of Behavior-based Mobile Robot in Dynamic Environment. Proceedings of IEEE International Conference on Robotics and Automation. Nice, France, **3**:2711-2718. [doi:10.1109/ROBOT.1992.219996]

Wooldridge, M., 1997. Agent-based software engineering. *IEEE Proc. on Software Engineering*, **144**(1):26-37. [doi:10.1049/ip-sen:19971026]

Yao, Z., Zheng, Q.L., Chen, G.L., 2005. AOP++: A Generic Aspect-oriented Programming Framework in C++. GPCE 2005, p.94-108.

Zhu, M.L., Zhang, X., Wang, X., Tang, W.J., 2000. Computer integration system of autonomous intelligent robot self-organization structure IRASO. *Pattern Recognition and Artificial Intelligence*, **13**(1):36-41 (in Chinese).