SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2006; **36**:1711–1733
Published online 13 July 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.742

Mobile-C: a mobile agent platform for mobile C/C++ agents

Bo Chen[1], Harry H. Cheng[1,*,†] and Joe Palen[2]

[1]*Integration Engineering Laboratory, Department of Mechanical and Aeronautical Engineering, University of California, Davis, CA 95616, U.S.A.*
[2]*Office of New Technology and Research, California Department of Transportation, Box 942873, MS83, Sacramento, CA 94273, U.S.A.*

SUMMARY

This article presents the design, implementation and application of Mobile-C, an IEEE Foundation for Intelligent Physical Agents (FIPA) compliant agent platform for mobile C/C++ agents. IEEE FIPA standards are a set of specifications designed to ensure the interoperation between agents in a heterogeneous network. Mobile-C conforms to the FIPA standards both at agent and platform level. Mobile-C extends FIPA standards to support mobile agents by integrating an embeddable C/C++ interpreter—Ch—into the platform as a mobile agent execution engine and defining a mobile agent mobility protocol to direct agent migration process. Agent migration in Mobile-C is achieved through FIPA agent communication language (ACL) messages encoded in XML. Using FIPA ACL messages for agent migration provides a straightforward but effective way for inter-platform agent migration in FIPA compliant agent systems as both agent communication and migration can share the same communication mechanism. Choosing scriptable C/C++ as a mobile agent language allows mobile agents easy interfacing with low-level software modules and underlying hardware. Mobile-C has been used to simulate highway traffic detection and management. The agent-based traffic detection and management system uses agent technology for real-time distributed traffic information fusion. Mobile agents in the system are used for dynamic code deployment and performing unanticipated actions. The application of agent technology shows a great potential for enhancing the interoperability, flexibility and distributed computing capabilities of intelligent transportation systems. Copyright © 2006 John Wiley & Sons, Ltd.

Received 25 July 2005; Revised 11 November 2005; Accepted 30 January 2006

KEY WORDS: agent technology; mobile agents; mobility; FIPA, C/C++ interpreter; Ch

*Correspondence to: Harry H. Cheng, Integration Engineering Laboratory, Department of Mechanical and Aeronautical Engineering, University of California, Davis, CA 95616, U.S.A.
†E-mail: hhcheng@ucdavis.edu

Contract/grant sponsor: California Department of Transportation

1. INTRODUCTION

Over the past decade, distributed middleware, such as Object Management Group's (OMGs) Common Object Request Broker Architecture (CORBA), Sun's Java Remote Method Invocation (RMI), Microsoft's Distributed Component Object Model (DCOM), SOAP, the OMG's CORBA Common Object Services, Sun's Java 2 Enterprise Edition (J2EE) and Microsoft's .NET Web services, has emerged as a set of software protocols and service layers that help to solve the problems specifically associated with heterogeneity and interoperability in distributed systems [1]. The rapid progress of distributed object computing (DOC) middleware has provided enhanced environments for managing distributed resources and the systems built on this technology have gained wide applications in desktop business information systems. However, the commercial off-the-shelf DOC middleware technology is not yet mature enough to cover the realm of autonomous, mobile, dynamic, time-critical and embedded distributed systems [2]. To satisfy these requirements, further research is needed to generate more flexible middleware that will be able to adapt robustly to dynamically changing requirements and environmental conditions.

Agent-based computing has emerged as a powerful technology for the development of distributed complex software systems in the 1990s [3–5]. Indeed, many researchers believe that agents represent the most important new paradigm for software development since object-oriented design [6], and the concept of intelligent agents has already found a diverse range of applications in manufacturing [7–9], process control and real-time control systems [10–12], electronic commerce [13–15], network management [16,17], transportation systems [18,19], information management [20,21], scientific computing [22,23], health care [24] and entertainment [25]. The reason for the growing success of agent technology in these areas is that the inherent distribution allows for a natural decomposition of the system into multiple agents that interact with each other to achieve a desired global goal [19]. The agent technology can significantly enhance the design and analysis of problem domains under the following three conditions [26]: (1) the problem domain is geographically distributed; (2) the subsystems exist in a dynamic environment; and (3) subsystems need to interact with each other more flexibly. Mobile agents are software components that are able to move between different execution environments [27]. Several benefits that mobile agents provide make mobile agent technology a good programming paradigm for building agile distributed systems [28]. Mobile agents can be created dynamically at runtime and dispatched to source systems to perform tasks with the most updated code and algorithms. Mobility significantly enhances the flexibility and adaptability of large scale distributed systems.

This article presents Mobile-C [29,30], an IEEE the Foundation for Intelligent Physical Agents (FIPA) [31] compliant agent platform, and its application in real-time traffic detection and management systems. FIPA is one of two international agent standards, FIPA and OMGs Mobile Agent System Interoperability Facility (MASIF) [32]. The strength of FIPA standards is that it promotes the interoperation of agents and agent systems across heterogeneous agent platforms. To support this, FIPA has been working on specifications that range from agent platform architecture to support inter-agent communication, communication languages and content languages for expressing exchanging messages, ontologies to define semantic contents of messages and interaction protocols that direct agent communication through an admissible sequence of messages between agents. FIPA has been increasingly accepted in the agent community, and the compliance with FIPA standards has been recognized as a crucial property for the interoperability of agents.

DOI: 10.1002/spe

The development of Mobile-C is primarily motivated by applications that contain low-level hardware, such as detection systems and control systems. Since most of these types of systems are written in C/C++, Mobile-C chooses C/C++ as the mobile agent language for easy interfacing with control programs and underlying hardware. In an application presented in Section 6, distributed control programs are designed to be wrapped into distributed stationary agents. Being compliant with FIPA standards allows these heterogeneous stationary agents and mobile agents in the system to interoperate with each other to accomplish complicated tasks. Since the mobility service is optional in the FIPA standards, there is no mandatory component to support mobile agents in the platform abstract architecture. Mobile-C extends FIPA standards and integrates an embeddable C/C++ interpreter—Ch [33–35]—into the agent platform as a mobile agent execution engine. An agent mobility protocol has been designed to regulate the entire migration operation. Mobile agent migration in Mobile-C is achieved through FIPA agent communication language (ACL) messages. Using FIPA ACL messages for agent migration in FIPA compliant agent systems simplifies agent platform, reduces development effort and easily achieves inter-platform migration through well-designed communication mechanisms provided in the agent platform. Messages for agent communication and migration are expressed in FIPA ACL and encoded in XML.

Mobile-C has been used to simulate highway traffic detection and management for Intelligent Transportation Systems (ITSs). Agent-based real-time traffic detection and management system is a decentralized, autonomous, interoperable and scalable system. The system includes both stationary agents and mobile agents. The stationary agents are usually located at distributed detection stations and transportation management center (TMC). The communication between agents allows detection stations to cooperate with each other to perform distributed real-time traffic information fusion, which will dramatically reduce data transmission and the response time to incidents. The system also used mobile agents to perform unanticipated actions by dynamically deploying new algorithms and code.

The remainder of the article is structured as follows. Section 2 examines several notable mobile agent systems. Section 3 introduces the system architecture of Mobile-C. Section 4 presents Mobile-C mobility facilities. Section 5 discusses the implementation issues of the main components of Mobile-C agent platform. Section 6 describes a sample application of Mobile-C for highway traffic detection and management. Sections 7 and 8 discuss and summarize the presented work, respectively.

2. RELATED WORK

An increasing interest in agent technology results in numerous mobile agent systems being developed in the past decade. This section reviews seven notable mobile agent systems: Mole, Aglets, Concordia, D'Agents, Ara, TACOMA, and JADE. The first six mobile agent systems are not compliant to agent standards, whereas JADE is a FIPA compliant agent system.

Mole [36] is one of the earliest mobile agent systems to have been developed in Java. Mole supports weak migration. Mole offers the common means of communication mechanisms, including message passing and remote method invocation. In addition, Mole provides a session-oriented communication concept. Agents that want to communicate with each other have to establish a session before the actual communication begins.

Aglets [37] is another one of the earliest Java-based mobile agent systems and has been used in some commercial applications. The Aglet system supports weak migration only, using the standard

DOI: 10.1002/spe

Java object serialization mechanism to marshal and unmarshal the state of the aglet into a stream. Aglet defines its own transport protocol, Agent Transfer Protocol, modelled on top of HTTP. It also provides access control and message protection to prevent attacks from malicious agents and server authentication to avoid agents moving to malicious hosts and blocking suspicious agents coming from malicious hosts [38].

Concordia [39] is a typical Java-based agent system, running on top of a standard Java virtual machine. The Concordia server, which consists of a set of Concordia managers, is the major building block of the system and runs on each node of a Concordia network. It provides services for agent communication, migration, security, persistence and other high-level services. Concordia provides two inter-agent communication mechanisms, asynchronous distributed events and collaboration, built on top of a standard Java remote method invocation. Concordia supports weak migration only. Within Concordia, an agent can travel to multiple hosts based on its Itinerary, which specifies multiple destinations to which an agent travels and the work for the agent to accomplish at each location.

Agent-Tcl [40] is a mobile agent system originally supporting mobile agents written in Tcl scripting language. Subsequently, the system has been extended to allow the programmer to write mobile agents in multiple mobile agent languages and renamed to D'Agents [41]. The most recent version of D'Agents supports three different programming languages, Tcl, Scheme and Java. D'Agents supports strong mobility for Tcl and Java, moving all variables and heap data with the agent. The agent's code is also shipped to the target host as part of the agent's state image. The core D'Agents supports three low-level communication mechanisms, messages, streams and events. High-level communication services, such as directory service, are provided through libraries or stationary service agents.

Ara [42,43] is a platform that supports multiple agent languages and strong migration. Agents in Ara can be written in three languages: Tcl, C/C++, and Java. Agents programmed in one of these three languages are executed within an interpreter for the language and supported by a special runtime system, the core of Ara. Ara offers synchronous communication via service points and asynchronous communication by means of a tuple space for local agent communication. The basic security in the Ara is ensured by interpreters.

TACOMA [44] (Tromsø And Cornell Mobile Agents) project is aimed to study scientific and engineering issues raised in the agent programming paradigm. A series of TACOMA prototypes have been experimented. Version 1.0 provided basic support for agents written in the Tcl language. Later, the system has been extended to support multiple languages. To support agents on small PDA devices, the TACOMA team built TACOMA LITE with a small footprint. TACOMA supports weak mobility. Agent states and initialization data are encapsulated in a data structure called briefcase. Agents in TACOMA communicate with each other through passing briefcases.

JADE (Java Agent DEvelopment Framework) [45] is a software framework for the implementation of agent systems through a middle-ware that complies with the FIPA specifications. JADE employs three methods to dispatch messages among agents. A message is delivered through Java local calls when agents are in the same container, Remote Method Invocation (RMI) calls among agents in different containers of the same platform and FIPA delivery when crossing platform boundaries. The agent mobility in JADE is achieved through Java object serialization.

Although numerous mobile agent systems have been implemented, many of them have been implemented in Java. There are several mobile agent systems supporting mobile C/C++ agents, including Ara and TACOMA described above. These two systems are not compliant to agent standards. In Ara, mobile agents written in C/C++ are interpreted by its customized C/C++ interpreter—Mobile

DOI: 10.1002/spe

Agent Computing Environment (MACE) [42,46]. MACE interprets customized MACE byte code in a way similar to Java interpreting Java byte code. MACE byte code is generated in two steps. First, a GUN C front end generates Register Transfer Language (RTL) code from the source code of an agent in a format for the Motorola MC680X0 microprocessor. Next, a MACE compiler transforms RTL code into MACE byte code. Since MACE byte code is about five times longer than the equivalent Motorola MC680X0 native code [42], the Ara core compresses the agent code during migration. In TACOMA, mobile agent written in C is not interpreted. Instead, mobile agent source code sent to remote hosts is compiled by different C compilers in the remote hosts and the resulting binary code is then executed by vm_bin virtual machines [47].

Nine implementations in ten publicly available FIPA-compliant agent platforms provided at [48] have been implemented in Java. The remaining one is implemented by April programming language and the InterAgent Communication System. The mobile agent system Mobile-C presented in this article is written in C with a small footprint. Mobile-C uses an embeddable C/C++ interpreter—Ch [33–35]—to support the execution of mobile agent C/C++ source code. The interface between the mobile agent platform and mobile agents can be easily achieved through Ch Software Development Kit (SDK) and Embedded Ch. Mobile-C is compliant to the FIPA standards and the inter-agent communication and inter-platform agent migration are based on FIPA ACL messages encoded in XML.

3. THE SYSTEM ARCHITECTURE OF MOBILE-C

The system architecture of Mobile-C is shown in Figure 1. Agencies are the major building blocks of the system and are installed in each node of Mobile-C. They are the actual runtime environment for stationary agents (SA) and mobile agents (MA). They also serve as 'home bases' for locating and messaging mobile and detached agents, collecting knowledge about a group of agents and providing an environment in which a mobile agent executes [49]. The core of an agency is the agent platform, which provides local services for agents and proxies to access remote agencies. An agent platform represents the minimal functionality required by an agency in order to support the execution of agents. The main functionalities of an agent platform can be summarized as follows.

- Agent Management System (AMS): AMS manages the life cycle of agents. It controls the creation, registration, retirement, migration and persistence of agents. AMS maintains a directory of Agent Identifiers (AID), which contains transport addresses (amongst other things) for registered agents. Each agent must register with an AMS in order to get a valid AID.
- Agent Communication Channel (ACC): ACC routes messages between local and remote entities, realizing messages using FIPA ACL. This service is responsible for all of the remote interactions that take place between the distributed components, such as inter-agent communication and inter-platform agent migration. All of the interactions can be performed via ACL message exchange.
- Agent Security Manager (ASM): ASM is responsible for maintaining security policies for the platform and infrastructure, such as communication and transport-level security.
- Directory Facilitator (DF): DF serves yellow page services. Agents in the system can register their services with DF for providing to the community. They can also look up required services with DF.

DOI: 10.1002/spe

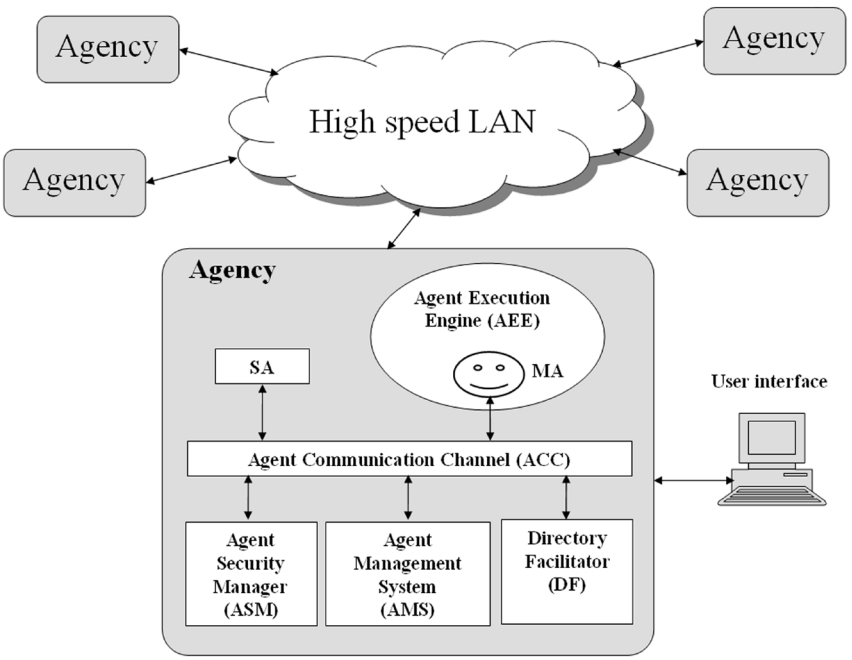

Figure 1. The system architecture of Mobile-C.

- Agent Execution Engine (AEE): AEE serves as the execution environment for the mobile agents. Mobile agents must reside inside an engine to execute. AEE has to be platform independent in order to support a mobile agent executing in a heterogeneous network.

4. SYSTEM SUPPORT FOR MOBILE AGENTS IN MOBILE-C

The mobility service is optional in the FIPA standards. There are no mandatory components to support mobile agents in the agent platform abstract architecture. One of the FIPA specifications, which specifies the minimum requirements and technologies to allow agents to take advantage of mobility, is in 'Deprecated' status and may be obsolete after a grace period. To support mobile agents, Mobile-C has extended FIPA standards that we will describe in the remainder of this section.

4.1. Mobile agent execution engine

Running foreign code on a host introduces numerous new requirements on a runtime environment. The most obvious requirements are the portability and security of mobile code execution [41].

DOI: 10.1002/spe

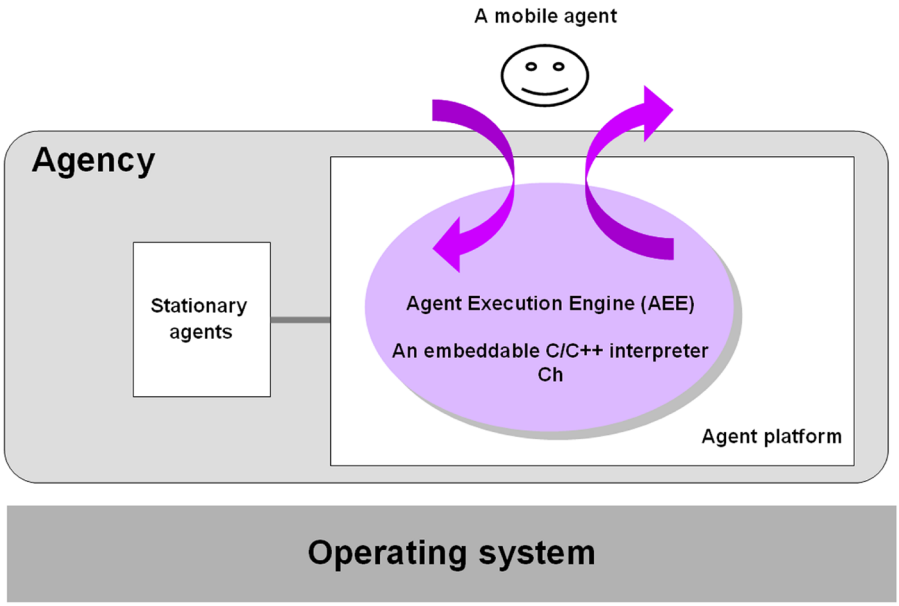

Figure 2. System support for mobile agents.

A common solution for these requirements is executing mobile agents in a virtual environment, an interpreter or a virtual machine. This approach is appropriate when considering the requirements of portability and security. Since mobile agents run in a virtual environment, they are platform independent once the environment has been ported to all necessary platforms. Furthermore, the host protection can be achieved because the virtual environment can completely control the access of interpreted mobile agents to the host systems.

Mobile-C chooses C/C++ as mobile agent language. C/C++ is suitable for creating complex mobile agents because C/C++ is highly structured with a large and powerful set of functions. C/C++ is one of the most popular programming languages, which has been widely used for system programming and also for different applications. Choosing C/C++ as mobile agent language makes a huge body of existing C/C++ code available for reuse in mobile agent programs. Moreover, comparing with other languages, C has a unique advantage of easy interface with low-level hardware. For agent systems that are implemented in C/C++, there is an additional advantage for C/C++ mobile agents. C/C++ mobile agents can easily communicate with the agent platform because both programs share the same data structures, function prototypes and language syntaxes. The interface between platform space and the mobile agent space can be easily achieved.

Mobile-C integrates an embeddable C/C++ interpreter—Ch [33–35]—into the agent platform as a mobile agent execution engine as shown in Figure 2. To prevent malicious mobile agents from tampering outside their own boundary, mobile agents executing within Mobile-C are protected from each other, as well as agent platforms. Each mobile agent is interpreted inside a Ch, which is embedded

DOI: 10.1002/spe

into the system as a separate thread. A mobile agent runs within an interpreter, controlled and served by the agent platform. An agency creates an AEE thread when receiving a request to execute a mobile agent and terminates this thread after the execution of the mobile agent. The agent platform of an agency mediates mobile agents to communicate with other agents or access the host system. The interpreter restricts mobile agents by invoking only allowed functions and accessing their own address space in the interpreter space. The agent platform memory space exists outside the interpreter space. The interface between interpreter space and agent platform space can be accomplished by Ch SDK and Embedded Ch, which will be introduced in detail in the next subsection.

Ch is an embeddable C/C++ interpreter for cross-platform scripting, shell programming, numerical computing and embedded scripting. Ch supports all of the language features and standard libraries of the ISO C90 standard, major new features in C99 and classes in C++. It also supports an increasing number of C/C++ libraries, including POSIX, TCP/IP socket, Winsock, Win32, X11/Motif, GTK+, OpenGL, ODBC, SQLite, CGI, LAPACK, LDAP, PCRE, Gnome Libxml2, Oracle XDK for XML, NAG statistics library, Intel OpenCV for computer vision, ImageMagick for image processing, SigLib for signal processing and National Instruments' NI-DAQ and NI-Motion. For the engineering application purpose, Ch extends C and provides two- and three-dimensional graphical libraries for graphical output, computational arrays for matrix computation and advanced numerical analysis functions based on LAPACK for linear system analysis. Since Ch is a super set of C, the mobile agent language in Mobile-C follows the C standard. All of the functions and APIs available in the C standard and C/C++ libraries listed above can be used to model mobile agents. More information about Ch can be found in [35].

Ch has the following security mechanisms. First, from the language point of view, Ch is designed to be type-safe and secure. Pointers and dynamic memory allocation are powerful features of C/C++. However, inexperienced developers might lack the confidence to use them. Inappropriate handling of pointers and memory allocation/deallocation may cause a program crash. Ch provides an additional built-in string type with automatic memory management to resolve this problem. It also automatically checks the range of strings and arrays to avoid memory corruption. In addition, safe Ch is introduced as a restricted shell to run remote code obtained from the open network. Similarly to a Java virtual machine executing Java applets, safe Ch uses a sandbox security model to provide security protection.

4.2. The interface between the mobile agent space and the agent platform space

An agent execution engine is a dedicated environment for executing mobile agent code. In Mobile-C, an embeddable C/C++ interpreter—Ch—is used as the agent execution engine for mobile agents. Ch runs in a separate thread, and each mobile agent runs in its own Ch environment. Ch is dynamically embedded into the agent platform on arrival of each mobile agent. Ch SDK and Embedded Ch [35] serve as the interface between mobile agent space (Ch space) and agent platform space (binary C/C++ space) as shown in Figure 3. Ch SDK enables mobile agents to access global variables in the agent platform space or call functions in the compiled C/C++ libraries, such as static libraries, shared libraries or Dynamically Linked Libraries (DLLs). Embedded Ch allows agent platforms to run mobile agent code, invoke mobile agent functions, or access variables in the mobile agent space. With the power of Ch SDK and Embedded Ch, mobile agents in the interpreter space can easily access the host system—calling system functions and accessing variables in the binary space. On the other hand, C/C++ programs can also call functions and access variables in the mobile agent space.

DOI: 10.1002/spe

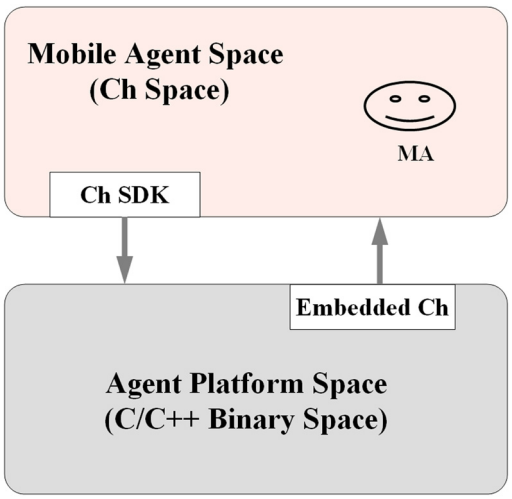

Figure 3. The interface between mobile agent space and agent platform space.

4.3. Mobile agent migration

Agent migration has been defined as a process in which an agent changes its executing host during the life cycle of the agent. An agent state is composed of the execution state and the data state. An agent's execution state mainly refers to the current control information, such as program counter, and all of the stack information that is required to characterize the current execution point. An agent's data state consists of essential properties of the agent, the internal and global data. According to whether an execution state is transferred, an agent can migrate in two different ways: strong migration, and weak migration. *Strong migration* means that all of the agent execution state and data state are captured and transferred to a target host. After a strong migration, the agent continues to execute its program exactly at the point at which it has been interrupted before the migration. *Weak migration* means that only the data state is transferred from one host to another.

Since the current version of Ch does not offer the capability to capture the execution state of a process or thread, Mobile-C supports weak migration in the current implementation. The task of a mobile agent is divided into multiple subtasks similar to the approach presented in [50]. These subtasks can be performed in different hosts and are listed in a task list. New subtasks can be added into the list and the task list can be modified dynamically, based on the new conditions. The ability to dynamically change the task list significantly improves the flexibility of a mobile agent. However, once a subtask has started to execute in a certain host, the mobile agent cannot move until the termination of the execution. Agent migration in Mobile-C is achieved through ACL messages encoded in XML, which convey agents as the content of messages. Using FIPA ACL messages for agent migration

DOI: 10.1002/spe

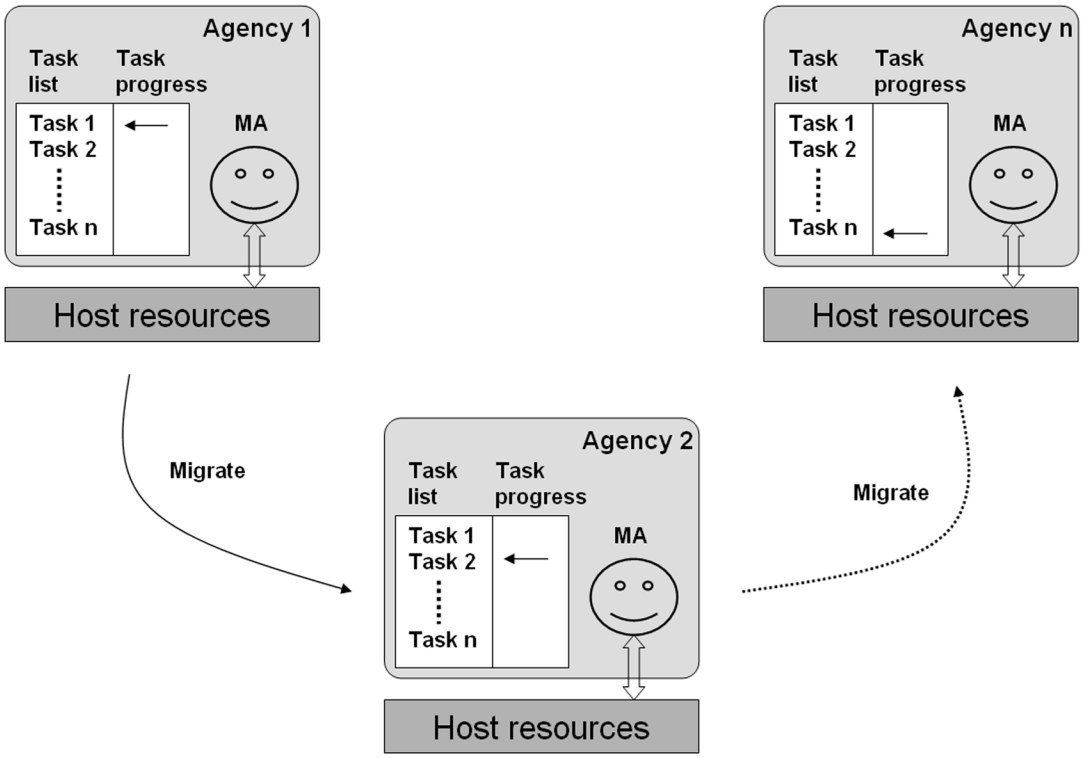

Figure 4. Agent migration based on a task list and a task progress pointer.

in FIPA compliant systems achieves inter-platform migration through mandatory communication mechanisms in the agent platform. A mobile agent message contains the data state and code of an agent. The data state of a mobile agent includes general information of a mobile agent and tasks that the mobile agent is going to perform in destination hosts. The general information of a mobile agent includes agent name, agent owner and the home agency that creates the mobile agent. Task information includes number of tasks, a task progress pointer and the description of each task. During agent migration, the return results from previous tasks and the values of global variables are also encapsulated into the agent message. The strategy to resume agent execution after migration uses a task progress pointer as shown in Figure 4.

4.4. Agent mobility protocol

The process of agent migration can be triggered by an agent itself or users through the graphical user interface (GUI) via AMS. The agent mobility protocol that realizes the entire migration operation can

DOI: 10.1002/spe

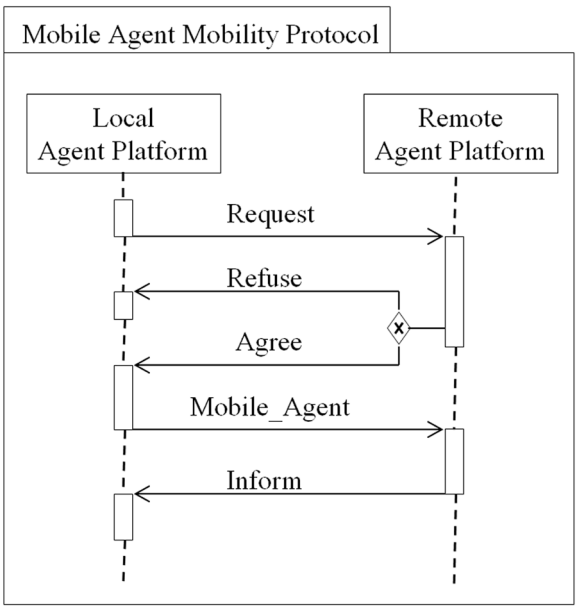

Figure 5. The agent mobility protocol.

be expressed in agent UML [51] as shown in Figure 5. The description of each interaction step is given below.

1. The local agent platform sends a 'Request' message with the mobility requirements of the mobile agent to the remote agent platform.
2. If the migration request is accepted, the remote agent platform sends an 'Agree' message; otherwise, a 'Refuse' message.
3. Once the migration is accepted, the local agent platform wraps the information related to the mobile agent, including agent code and data state necessary to resume agent state, into a mobile agent message. Then, the ACL mobile agent message with type 'Mobile-Agent' is delivered to the destination agent platform.
4. When the remote agent platform receives this ACL mobile agent message, the remote agent platform retrieves the agent code and data state from the mobile agent message. It then requests to create a new thread and resumes the execution of the mobile agent.
5. After the remote agent platform successfully resumes the execution of the mobile agent, it notifies the source agent platform with an 'Inform' ACL message. The source agent platform then destroys the mobile agent and terminates the agent thread.

DOI: 10.1002/spe

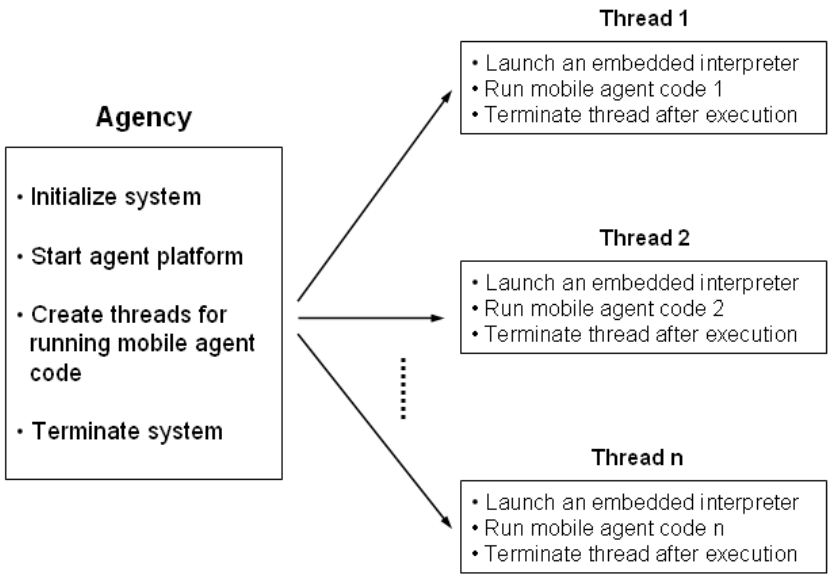

Figure 6. The program structure of an agency.

5. THE IMPLEMENTATION OF AGENCIES

Agencies run in each node of the network in Mobile-C. From the programming point of view, an agency is a multi-thread server program. The program structure of an agency is shown in Figure 6. When the execution starts, the agency first initializes the system and creates threads for all of the components in the agent platform. After system initialization, the agency waits for defined events. When the agency receives a request to run a mobile agent, it creates a new thread and embeds an embeddable C/C++ interpreter—Ch—into the thread for executing mobile agent code. After the mobile agent migrates to other hosts, this thread is terminated automatically. If the agency receives a termination request, it terminates the execution of agent platform and the system itself. In the current Mobile-C implementation, each mobile agent runs in an Embedded Ch inside its own thread. The use of multiple threads to implement multiple agents is much more efficient than a multi-process approach because the start-up and termination of multi-processes and expensive Inter-Process Communication (IPC) are avoided. Multiple mobile agents in a common address space will not introduce a memory protection problem because protection is already ensured by interpreters. The agency governs mobile agent threads, and the agent platform mediates interaction among agents and controls mobile agents accessing to the host system.

The agent platform is the core of an agency. As described in Section 3, the main components of an agent platform are AMS, ASM, DF, ACC and AEE. The agent platform is initiated when the

DOI: 10.1002/spe

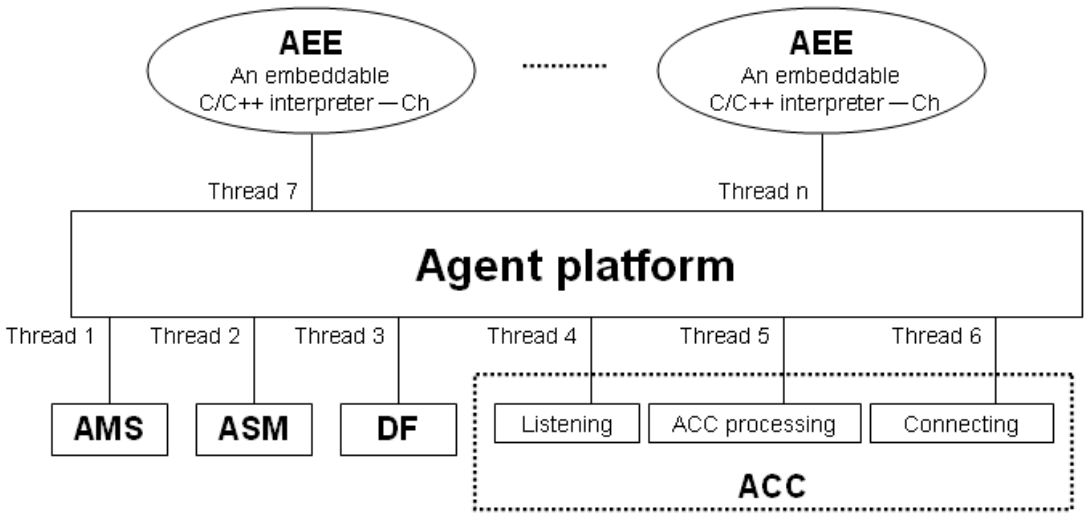

Figure 7. The multi-thread implementation of an agent platform.

agent system starts. The agent platform is also a multi-thread program. Each component works in an individual thread, which ensures that all of the components can work concurrently as shown in Figure 7. AMS is responsible for managing agents, including creation, registration and migration. All of the registered agents are records in an agent list. The node of the agent list contains information for each agent, such as agent type, AID for stationary agent and data state for mobile agents. AMS is also responsible for accepting and dispatching mobile agents, including creating an object of a mobile agent from a mobile agent message, requesting for executing and suspending the execution of a mobile agent and generating a mobile agent message to pack the data state and code of a mobile agent. The DF manages agent services registered with it. To speed up searching a request service, a Service Hash Table and two types of linked lists are designed to record available agent services.

According to the FIPA specifications, an agency should provide mechanisms for both receiving and sending messages. This requirement is satisfied by a listening thread, a connecting thread and an ACC processing thread as shown in Figure 7. The listening thread is used to listen for client connections. Once it accepts a new client connection, it adds this client to a connection list. On the other hand, the connecting thread is responsible for connecting to other hosts. The ACC processing thread processes linked lists of client connections and requests for connecting remote hosts. ACC facilitates the remote horizontal communication via ACL messages, such as remote agent to agent communication and agent platform to agent platform communication. Remote horizontal communication in Mobile-C is implemented on top of TCP/IP. The transport protocol uses HTTP. Messages are sent asynchronously to a recipient agent identified by its global unique identifier. The local agent to agent platform information exchange is achieved by Ch SDK and Embedded Ch, an interface between agent platform space (binary C/C++ space) and mobile agent space (Ch space or C/C++ script space).

DOI: 10.1002/spe

6. APPLICATION: AGENT-BASED REAL-TIME TRAFFIC DETECTION AND MANAGEMENT SYSTEM

6.1. Overview

The domain of traffic and transportation management is well suited for an agent-based approach because of its geographically distributed and dynamic changing nature [52]. However, the applications of agent technology in this field are primarily seeking the usage of intelligent agents for simulation purposes [53,54]. Most applications that address real-world roadway network management are still at the research stage [18,19,26]. Our research project, agent-based real-time traffic detection and management system (ABRTTDMS) [55] aims at providing autonomous traffic information fusion for ITSs. Real-time traffic information is the core of ITSs. The current ITS information architecture is typically centralized. The raw data from the field surveillance systems are transmitted to the TMC, and the computation of multi-site statistics has to be performed at the TMC. The delay of data transmission and processing would affect the evaluation of traffic flow at real-time and slows the response process to traffic incidents. Furthermore, most of the existing surveillance systems are proprietary and are not able to interoperate with each other. The integration of different detection systems is, therefore, difficult and costly.

The agent and advanced communication technology allow application developers to enhance interoperability and distributed computing capability of information systems in ITSs. The interoperability and cooperation are critically needed in making decisions based on information across systems, organizational and jurisdictional boundaries. The distributed computing capability enables detection stations to cooperate with each other in a certain area to perform traffic information fusion, which will dramatically reduce the response time to incidents in an emergency. The goal of the ABRTTDMS is to add interoperability and distributed computing capability to the existing centralized traffic detection and management information systems. This goal can be achieved by wrapping detection stations to distributed agents. Compliance with FIPA standards allows distributed agents interoperating with each other and exchanging information across multiple sites. The integration system can combine information from multiple detection stations and systems, evaluate traffic flow, respond traffic flow changes and evaluate operational responses to traffic flow changes in real-time. The integration of data from multiple detection stations and systems will help operations become more efficient by enabling a more comprehensive view of the environment. In addition to stationary distributed agents, ABRTTDMS uses mobile agent technology to further enhance the flexibility and adaptability of traffic detection and management systems. The use of mobile agents in the system will be further discussed in Section 6.2.

The system architecture of the ABRTTDMS has been designed to have multiple levels as shown in Figure 8. The lowest level is composed of various detectors such as loop detectors [56], laser detectors [57] and video cameras [58]. Laser detection agents (LRDAs), loop detector agents (LPDAs), and video camera detector agents (VCDAs) are used to process the output of the corresponding detectors and provide the desired intelligence. A traffic detection execution system (TDES) agent coordinates all of the LRDAs, LPDAs, and VCDAs in a sub-network and communicates with other TDES agents and an upper-level TMC agent. The TMC agent delegates tasks to the lower-level agents and analyzes the information from these agents. It also dispatches mobile agents to different agencies to perform certain tasks. As the system architecture is open, new detection systems can be easily integrated into the system by wrapping them into agent-based sub-systems.

DOI: 10.1002/spe

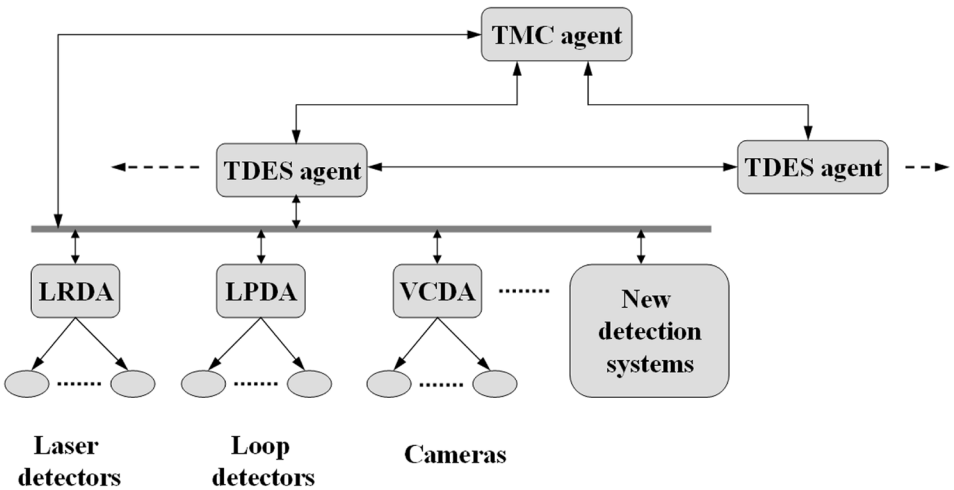

Figure 8. The system architecture of the ABRTTDMS.

Either laser detectors or loop detectors can detect the real-time traffic parameters. In the current ABRTTDMS implementation, real-time traffic parameters are detected by the laser-based vehicle detection system shown in Figure 9. The detection system was developed in the Integration Engineering Lab at the University of California, Davis sponsored by Caltrans [57,59,60]. The laser-based detectors are designed to detect the delineations, such as length and width, as well as velocities of passed vehicles on highways. The real-time detection data are sent to LRDAs continuously.

The video cameras in the system are used to verify the incidents. Multiple types of cameras may be employed to meet different purposes. For example, omni-view cameras can be used to provide a much broader range of coverage than standard rectilinear cameras, and infrared cameras can be used during the night time [61]. The VCDA controls the operation of different cameras, processes the video stream and estimates the required traffic parameters.

6.2. Mobile agents in ABRTTDMS

The ABRTTDMS has two types of agents: stationary agents, and mobile agents. LRDAs, LPDAs, VCDAs, TDES agents and TMC agent are stationary agents. Mobile agents in the ABRTTDMS are used for dynamic code deployment and performing unanticipated actions. There are instances where Homeland Security attempts to track individual vehicles. A mobile agent with vehicle signatures filtering and re-correlating data between stations as needed might be a good solution for this purpose. One of the other useful applications is to obtain traffic information that is not available in the system. The TMC incoming data, data format and traffic parameters calculated from these data are fixed in the current traffic information systems. For the functionality that was not implemented at the design stage, the existing code cannot provide required information. To obtain additional information, raw data

DOI: 10.1002/spe

Figure 9. A laser-based vehicle detection system.

have to be sent to the TMC. The raw data usually generated by detectors are huge. Instead of moving raw data, we move computation to the data source. A mobile agent equipped with an appropriate computational logic can migrate through specified stations in the network. At each station it accesses raw data, processes raw data locally with new algorithms and sends results back to its dispatcher—the TMC agent. Using mobile agents can avoid transferring a huge volume of raw data to the TMC, reduce the network transmission load, overcome the network latency and provide secure data access. We will now give a simulated application example of using Mobile-C in the ABRTTDMS.

In this example, the real-time vehicle information in the ABRTTDMS is saved in geographically distributed databases in detection stations as shown in Table I. The databases have been implemented in a popular open source database—MySQL. For the quick evaluation of traffic conditions at several nodes in the network, a mobile agent is a promising approach for performing this dynamic information query task. Figure 10 shows an example of using a mobile agent to find the average vehicle velocities at Detection station 4 (Mace) and Detection station 3 (UC Davis) in the sub-network 2 (Davis) during the time interval from 12:25 pm to 12:40 pm on 1 March 2005. To get this information, a mobile agent visits these detection stations and makes database queries to retrieve the required information. The first four fields of the database in Table I record the time when the vehicles have been detected. The average vehicle velocities at detection stations are based on the values of the fields V_FEND and V_BEND in the database, which are the front velocity and rear velocity of the passed vehicles. The fields 1–6 in the

DOI: 10.1002/spe

Table I. Vehicle information database (field 1–6).

T_DAY	T_HOUR	T_MINUTE	T_SECOND	V_FEND	V_BEND
1	12	26	32	42.1	42.25
1	12	28	19	52.5	55.9
1	12	31	15	41.5	47.35
1	12	36	12	41.61	48.25
1	12	36	50	51.8	52.65
1	12	37	13	53.8	51.35

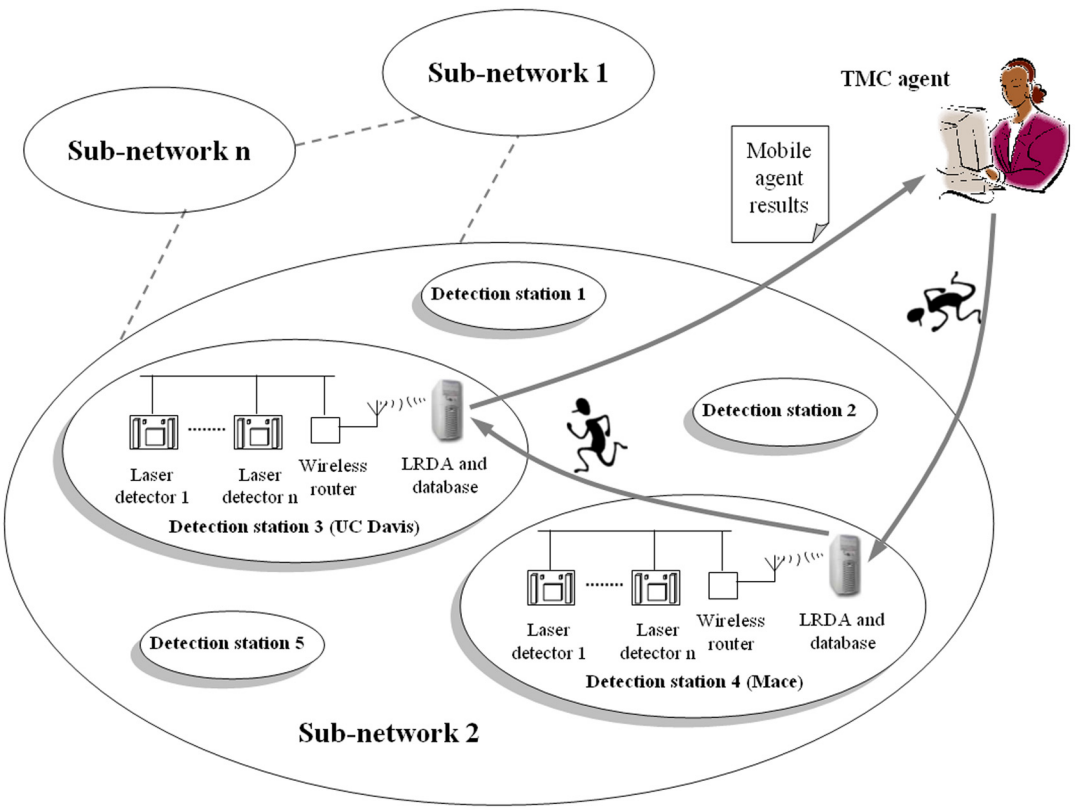

Figure 10. Mobile agents for remote information query.

DOI: 10.1002/spe

```
<NAME> mobagent </NAME>
<OWNER> IEL </OWNER>
<HOME> iel2.engr.ucdavis.edu:5138 </HOME>
<TASK task = "2" num = "0">
  <DATA number_of_elements="0" name ="results_mouse2"
        complete ="0" server="mouse2.engr.ucdavis.edu:5167">
    <DATA_ELEMENT> </DATA_ELEMENT>

    <AGENT_CODE>
       Mobile agent code on mouse2.
    </AGENT_CODE>
  </DATA>
  <DATA number_of_elements = "0" name = "results_bird1"
        complete = "0" server = "bird1.engr.ucdavis.edu:5125">
    <DATA_ELEMENT> </DATA_ELEMENT>

    <AGENT_CODE>
       Mobile agent code on bird1.
    </AGENT_CODE>
  </DATA>
</TASK>
```

Figure 11. The content of the mobile agent message from the host *iel2* to the host *mouse2*.

database at the Mace detection station on 1 March 2005 at 12:00 pm are shown in Table I. Other fields include vehicle acceleration, length, lane number and detection station domain name.

In Figure 10, the host name of TMC agent, Detection station 4, and Detection station 3 is *iel2*, *mouse2* and *bird1*, respectively. The mobile agent dispatched by host *iel2* visits remote host *mouse2* and *bird1*. Figure 11 shows part of the mobile agent message sent from host *iel2* to host *mouse2*. The mobile agent message contains the general information of the agent, including agent name, agent owner and home agent platform. The agent tasks are described within the TASK tag. The *task* attribute of the TASK element specifies how many tasks the mobile agent has. The *num* attribute indicates how many tasks have been completed. The DATA element contains agent data, code and return results related to each task. For example, the attributes, *number_of_elemets*, *name*, *complete* and *server* specify the size of return data array, the name of return data array, the number of times that the mobile agent code has been executed on the remote host and the remote host domain name and port number, respectively. The sub-element DATA_ELEMENT contains the return data from the task. The sub-element AGENT_CODE includes mobile agent code to be executed in the remote host. For simplicity, the mobile agent code is omitted in Figure 11. The mobile agent in this example has two tasks and no task has yet been completed. Two hosts that the mobile agent is going to visit are *mouse2.engr.ucdavis.edu* and *bird1.engr.ucdavis.edu*. The server port numbers of these two hosts are 5167 and 5125.

The mobile agent accesses the vehicle information databases at *mouse2* and *bird1* through Open Database Connectivity (ODBC) [62] as shown in Figure 12. Ch has a binding to ODBC,

DOI: 10.1002/spe

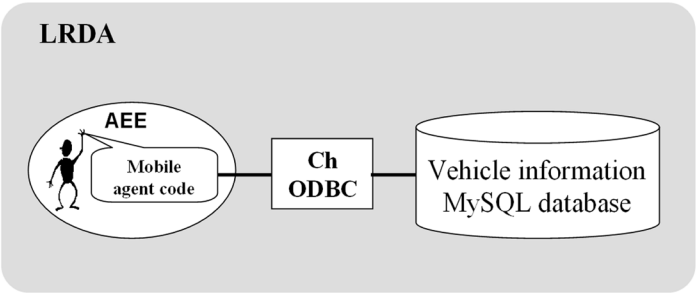

Figure 12. Access databases through Ch ODBC.

called Ch ODBC [63]. With Ch ODBC, the mobile agent running in Ch can use ODBC APIs to access databases in different platforms. At each detection station, the mobile agent initiates an ODBC driver, connects to the database and performs any specified queries. When the mobile agent completes its task at each detection station, it migrates to the next detection station as shown in Figure 10, and finally sends the results back to the TMC agent as shown in Figure 13.

7. DISCUSSIONS

As previously described, the motivation of developing Mobile-C is for the applications in distributed systems that have a high degree of coupling with low-level hardware. The characteristics of distributed nodes in these systems, such as real-time mechatronic systems and sensor networks, are small, relatively resource poor, and involving real-time information fusion and hardware control. Most real-time operating systems (OSs) and application programs run in real-time OSs are implemented in C. To easily interface with application programs and underlying hardware, C is the language of choice for the implementation of mobile agents in this type of applications. Also, a mobile agent system with a small footprint is desirable for integration with systems that have limited resources.

The design philosophy of Mobile-C is to be small, embeddable and easy to interface with low-level software and hardware. The footprint of Ch is smaller than the Java virtual machines used in most FIPA compliant agent systems. To further minimize the size of the system, Ch supports MiniXML to generate and parse XML messages. From the agent platform point of view, Mobile-C, to the best of our knowledge, is the first system that uses mobile agent messages for mobile agent migration. Using FIPA ACL messages for agent migration simplifies the agent platform because both agent communication and migration can share the same communication mechanism. For small systems that need to store distributed data locally, an embeddable database SQLite might be a good choice to meet the need. The SQLite is a small C library that implements a self-contained, embeddable, zero-configuration SQL database engine. By using SQLite, data can be stored in data files without need of configuration. Ch has a package called Ch SQLite [64], which allows mobile agents to access SQLite database directly. Ch is embeddable. Using Ch as a mobile agent execution engine makes Mobile-C easily embeddable into

DOI: 10.1002/spe

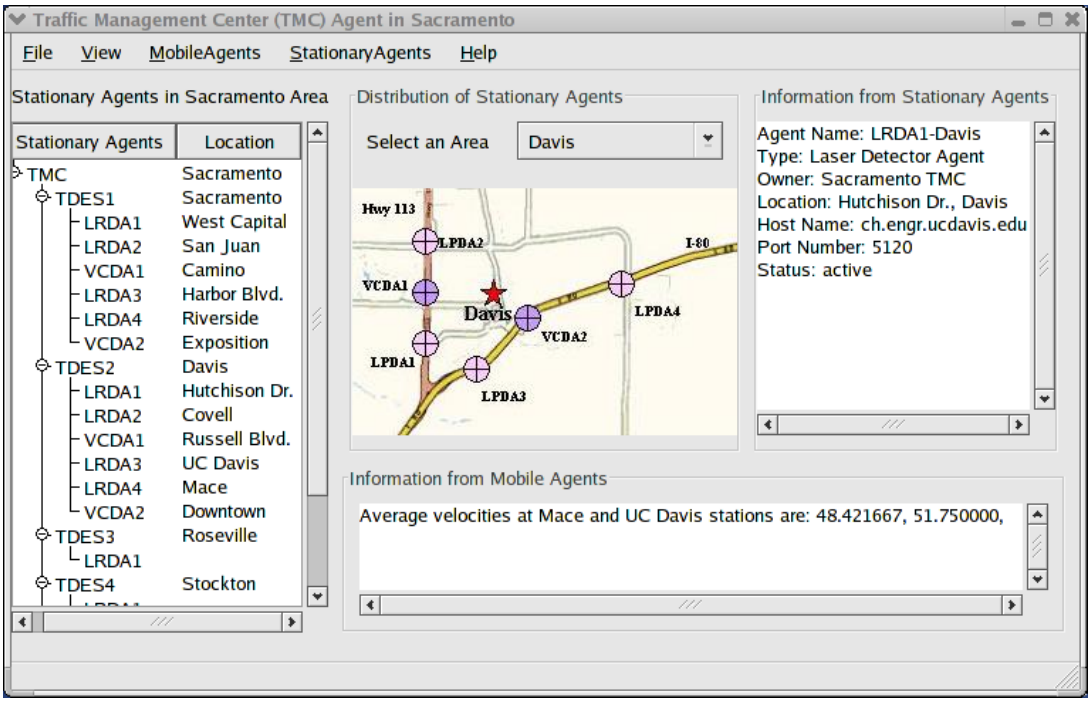

Figure 13. The graphical user interface of the TMC agent.

other C application programs. Ch SDK and Embedded Ch allow mobile agents and applications to interface with each other.

Comparing with other C/C++ mobile agent systems that are not compliant with agent standards, Mobile-C uses a source-level C/C++ interpreter to support the execution of mobile agents. Although the interpreted C/C++ mobile agent code, to a certain extent, sacrifices the execution speed of the mobile agent code comparing with binary C/C++ code, the benefit it brings to the system is significant. It allows mobile agent code works in heterogeneous platforms, provides secure execution and simplifies the implementation of the agent system. In addition, since the major computation of the system is performed by stationary agents, the interpretation of mobile agent code should not cause significant computational problems in most cases. Contrary to most agent systems, Mobile-C supports both stationary agents and mobile agents. This choice was made because we recognize the necessity of stationary agents for autonomous local information fusion and control, providing required intelligence and, more importantly, working collaboratively with other stationary agents through agent communication. The use of mobile agents is mainly for increasing the flexibility and adaptability of the systems.

DOI: 10.1002/spe

One additional advantage of Mobile-C, to be presented in a separate paper, is that mobile agents in Mobile-C can directly process XML data through an interpretive software package Ch XML [65]. This salient feature exceeds the normal use of XML for agent communication messages and provides a solid foundation for using XML to form a programmable information base in the mobile agent systems, including agent data, communication messages and access control policies.

8. CONCLUSIONS

The design, implementation and application of an IEEE FIPA compliant agent platform, Mobile-C, has been presented in this article. Mobile-C conforms to the IEEE FIPA standards at both agent-level and platform-level. To support mobile agents, Mobile-C integrates an embeddable C/C++ interpreter—Ch—into the platform as a mobile agent execution engine and defines an agent mobility protocol to regulate entire agent migration operation. Mobile agent migration is realized through ACL messages. Mobile agents, including its data state and code, are transported to a remote agent platform via ACL messages, and the execution of mobile agents is resumed by a task progress pointer. ACL messages are encoded in XML. Our experience shows that using XML to encode different types of messages, including simple agent communication messages and messages containing mobile agents, is simple, convenient and easy to change.

Mobile-C is designed for the applications in large-scale hardware related distributed systems. Mobile-C is small, embeddable and easy to interface with low-level software and underlying hardware. Mobile-C has potential to be used to support code mobility in many engineering applications, such as agile manufacturing systems, autonomous sensor fusion networks and intelligent transportation systems. Mobile-C has been used to simulate highway traffic detection and management. The ABRTTDMS is designed for the traffic information fusion from different detection stations on highways to perform real-time traffic condition evaluation. The mobile agents in the system increase the flexibility and adaptability of the traffic information system. The simulation results demonstrate the effectiveness of mobile agents for the dynamic code deployment and performing unanticipated actions.

ACKNOWLEDGEMENTS

This work was partially funded by the California Department of Transportation through the PATH program. The authors would like to thank David D. Linz for his contribution in the implementation of the agent platform described in this article.

REFERENCES

1. Schantz RE, Schmidt DC. Research advances in middleware for distributed systems. *Proceedings of the IFIP 17th World Computer Congress-TC6 Stream on Communication Systems: The State of the Art*. Kluwer: Norwell, MA, 2002; 1–36.
2. Schmidt DC. Middleware for real-time and embedded systems. *Communications of the ACM* 2002; **45**(6):43–48.
3. Zambonelli F, Van Dyke Parunak H. Signs of a revolution in computer science and software engineering. *Engineering Societies in the Agents World III: Third International Workshop* (*Lecture Notes in Computer Science*, vol. 2577), Petta P, Tolksdorf R, Zambonelli F (eds.). Springer: Berlin, 2003; 13–28.
4. Jennings NR. An agent-based approach for building complex software systems—why agent-oriented approaches are well suited for developing complex, distributed systems. *Communications of the ACM* 2001; **44**(4):35–41.
5. Wooldridge MJ, Ciancarini P. Agent-oriented software engineering: The state of the art. *Proceedings of Agent-Oriented Software Engineering: The 1st International Workshop (AOSE 2000)*. Revised Paper (*Lecture Notes in Computer Science*, vol. 1957), Ciancarini P, Wooldridge MJ (eds.). Springer: Berlin/Heidelberg, 2001; 1–28.
6. Luck M. Guest editorial: Challenges for agent-based computing. *Autonomous Agents and Multi-Agent Systems* 2004; **9**(3):199–201.

DOI: 10.1002/spe

7. Shen W, Xue D, Norrie DH. An agent-based manufacturing enterprise infrastructure for distributed integrated intelligent manufacturing systems. *Proceedings of the 3rd International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-98)*. Practical Application: Blackpool, UK, 1998; 533–548.
8. Wada H, Okada S. An autonomous agent approach for manufacturing execution control systems. *Integrated Computer-Aided Engineering* 2002; **9**(3):251–262.
9. Van Dyke Parunak H, Baker AD, Clark SJ. The AARIA agent architecture: From manufacturing requirements to agent-based system design. *Integrated Computer-Aided Engineering* 2001; **8**(1):45–58.
10. Eo SY, Chang TS, Shin DG, Yoon ES. Cooperative problem solving in diagnostic agents for chemical processes. *Computers and Chemical Engineering* 2000; **24**(2–7):729–734.
11. Jennings NR, Bussmann S. Agent-based control systems—why are they suited to engineering complex systems? *IEEE Control Systems Magazine* 2003; **23**(3):61–73.
12. Brennan RW, Fletcher M, Norrie DH. An agent-based approach to reconfiguration of real-time distributed control systems. *IEEE Transactions on Robotics and Automation* 2002; **18**(4):444–451.
13. Yokoo M, Fujita S. Trends of internet auctions and agent-mediated Web commerce. *New Generation Computing* 2001; **19**(4):369–388.
14. Sandholm T. eMediator: A next generation electronic commerce server. *Computational Intelligence* 2002; **18**(4):656–676.
15. Choi SPM, Liu JM, Chan SP. A genetic agent-based negotiation system. *Computer Networks—The International Journal of Computer and Telecommunications Networking* 2001; **37**(2):195–204.
16. Chen WSE, Hu CL. A mobile agent-based active network architecture for intelligent network control. *Information Sciences* 2002; **141**(1–2):3–35.
17. Chou LD, Shen KC, Tang KC, Kao CC. Implementation of mobile-agent-based network management systems for National Broadband Experimental Networks in Taiwan. *Holonic and Multi-Agent Systems for Manufacturing* (*Lecture Notes in Computer Science*, vol. 2744), Marík V, McFarlane D, Valckenaers P (eds.). Springer: Berlin, 2003; 280–289.
18. Logi F, Ritchie SG. A multi-agent architecture for cooperative inter-jurisdictional traffic congestion management. *Transportation Research Part C-Emerging Technologies* 2002; **10**(5–6):507–527.
19. Hernandez JZ, Ossowski S, Garcia-Serrano A. Multiagent architectures for intelligent traffic management systems. *Transportation Research Part C—Emerging Technologies* 2002; **10**(5–6):473–506.
20. Stathis K, de Bruijn O, Macedo S. Living memory: Agent-based information management for connected local communities. *Interacting with Computers* 2002; **14**(6):663–688.
21. Tu HC, Hsiang J. An architecture and category knowledge for intelligent information retrieval agents. *Decision Support Systems* 2000; **28**(3):255–268.
22. Boloni L, Marinescu DC, Rice JR, Tsompanopoulou P, Vavalis EA. Agent based scientific simulation and modeling. *Concurrency Practice and Experience* 2000; **12**(9):845–861.
23. Casanova H, Dongarra J. Using agent-based software for scientific computing in the netsolve system. *Parallel Computing* 1998; **24**(12–13):1777–1790.
24. Huang J, Jennings NR, Fox J. Agent-based approach to health care management. *Applied Artificial Intelligence* 1995; **9**(4):401–420.
25. Noda I, Stone P. The RoboCup Soccer Server and CMUnited clients: Implemented infrastructure for MAS research. *Autonomous Agents and Multi-Agent Systems* 2003; **7**(1–2):101–120.
26. Adler JL, Blue VJ. A cooperative multi-agent transportation management and route guidance system. *Transportation Research Part C—Emerging Technologies* 2002; **10**(5–6):433–454.
27. Fuggetta A, Picco GP, Vigna G. Understanding code mobility. *IEEE Transactions on Software Engineering* 1998; **24**(5):342–361.
28. Lange DB, Oshima M. Seven good reasons for mobile agents. *Communications of the ACM* 1999; **42**(3):88–89.
29. Chen B, Cheng HH. A runtime support environment for mobile agents. *Proceedings of the 2005 ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications (MESA05)*, Long Beach, CA, September 2005. American Society of Mechanical Engineers: New York, 2005; 37–46.
30. Chen B, Runtime support for code mobility in distributed systems. *PhD Thesis*, Department of Mechanical and Aeronautical Engineering, University of California, Davis, CA, 2005.
31. IEEE Foundation for Intelligent Physical Agents (FIPA). Agent Communication Language Specifications. http://www.fipa.org/repository/aclspecs.html [2006].
32. Object Management Group. Mobile Agent System Interoperability Facilities Specification. http://www.omg.org/docs/orbos/97–10-05.pdf [2006].
33. Cheng HH. Scientific computing in the Ch programming language. *Scientific Programming* 1993; **2**(3):49–75.
34. Cheng HH. Ch: A C/C++ interpreter for script computing. *C/C++ User's Journal* 2006; **24**(1):6–12.
35. Softintegration, Inc. Ch—An Embeddable C/C++ Interpreter. http://www.softintegration.com [2006].

DOI: 10.1002/spe

36. Baumann J, Hohl F, Rothermel K, Strasser M, Theilmann W. MOLE: A mobile agent system. *Software—Practice and Experience* 2002; **32**(6):575–603.
37. Lange DB, Oshima M. *Programming and deploying Java mobile agents with Aglets*. Addison-Wesley: Reading, MA, 1998.
38. Ono K, Tai H. A security scheme for Aglets. *Software—Practice and Experience* 2002; **32**(6):497–514.
39. Wong D, Paciorek N, Walsh T, DiCelie J, Young M, Peet B. Concordia: An infrastructure for collaborating mobile agents. *Proceedings of the 1st International Workshop on Mobile Agents (MA'97)* (*Lecture Notes in Computer Science*, vol. 1219). Springer: Berlin, 1997; 86–97.
40. Gray RS. Agent Tcl: A flexible and secure mobile-agent system. *Proceedings of the 4th Annual USENIX Tcl/Tk Workshop*, Monterey, CA, 1996. USENIX Association: Berkeley, CA, 1996; 9–23.
41. Gray RS, Cybenko G, Kotz D, Peterson RA, Rus D. D'Agents: Applications and performance of a mobile-agent system. *Software—Practice and Experience* 2002; **32**(6):543–573.
42. Peine H. Run-time support for mobile code. *PhD Thesis*, Department of Computer Science, University of Kaiserslautern, Germany, 2002.
43. Peine H. Application and programming experience with the Ara mobile agent system. *Software—Practice and Experience* 2002; **32**(6):515–541.
44. Johansen D, Lauvset KJ, van Renesse R, Schneider FB, Sudmann NP, Jacobsen K. A TACOMA retrospective. *Software—Practice and Experience* 2002; **32**(6):605–619.
45. JADE-Board. Java Agent DEvelopment Framework. http://jade.tilab.com/ [2006].
46. The University of Kaiserslautern. MACE—Mobile Agent Code Environment. http://wwwagss.informatik.uni-kl.de/Projekte/Ara/mace.html [2006].
47. Sudmann NP, Johansen D. Adding mobility to non-mobile Web robots. *Proceedings of the IEEE ICDCS'00 Workshop on Knowledge Discovery and Data Mining in the World-Wide Web*. IEEE: Piscataway, NJ, 2000; 73–79.
48. IEEE Foundation for Intelligent Physical Agents (FIPA). Publicly Available Agent Platform Implementations. http://www.fipa.org/resources/livesystems.html#zeus [2006].
49. Griss ML. Software agents as next generation software components. *Component-Based Software Engineering: Putting the Pieces Together*, Heineman GT, Councill WT (eds.). Addison-Wesley: Boston, MA, 2001; 641–657.
50. Chong C, Jiwen H, Kai B, Zhongfan M. Mobile software agent model and the architecture of JMSAS system. *Proceedings of the 1st International Workshop on Mobile Agent for Telecommunication Applications*, Ottawa, Canada, 1999. World Scientific: Singapore, 1999; 37–52.
51. Odell J, Van Dyke Parunak H, Bauer B. Representing agent interaction protocols in UML. *Agent-Oriented Software Engineering*, Ciancarini P, Wooldridge MJ (eds.). Springer: Berlin, 2001; 121–140.
52. Jennings NR, Sycara K, Wooldridge MJ. A roadmap of agent research and development. *International Journal of Autonomous Agents and Multi-Agent Systems* 1998; **1**(1):7–38.
53. Dia H. An agent-based approach to modelling driver route choice behaviour under the influence of real-time information. *Transportation Research Part C—Emerging Technologies* 2002; **10**(5–6):331–349.
54. Hidas P. Modelling lane changing and merging in microscopic traffic simulation. *Transportation Research Part C—Emerging Technologies* 2002; **10**(5–6):351–371.
55. Chen B, Cheng HH, Palen J. Agent-based real-time computing and its applications in traffic detection and management systems. *Proceedings of the ASME 24th Computers in Engineering Conference*, Salt Lake City, UT, 2004. American Society of Mechanical Engineers: New York, 2004; 543–552.
56. Ostland M, Petty KF, Bickel P, Jiang J, Rice J, Ritov Y, Schoenberg F. Simple travel time estimation from single-trap loop detectors. *Intellimotion* 1997; **6**(1):4–5 and 11.
57. Cheng HH, Shaw BD, Palen J, Larson JE, Hu XD, Van Katwyk K. A real-time laser-based detection system for measurement of delineations of moving vehicles. *IEEE/ASME Transactions on Mechatronics* 2001; **6**(2):170–187.
58. Malik J, Russell S. Measuring traffic parameters using video image processing. *Intellimotion* 1997; **6**(1):6–7 and 12–13.
59. Cheng HH, Shaw BD, Palen J, Lin B, Chen B, Wang Z. Development and field test of a laser-based nonintrusive detection system for identification of vehicles on the highway. *IEEE Transactions on Intelligent Transportation Systems* 2005; **6**(2):147–155.
60. Wang Z, Chen B, Cheng HH, Shaw B, Palen J. Performance analysis for design of a high-precision electronic opto-mechanical system for vehicle delineation detection on highway. *ASME Journal of Mechanical Design* 2003; **125**(4):802–808.
61. Trivedi MM, Mikic I, Kogut G. Distributed video networks for incident detection and management. *Proceedings of IEEE Intelligent Transportation Systems*, Dearborn, MI, October 2000. IEEE: Piscataway, NJ, 2000; 155–160.
62. Microsoft Corporation. Open Database Connectivity. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odbc/htm/dasdkodbcoverview.asp [2006].
63. Softintegration, Inc. Ch ODBC. http://www.softintegration.com/products/toolkit/odbc/ [2006].
64. Integration Engineering Laboratory, The University of California, Davis. Ch SQLite. http://chsqlite.sourceforge.net [2006].
65. Wang Z, Cheng HH. Portable C/C++ code for portable XML data. *IEEE Software* 2006; **23**(**1**):76–81.

DOI: 10.1002/spe