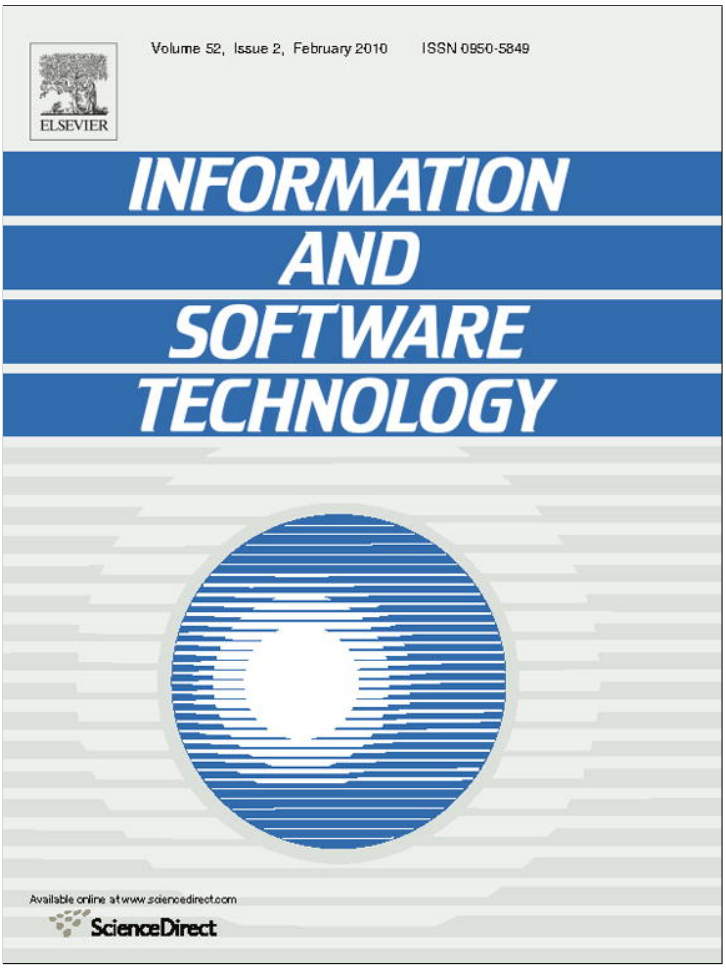

Information and Software Technology 52 (2010) 185–196

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

An embeddable mobile agent platform supporting runtime code mobility, interaction and coordination of mobile agents and host systems

Yu-Cheng Chou, David Ko, Harry H. Cheng *

Integration Engineering Laboratory, Department of Mechanical and Aeronautical Engineering, University of California, Davis, CA 95616, USA

ARTICLE INFO

Article history:
Received 16 February 2008
Received in revised form 2 August 2009
Accepted 24 September 2009
Available online 12 October 2009

Keywords:
Mobile agent library
Mobile agent API
Embeddable IEEE FIAP standard compliant mobile agent platform
C/C++ mobile agents
Ch, C/C++ interperter

ABSTRACT

Agent technology is emerging as an important concept for the development of distributed complex systems. A number of mobile agent systems have been developed in the last decade. However, most of them were developed to support only Java mobile agents. In order to provide distributed applications with code mobility, this article presents a library, the Mobile-C library, that allows a mobile agent platform, Mobile-C, to be embeddable in an application to support mobile C/C++ codes carried by mobile agents. Mobile-C uses a C/C++ interpreter as its Agent Execution Engine (AEE). Through the Mobile-C library, Mobile-C can be embedded into an application to support mobile C/C++ codes carried by mobile agents. Using mobile C/C++ codes, it is easy to interface a variety of low-level hardware devices and legacy systems. Through the Mobile-C library, Mobile-C can run on heterogeneous platforms with various operating systems. The Mobile-C library has a small footprint to meet the stringent memory capacity for applications in mechatronic and embedded systems. The Mobile-C library contains different categories of Application Programming Interfaces (APIs) in both binary and agent spaces to facilitate the design of mobile agent based applications. In addition, a rich set of existing APIs for the C/C++ interpreter employed as the AEE allows an application to have complete information and control over the mobile C/C++ codes residing in Mobile-C. With the synchronization mechanism provided by the Mobile-C library for both binary and agent spaces, simultaneous processes across both spaces can be coordinated to get correct runtime order and avoid unexpected race condition. The study of performance comparisons indicates that Mobile-C is about two times faster than JADE in agent migration. The application of the Mobile-C library is illustrated by dynamic runtime control of a mobile robot's behavior using mobile agents.

1. Introduction

Agent-based computing emerged in the past decade as a promising strategy for developing distributed complex systems [1–3]. It has been applied to a variety of distributed applications, such as manufacturing [4–6], real-time control systems [7–9], electronic commerce [10–12], network management [13,14], transportation systems [15,16], information management [17,18], scientific computing [19,20], health care [21] and entertainment [22]. The agent technology can significantly enhance the design and analysis of systems whose problem domain is geographically distributed, and whose subsystems exist in a dynamic environment and need to interact with each other more flexibly [23]. An agent that does not or cannot leave the execution environment is a stationary agent. On the other hand, a mobile agent is a software component that can travel among different execution environments autonomously [24]. Mobile agents provide a unifying framework to construct a variety of distributed applications. Mobile agents can be created dynamically during runtime and dispatched to destination systems to perform tasks with the most up-to-date code. The mobility of mobile agents provides applications with significant flexibility and adaptability that are both essential for satisfying dynamically changing requirements and conditions in a distributed environment.

In the past decade, many mobile agent platforms have been developed. Some of the well-known mobile agent platforms include Mole [25], Aglets [26], Concordia [27], D'Agents [28], Ara [29], TACOMA [30] and JADE [31,32]. However, most of them, such as Mole, Aglets, Concordia and JADE, were developed to support only Java mobile agents. D'Agents supports mobile agents written in Tcl, Scheme and Java, whereas Ara supports those written in Tcl, C, C++ and Java. TACOMA was originally developed to support Tcl mobile agents, and subsequently extended to support mobile agents written in multiple languages including C, Tcl, Perl, Python and Scheme.

The majority of mobile agent platforms in use are Java based. However, adopting a standard language as the mobile agent code language that provides both high-level and low-level functionalities is a good choice to accommodate the diversity of distributed

* Corresponding author. Tel.: +1 530 752 5020; fax: +1 530 752 4158.
E-mail address: hhcheng@ucdavis.edu (H.H. Cheng).

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.09.002

applications, especially in the field of networked mechatronic and embedded systems. C/C++ is a proper choice for such a mobile agent code language. First, compared to other scripting languages such as Tcl/Tk, Perl and Python, C/C++ provides more powerful functions in terms of memory access. Second, a vast number of existing C/C++ programs can be reused to construct mobile agent codes. Third, C is a middle-level language with industry standards. It can easily interface with a variety of low-level hardware devices.

Mobile-C [33–35] was originally developed as a standalone, IEEE FIPA compliant mobile agent platform to accommodate applications where low-level hardware is involved, such as networked mechatronic and embedded systems. Since most of the systems are written in C/C++, Mobile-C chose C/C++ as the mobile agent language for easy interfacing with control programs and underlying hardware. Additionally, Mobile-C adopted an embeddable C/C++ interpreter, Ch [36–38], as the Agent Execution Engine (AEE) to support the interpretive execution of C/C++ mobile agent codes. As opposed to a standalone mobile agent platform, the Mobile-C library was developed thereafter to make Mobile-C embeddable in any C/C++ programs to support code mobility. The original standalone Mobile-C was only supported by Linux operating systems on general purpose computers. However, the Mobile-C library allows the embeddable Mobile-C to run on heterogeneous platforms with various operating systems. Mobile-C is now supported by Windows, Linux, Solaris, HP-UX, FreeBSD, Mac OS X and QNX operating systems on general purpose computers and Linux operating systems on tiny and single-board computers.

The main differences between Mobile-C and the other platforms supporting C/C++ mobile agents, Ara and TACOMA, are as follows. Unlike Ara and TACOMA that are not compliant to any of the two international agent standards, the IEEE Foundation for Intelligent Physical Agents (FIPA) [39] and the OMG Mobile Agent System Interoperability Facility (MASIF) [40], Mobile-C is compliant to the IEEE FIPA standard. Such a compliance ensures the interoperability between a Mobile-C agent and other agents from heterogeneous FIPA compliant mobile agent platforms. In Ara and TACOMA, before a C/C++ mobile agent code can be executed at a remote host, it has to be compiled into a customized byte code for Ara [41], and a binary code for TACOMA [42]. Conversely, a C/C++ mobile agent code can be readily executed in Mobile-C without a pre-compilation step. As a result, Mobile-C allows for executing mobile agent codes that are dynamically generated or modified at runtime. Moreover, Mobile-C is able to dynamically respond to changes occurring in the environment. Therefore, Mobile-C provides users with significant flexibility to facilitate the development and implementation of mobile agent-based applications. In Ara, a byte code generated from a C/C++ mobile agent code is received by a remote host. In TACOMA, either a C mobile agent code or a binary code generated from a C mobile agent code is received by a remote host. In Mobile-C, a C/C++ mobile agent code is what a remote host receives. For the situation where a C/C++ mobile agent code is received by a remote host, the security of the remote host can be easily maintained. Because if desired, the C/C++ mobile agent code can be parsed and examined before it is executed. However, it is difficult to examine what is contained in a byte code or a binary code. Therefore, Mobile-C can provide a high level of security for remote hosts if desired.

The operating systems on general purpose computers that support Ara and/or TACOMA include Linux, Solaris, HP-UX and FreeBSD. Besides these operating systems, Mobile-C is also supported by other operating systems on general purpose computers such as Windows, Mac OS X and QNX. In addition, Mobile-C also works in tiny computers and single-board computers with supported Linux operating systems. Particularly, in Mobile-C, a C/C++ mobile agent code developed using standard C/C++ functions can be run by the AEE across Windows and Unix-like operating systems without the need to modify the mobile agent code. Therefore, Mobile-C provides a high degree of portability for mobile agent codes.

This article presents the Mobile-C library that can embed Mobile-C into any C/C++ programs to facilitate the design of mobile agent-based applications. Mobile agents in an application can control the agent platform, its modules and other mobile agents, as well as smoothly interface with a variety of low-level hardware devices. The Mobile-C library has a small footprint to satisfy the small memory requirement for various mechatronic and embedded systems. The interface between the binary and mobile agent spaces has been designed and implemented. The Mobile-C functionality in the binary space has been extended to the mobile agent space. Flexible synchronization mechanisms have been added in both binary and mobile agent spaces for concurrent execution and interaction of multiple mobile agents.

The rest of the article is organized as follows. Section 2 describes the design considerations for the Mobile-C library. Section 3 presents the architecture of the Mobile-C library with an illustration of its APIs. A proof of concept example is given in this section as well. Section 4 describes the interface allowing for agency-to-agent interaction through accessing the mobile agent space from the host program space. Section 5 extends the Mobile-C functionality into the mobile agent space, allowing for agent-to-agency and agent-to-agent interactions. Section 6 presents the Directory Facilitator (DF) in Mobile-C. An example is given to demonstrate the dynamic algorithm alteration methodology through the DF, which provides an algorithm experimentation functionality for computational steering. Section 7 illustrates the synchronization support in Mobile-C. Section 8 evaluates the performance of Mobile-C in terms of agent migration. Section 9 presents an application of dynamic runtime control of a mobile robot's behavior with mobile agents. Section 10 summarizes the investigation that has done.

2. Mobile-C library design

The Mobile-C library allows a mobile agent platform to be embedded in a program to support C/C++ mobile agents. This mobile agent platform is referred to as Mobile-C or the Mobile-C agency in this article. Additionally, the host program space is defined as the C/C++ binary space where a host program and the Mobile-C agency reside. The mobile agent space is defined as the C/C++ script space where a mobile agent resides.

An embeddable C/C++ interpreter, Ch [36–38], was chosen to be the Agent Execution Engine (AEE) to run C/C++ mobile agent codes. Using Ch as a runtime execution environment has several advantages over other alternatives. As a superset of C, all standard C functions are supported by Ch. Besides, an increasing number of C/C++ toolkits and packages are available for solving complicated engineering problems using Ch. Ch runs on most of the existing Windows and Unix-like operating systems for general purpose computers. It also runs on Linux operating systems for single board computers and tiny computers such as Gumstix [35]. It is suitable for use in a heterogeneous environment.

Since an agency is embedded in a host program to support mobile agents, a host program can protect itself from malicious agents by controlling the operation of the embedded agency and mobile agents. The Mobile-C library was designed to provide APIs relevant to an agency, different modules of an agency, agents, and other functionalities of an agency. These APIs can be called in a host program to have control over the embedded agency, different modules of the agency, and mobile agents operating within the agency.

There are many situations that cannot be foreseen at the development stage of a host program. A mobile agent can be dynamically created to provide solutions for those unexpected scenarios

to enrich the functionality of a host program by supplying new or updated services. Dynamic population and propagation are distinctive and natural advantages offered by mobile agents. Host program functionality enrichment is accomplished by providing mobile agents with the ability to interact with the host program space through a set of mobile agent space APIs. Therefore, the Mobile-C library was also designed to extend most of the functionality from the host program space to the mobile agent space.

For certain scientific and engineering applications, a host program can consist of an agency and many user-defined routines to perform a variety of complicated tasks or intensive computations. Some resources might be shared between different agents or between agents and user-defined routines. Thus, the Mobile-C library was also designed to support synchronization among mobile agents as well as synchronization between mobile agents and their host program.

One of the key features of a mobile agent is the social ability that allows a mobile agent to interact with other agents or host programs to accomplish its task on the user's behalf. However, there is generally little support to encourage interaction and coordination among multiple mobile agents [43]. With the interactivity in mind, the Mobile-C APIs were designed to fulfill agency-to-agency, agency-to-agent, agent-to-agency and agent-to-agent interactions.

3. Mobile-C library architecture

The functionalities of each API provided by the Mobile-C library is described in this section. Fig. 1 shows the architecture of the Mobile-C library. APIs in the Mobile-C library can be organized into eight categories: Agency, AMS, ACC, DF, AEE, Agent, Synchronization and Miscellaneous APIs.

3.1. Agency API

The main purpose of the Agency API is to start a Mobile-C agency inside a host program to support C/C++ mobile agents. When a Mobile-C agency is started, modules and data structures maintained by the agency are initialized as well. Afterwards, APIs in other categories can be used to access data structures associated with different modules.

Fig. 2 shows the Mobile-C agency data structure. As shown in Fig. 2, a Mobile-C agency maintains different threads and data structures in the host program space. As an IEEE FIPA compliant mobile agent platform, a Mobile-C agency comprises three FIPA normative modules, Agent Management System (AMS), Agent Communication Channel (ACC) and Directory Facilitator (DF). Two additional modules, Agent Execution Engine (AEE) and Agent Security Manager (ASM), are included in a Mobile-C agency as well. Each module has different functionalities that are implemented as

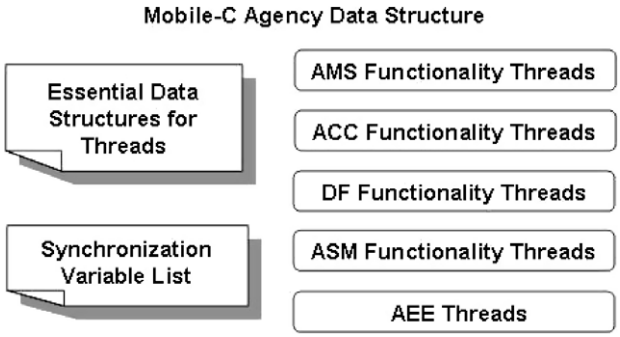

Fig. 2. Mobile-C agency data structure.

independent threads. These threads are classified into five categories: AMS functionality threads, ACC functionality threads, DF functionality threads, ASM functionality threads and AEE threads, as shown in Fig. 2. Each AEE thread is launched for one mobile agent by one of the AMS functionality threads.

Other representative functions of the Agency API include those for specifying which thread needs to be active or inactive when an agency is started, halting and resuming an agency's operation, and ending an agency.

3.2. AMS API

The AMS module controls the creation, registration, execution, migration, persistence and termination of a mobile agent. It maintains a directory of Agent Identifiers (AIDs) for registered mobile agents. Each mobile agent must register with the AMS module in order to have a valid AID. The AMS API is used to instruct the AMS module to perform operations related to the life cycle of an agent. The representative operations include indefinitely waiting and processing incoming agents, adding, removing, duplicating and retrieving agents.

3.3. ACC API

The ACC module routes messages between distributed entities. The ACC API is used to instruct the ACC module to perform operations regarding interactions between agents and agencies including inter-agent communication and inter-agency agent migration. The interactions can be performed through Agent Communication Language (ACL) messages. In Mobile-C, an agent migration message is an XML based ACL message that contains all the information about a mobile agent, such as the name, owner, source machine, code and data of the mobile agent. Unlike the previous standalone Mobile-C implementation [34] that used Libxml2 [44], Mini-XML [45], a small XML parsing library, is used in the Mo-

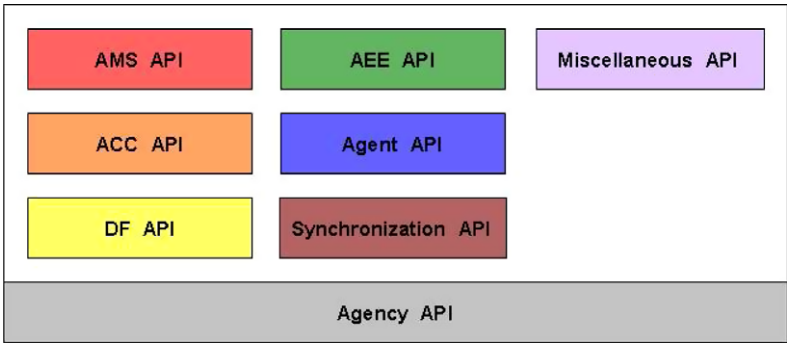

Fig. 1. Architecture of the Mobile-C library.

bile-C library to parse agent migration messages so that Mobile-C can more applicable to resource constrained applications.

3.4. DF API

The DF module provides yellow page services. Agents intended to advertise their services should register with the DF module. Visiting mobile agents can search the DF module for agents providing the services they desire. The DF API is used to instruct the DF module to perform operations regarding services provided by agents. Functions of this API can be used to register and deregister a service with the DF module, as well as search the DF module for a service.

3.5. AEE API

The AEE serves as the execution environment for mobile agent codes. The AEE has to be platform independent in order to support the execution of mobile agent codes in a heterogeneous environment. There will be multiple AEEs running concurrently in a Mobile-C agency to support multiple agents. Each AEE is associated with only one agent. The AEE API is used to instruct the AEE to perform operations on an agent. Some important functions of the AEE API include retrieving the AEE associated with an agent, obtaining and setting an agent's status, halting a running agent, and calling a function defined in an agent. In addition, all the functions of the Embedded Ch [46], a toolkit associated with the AEE, can be used to develop host programs.

3.6. Agent API

The Agent API is used to obtain the information regarding an agent. Representative functions in this API allow for retrieving an agent by its ID or name, obtaining an agent's ID, name, code and data.

3.7. Synchronization API

As shown in Fig. 2, Mobile-C agency maintains a list of synchronization variables that can be used with the Synchronization API to ensure synchronization within a host program where synchronization might be needed among agents, between agents and user-defined routines, as well as among user-defined routines.

3.8. Miscellaneous API

This API currently contains functions for setting up a steerable binary space function, retrieving a steering command sent from the mobile agent space, and send a command to control a steerable binary space function.

3.9. ASM module

The ASM module is responsible for maintaining security policies for a host program. Some sample tasks of the ASM module include identifying users, protecting host resources, authenticating and authorizing mobile agents, and ensuring the security and integrity of mobile agents. The protocol of the ASM module is based on the SSH protocol [47]. Currently, this Mobile-C module provides a secure transmission process for mobile agents and ACL messages from one agency to another. Two agencies must successfully authenticate each other before a transmission process begins. A successful authentication establishes a trust between these two agencies. A mobile agent or ACL message to be sent out will be encrypted so that its integrity can be maintained upon being received. Therefore, the ASM module helps protect against man-in-the-middle attacks and eavesdropping. The ASM module can be enabled or disabled during program compilation with the Mobile-C library. There is no separate ASM API needed. For a resource constrained application running in a secure intranet, the ASM module can be disabled.

3.9.1. Example 1: sample application programs

The concept of embedding a Mobile-C agency into an application to support code mobility is illustrated in this example. Program 1 starts an agency that keeps receiving mobile agents and executing mobile agent codes. The variable `agency`, of type `MCAgency_t`, is a handle that contains information about an agency. The function `MC_Initialize()` takes two parameters, an integer and the address of an `MCAgencyOptions_t` variable. An `MCAgencyOptions_t` variable is a structure that contains information specified by a user regarding the threads to be activated, default agent status, and settings for the ASM module. Here, a `NULL` pointer is passed to `MC_Initialize()` to start an agency with default settings. A local agency will be initialized to listen on port `5130`. The function `MC_MainLoop()` makes the agency continuously receive mobile agents and execute mobile agent codes. As shown in Program 1, the function `MC_End()` will be called to terminate the agency when `MC_MainLoop()` fails.

Program 2 starts an agency that sends a mobile agent to a remote agency. In Mobile-C, a mobile agent is an ACL message in XML format. The function `MC_SendAgentMigrationMessageFile()` takes four parameters including an `MCAgency_t` variable, the filename of a mobile agent, and the host name and port number of a remote agency. Here, `MC_SendAgentMigrationMessageFile()` sends a mobile agent, saved as `mobileagent_ex1.xml`, to a remote agency which runs on host `iel2.engr.ucdavis.edu` and listes on port `5130`. The mobile agent is shown in Program 3. The tag `MOBILEC_MESSAGE` indicates that the following content is a Mobile-C message. The tag `MESSAGE` reveals the message type through the attribute `message`. Here, the message type is `MOBILE_AGENT`. The tag `MOBILE_AGENT` indicates that the following content is a mobile agent. The tag `AGENT_DATA` embraces all the information pertinent to an agent including the name, owner, home address and tasks of an agent. Specifically, the tag `TASKS` reveals the number of tasks an agent has and the number of tasks an agent has completed via attributes `task` and `num`, respectively. Here, the agent has one task which has not been completed. Furthermore, the tag `TASK` reveals information pertinent to one task of an agent, such as the ordinal number, return variable, completeness, persistence and execution host of the task, through different attributes. Particularly, the attribute `persistent` specifies the persistence of an agent. This attribute can be set to 1 for a persistent agent, and 0 or removed for a non-persistent agent. Here, the agent shown in Program 3 is a per-

```
#include <libmc.h>

int main() {
    MCAgency_t agency;
    int local_port = 5130;

    agency = MC_Initialize(local_port, NULL);
    if(MC_MainLoop(agency) != 0) {
        MC_End(agency);
        return -1;
    }
    return 0;
}
```

Program 1. The server program in Example 1.

```
#include <libmc.h>

int main() {
    MCAgency_t agency;
    int local_port = 5050, server_port = 5130;
    char *file_name = "mobileagent_ex1.xml";
    char *server_name = "iel2.engr.ucdavis.edu";

    agency = MC_Initialize(local_port, NULL);
    MC_SendAgentMigrationMessageFile(agency, file_name, server_name, server_port);
    MC_End(agency);
    return 0;
}
```

Program 2. The client program in Example 1.

```
<?xml version="1.0"?>
<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">
<MOBILEC_MESSAGE>
 <MESSAGE message="MOBILE_AGENT">
  <MOBILE_AGENT>
   <AGENT_DATA>
    <NAME>mobileagent1</NAME>
    <OWNER>IEL</OWNER>
    <HOME>bird1.engr.ucdavis.edu:5050</HOME>
    <TASKS task="1" num="0">
     <TASK num="0"
           return="no-return"
           complete="0"
           persistent="1"
           server="iel2.engr.ucdavis.edu:5130">
     </TASK>
     <AGENT_CODE>
      <![CDATA[
#include <stdio.h>
int main() {
    printf("Hello World!\n");
    return 0;
}
int func() {
    printf("func() is called.\n");
    return 0;
}
      ]]>
     </AGENT_CODE>
    </TASK>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</MOBILEC_MESSAGE>
```

Program 3. A mobile agent in Example 1.

sistent agent. A persistent agent will not be removed from an agency after the agent code is executed so that the variables and functions in the agent code can still be accessed later on. The tag `AGENT_CODE` embraces a C/C++ code representing the task that an agent will perform. In this example, after the mobile agent is received, the encapsulated agent code will be executed to print `Hello World!` to the console terminal. The agent code also defines a function, `func()`, that can be called from the host program space. The details about accessing the mobile agent space from the host program space will be described in the next section.

4. Access mobile agent space from host program space

The ability to access the mobile agent space from the host program space can create benefits for a host program. One of the benefits is relevant to the security of a host program. As mentioned in Section 2, a host program can protect itself from malicious agents by controlling the operation of mobile agents. Because mobile agents operate within an agency which is embedded in a host program, the agency-to-agent interaction therefore plays an important role in securing a host program. The agency-to-agent interaction allows an agency to have complete control over a mobile agent code. Therefore, through the embedded agency, a host program can control the execution and debugging of a mobile agent code and evaluate intermediate status or dynamic properties of an agent by accessing variables or functions defined in the agent code during runtime.

The Mobile-C library provides APIs to carry out such an agency-to-agent interaction. These APIs can be called in a host program to control not only mobile agents, but also the Mobile-C agency and its different modules. Additionally, since Ch is adopted as the AEE in Mobile-C, a large number of Embedded Ch functions [46] can be readily used in the host program space to access information existing in the mobile agent space.

The Embedded Ch toolkit reduces the complexity of heterogeneous development environment for both embedded scripting and applications. With the consistent C/C++ code base, it can significantly reduce the effort in the software development and maintenance. Moreover, through the Embedded Ch toolkit, C/C++ applications can be extended with all the features of Ch including built-in string type for scripting. The pointer and time deterministic nature of the C language provide a perfect interface with hardware in real-time systems.

4.1. Example 2: invoke mobile agent space function from host program space

This example illustrates how to call a function defined in the mobile agent code by using the Mobile-C library and Embedded Ch toolkit. The mobile agent in this example, shown in Program 3, is a persistent agent which will not be removed from the agency after the agent code is executed. The agent code is a simple but complete C program which also defines a function `func()` that is called in the server program shown in Program 4.

In Program 4, the function `MC_WaitSignal()` blocks program execution until a given signal is received. The signal `MC_EXEC_AGENT` is used to unblock program execution once the first received agent is executed. The function `MC_ResetSignal()` is called to resume the operation of the agency. The mobile agent is found by the function `MC_FindAgentByName()`, and the AEE associated with

```
#include <stdio.h>
#include <libmc.h>
#include <embedch.h>

int main() {
    MCAgency_t agency;
    MCAgent_t agent;
    ChInterp_t interp;
    int local_port = 5130, retval;

    agency = MC_Initialize(local_port, NULL);
    MC_WaitSignal(agency, MC_EXEC_AGENT);
    MC_ResetSignal(agency);

    agent = MC_FindAgentByName(agency, "mobileagent1");
    interp = (ChInterp_t)MC_GetAgentExecEngine(agent);
    Ch_CallFuncByName(interp, "func", &retval);
    MC_End(agency);
    return 0;
}
```

Program 4. The server program in Example 2.

the mobile agent is obtained by the function `MC_GetAgentExecEngine()`. The variable returned by `MC_GetAgentExecEngine()` is a Ch interpreter of type `ChInterp_t`. There are several different methods to call functions defined in the mobile agent space from the host program space using the Embedded Ch toolkit. In this example, the function `func()` is invoked by its name through an Embedded Ch function `Ch_CallFuncByName()` to print `func() is called!` to the console terminal.

5. Extend Mobile-C functionality to mobile agent space

The key concepts of mobile agents are their interoperability and autonomy. These concepts set mobile agents apart from conventional objects in such a way that mobile agents should be able to refuse an action. Therefore, mobile agents must be able to interact with each other to decide what information to retrieve or what physical action to take, such as shutting down an assembly line or avoiding a collision with another robot. These interactions occur among agents as well as between agents and host programs. These interactions are achieved by providing mobile agents with the ability to interact with the host program space through the mobile agent space APIs. The Mobile-C library has extended most of the functionality from the host program space to the mobile agent space. The mobile agent space APIs allow a mobile agent to interact with an agency, different modules of an agency, and other agents.

Fig. 3 shows how mobile agent code interfaces with the Mobile-C library. When the function `mc_Function()` is called in mobile agent code, Ch searches the corresponding interface function `MC_Function_chdl()` in the Mobile-C library, and passes arguments to it by calling the function. Subsequently, the interface function `MC_Function_chdl()` invokes the target function `MC_Function()`, and passes the return value back to the mobile agent space [46].

The prototypes of mobile agent space functions and a number of enumerations, data types, and special variables are all considered *built-in* in the mobile agent space as no header file or extra code is needed to access them. They are declared using the Embedded Ch toolkit.

6. Directory Facilitator in Mobile-C

As mentioned in Section 3, the DF in Mobile-C provides yellow page services. Agents intended to advertise their services should register with the DF. Visiting mobile agents can search the DF for agents providing their desired services. The Mobile-C library provides functions to instruct the DF to perform operations including registering and deregistering a service with the DF, as well as searching the DF for a desired service.

For pervasive systems consisting of small devices with limited communication and processing power, the functionality of Directory Facilitator is important. Each one of these small devices typically offers a specific service to the user, and they need to collaborate with each other to build up more complex services. In order to achieve this, devices should be able to dynamically discover and share their services. For example, sensors and air conditioning systems in an intelligent building might interact with a user's PDA to automatically adapt the environment to a user's need or preference.

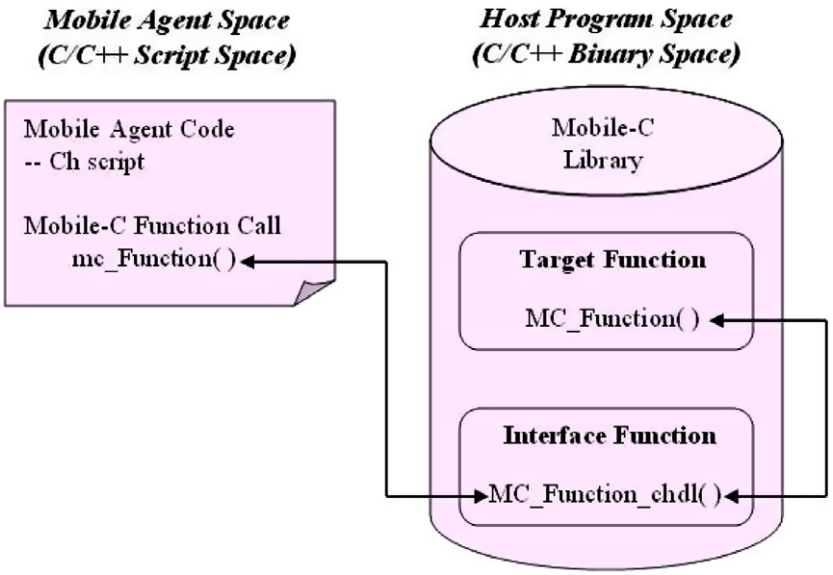

Fig. 3. Interface of mobile agent code with the Mobile-C library.

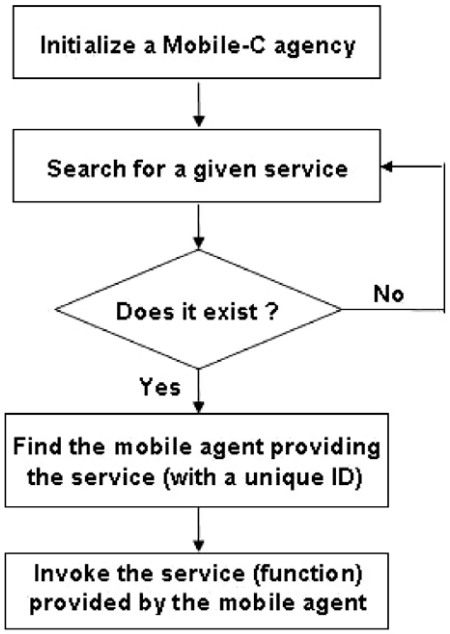

Fig. 4. The flowchart for the server program in Example 3.

6.1. Example 3: dynamic algorithm alteration

A simulation program typically consists of multiple functions, each of which performs a specific computation using an algorithm. Therefore, support of dynamic algorithm alteration allows a user to change the implementation of functions while a simulation is in progress. With the Mobile-C library, each function in a simulation program can be defined in the mobile agent code and registered as a service with the DF, so that an algorithm can be altered by replacing the current service with a new one containing the modified function implementation [48]. This section gives an example that demonstrates how to dynamically change an algorithm in a running simulation through the Mobile-C library, which cannot be accomplished by some commonly-used computational steering libraries [49–52]. In this example, there are two mobile agents sent from a client program to the server program. The two mobile agents have the same mobile agent code except for the difference in the function to be registered with the DF. The programs for this example can be downloaded from the Web [53].

As shown in Fig. 4, the server program repeatedly searches for a service and calls a mobile agent space function that represents the service. A Mobile-C agency is initialized as a low priority back-end thread to receive and execute mobile agents to update the function.

Fig. 5 illustrates the flowchart for the main function of the mobile agent code. The main function first searches for a given service. If no mobile agent has provided the service, the service provided by the current mobile agent will be registered with the DF. On the other hand, if an existing mobile agent is found to provide the service, the service provided by this mobile agent will be deregistered. Afterwards, the current mobile agent's service will be registered with the DF.

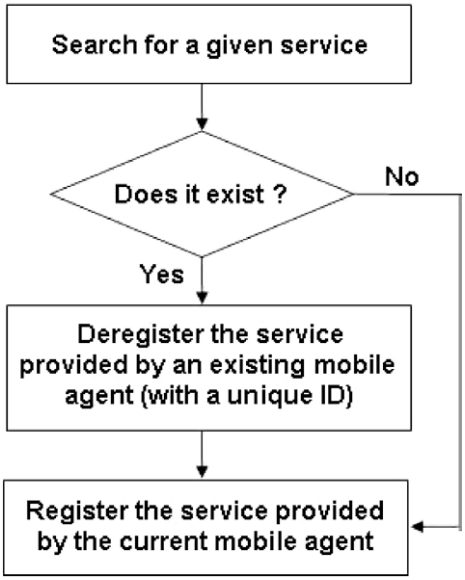

Fig. 5. The flowchart for the main function of the mobile agent code in Example 3.

```
<?xml version="1.0"?>
<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">
<MOBILEC_MESSAGE>
 <MESSAGE message="MOBILE_AGENT">
  <MOBILE_AGENT>
   <AGENT_DATA>
    <NAME>mobileagent1</NAME>
    <OWNER>IEL</OWNER>
    <HOME>bird1.engr.ucdavis.edu:5050</HOME>
    <TASKS task="1" num="0">
     <TASK num="0"
           return="no-return"
           persistent="1"
           complete="0"
           server="iel2.engr.ucdavis.edu:5130">
     </TASK>
     <AGENT_CODE>
      <![CDATA[
int data;

int main() {
    int mutex_id = 55;
    mc_SyncInit(mutex_id);
    return 0;
}

void WriteData(int i) {
    data += i;
    if(data > 1000) {
        data = 0;
    }
}

int ReadData() {
    return data;
}
      ]]>
     </AGENT_CODE>
    </TASKS>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</MOBILEC_MESSAGE>
```

Program 5. A mobile agent that contains a global variable and defines functions to access the global variable in Example 4.

7. Synchronization support in Mobile-C

In a Mobile-C agency, mobile agents are executed by independent AEEs. A user might also need to design a multi-threaded application where a Mobile-C agency itself is one of the many threads that handle different tasks. The Mobile-C library supports synchronization among mobile agents and threads. The synchronization API functions are used to protect shared resources as well as to provide a method of deterministically timing the execution of mobile agents and threads.

The internal implementation consists of a linked list of Portable Operating System Interface for UNIX (POSIX) compliant synchronization variables, namely, mutexes, condition variables and semaphores. Each node in the linked list is a synchronization variable which is assigned or given a unique identification number. The API functions can be called from the binary or mobile agent space to initialize the synchronization variables and access them by their unique identification numbers in the list.

As opposed to traditional synchronization variables, a Mobile-C synchronization variable is an abstract variable. Once it has been initialized, it may be used as a mutex, condition variable, or semaphore. No further function calls are necessary to change a generic synchronization variable to one of the types. However, once a synchronization variable is used as a mutex, condition variable, or semaphore, it should not be used again as a different type.

The example below demonstrates the ability of a Mobile-C mutex to protect a resource that may be shared between two agents. Any real or imaginary resource that should not be accessed simultaneously by more than one entity at a time should be guarded by a mutex. The resource may be a shared variable, or something more abstract such as control of a robot arm. If there is only one robot arm, then only one entity, an agent in this case, should be able to control it at a time. In the following example, the tasks of agents include reading and writing a global variable existing in an agent code.

7.1. Example 4: synchronization using mutex in mobile agent space

As shown in Program 5, the mobile agent `mobileagent1` initializes a mutex with an ID `55` via the function `mc_SyncInit()` and defines two functions, `WriteData()` and `ReadData()`, for writing and reading a global variable, `data`. As shown in Program 6, the mobile agent `mobileagent2` continuously performs a writing operation. The operation includes locking the mutex via the function `mc_MutexLock()`, writing the global variable by calling `WriteData()` through the function `mc_CallAgentFunc()`, and unlocking the mutex via the function `mc_MutexUnlock()`. Likewise, as shown in Program 7, the mobile agent `mobileagent3` locks the mutex, reads the global variable, and unlocks the mutex afterwards. By using a mutex, it is guaranteed that the global var-

```
<?xml version="1.0"?>
<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">
<MOBILEC_MESSAGE>
 <MESSAGE message="MOBILE_AGENT">
  <MOBILE_AGENT>
   <AGENT_DATA>
    <NAME>mobileagent2</NAME>
    <OWNER>IEL</OWNER>
    <HOME>bird1.engr.ucdavis.edu:5050</HOME>
    <TASKS task="1" num="0">
     <TASK num="0"
           return="no-return"
           complete="0"
           server="iel2.engr.ucdavis.edu:5130">
     </TASK>
     <AGENT_CODE>
      <![CDATA[
#include <stdio.h>
int main() {
    MCAgent_t agent;
    int i = 0, mutex_id = 55, retval;
    agent = mc_FindAgentByName("mobileagent1");
    while(1) {
        mc_MutexLock(mutex_id);
        mc_CallAgentFunc(agent, "WriteData", NULL, i);
        mc_MutexUnlock(mutex_id);
        i++;
        if(i == 20) {
            i = 0;
        }
    }
    return 0;
}
      ]]>
     </AGENT_CODE>
    </TASKS>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</MOBILEC_MESSAGE>
```

Program 6. A mobile agent that continuously updates a variable in Example 4.

```
<?xml version="1.0"?>
<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">
<MOBILEC_MESSAGE>
 <MESSAGE message="MOBILE_AGENT">
  <MOBILE_AGENT>
   <AGENT_DATA>
    <NAME>mobileagent3</NAME>
    <OWNER>IEL</OWNER>
    <HOME>bird1.engr.ucdavis.edu:5050</HOME>
    <TASKS task="1" num="0">
     <TASK num="0"
           return="no-return"
           complete="0"
           server="iel2.engr.ucdavis.edu:5130">
     </TASK>
     <AGENT_CODE>
      <![CDATA[
#include <stdio.h>
int main() {
    MCAgent_t agent;
    int i, mutex_id = 55, retval;
    agent = mc_FindAgentByName("mobileagent1");
    mc_MutexLock(mutex_id);
    mc_CallAgentFunc(agent, "ReadData", &retval, NULL);
    printf("data: %d\n", retval);
    mc_MutexUnlock(mutex_id);
    return 0;
}
      ]]>
     </AGENT_CODE>
    </TASKS>
   </AGENT_DATA>
  </MOBILE_AGENT>
 </MESSAGE>
</MOBILEC_MESSAGE>
```

Program 7. A mobile agent that reads a variable of another agent (Program 5) in Example 4.

iable `data` of `mobileagent1` is only accessed by one agent at a time.

8. Performance of agent migration in Mobile-C

The agent mobility of a mobile agent platform is relevant to and affected by the agent migration performance of that platform. The performance of agent migration is specifically important to applications with a large number of mobile agents. A number of benchmark tests were proposed to evaluate the performance of agent migration for JADE [54,31]. Therefore, this section presents performance comparison of agent migration for Mobile-C and JADE using three essential benchmark tests proposed for JADE. The agent migration benchmark programs for both Mobile-C and JADE can be downloaded from the Web [55].

Each of the three experiments simulates a relay-race. Interactions between any two runner agents in the same team are accomplished by exchanging ACL messages. A homogeneous environment is set up to perform the tests. The environment consists of four identical Linux machines. Each machine has an Intel Pentium 4 processor running at 3.2 GHz and 512 Mb of RAM. All these machines are connected through a switch with a 100 Mbit/s transmission rate.

8.1. Test 1: four agent teams with different numbers of agencies

The first test is where four agent teams migrate among different numbers of agencies. The number of agencies ranges from two to six. For situations where there are two to four agencies, each machine has one agency running on it. For situations where there are five and six agencies, there are two machines that have two agencies running on them. The migration routes for five and six agencies are set up in a way that no agents will migrate between two agencies running on the same machine.

Each team has one trigger agent and the same number of runner agents as that of agencies. For each team, at the beginning, each runner agent stands by on each agency. The trigger agent is then sent to the first agency to signal the runner agent to start the relay-race. The trigger agent becomes another runner agent once it signals the runner agent to depart from the first agency. In order to make all the runner agents leave the first agency almost simultaneously, a trigger agent does not signal its runner agent until all the other trigger agents arrive the first agency. Each team is considered to complete one lap when the runner agent on the last agency arrives the first agency. A total of five laps need to be completed by each team. The moment when all the trigger agents arrive the first agency to start the first lap is recorded as the start time. The moment when all the runner agents on the last agency arrive the first agency to complete the fifth lap is recorded as the stop time. A migration time is calculated as the interval between the start time and stop time.

Fig. 6 shows the migration times of Mobile-C and JADE for test 1. Each migration time is obtained by averaging 20 migration results. The migration times of Mobile-C are `716.5`, `1076.4`, `1417.85`, `1716.25` and `2064.75` milliseconds for two, three, four, five and six agencies, respectively. Likewise, the migration times of JADE are `1684`, `2071.5`, `2533.5`, `2797.5` and `2064.75` milliseconds for two, three, four, five, and six agencies, respectively. Therefore, the migration times of Mobile-C are about `42.55%`, `51.96%`, `55.96%`, `61.35%` and `63.79%` of the corresponding ones of JADE for two to six agencies. The results also show that the performance of Mobile-C is more linear than JADE.

8.2. Test 2: four agencies with different numbers of agent teams

The second test is where different numbers of agent teams migrate among four agencies. The number of agent teams ranges from one to six. Each machine has one agency running on it. Each team has one trigger agent and four runner agents. The methods for starting the relay-race and calculating a migration time are the same as those in test 1.

Fig. 7 shows the migration times of Mobile-C and JADE for test 2. Each migration time is obtained by averaging twenty migration results. The migration times of Mobile-C are calculated to be `34.61%`, `45.24%`, `50.02%`, `57.73%`, `58.9%` and `48.94%` of the corresponding ones of JADE for one to six agent teams.

8.3. Test 3: four agencies with large numbers of agent teams

The third test is the same as test 2 except that large numbers of agent teams are involved in the relay-race. The numbers of agent teams in test 3 are 20, 40, 60, 80 and 100. Each machine has one agency running on it. Each team has one trigger agent and four runner agents. The methods for starting the relay-race and calculating a migration time are the same as those in test 1.

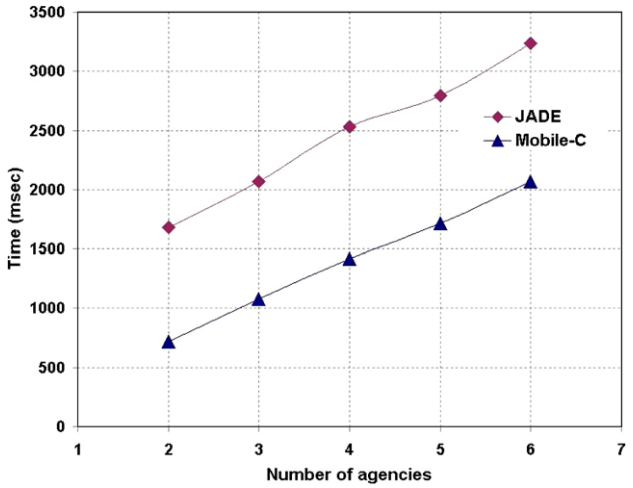

Fig. 6. Migration times for four agent teams with different number of agencies.

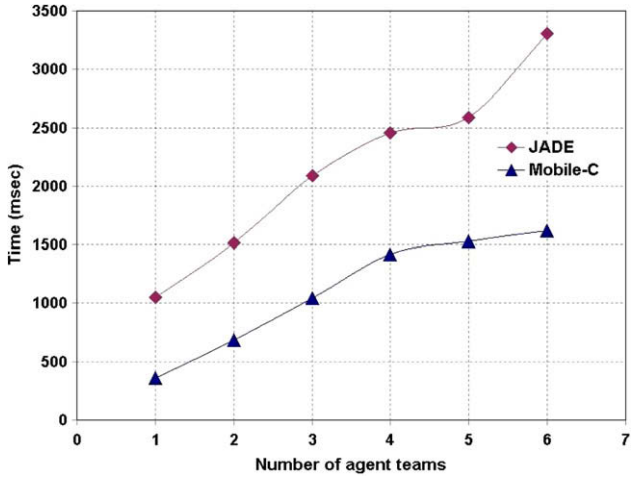

Fig. 7. Migration times for different numbers of agent teams with four agencies.

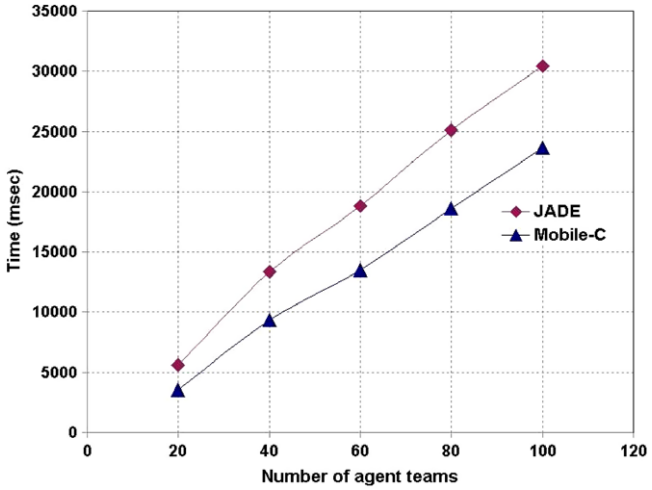

Fig. 8. Migration times for different large numbers of agent teams with four agencies.

Fig. 8 shows the migration times of Mobile-C and JADE for test 3. Each migration time is obtained by averaging twenty migration results. The migration times of Mobile-C are about 62.46%, 69.87%, 71.8%, 74.01% and 77.82% of the corresponding ones of JADE for 20 to 100 agent teams.

As shown in Figs. 6–8, the performance of agent migration in Mobile-C is better than that in JADE for different numbers of agencies and different numbers of agent teams in a homogeneous environment consisting of networked Linux platforms.

9. Application: mobile agent-based dynamic runtime behavior control for a mobile robot

Robots are growing in complexity and their use in industry is becoming more widespread. One major use of robots has been in the automation of mass production industries, where the same, definable tasks must be performed repeatedly in exactly the same fashion. Robots are appropriate for such tasks because the tasks can be accurately defined and performed the same every time, with little need for feedback to perform the same process. Mobile robots are increasingly being deployed to perform tasks that are unpleasant or dangerous for human beings, for instance, bomb disposal, space or undersea exploration, mining and cleaning of toxic waste. Uncertain and unforeseen events are very likely to occur in those unstructured environments. A mobile robot often cannot be preprogrammed to handle those uncertainties. Mobile agents allow mobile robots to rapidly respond to unanticipated events. With the Mobile-C library, which uses Ch as the AEE, a robot control program can be directly sent to a mobile robot for execution.

The use of the Mobile-C library to control and change a mobile robot's behavior on the fly has been validated through a real-world experiment with a K-Team Khepera III mobile robot [56]. The Khepera III mobile robot is equipped with the KoreBot board, an ARM based computer, in this experiment. The KoreBot board has 64 Mbytes of RAM and an embedded Linux operating system which provides a standard GNU C/C++ environment for the development of applications using the Mobile-C library. Through the KoreBot board, the Khepera III mobile robot is also able to host a standard CompactFlash extension card supporting Wi-Fi, Bluetooth, or extra storage space. The Khepera III mobile robot base includes nine infrared sensors for obstacle detection and five ultrasonic sensors for long range object detection.

In this experiment, a program with an embedded agency is running in the mobile robot. Agent `service_provider_1` containing

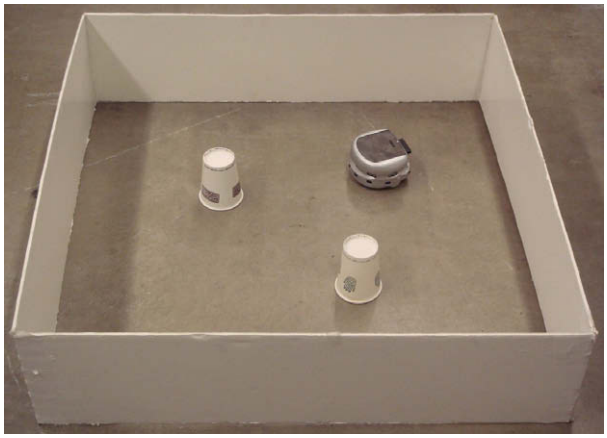

Fig. 9. An experimental environment for Khepera III mobile robot to perform obstacle avoidance.

five functions is first sent to the robot to register those functions as services with the DF. Agent `mobagent1` is then sent to the robot, triggering the registered services to make the robot avoid obstacles. The service that leads to this obstacle avoidance behavior is represented and provided by function `RobotBehaviour()`. The implementation of this function is based on the Braitenberg obstacle avoidance algorithm [57]. The experimental environment for the Khepera III robot is a square area formed by four cardboard pieces with two cups in it as obstacles, as shown in Fig. 9. Therefore, the mobile robot will wander through the field avoiding the walls and cups.

While the mobile robot is wandering through the field and avoiding obstacles, agent `service_provider_2` carrying another `RobotBehaviour` service is sent to the robot. The intention of sending this agent is to change the robot behavior on the fly. Therefore, agent `service_provider_2` deregisters the `RobotBehaviour` service provided by agent `service_provider_1` and registers its own `RobotBehaviour` service with the DF, according to the methodology illustrated in Section 6.

The second `RobotBehaviour` service will cause the robot to follow a moving object. Therefore, once agent `service_provider_2` is sent to the robot, the walls and one of the cups are removed. The remaining cup is dragged by a hand around the robot for it to follow [58]. The object following behavior is implemented by modifying the Braitenberg obstacle avoidance algorithm. The sensor weights of the Braitenberg obstacle avoidance algorithm are swapped to drive the opposite motor rather than the one originally specified. Also, the weighted sensor values, which are directly associated with the distance between the robot and the cup, are negated and then summed up to determine the speed of each motor. These modifications provide the opposite effect of obstacle avoidance by making the robot move towards the cup. In addition, the speed limit is set so that the robot will not bump into the followed target. Therefore, the robot will follow the moving cup until it reaches the speed limit, for either the left or right motor, at which time it will stop.

The program flowchart of agent `mobagent1` is identical to Fig. 4 for the part regarding `RobotBehaviour` service. Therefore, the mobile robot behavior is changed seamlessly on the fly without the need to stop and modify the running agent `mobagent1`. Besides, because agents `service_provider_1` and `service_provider_2` are both persistent agents, the deregistered `RobotBehaviour()` service that belongs to agent `service_provider_1` still exists inside the agency that runs in the robot. Thus, if needed, the deregistered `RobotBehaviour()` service that allows for obstacle avoidance can be easily registered with the DF

again through a mobile agent to replace the current `RobotBehaviour()` service.

Using the Mobile-C library, robot behaviors can be easily modified by simply sending over a new mobile agent to the mobile robot. The application programs described in this section can be downloaded from the Web [58].

10. Conclusions

The design, implementation and application of the Mobile-C library have been presented in this paper. Using the Mobile-C library, an IEEE FIPA compliant mobile agent platform can be embedded in an application to support C/C++ mobile agents. Although it is a general purpose mobile agent platform, Mobile-C is especially developed for resource constrained applications in mechatronic and embedded systems. The Mobile-C library has a small footprint and is supported in multiple platforms. The Mobile-C APIs in the binary space along with a large number of the Embedded Ch APIs provide a rich interface for agency-to-agent interaction and allow a host program to have complete control over the execution and bebugging of the mobile C/C++ code carried by mobile agents. A mobile agent can be dynamically created to provide solutions for those situations that are not anticipated at the development stage of a host program. The additional functionality of the host program is achieved by using mobile agents which can interact with the host program through a set of Mobile-C APIs in the agent space. The Directory Facilitator of Mobile-C provides a powerful algorithm experimentation capability for computational steering that can dynamically change an algorithm in a running simulation, which cannot be accomplished by other computational steering libraries. The Mobile-C library supports synchronization among mobile agents and threads. The synchronization functions protect shared resources and provide a way of deterministically timing the execution of mobile agents and threads. A set of agent migration tests reveals that the performance of agent migration in Mobile-C is about two times faster than JADE for different numbers of agencies and different numbers of agent teams. A dynamic runtime control of a mobile robot's behavior demonstrates the potential applications of the Mobile-C library in a wide variety of mechatronic and embedded systems.

References

[1] F. Zambonelli, H.V.D. Parunak, Signs of a revolution in computer science and software engineering, in: Engineering Societies in the Agents World III: Third International Workshop (Lecture Notes in Computer Science), vol. 2577, 2003, pp. 13–28.

[2] N.R. Jennings, 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 44 (4) (2001) 35–41.

[3] M. Wooldridge, P. Ciancarini, Agent-oriented software engineering: the state of the art, in: Proceedings of Agent-Oriented Software Engineering: The First International Workshop (Lecture Notes in Computer Science), vol. 1957, 2001, pp. 1–28.

[4] W. Shen, D. Xue, D.H. Norrie, An agent-based manufacturing enterprise infrastructure for distributed integrated intelligent manufacturing systems, in: Proceedings of the 3rd International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-98), London, UK, 1998, pp. 533–548.

[5] H. Wada, S. Okada, An autonomous agent approach for manufacturing execution control systems, Integrated Computer-Aided Engineering 9 (3) (2002) 251–262.

[6] H.V.D. Parunak, A.D. Baker, S.J. Clark, The AARIA agent architecture: from manufacturing requirements to agent-based system design, Integrated Computer-Aided Engineering 8 (1) (2001) 45–58.

[7] S.Y. Eo, T.S. Chang, D.G. Shin, E.S. Yoon, Cooperative problem solving in diagnostic agents for chemical processes, Computers and Chemical Engineering 24 (2–7) (2000) 729–734.

[8] N.R. Jennings, S. Bussmann, Agent-based control systems – why are they suited to engineering complex systems?, IEEE Control Systems Magazine 23 (3) (2003) 61–73

[9] R.W. Brennan, M. Fletcher, D.H. Norrie, An agent-based approach to reconfiguration of real-time distributed control systems, IEEE Transactions on Robotics and Automation 18 (4) (2002) 444–451.

[10] M. Yokoo, S. Fujita, Trends of internet auctions and agent-mediated web commerce, New Generation Computing 19 (4) (2001) 369–388.

[11] T. Sandholm, eMediator: a next generation electronic commerce server, Computational Intelligence 18 (4) (2002) 656–676.

[12] S.P.M. Choi, J. Liu, S. Chan, A genetic agent-based negotiation system, Computer Networks: The International Journal of Computer and Telecommunications Networking 37 (2) (2001) 195–204.

[13] W.E. Chen, C. Hu, A mobile agent-based active network architecture for intelligent network control, Information Sciences 141 (1–2) (2002) 3–35.

[14] L. Chou, K. Shen, K. Tang, C. Kao, 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) 2744 (2003) 280–289.

[15] F. Logi, S.G. Ritchie, A multi-agent architecture for cooperative inter-jurisdictional traffic congestion management, Transportation Research Part C—Emerging Technologies 10 (5–6) (2002) 507–527.

[16] J.Z. Hernandez, S. Ossowski, A. Garcia-Serrano, Multiagent architectures for intelligent traffic management systems, Transportation Research Part C—Emerging Technologies 10 (5–6) (2002) 473–506.

[17] K. Stathis, O. DeBruijn, S. Macedo, Living memory: agent-based information management for connected local communities, Interacting with Computers 14 (6) (2002) 663–688.

[18] H. Tu, J. Hsiung, An architecture and category knowledge for intelligent information retrieval agents, Decision Support Systems 28 (3) (2000) 255–268.

[19] L. Boloni, D.C. Marinescu, J.R. Rice, P. Tsompanopoulou, E.A. Vavalis, Agent based scientific simulation and modeling, Concurrency Practice and Experience 12 (9) (2000) 845–861.

[20] H. Casanova, J. Dongarra, Using agent-based software for scientific computing in the netsolve system, Parallel Computing 24 (12–13) (1998) 1777–1790.

[21] J. Huang, N.R. Jennings, J. Fox, Agent-based approach to health care management, Applied Artificial Intelligence 9 (4) (1995) 401–420.

[22] I. Noda, P. Stone, The RoboCup soccer server and CMUnited clients: implemented infrastructure for MAS research, Autonomous Agents and Multi-Agent Systems 7 (1–2) (2003) 101–120.

[23] J.L. Adler, V.J. Blue, A cooperative multi-agent transportation management and route guidance system, Research Part C—Emerging Technologies 10 (5–6) (2002) 433–454.

[24] A. Fuggetta, G.P. Picco, G. Vigna, Understanding code mobility, IEEE Transactions on Software Engineering 24 (5) (1998) 342–361.

[25] J. Baumann, F. Hohl, K. Rothermel, M. Strasser, W. Theilmann, MOLE: a mobile agent system, Software-Practice and Experience 32 (6) (2002) 575–603.

[26] D.B. Lange, M. Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley, MA, 1998.

[27] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, B. Peet, Concordia: an infrastructure for collaborating mobile agents, in: Proceedings of the 1st International Workshop on Mobile Agents (MA'97, Lecture Notes in Computer Science), vol. 1219, 1997, pp. 86–97.

[28] R.S. Gray, G. Cybenko, D. Kotz, R.A. Peterson, D. Rus, D'Agents: applications and performance of a mobile–agent system, Software-Practice and Experience 32 (6) (2002) 543–573.

[29] H. Peine, Application and programming experience with the ara mobile agent system, Software-Practice and Experience 32 (6) (2002) 515–541.

[30] D. Johnansen, K.J. Lauvset, R. van Renesse, F.B. Schneider, N.P. Sudmann, K. Jacobsen, A TACOMA retrospective, Software-Practice and Experience 32 (6) (2002) 605–619.

[31] F. Bellifemine, G. Caire, A. Poggi, G. Rimassa, JADE: a software framework for developing multi-agent applications. Lessons learned, Information and Software Technology 50 (1–2) (2008) 10–21.

[32] JADE - Java Agent Development Framework. <http://jade.tilab.com/>.

[33] B. Chen and H.H. Cheng, A run-time support environment for mobile agents, in: Proceeding of ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications, No. DETC2005-85389, Long Beach, California, (September) 2005.

[34] B. Chen, D. Linz, H.H. Cheng, XML-based agent communication, migration and computation in mobile agent systems, Journal of Systems and Software 81 (8) (2008) 1364–1376.

[35] Mobile-C: A Multi-Agent Platform for Mobile C/C++ Code. <http://www.mobilec.org> (accessed 1.08.09).

[36] H.H. Cheng, Scientific computing in the Ch programming language, Scientific Programming 2 (3) (1993) 49–75.

[37] H.H. Cheng, Ch: A C/C++ interpreter for script computing, C/C++ User's Journal 24 (1) (2006) 6–12.

[38] H.H. Cheng, Ch — an Embeddable C/C++ Interpreter. <http://www.softintegration.com> (accessed 15.04.09).

[39] FIPA: The Foundation for Intelligent Physical Agents. <http://www.fipa.org/> (accessed 29.10.08).

[40] Object Management Group, Mobile Agent System Interoperability Facilities Specification. <http://www.omg.org/docs/orbos/97-10-05.pdf>.

[41] MACE – Mobile Agent Code Environment. <http://wwwagss.informatik.uni-kl.de/Projekte/Ara/mace.html> (accessed 10.08.04).

[42] N.P. Sudmann, D. Johansen, Adding mobility to non-mobile web robots, in: Proceedings of the IEEE ICDCS00 Workshop on Knowledge Discovery and Data Mining in the World-Wide Web, Taipei, Taiwan, 2000, pp. 73–79.

[43] D.L. Martin, A.J. Cheyer, D.B. Moran, The open agent architecture: a framework for building distributed software systems, Applied Artificial Intelligence 13 (1–2) (1999) 91–128.
[44] The XML C parser and toolkit of Gnome. <http://xmlsoft.org/index.html>.
[45] Mini-XML Home Page. <http://www.easysw.com/~mike/mxml/>.
[46] Embedded Ch, SoftIntegration, Inc. <http://www.softintegration.com/products/sdk/embedded_ch/> (accessed 15.04.09).
[47] D.J. Barret, R.E. Silverman, R.G. Byrnes, SSH: The Secure Shell (The Definitive Guide), second ed., O'Reilly, 2005.
[48] Y.-C. Chou, D. Ko, H.H. Cheng, Mobile agent-based computational steering for distributed applications, Concurrency and Computation: Practice and Experience (2009), doi:10.1002/cpe.1458.
[49] J.A. Kohl, T. Wilde, D.E. Bernholdt, CUMULVS: interactive with high-performance scientific simulations for visualization, steering and fault tolerance, The International Journal of High Performance Computing Applications 20 (2) (2006) 255–285.
[50] S.M. Pickles, R. Haines, R.L. Pinning, A.R. Porter, A practical toolkit for computational steering, Philosophical Transactions of the Royal Society A—Mathematical, Physical and Engineering Sciences 363 (1833) (2005) 1843–1853.
[51] A. Modi, N. Sezer-Uzol, L.N. Long, P.E. Plassmann, Scalable computational steering for visualization/control of large-scale fluid dynamics simulations, Journal of Aircraft 42 (4) (2005) 963–975.
[52] K. Brodlie, J. Wood, D. Duce, M. Sagar, gViz: visualization and computational steering on the grid, in: Proceedings of the UK e-Science All Hands Meeting, Nottingham, UK, 2004, pp. 54–60.
[53] Mobile Agent-Based Computational Steering for Distributed Applications. <http://www.mobilec.org/apps/csteering/> (accessed 1.08.09).
[54] K. Chmiel, M. Gawineck, P. Kaczmarek, M. Szymczak, M. Paprzycki, Efficiency of JADE agent platform, Scientific Programming 13 (2) (2005) 159–172.
[55] Mobile-C: Performance Analysis. <http://www.mobilec.org/performance.php> (accessed 1.08.09).
[56] K-TEAM Corporation. <http://www.k-team.com/> (accessed 1.08.09).
[57] V. Braitenberg, Vehicles: Experiments in Synthetic Psychology, MIT Press, Cambridge, MA, 1984.
[58] Mobile Agent-Based Control of a Khepera III Robot using Mobile-C. <http://www.mobilec.org/apps/khepera/>.