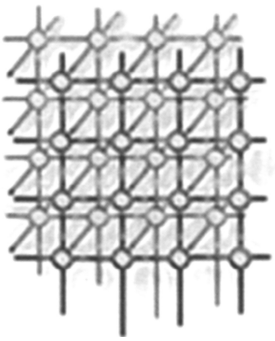
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2009)
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1458



Mobile agent-based computational steering for distributed applications

Yu-Cheng Chou[‡], David Ko[‡] and Harry H. Cheng[*, †, §]

Integration Engineering Laboratory, Department of Mechanical and Aeronautical Engineering, University of California, Davis, CA 95616, U.S.A.

SUMMARY

The mobile agent-based computational steering (MACS) for distributed applications is presented in this article. In the MACS, a mobile agent platform, Mobile-C, is embedded in a program through the Mobile-C library to support C/C++ mobile agent code. Runtime replaceable algorithms of a program are represented as agent services in C/C++ source code and can be replaced with new ones through mobile agents. In the MACS, a mobile agent created and deployed by a user from the steering host migrates to computing hosts successively to replace algorithms of running programs that constitute a distributed application without the need of stopping the execution and recompiling the programs. The methodology of dynamic algorithm alteration in the MACS is described in detail with an example of matrix operation. The Mobile-C library enables the integration of Mobile-C into any C/C++ programs to carry out computational steering through mobile agents. The source code level execution of mobile agent code facilitates handling issues such as portability and secure execution of mobile agent code. In the MACS, the network load between the steering and computing hosts can be reduced, and the successive operations of a mobile agent on multiple computing hosts are not affected whether the steering host stays online or not. The employment of the middle-level language C/C++ enables the MACS to accommodate the diversity of scientific and engineering fields to allow for runtime interaction and steering of distributed applications to match the dynamic requirements imposed by the user or the execution environment. An experiment is used to validate the feasibility of the MACS in real-world mobile robot applications. The experiment replaces a mobile robot's behavioral algorithm with a mobile agent at runtime. Copyright © 2009 John Wiley & Sons, Ltd.

Received 8 May 2008; Revised 5 April 2009; Accepted 10 April 2009

KEY WORDS: computational steering; distributed computing; mobile agent library; mobile agent platform; Ch; C/C++ interpreter

[*]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
[‡]Graduate Research Assistant.
[§]Professor.

1. INTRODUCTION

A distributed application is typically composed of distinct software computing components running on different computers connected through a network. Dependencies and relationships do not necessarily exist among the components. Each component is a program that implements different algorithms. Typically, any changes in any algorithm of a component requires at least interrupting, recompiling, and restarting the specific component. It may even need to stop and recompile the whole application to make a valid change in an algorithm of a component.

Computational steering is a powerful and widely applicable concept that allows users to control a computational process during its execution. Computational steering can be employed for three main uses—model exploration, algorithm experimentation, and performance optimization [1] in scientific and engineering applications. For algorithm experimentation, computational steering enables users to replace algorithms of a program, e.g. to experiment with newly prototyped numerical methods, while the program is running. Therefore, certain sections of the program can be replaced at runtime, i.e. algorithm experimentation corresponds to program code replacement.

In this article, the target application is a distributed application consisting of multiple programs running on different computers. Therefore, an algorithm experimentation regarding such an application involves code mobility. In the simplest case, a piece of program code is transferred from the steering host to a specific computing host to replace another piece of program code at runtime. There are three main design paradigms exploiting code mobility: code on demand, remote evaluation, and mobile agent [2]. Code on demand represents the situation where a computing host downloads the necessary program code. This is not applicable to computational steering where the steering host plays the proactive role. Remote evaluation and mobile agent are two paradigms that match the idea of computational steering in the sense that program code is sent out from the steering host to a specific computing host. One difference between mobile agent and remote evaluation paradigms is that mobile agent paradigm has the option to allow for migrating the execution state of mobile agent code. Another difference between these two paradigms lies within the degree of code mobility, which is essential to steering multiple components of a distributed application.

Remote evaluation paradigm allows for single-hop code mobility, whereas mobile agent paradigm allows for multiple-hop code mobility. In remote evaluation paradigm, a piece of code is sent out from the steering host to a single computing host where the piece of code is executed. In mobile agent paradigm, a mobile agent can contain multiple pieces of code for multiple computing hosts. Such a mobile agent can move from the steering host to computing hosts one after another for code execution. An advantage of using mobile agent paradigm over remote evaluation paradigm to perform code replacement for distributed applications is that it reduces the network load between the steering and computing hosts. Another advantage is that once such a multiple-hop mobile agent has been sent out, the steering host can be disconnected from the network if desired, e.g. to minimize the energy consumption of the steering host for some application scenarios.

The majority of mobile agent platforms in use are Java-oriented. Several mobile agent platforms supporting Java mobile agent code include Mole [3], Aglets [4], Concordia [5], JADE [6], and D'Agents [7]. Adopting a standard language as the mobile agent code language that provides both high-level and low-level functionalities is a good choice to deal with the diversity of distributed applications. C/C++ is a proper choice for such a mobile agent code language. C/C++ provides powerful functions in terms of memory access. A huge number of existing C/C++ programs can be

DOI: 10.1002/cpe

used as mobile agent code. Moreover, C is an internationally standardized language, and can easily interface with a variety of low-level hardware devices, which is especially useful for mechatronic and embedded systems. Ara [8,9] and TACOMA [10] are two mobile agent platforms supporting C mobile agent code, whereas Ara also supports C++ one. Mobile agent code is compiled as byte code [11] and machine code [12] for execution in Ara and TACOMA, respectively.

Mobile-C [13–16] was originally developed as a stand-alone mobile agent platform to support C/C++ mobile agent code. In contrast to the approach of running compiled C/C++ mobile agent code adopted by Ara and TACOMA, Mobile-C chose an embeddable C/C++ interpreter—Ch [17–19] to run C/C++ mobile agent source code. The interpretive approach can avoid some potential problems, such as platform portability, secure execution, and system implementation issues that could be induced by the compiling approach. In our previous work, we developed the Mobile-C library [20] to enable the integration of Mobile-C into any C/C++ programs that attempt to exploit and benefit from the mobile agent technology. A mobile agent of Mobile-C is represented in XML. The term, *mobile agent code* (also called *mobile code* or *agent code*) in this article refers to the C/C++ source code inside a mobile agent of Mobile-C.

In this article, the mobile agent-based computational steering (MACS) for distributed applications is presented. In the MACS, a mobile agent sent out from the steering host migrates to computing hosts successively to replace algorithms of running programs that constitute a distributed application. The network load between the steering and computing hosts can be reduced, and the successive operations of a mobile agent on multiple computing hosts are not affected whether the steering host stays online or not. In the MACS, programs running on the steering and computing hosts are C/C++ programs and have an embedded mobile agent platform, Mobile-C, to handle the operations regarding mobile agents. Runtime replaceable algorithms are represented as agent services in C/C++ source code and can be replaced with new ones through mobile agents. The source code level execution of mobile agent code facilitates handling issues such as portability and secure execution of mobile agent code. The Mobile-C library enables the integration of Mobile-C into any C/C++ program to carry out computational steering through mobile agents. The employment of the middle-level language C/C++ allows the MACS to accommodate the variety of distributed applications.

The rest of the article is organized as follows. Section 2 presents some related work in computational steering, dynamic software reconfiguration for mobile robots and sensor networks, and mobile agent-based applications. Section 3 presents the Mobile-C library adopted in the MACS to embed a mobile agent platform in a program to support C/C++ mobile agent code. Section 4 describes the MACS framework. Section 5 describes the methodology of dynamic algorithm alteration in the MACS. Section 6 presents an experiment used to validate the feasibility of the MACS in real-world mobile robot applications. The experiment is to replace a mobile robot's behavioral algorithm with a mobile agent at runtime. Section 7 gives the conclusions for the investigation we have done.

2. RELATED WORK

Several computational steering-oriented libraries, such as CUMULVS [21], RealityGrid [22,23], POSSE [24], and gViz [25], allow users to adjust parameters of an application algorithm at runtime. As opposed to computational steering libraries, Wenisch *et al.* [26] developed an integrated

DOI: 10.1002/cpe

development environment (IDE) called Computational Steering Environment (CSE) for computational fluid dynamics (CFD) applications. CSE provides support for interactively adding, removing, and modifying fluid obstacles and boundary conditions for indoor thermal comfort application and assessment. Shenfield *et al.* [27] proposed an IDE that allows users to alter parameters on-the-fly to influence the quality of solutions produced by a multi-objective evolutionary algorithm for engineering design. These computational steering-oriented libraries and IDEs, however, do not support the functionality to replace an algorithm at runtime.

VASE [28,29] and SCIRun [30,31] are two computational steering-oriented IDEs that support algorithm experimentation [1]. A VASE application is based on a control-flow graph structure consisting of logical blocks and control-flow arcs. Breakpoint scripts can be set between logical blocks and contain algorithms for tasks such as the visualization of intermediate results or the calculation of steerable parameters. A user can modify breakpoint scripts at runtime. A SCIRun application is based on a data-flow structure comprising a network of modules. A user can interchange existing modules containing different algorithms at runtime. However, new modules cannot be added at runtime. Because VASE and SCIRun are IDEs instead of libraries, they were intended for developing new applications using their built-in software tools and components as well as running the created applications. Therefore, it will take a significant effort to integrate the algorithm experimentation functionality of VASE or SCIRun into the existing programs. In addition, the network connection should exist between the steering and computing hosts during the entire application execution.

Cragg *et al.* [32,33] used mobile agent paradigm for multi-robot architecture development to benefit from mobile agents in several aspects among which is dynamic software reconfiguration for networked mobile robots. MacDonald and coworkers [34,35] and Lee and coworkers [36,37] adopted the Common Object Request Broker Architecture (CORBA)-distributed object framework for dynamic reconfiguration of mobile robot software. In those CORBA frameworks, a user manipulates the software configuration of a mobile robot by switching between available software objects that are physically hosted on some object servers through CORBA mechanisms. Therefore, the main difference between the above CORBA and mobile agent-based approaches is the constant connection between mobile robots and software servers in the above CORBA-based approach. In addition, the main difference between Cragg *et al.*'s and our approaches is that we chose C/C++ as the mobile agent code language whereas they chose Java.

Global code update and mobile code principles have been used for software reconfiguration on sensor networks. Global code update principle is supported in [38–40], whereas mobile code principle is supported in [41–43]. The advantage of mobile code approach is clear with respect to the bytes transferred in the network and the energy consumed as the network scales for the object tracking application scenario [44]. Although our system is not specifically aimed at extremely resource-constrained sensor networks, it is applicable to embedded systems such as tiny computers or single-board computers, among which Gumstix [45] is an example. Our system provides a higher generality and broader functionality to support mobile code-based software reconfiguration for a wide range of distributed embedded systems.

Different from this article's emphasis on user-driven dynamic replacement of algorithms continually invoked in programs running on multiple computers, mobile agent technology has been utilized in a variety of different distributed applications. For example, mobile agents have been used as the load balancing scheme for a digital library [46,47]. An architecture Mobile Agent-based GriD

DOI: 10.1002/cpe

Architecture (MAGDA) was designed to support the programming and execution of mobile agent-based applications for Grid systems [48]. Other mobile agent-based distributed applications may also be found in manufacturing [49], electronic commerce [50], network management [51], transportation systems [52], and information management [53].

3. MOBILE-C LIBRARY

Due to C/C++'s universality, portability, and flexibility, C/C++ is widely used in many applications in scientific and engineering fields. Therefore, C/C++ was chosen to be the mobile agent code language for Mobile-C [13–16]. An embeddable C/C++ interpreter, Ch [17–19], is incorporated as the Agent Execution Engine (AEE) to support the execution of C/C++ mobile agent code in Mobile-C. Ch is a cross-platform computing environment that supports standard and other commonly used C libraries. A C program can typically run interpretively in different platforms in Ch without any modification and compilation.

Mobile-C was originally developed as a stand-alone, IEEE FIPA [54] compliant mobile agent platform to accommodate applications where low-level hardware gets involved, such as networked mechatronic and embedded systems. The Mobile-C library [20] was thereafter developed to facilitate the design of mobile agent-based applications. This library allows Mobile-C to be embedded in a program to support C/C++ mobile agent code. A C/C++ program with an embedded Mobile-C is compiled into machine code, whereas mobile agent code is C/C++ source code. Therefore we define the binary space is where a compiled program exists and the mobile agent space or agent space is where mobile agent code exists. The Mobile-C library provides functions that can be organized into eight categories: Agency, Agent Management System (AMS), Agent Communication Channel (ACC), Directory Facilitator (DF), AEE, Agent, Synchronization, and Miscellaneous APIs, as shown in Figure 1. Most of these functions have two versions, one for the binary space and the other for the mobile agent space. The binary space functions can be called in a program to manipulate the operations associated with the embedded Mobile-C. On the other hand, the mobile agent space functions can be called in mobile agent code to perform the same operations as those done by their binary space counterparts.

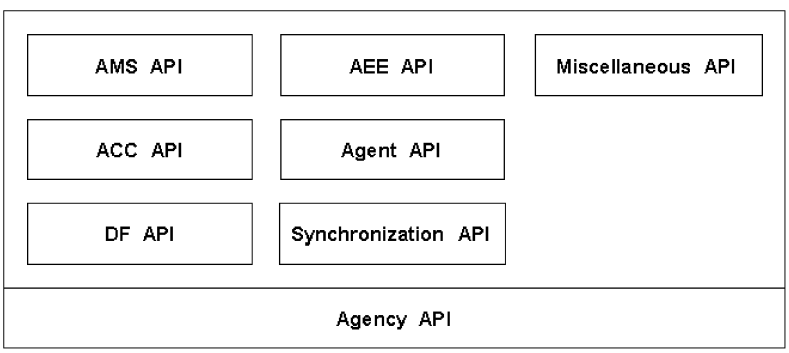


Figure 1. Architecture of the Mobile-C library.

DOI: 10.1002/cpe

The Embedded Ch toolkit [55], associated with the AEE of Mobile-C, was designed to allow a binary C/C++ program to interface with Ch scripts at runtime for operations such as calling functions and accessing global arrays/variables defined in the Ch scripts. Several Mobile-C binary space functions were therefore developed on top of the Embedded Ch toolkit in order to call functions and access global arrays/variables defined in the mobile agent space from the binary space. Besides those Mobile-C binary space functions, all the Embedded Ch functions can be directly called from the binary space to interface with the mobile agent space.

In Mobile-C, a mobile agent is associated with an AEE, and each AEE is independent of each other. Functions and global arrays/variables associated with a mobile agent are therefore independent of those associated with other mobile agents. The Mobile-C binary space functions mentioned in the previous paragraph have their counterparts in the mobile agent space. Those mobile agent space functions, on the other hand, allow a mobile agent code to call functions and access global arrays/variables defined in another mobile agent code.

To coordinate simultaneous processes to complete a task in order to get correct runtime order and avoid unexpected race conditions, a set of synchronization mechanisms can be invoked from the binary or mobile agent space through a set of Mobile-C functions for different types of synchronization.

4. MACS FRAMEWORK

In order to utilize mobile agents to replace program algorithms, a program needs to possess the functionality to receive mobile agents and run their mobile agent code. The mobile agent code in this article is a C/C++ program consisting of the optional function main() with or without runtime replaceable algorithms implemented as additional functions and/or global arrays/variables. The function main() of the mobile agent code can specify the operations, for example, the algorithm alteration procedure, that will be performed by the mobile agent.

In the MACS framework shown in Figure 2, a distributed application consists of multiple user programs, each of which is running on a computing host. With the Mobile-C library, each user program can encompass a Mobile-C agency to support multiple mobile agents. Besides a Mobile-C agency, a user program may consist of functions and data in the binary space. A steering host contains a Mobile-C-enabled program used to monitor and control a distributed application through mobile agents. Data structures associated with each module of a Mobile-C agency can be accessed through the Mobile-C functions from within the binary and mobile agent spaces.

From a binary program's perspective, it can call functions and access data in any mobile agent code through the Mobile-C and Embedded Ch functions, as mentioned in Section 3. The data refer to the global arrays/variables defined in the mobile agent code. On the other hand, from a mobile agent's perspective, it can, from within its agent code, call functions and access data associated with other mobile agents in the same agency through the Mobile-C functions, as mentioned in Section 3. In addition, if desired, the binary space functions and data in a user program can be managed so that they can be called and accessed from within the mobile agent space via the Ch Software Development Kit (Ch SDK) [56]. The Ch SDK, associated with the AEE of Mobile-C, was designed to allow Ch scripts to access global variables or call functions in a compiled C/C++ library such as a static library, shared library, or dynamically linked library.

DOI: 10.1002/cpe

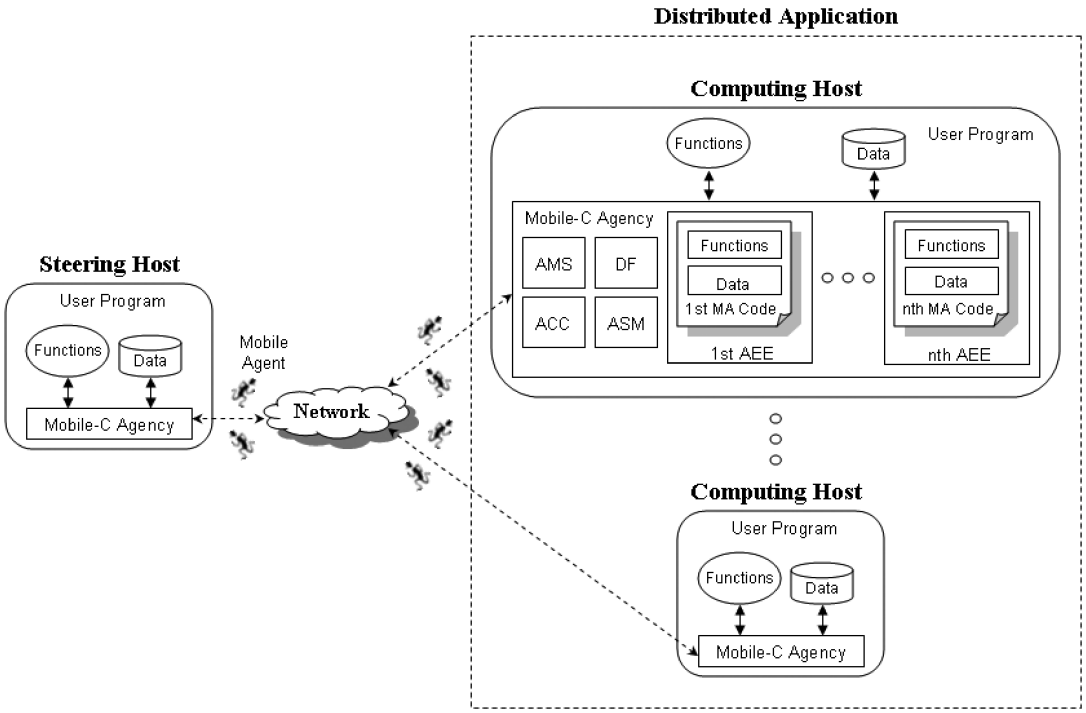


Figure 2. MACS framework.

Regarding a user's controllability perspective, for example, a user from the steering host can generate a mobile agent, *MA_2*, and send it on the fly to control a particular mobile agent, *MA_1*, that runs on a specific agency, *Agency_1*. The control activities performed on *MA_1* by *MA_2* may include calling the agent space functions of *MA_1*, terminating the execution of *MA_1*'s agent code, removing *MA_1* from *Agency_1*, replacing the agent space functions of *MA_1* with those carried by *MA_2*. A user, responsible for allocating the rights for those activities, can specify those activities in the function main() of *MA_2*'s agent code through necessary Mobile-C agent space functions. When *MA_2* arrives at *Agency_1*, its agent code will be executed by a newly launched AEE to perform the desired control activities.

In the MACS, a mobile agent code can typically be executed by an AEE without any modification, because the AEE-Ch supports standard and other commonly used C functions. However, if a mobile code in a mobile agent uses a special library, this library needs to be installed beforehand, so that functions in the library can be invoked locally in target machines. For instance, if a mobile agent code contains functions of the Open Source Computer Vision (OpenCV) C library, the Ch OpenCV package created using the Ch SDK needs to be installed on the machine where the agent code will be executed.

In the MACS, when a mobile agent is received by an agency, the agent code carried by this mobile agent will be pulled out and saved as a temporary file, which will be run by an AEE.

DOI: 10.1002/cpe

This temporary file will be removed once it has been run by an AEE. Thus, as long as the user's computer has sufficient hard drive capacity to accommodate a temporary file for an agent code, there is no limit imposed on the size of a mobile agent code. A mobile agent in the MACS can also travel across multiple agencies that reside on different machines to perform the same or different tasks.

Among the Mobile-C modules shown in Figure 2, the AMS, ACC, DF will be initialized when Mobile-C agency is started in a user program via a Mobile-C function that starts Mobile-C agency. These modules can therefore be viewed as being hardwired into a program when Mobile-C agency is started. The AMS is related to the creation, registration, execution, migration, persistence, and termination of a mobile agent. The ACC in the article is related to the inter-agency mobile agent transport. The DF is related to yellow page activities. An AEE is automatically created to support the agent code execution for an arrived mobile agent. In this article, it is assumed that mobile agents are authorized agents that will not harm the security of an agency. Thus, the ASM is not required to be initialized in a user program. By default, the ASM is not initialized when a Mobile-C agency is started unless specified otherwise.

5. DYNAMIC ALGORITHM ALTERATION METHODOLOGY

A program typically consists of multiple functions, and each function performs a specific computational task based on an algorithm. The dynamic algorithm alteration allows a user to change the underlying implementation of functions while a program is in progress. With the Mobile-C library, each function in a program can be defined in a mobile agent code and registered as a service with the DF. Therefore, an algorithm can be altered by replacing an existing service with a new service that implements a new algorithm through mobile agents. In this article, a service refers to a function that is defined in the agent code of a mobile agent and has been or will be registered with the DF of Mobile-C agency by that mobile agent.

Through an example, this section demonstrates how to dynamically change an algorithm in a running application through the Mobile-C library. Such dynamic algorithm alteration cannot be accomplished using other notable computational steering libraries [21,22,24,25]. In this example, there are two mobile agents sent from a client program to the server program. These two mobile agents have the same mobile agent code except for the difference in the function to be registered as a service with the DF. The programs for this example can be downloaded from the Internet [57].

From the flowchart shown in Figure 3, 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 to receive mobile agents and execute mobile agent codes to update the service. An authorized mobile agent is represented in the form of a mobile agent structure inside an agency, as shown in Figure 4. A mobile agent structure contains a pointer to the mobile code, a handle for the AEE, and other information such as the name, ID, and status of a mobile agent. The *agent managing thread* starts an *agent executing thread* for each mobile agent to initialize the AEE, which in turn runs the mobile agent code. Once the execution of the mobile agent code is completed, the *agent executing thread* updates the agent status according to which the *agent managing thread* can process the mobile agent subsequently. If a mobile agent is specified to be persistent, it is not removed from an agency

DOI: 10.1002/cpe

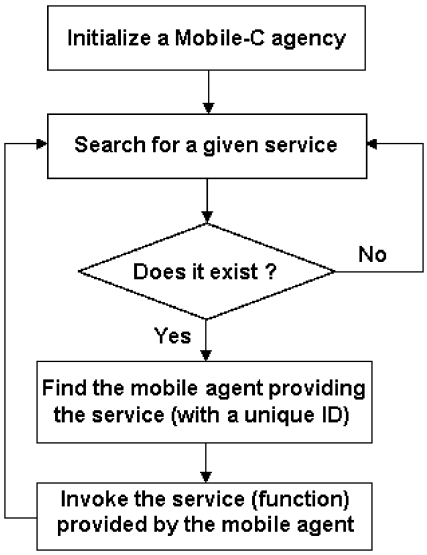


Figure 3. Flowchart of the server program.

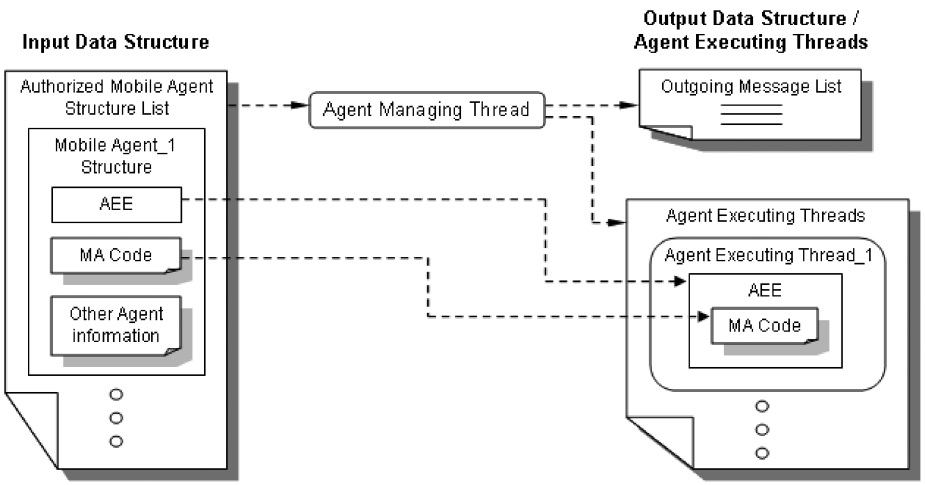


Figure 4. Agent managing thread in Mobile-C starts an agent executing thread to execute a mobile code.

once its mobile code has been executed, so that all the functions and data in its mobile code can be accessed from both the binary and mobile agent spaces. Here the data are global arrays/variables that are defined to contain results of interest from the execution of the mobile agent code.

The server program shown in Program 1 performs matrix computation for two matrices and prints the resultant matrix to the screen. A Mobile-C agency is initialized to listen on port *5130*

DOI: 10.1002/cpe

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

int main() {
 int i, j, local_port = 5130, mutex_id = 55, *agentID, numResult;
 char *funcname = "matrix_operate", **agentName, **serviceName;
 double a[2][2], b[2][2], c[2][2];
 MCAgency_t agency;
 MCAgent_t agent;

 for(i=0; i<2; i++) {
  for(j=0; j<2; j++) {
   a[i][j] = 2*i+j+1;
   b[i][j] = 2*i+j+5;
  }
 }
 agency = MC_Initialize(local_port, NULL);
 MC_SyncInit(agency, mutex_id);
 while(1) {
  MC_MutexLock(agency, mutex_id);
 MC_SearchForService(agency, funcname, &agentName, &serviceName, &agentID, &numResult);
  while(numResult == 0) {
   /* If no agent is found to have provided the desired service, unlock the Mutex variable
     to allow an agent to register the desired service with the DF before locking the same
     Mutex variable again for the next search of the desired service. */
   MC_MutexUnlock(agency, mutex_id);
   MC_MutexLock(agency, mutex_id);
  MC_SearchForService(agency, funcname, &agentName, &serviceName, &agentID, &numResult);
  }
  agent = MC_FindAgentByID(agency, agentID[0]);
  MC_CallAgentFunc(agent, funcname, &c[0][0], 2, a, b);
  MC_MutexUnlock(agency, mutex_id);
  MC_DestroyServiceSearchResult(agentName, serviceName, agentID, numResult);

  /* Output array c containing the matrix computation result to the screen. */
  printf("\n");
  for(i=0; i<2; i++) {
   for(j=0; j<2; j++) {
    printf("%8.2f", c[i][j]);
   }
   printf("\n");
  }
  /* Wait for 1 second to make the screen output of array c for each iteration clearly
    readable. */
  sleep(1);
 }
 return 0;
}
```

Program 1: A server program with a Mobile-C agency.

with default settings by function *MC_Initialize()*. A mutex variable is initialized to have an ID *55* by function *MC_SyncInit()*. As shown in Program 1, when this mutex variable is locked by function *MC_MutexLock()*, it is guaranteed that the desired service and the agent providing the service on the local agency are protected. Thus, the simultaneous access of the desired service and

DOI: 10.1002/cpe

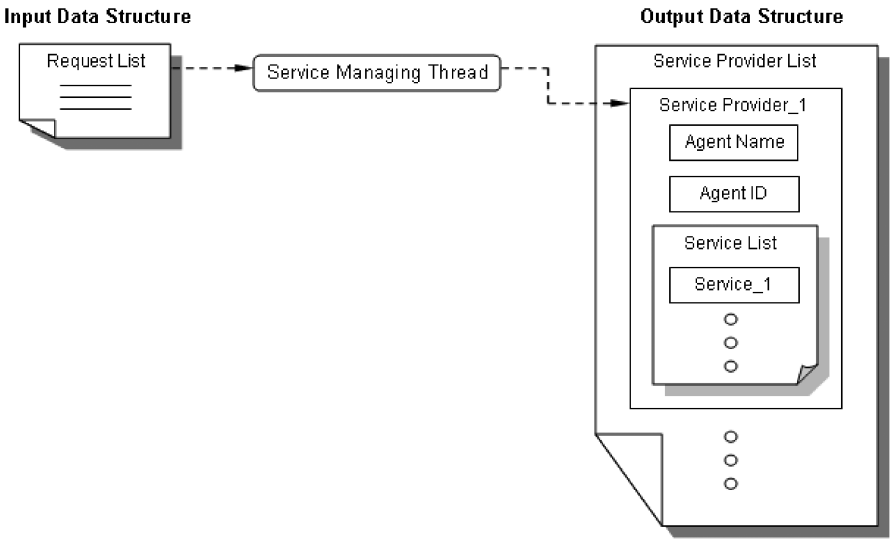


Figure 5. Service managing thread in Mobile-C.

the agent on the local agency can be avoided. Function *MC_SearchForService()* is used to search for a service in an agency after the mutex variable is locked. The mutex variable can be unlocked by function *MC_MutexUnlock()* to allow for the access of the desired service and the agent by other mobile agents. As shown in Figure 3, the server program will keep searching for a desired service if the service cannot be found. Therefore as shown in Program 1, when this happens, the server program will unlock the mutex variable to allow any mobile agent to register the desired service with the DF, and then lock the mutex variable again for the next search of the desired service.

Once function *MC_SearchForService()* is called; it places a *search* request in the request list shown in Figure 5. The *service managing thread* processes a *search* request by searching for a specified service in the service provider list. Each node in the service provider list points to a persistent agent whose single or multiple services have been registered with the DF. After the *service managing thread* completes processing a search request, the results such as the names, IDs, and number of all the persistent agents providing the specified service will be returned and stored in function *MC_SearchForService()*'s arguments. As shown in Program 1, the first agent providing the *matrix_operate* service is obtained with its ID through function *MC_FindAgentByID()*. Function *MC_CallAgentFunc()* is then used to call the agent space function, *matrix_operate()*, which implements the desired service. Arrays *a* and *b* are passed to function *matrix_operate()* as two arguments for computation purposes, and array *c* is used to take the resultant matrix returned by function *matrix_operate()*. Function *MC_DestroyServiceSearchResult()* is used to release the previous search result obtained by function *MC_SearchForService()* regarding the names and IDs of all the persistent agents that provide the specified service.

Program 2 shows the template for a persistent mobile agent with a single task sent to the server shown in Program 1. The agent carries three kinds of information. First, information about itself

DOI: 10.1002/cpe

```
<?xml version="1.0"?>
<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">
<MOBILEC_MESSAGE>
 <MESSAGE message="MOBILE_AGENT">
  <MOBILE_AGENT>
   <AGENT_DATA>
    <NAME>mobileagent</NAME>
    <OWNER>IEL</OWNER>
    <HOME>bird2.engr.ucdavis.edu:5050</HOME>
    <TASKS task="1" num="0">
     <TASK num="0"
         complete="0"
         return="no-return"
         persistent="1"
         server="bird1.engr.ucdavis.edu:5130">
    </TASK>
    <AGENT_CODE>
     <![CDATA[

     C/C++ mobile agent code

     ]]>
    </AGENT_CODE>
   </TASKS>
  </AGENT_DATA>
 </MOBILE_AGENT>
</MESSAGE>
</MOBILEC_MESSAGE>
```

Program 2: Template for a persistent mobile agent with a single task sent to the server program shown in Program 1.

including its name, owner, home address. Second, overall information about the task it has to do. The statement

$$<\text{TASKS task}=\text{“1”num}=\text{“0”}>$$

shows that this mobile agent has one task to perform and no task has been finished yet. Third, detailed information about the task including the persistence of the agent, the name of the task's return variable, the completeness of the task, the host to perform the task, and most importantly, the C/C++ agent code that implements the task. Since the flag *persistent* is set to 1, the agent is a persistent agent that will not be removed from an agency once its code has been executed. The flag *persistent* can be set to 0 for a non-persistent mobile agent. A mobile agent is not persistent by default if this attribute is omitted. In addition to the template shown in Program 2, in Mobile-C, a mobile agent can also be composed in such a way that it contains multiple tasks with a single agent code block or multiple tasks with multiple agent code blocks [16].

The Mobile-C library provides mobile agent space functions that can be called in a mobile agent code to perform tasks just like their counterparts could do in the binary space. For instance, the binary space function *MC_SearchForService()* mentioned above has a counterpart in the mobile agent space, function *mc_SearchForService()*, which can be called in a mobile agent code to search for a given service in an agency.

DOI: 10.1002/cpe

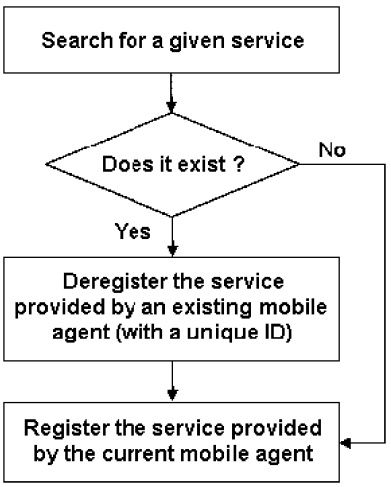


Figure 6. Flowchart of the main() function of the mobile agent code used in the matrix computation example.

Figure 6 illustrates the flowchart of the main function of the mobile agent code in this example. 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 have already provided the service, the service provided by this mobile agent will be deregistered first. Afterwards, the current mobile agent's service will be registered with the DF. Since the flag *persistent* for a mobile agent providing services is set to 1, deregistering the services provided by such a mobile agent only removes the services from the service list shown in Figure 5 associated with the mobile agent. It does not remove the mobile agent from the agency. However, if the incoming mobile agent code contains function *mc_DeleteAgent()* to delete an existing mobile agent described above, the existing mobile agent's status will be changed so that the *Agent Managing Thread* will remove the mobile agent structure for this agent from the mobile agent structure list shown in Figure 4.

Two mobile agents, *mobileagent1.xml* and *mobileagent2.xml*, based on the template shown in 2 are sent to the server. Program 3 shows the agent code in the mobile agent *mobileagent1.xml*. The mobile agent *mobileagent2.xml* is the same as *mobileagent1.xml* except for the difference in the implementation of function *matrix_operate()* shown in Program 4. As illustrated in Program 3, function *mc_SearchForService()* is called to search for a given service in an agency. If no mobile agent provides service *matrix_operate*, then service *matrix_operate* provided by the current mobile agent is registered through function *mc_RegisterService()* with system variable *mc_current_agent*. Once function *mc_RegisterService()* is called, it places a *register* request in the request list. The *service managing thread* processes a *register* request by storing the name and ID of the specified mobile agent and the services the mobile agent is intended to provide in the service provider list. In this example, since only one mobile agent is sent to the server at a time, no matter which agent is sent first, function *mc_DeregisterService()* is used to remove the information of the first

DOI: 10.1002/cpe

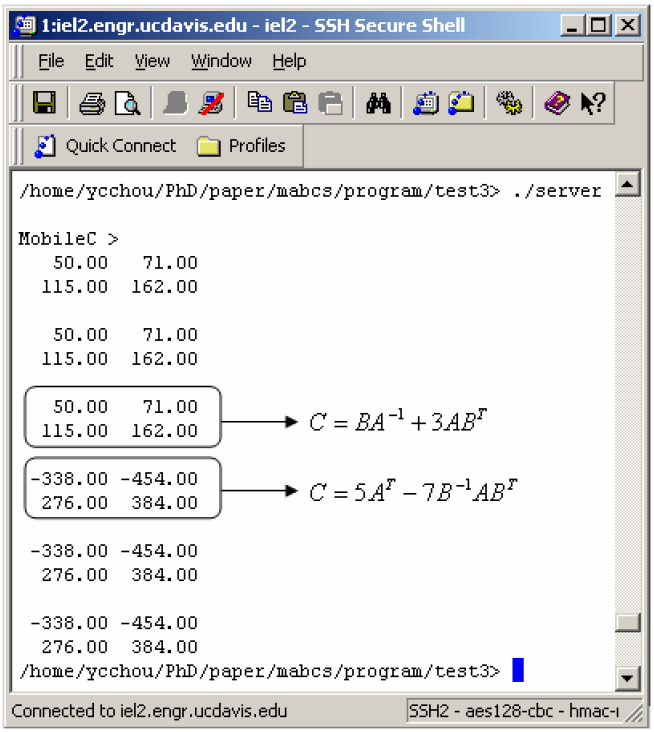


Figure 7. Output from the server for the matrix computation example.

mobile agent from the service provider list for service *matrix_operate*. Once the service provided by the first mobile agent is deregistered, the service provided by the current mobile agent is then registered. Programs 3 and 4 implement service *matrix_operate* with the following two different algorithms:

$$C = BA^{-1} + 3AB^{\mathrm{T}} \tag{1}$$

$$C = 5A^{\mathrm{T}} - 7B^{-1}AB^{\mathrm{T}} \tag{2}$$

where A, B, and C are 2×2 matrices. Arrays *a* and *b* passed from the binary space can be cast to Ch computational arrays *a* and *b* in the mobile agent space. With Ch computational arrays, we can specify vector and matrix operations directly in the same way as we do with scalars in the mobile agent code. As a running example, mobile agent *mobileagent1.xml* with an agent code shown in Program 3 is first sent to the agency initialized in the server shown in Program 1. Afterwards, mobile agent *mobileagent2.xml*, whose mobile code is to replace the previous *matrix_operate* service, is sent to the same agency. The output from the server is shown in Figure 7. In this example, matrices

DOI: 10.1002/cpe

```
#include <stdlib.h>
#include <string.h>
#include <array.h>

int main() {
 int i, numService = 1, mutex_id = 55, *agentID, numResult;
 char *funcname = "matrix_operate", **service, **agentName, **serviceName;
 MCAgent_t agent;

 service = (char **)malloc(sizeof(char *)*numService);
 for(i=0; i<numService; i++) {
  service[i] = (char *)malloc(sizeof(char)*(strlen(funcname)+1));
 }

 strcpy(service[0], funcname);

 mc_SearchForService(service[0], &agentName, &serviceName, &agentID, &numResult);

 if(numResults < 1) {
  /* No agent is found to have provided such a service. */
  mc_RegisterService(mc_current_agent, service, numService);
 }
 else {
  /* An existing agent is found to have provided such a service. */
  mc_MutexLock(mutex_id);
  mc_DeregisterService(agentID[0], service[0]);
  mc_RegisterService(mc_current_agent, service, numService);
  mc_MutexUnlock(mutex_id);
  mc_DestroyServiceSearchResult(agentName, serviceName, agentID, numResult);
 }

 for(i=0; i<numService; i++) {
  free(service[i]);
 }
 free(service);

 return 0;
}

array double matrix_operate(array double a[2][2], b[2][2])[2][2] {
 return b*inverse(a) + 3*a*transpose(b);
}
```

Program 3: The mobile agent code for the mobile agent *mobileagent1.xml* in Program 2.

```
array double matrix_operate(array double a[2][2], b[2][2])[2][2] {
 return 5*transpose(a) - 7*inverse(b)*a*transpose(b);
}
```

Program 4: Service *matrix_operate* provided by *mobileagent2.xml*.

A and B are specified as follows:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

DOI: 10.1002/cpe

The resultant matrices are shown as follows:

$$C = \begin{bmatrix} 50 & 71 \\ 115 & 162 \end{bmatrix} \quad \text{for the algorithm in Equation (1)}$$

$$C = \begin{bmatrix} -338 & -454 \\ 276 & 384 \end{bmatrix} \quad \text{for the algorithm in Equation (2)}$$

In this example, regardless of the output portion for array *c* in Program 1, when there is already an agent in the local agency to provide the desired service *matrix_operate*, the average time to complete one iteration of the outer while-loop in Program 1 is 260 μs on a Linux machine with 256 MB of RAM and an Intel Pentium 4 CPU running at 1400 MHz. Such an iteration includes function calls to *MC_MutexLock()*, *MC_SearchForService()*, *MC_FindAgentByID()*, *MC_CallAgentFunc()*, *MC_MutexUnlock()*, and *MC_DestroyServiceSearchResult()*. When a new mobile agent arrives at the local agency to replace the existing algorithm of the desired service *matrix_operate*, the average time to complete one iteration of the outer while-loop in Program 1 is 1118 μs. Afterwards, the time to complete such an iteration reverts back to an average of 260 μs. The overhead, $1118 - 260 = 858$ μs, is a one-time overhead associated with the algorithm replacement. The specific factors that contribute to the overhead include the parsing and execution of the new mobile agent as well as the deregistration and registration of services performed by the new mobile agent. The factors that contribute to the overhead are unaffected by the algorithm of a service. Therefore, even when the algorithm of a service becomes complicated, the overhead due to the algorithm replacement remains constant.

6. APPLICATION: RUNTIME REPLACEMENT OF A MOBILE ROBOT'S BEHAVIORAL ALGORITHM WITH A MOBILE AGENT

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 pre-programmed to handle those uncertainties. With mobile agent paradigm, a user can create mobile agents that contain newly developed algorithms and migrate among desired mobile robots to replace improper algorithms at runtime.

A scenario that can prominently show mobile agent paradigm's advantage is that a mobile agent carries pieces of code to be run on multiple mobile robots. A piece of code can represent an algorithm. For example, there are 10 mobile robots (MRobot1 to MRobot10) with one steering host (Steer). A piece of code on MRobot1 to MRobot5 is to be replaced with Code1, and a piece of code on MRobot6 and MRobot10 is to be replaced with Code2. A mobile agent, MAgent1, is created by a user in such a way that MAgent1 carries Code1 and Code2 and will visit MRobot1 to MRobot5 successively for Code1 execution and will visit MRobot6 to MRobot10 successively for Code2 execution. The network transformation between the steering host and mobile robots only includes migrating MAgent1 from Steer to MRobot1. Once MAgent1 is sent out from Steer, it will perform its tasks regardless of whether Steer stays online or not.

DOI: 10.1002/cpe

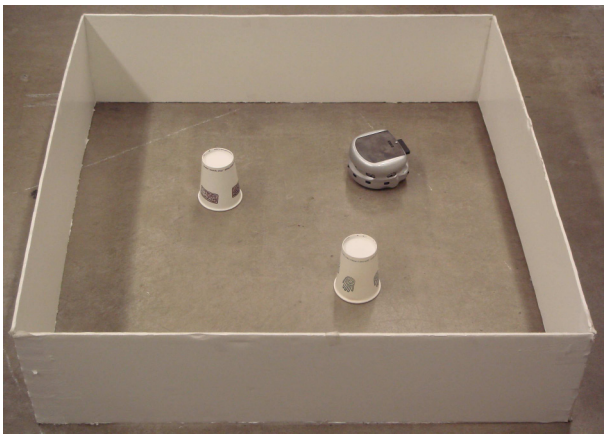


Figure 8. Khepera III mobile robot performing object avoidance.

Figure 9. Khepera III mobile robot performing object following.

Control programs for mobile robots are typically written in C/C++ because the underlying hardware interface packages for mobile robots are typically developed in C/C++. In the MACS, through the Mobile-C library, a program running on a mobile robot can encompass Mobile-C to support C/C++ mobile agent code that represents a runtime replaceable algorithm. The C/C++ mobile agent code is run interpretively as a script in Mobile-C. Therefore, the MACS allows a user to compose a mobile agent with newly developed algorithms as the C/C++ agent code and directly send out the mobile agent to perform algorithm replacement on multiple desired mobile robots.

In this section, we will validate the MACS by replacing a mobile robot's behavioral algorithm via a mobile agent at runtime through an experiment with a K-Team Khepera III mobile robot [58] as shown in Figures 8 and 9 [59]. Even though this validation only involves algorithm replacement on a single mobile robot due to the number of mobile robots we currently have, it does not impede the MACS's ability to successfully carry out the multiple mobile robot scenario described above.

The Khepera III mobile robot is equipped with the KoreBot board, an ARM -based computer. The KoreBot board has 64 Mbytes of RAM and an embedded Linux operating system that provides

DOI: 10.1002/cpe

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 object 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. The agency is started with default modules, the AMS, ACC, and DF. An AEE will be launched for each mobile agent arriving at the agency.

Agent *service_provider_1* containing five functions, *InitRobot()*, *SensorMatrix()*, *RobotBehaviour()*, *MoveRobot()*, and *StopRobot()*, 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 objects. The service leading to this object avoidance behavior is service *RobotBehaviour* that corresponds to function *RobotBehaviour()*. The implementation of this function is based on the Braitenberg object avoidance algorithm [60]. As shown in Figure 8, the Khepera III mobile robot will wander through the field avoiding the walls and cups.

The mobile robot's trajectory depends on service *RobotBehaviour*, which implements the robot behavioral algorithm. The robot behavioral algorithm calculates the robot's left and right wheel speeds, which will be fed in the robot's left and right motors as inputs. Therefore we use the root-mean-square (RMS) wheel speed of the mobile robot to indirectly show the mobile robot's runtime behavior. The RMS wheel speed of the mobile robot is calculated using Equation (3).

$$\text{RMS wheel speed} = \sqrt{\frac{(\text{Left wheel speed})^2 + (\text{Right wheel speed})^2}{2}} \tag{3}$$

In Figure 10, the X-axis value is the ordinal number for calling service *RobotBehaviour* on the mobile robot, and the Y-axis value is the RMS wheel speed of the mobile robot. As shown in Figure 10, during the first 100 times when service *RobotBehaviour* is called, the underlying algorithm that service *RobotBehaviour* performs is the object avoidance algorithm. This service is provided by agent *service_provider_1*. Some high RMS wheel speeds in this curve section are induced when the mobile robot rapidly changes its direction and moves away from the object.

While the mobile robot is wandering through the field and avoiding objects, 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 5.

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. As shown in Figure 9, the remaining cup is dragged by a hand around the robot for it to follow.

As shown in Figure 10, at the 101th time when service *RobotBehaviour* is called, the underlying algorithm that service *RobotBehaviour* performs is the object following algorithm. This service is provided by agent *service_provider_2*. The mobile robot now follows a moving object, which is

DOI: 10.1002/cpe

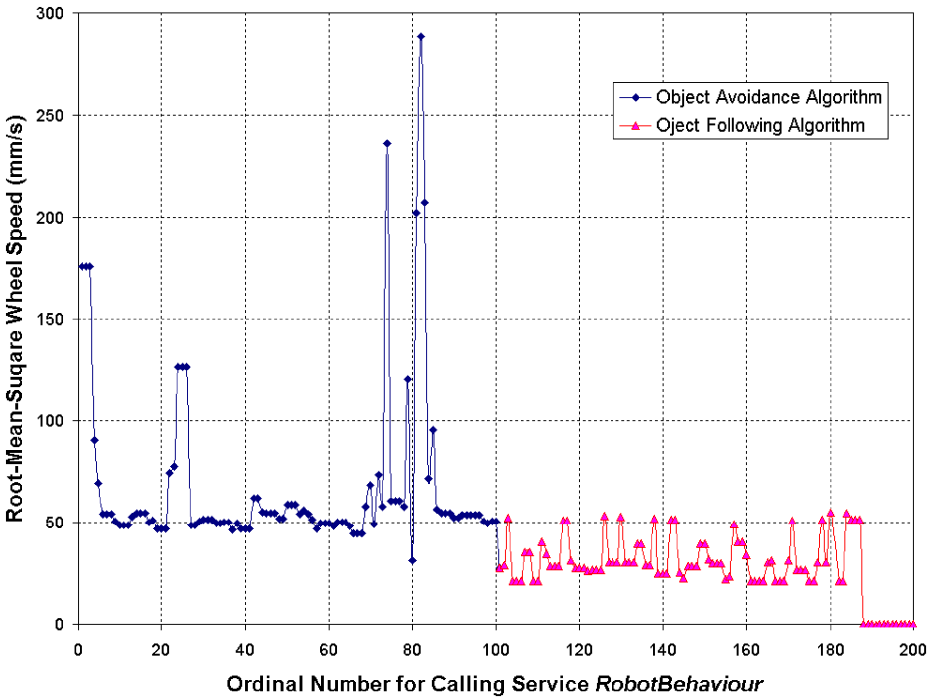


Figure 10. Root-mean-square wheel speeds of the Khepera III mobile robot in action under the object avoidance and object following algorithms.

slowly dragged by hand, does not avoid static objects as previously illustrated. As a result, the RMS wheel speeds of the mobile robot do not promptly increase, as shown in the second curve section of Figure 10.

The object following behavior is implemented by modifying the Braitenberg object avoidance algorithm. The sensor weights of the Braitenberg object avoidance algorithm are swapped to drive the opposite motor rather than the one originally specified. In addition, 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 object avoidance by making the robot move toward 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, as shown at the end of the second curve section of Figure 10.

The mobile code flowchart for agent *mobagent1* regarding the search and invocation of *Robot-Behaviour* service is shown in Figure 11. The flowchart is identical to the one shown in Figure 3 except that the Mobile-C agency is not initialized through an agent code. The mobile robot behavior is changed seamlessly on the fly without the need to stop and modify the running agent *mobagent1*. The main() function flowchart for agents *service_provider_1* and *service_provider_2* are the same as the one shown in Figure 6. Besides, because agents *service_provider_1* and

DOI: 10.1002/cpe

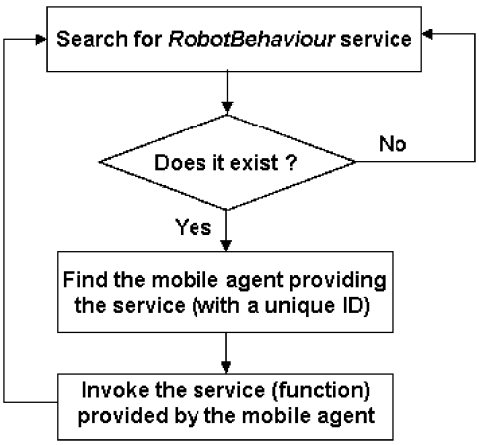


Figure 11. Flowchart for the mobile code of agent *mobagent1* regarding the search and invocation of *RobotBehaviour* service.

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 object 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 Internet [59].

7. CONCLUSIONS

The MACS for distributed applications has been presented in this article. In the MACS, programs running on the steering and computing hosts are C/C++ programs and encompasses a mobile agent platform, Mobile-C, to handle the operations regarding the mobile agents that carry C/C++ agent code. Mobile-C is embedded in a program through the Mobile-C library. In the MACS, runtime replaceable algorithms of a program are represented as agent services in C/C++ source code and can be replaced with new ones through mobile agents. In the MACS, a mobile agent created and deployed by a user from the steering host migrates to computing hosts successively to replace algorithms of running programs that constitute a distributed application without the need of stopping the execution and recompiling the programs. The methodology of dynamic algorithm alteration in the MACS has been described in detail with an example of matrix operation. An experiment in runtime replacement of a mobile robot's behavioral algorithm with a mobile agent has been used to validate the feasibility of the MACS in real-world mobile robot applications regarding dynamic software reconfiguration.

DOI: 10.1002/cpe

The Mobile-C library enables the integration of Mobile-C into any C/C++ programs to carry out computational steering through mobile agents. The source code level execution of mobile agent code facilitates handling issues such as portability and secure execution of mobile agent code. In the MACS, the network load between the steering and computing hosts can be reduced, and the successive operations of a mobile agent on multiple computing hosts are not affected whether the steering host stays online or not.

The employment of the middle-level language C/C++ allows the MACS to accommodate a variety of distributed applications. For example, in multidisciplinary research areas, scientific data sets that an application needs to process can be of multiple terabytes, located remotely in different computers, and are produced by a variety of sources such as different remote sensing instruments or applications. Such data sets are likely to encompass a variety of sampling rates, data geometries, and error characteristics. Multiple algorithms are available that enable the visualization and analysis of such data sets. To avoid huge raw data transmission, each algorithm needs to be executed on-site to process the raw data locally and may need to be replaced at runtime.

It is expected that the MACS will find its applications in a wide range of scientific and engineering fields to allow for runtime interaction and computational steering of distributed applications to match the dynamic requirements imposed by the user or the execution environment.

REFERENCES

1. Mulder JD, van Wijk JJ, van Liere R. A survey of computational steering environments. *Future Generation Computer Systems* 1999; **15**(1):119–129.
2. Fuggetta A, Picco GP, Vigna G. Understanding code mobility. *IEEE Transactions on Software Engineering* 1998; **24**(5):342–361.
3. Baumann J, Hohl F, Rothermel K, Strasser M, Theilmann W. MOLE: A mobile agent system. *Software—Practice and Experience* 2002; **32**(6):575–603.
4. Lange DB, Oshima M. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley: MA, 1998.
5. Wong D, Paciorek N, Walsh T, DiCelie J, Young M, Peet B. Concordia: An infrastructure for collaborating mobile agents. *Proceedings of the First International Workshop on Mobile Agents* (*MA'97*) (*Lecture Notes in Computer Science*, vol. 1219). Springer: Berlin, 1997; 86–97.
6. Bellifemine F, Caire G, Poggi A, Rimassa G. JADE: A software framework for developing multi-agent applications. Lessons learned. *Information and Software Technology* 2008; **50**(1–2):10–21.
7. 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.
8. Peine H. Run-time support for mobile code. *PhD Dissertation*, Department of Computer Science, University of Kaiserslautern, Germany, 2002.
9. Peine H. Application and programming experience with the Ara mobile agent system. *Software—Practice and Experience* 2002; **32**(6):515–541.
10. Johnansen D, Lauvset KJ, van Renesse R, Schneider FB, Sudmann NP, Jacobsen K. A TACOMA retrospective. *Software—Practice and Experience* 2002; **32**(6):605–619.
11. MACE—Mobile agent code environment. Available at: http://wwwagss.informatik.uni-kl.de/Projekte/Ara/mace.html [last modified 10 August 2004].
12. Sudmann NP, Johansen D. Adding mobility to non-mobile web robots. *Proceedings of the IEEE ICDCS00 Workshop on Knowledge Discovery and Data Mining in the World-wide Web*, Taipei, Taiwan, 2000; 73–79.
13. Chen B, Cheng HH. A run-time support environment for mobile agents. *Proceedings of ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications*, *No. DETC2005-85389*, Long Beach, CA, September 2005.
14. Chen B, Cheng HH, Palen J. Mobile-C: A mobile agent platform for mobile C/C++ agents. *Software—Practice and Experience* 2006; **36**(15):1711–1733.
15. Chen B, Linz D, Cheng HH. XML-based agent communication, migration and computation in mobile agent systems. *Journal of Systems and Software* 2008; **81**(8):1364–1376.

DOI: 10.1002/cpe

16. Mobile-C: A multi-agent platform for mobile C/C++ code. Available at: http://www.mobilec.org [last modified 12 May 2009].
17. Cheng HH. Scientific computing in the Ch programming language. *Scientific Programming* 1993; **2**(3):49–75.
18. Cheng HH. Ch: A C/C++ interpreter for script computing. C/C++ *User's Journal* 2006; **24**(1):6–12.
19. *Ch—An embeddable* C/C++ *interpreter*. Available at: http://www.softintegration.com [last modified 15 April 2009].
20. Chou Y-C, Ko D, Cheng HH. Embeddable mobile-C for runtime support of code mobility in multi-agent systems. *Proceedings of ASME 2007 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, *No. DETC2007-35747*, Las Vegas, NV, September 2007.
21. Kohl JA, Wilde T, Bernholdt DE. CUMULVS: Interactive with high-performance scientific simulations for visualization, steering and fault tolerance. *The International Journal of High Performance Computing Applications* 2006; **20**(2):255–285.
22. Pickles SM, Haines R, Pinning RL, Porter AR. A practical toolkit for computational steering. *Philosophical Transactions of the Royal Society A—Mathematical*, *Physical and Engineering Sciences* 2005; **363**(1833):1843–1853.
23. The realityGrid computational steering library and toolkit. Available at: http://www.rcs.manchester.ac.uk/research/realitygrid/downloads [last modified 12 May 2009].
24. Modi A, Sezer-Uzol N, Long LN, Plassmann PE. Scalable computational steering for visualization/control of large-scale fluid dynamics simulations. *Journal of Aircraft* 2005; **42**(4):963–975.
25. Brodlie K, Wood J, Duce D, Sagar M. gViz: Visualization and computational steering on the grid. *Proceedings of the U.K. e-Science All Hands Meeting*, Nottingham, U.K., 2004; 54–60.
26. Wenisch P, van Treeck C, Borrmann A, Rank E, Wenisch O. Computational steering on distributed systems: Indoor comfort simulations as a case study of interactive CFD on supercomputers. *International Journal of Parallel*, *Emergent and Distributed Systems* 2007; **22**(4):275–291.
27. Shenfield A, Fleming PJ, Alkarouri M. Computational steering of a multi-objective evolutionary algorithm for engineering design. *Engineering Applications of Artificial Intelligence* 2007; **20**(8):1047–1057.
28. Haber R, Bliss B, Jablonowski D, Jog C. A distributed environment for run-time visualization and application steering in computational mechanics. *Computing Systems in Engineering* 1992; **3**(1–4):501–515.
29. Jablonowski DJ, Bruner JD, Bliss B, Haber RB. VASE: The visualization and application steering environment. *Proceedings of Supercomputing '93*, Portland, OR, 1993; 560–569.
30. Johnson C, Parker S, Weinstein D, Heffernan S. Componet-based problem-solving environments for large-scale scientific computing. *Concurrency and Computation*: *Practice and Experience* 2002; **14**(13–15):1337–1349.
31. Parker SG, Weinstein DM, Johnson CR. The SCIRun computational steering software system. In *Modern Software Tools in Scientific Computing*, Arge E, Bruaset AM, Langtangen HP (eds.). Birkhauser: Basel, 1997; 5–44 [Online]. Available at: citeseer.ist.psu.edu/article/parker97scirun.html [last modified 12 May 2009].
32. Cragg L, Hu H, Voelker N. Modularity and mobility of distributed control software for networked mobile robots. *Software Engineering for Experimental Robotics* (*Springer Tracts in Advanced Robotics*, vol. 30), Brugali D (ed.). Springer: Berlin, 2007; 459–484.
33. Cragg L, Hu H. Mobile agent approach to networked robots. *International Journal of Advanced Manufacturing Technologies* 2006; **30**:979–987.
34. Yu Z, Warren I, MacDonald B. Dynamic reconfiguration for robot software. *Proceedings of the 2006 IEEE International Conference on Automation Science and Engineering*, Shanghai, China, 2006; 292–297.
35. MacDonald B, Hsieh BP-S, Warren I. Design for dynamic reconfiguration of robot software. *Proceedings of the Second International Conference on Autonomous Robots and Agents*, Palmerston North, New Zealand, 2004; 19–24.
36. Hong S, Lee J, Eom H, Jeon G. The robot software communications architecture (RSCA): Embedded middleware for networked service robots. *The 14th IEEE International Workshop on Parallel and Distributed Real-time Systems* (*WPDRTS '2006*), Island of Rhodes, Greece, 2006.
37. Lee J, Park J, Han S, Hong S. RSCA: Middleware supporting dynamic reconfiguration of embedded software on the distributed URC robot platform. *The First International Conference on Ubiquitous Robots and Ambient Intelligence* (*ICURAI*), Seoul, Korea, 2004; 426–437.
38. Han CC, Kumar R, Shea R, Kohler E, Srivastava M. A dynamic operating system for sensor nodes. *Technical Report*, Networked Embedded Systems Lab, University of California, Los Angeles, 2004.
39. Liu T, Martonosi M. Impala: A middleware system for managing autonomous, parallel sensor systems. *Proceedings of the ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming* (*PPoPP'03*), San Diego, CA, 2003.
40. Reijers N, Langendoen K. Efficient code distribution in wireless sensor networks. *Proceedings of the Second ACM International Workshop on Wireless Sensor Networks and Applications* (*WSNA'03*), San Diego, CA, 2003.
41. Levis P, Culler D. Matè: A tiny virtual machine for sensor networks. *ASPLOS-X*: *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM Press: New York, NY, U.S.A., 2002; 85–95.
42. Lifton J, Seetharam D, Broxton M, Paradiso JA. Pushpin computing system overview: A platform for distributed embedded, ubiquitous sensor networks. *Proceedings of the First International Conference on Pervasive Computing*, Zurich, Switzerland, 2002.

43. Boulis A, Han C-C, Srivastava MB. Design and implementation of a framework for efficient and programmable sensor networks. *MobiSys '03*: *Proceedings of the First International Conference on Mobile Systems*, *Applications and Services*. ACM Press: New York, NY, U.S.A., 2003; 187–200.
44. Boulis A. Programming sensor networks with mobile agents. *MDM'05*: *Proceedings of the Sixth International Conference on Mobile Data Management*. ACM: New York, NY, U.S.A., 2005; 252–256.
45. Gumstix Inc. Available at: http://www.gumstix.com/ [last modified 4 May 2009].
46. Christos G, Rana OF. Performance-sensitive service provision in active digital libraries. *Proceedings of the 2005 IEEE International Conference on e-Technology*, *e-Commerce and e-Service*, Hong Kong, 2005; 514–517.
47. Christos G, Rana OF. Choosing a load balancing scheme for agent-based digital libraries. *Proceedings*, *ISPA 2006* (*Lecture Notes in Computer Science*, vol. 4330). Springer: Berlin, 2006; 51–62.
48. Aversa R, Martino BD, Mazzocca N, Venticinque S. MAGDA: A mobile agent based grid architecture. *Journal of Grid Computing* 2006; **4**(4):395–415.
49. Wada H, Okada S. An autonomous agent approach for manufacturing execution control systems. *Integrated Computer-aided Engineering* 2002; **9**(3):251–262.
50. Sandholm T. eMediator: A next generation electronic commerce server. *Computational Intelligence* 2002; **18**(4):656–676.
51. Chou L, Shen K, Tang K, Kao C. 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). Springer: Berlin, 2003; 280–289.
52. 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.
53. Stathis K, DeBruijn O, Macedo S. Living memory: Agent-based information management for connected local communities. *Interacting with Computers* 2002; **14**(6):663–688.
54. FIPA: The foundation for intelligent physical agents. Available at: http://www.fipa.org/ [last modified 29 October 2008].
55. Embedded Ch, SoftIntegration, Inc. Available at: http://www.softintegration.com/products/sdk/embedded_ch/ [last modified 15 April 2009].
56. Ch SDK, SoftIntegration, Inc. Available at: http://www.softintegration.com/products/sdk/chsdk/ [last modified 15 April 2009].
57. Mobile agent-based dynamic interaction and computational steering. Available at: http://www.mobilec.org/apps/csteering/ [last modified 12 May 2009].
58. T K-TEAM Corporation. Available at: http://www.k-team.com/ [last modified 12 May 2009].
59. Dynamic mobile robot control with mobile agents. Available at: http://www.mobilec.org/apps/khepera/ [last modified 12 May 2009].
60. Braitenberg V. *Vehicles*: *Experiments in Synthetic Psychology*. MIT Press: Cambridge, MA, 1984.

DOI: 10.1002/cpe