Two Executable Mobility Design Patterns: mfold and mmap

Zara Field, Rick Dewar, Phil Trinder, Andre R. Du Bois

School of Mathematical and Computer Sciences,
Heriot-Watt University,
Riccarton, Edinburgh EH14 4AS, U.K.
zf1@macs.hw.ac.uk, rick@macs.hw.ac.uk,
trinder@macs.hw.ac.uk,dubois@ucpel.tche.br

Abstract. We present two mobility design patterns, *mfold* and *mmap*. The patterns are equipped with corresponding coordination specifications, *mobility skeletons*, implemented on top of a host object-orientated mobile code language, Voyager. The mobility skeletons provide a high-level of abstraction and control all coordination aspects of the mobility design patterns. We conclude by demonstrating, through a simple yet concrete example, how the composite of these patterns and skeletons can be used in the development of a practical distributed application, a mobile meeting scheduler.

1 Introduction

Mobile design paradigms, including Mobile Agent, Remote Evaluation and Code-on-Demand [AF98] and mobile code languages such as Telescript [Whi94], Aglets [LO99], Voyager [Whe05] and JoCaml [FFMS] have the potential to increase the efficiency and effectiveness of the development of complex and customizable, distributed systems. In spite of this, the greater flexibility offered by mobile computation comes at additional costs. Designing and implementing mobile code systems is more complex than systems based on the traditional client-server paradigm, as complete mobility of cooperating applications forms large-scale, loosely-coupled and complex distributed systems. Furthermore, mobile code system development is not yet fully supported by sound technologies or methodological background.

Design patterns are a recognized means for promoting the use of mobile code. Several mobile agent design patterns have already been proposed [SD98], [LMSF04], [TOH99], [AL98], [ken97]. The problems shared by these previous efforts are described in [KKPS98] and [Kue], which include lack of agreed definitions, duplicated efforts, complexity and the difficulty in identifying and specifying common abstractions above the level of single agents. Also, their documentation and classifications are often difficult to apply since their purposes are not clearly stated or are unnecessarily related to a specific platform [LMSF04]. They also tend to focus on *how* to build mobility systems, and not *why* to use computational mobility [Wei03].

This paper aims to go further and deeper in the explanation of mobility design patterns by identifying the real-world forces and contexts of the problems that gave rise to the mfold and mmap patterns and corresponding skeletons. Furthermore, the patterns will be described in a language, platform and domain independent manner. The patterns are therefore *macroarchitecture* [EGV95] patterns as they represent the outline of the system configuration and are not dependent on any specific mobile code platform .

To enhance these mobility design patterns we also provide implementations of corresponding mobility skeletons. Mobility skeletons, including mfold and mmap have already been implemented in mHaskell, an extension of Haskell, as a set of higher-order functions [DBTL05]. However, the mobility skeletons provided here are implemented in the object-orientated mobile code language, Java Voyager. Mobility skeletons use the template design method [EGV95] by setting up the skeleton of the coordination algorithms for each mobility pattern. At this point, the patterns are described at the *microarchitecture* layer [EGV95] that represents the detail of the system configuration and mobile code behavior in the specific platform, Voyager. A case study example is then used to illustrate the use and composition of the patterns and skeletons.

The patterns are therefore presented at different layers, where each layer refines the abstractions of the previous layer. Table 1 illustrates the abstraction relationship between mobility patterns, skeletons, mobile paradigms and a mobile code API.

Abstract Level	Mobility Patterns
High Level	Mobility Skeletons
Medium Level	Mobile Code Paradigms
Low Level	Mobile Code API Primitives

Table 1. Abstraction Layers for Mobile Computation

2 Mobile Code in Distributed Information Processing

Distributed information processing tasks are traditionally performed with a centralized, sequential information processing methodology based on the client-server architecture. Using this method, large amounts of unfiltered information are often retrieved on the client, who then processes the information in order to extract information of importance. This *information shipping* process is effective for simple, uniform procedures where access to information is restricted. The system is typically 'closed' in terms of what data can be accessed and how it can be accessed as the sever stores a set of fixed procedures for accessing the information.

Nowadays, *customization* and *flexibility* is key. People want to access information in a variety of ways, from a variety of devices, ranging from desktop PCs to handheld devices accessing the internet on a variety of connection modes and speeds. Evidentally, distributing computation load, providing flexibility and reducing network interactions would increase the efficiency and effectiveness of distributed information processing applications in this dynamic and disparate computing environment.

The application domains which may potentially benefit from mobile code applications and mobility patterns include Active Documents [CTZ02], Advanced Telecommunications Services [MRK96], Remote Device Control and Configuration, Network Management [BGP97] [BP98], Workflow Management and Cooperation [TKI$^+$97], [STO99], Active Networks [CH02], Grid Computing and Global Computing [KBD02], [MR04] and Advanced E-Commerce [WT02], [KA03], [HfL02].

Mobile agents [AF98] are regarded highly attractive for distributed information proccessing applications [CPV97], [Lan98], [BGM$^+$99]. A mobile agent is executable code that can autonomously migrate from one location to another, providing *dynamic function shipping.* In data collection problems where the data is remotely located, mobile agents can be used to migrate to the remote location to perform the data collection locally and reduce the data collected by either filtering or compressing the results, realizing network bandwidth savings. Using mobile agents for distributed information processing also addresses the problem of network reliability. With mobile agents, the agent can continue execution at one side of an unreliable link. When the network link is back up, the agent either continues migrating to the next information source or returns home with the results.

Remote Evaluation, a one-shot remote code mechanism that is more flexible than proxies but less complicated than fully-general mobile agents, is another promising mobile code paradigm for distributed information processing. Several advantages arise from limiting movement to one hop. By avoiding some of the security issues introduced by code that can roam from site to site, infra-structural support is simplified. Also, technical problems associated with maintaining and updating program state during migration are avoided, without losing much functionality, a view supported by [KGR02].

3 Mobility Design Patterns: mfold and mmap

3.1 Mobile Fold

1. **Name** mobile fold (mfold)
   The name for the mfold pattern is inspired by the `fold` higher order function in functional languages that captures a common recursion pattern for processing and reducing lists. In an mfold invocation, a mobile operation is performed at each remote location within a given list. The results are systematically reduced at the remote locations and returned when the mobile operation has been performed at the last location.
2. **Problem**
   – How can a customizable (mobile) computation be used to migrate sequentially through a list of remote locations performing a task at each

  location reducing the results (i.e. merging/filtering), while hiding the migratory and coordination details from the user?
- How can a programmer force a customizable computation to migrate sequentially through a list of remote locations, perform a distributed processing task, while separating the migration and coordination details from the task to be performed?

3. **Motivating Scenario**
   You are designing a flexible and customizable distributed information system. You need to implement an algorithm that will allow an arbitrary mobile computation to migrate sequentially through a known set of locations (i.e. a static itinerary), perform a task at each location and automatically process the results (i.e. reduce them) before migrating to the next location. You require this distributed reducing of intermediate results, for example through merging or filtering, to retain only those results of relevance to your requirements and to reduce the amount of information to be transmitted. The solution needs to be reliable against possible intermittent/low bandwidth connections, but security of the transmitted data (i.e. results) is not a significant issue. If the transmitted data is sensitive and security is an issue, the user should refer to the secure mfold pattern to be presented in a companion paper.
   The distributed task most suitable for this mobility pattern is distributed information retrieval of selective yet non-sensitive information. For example, e-commerce agents searching remote product sites for the best price or network management agents collating management information from remote network nodes. This pattern could also be used for distributed information dispersal applications, for example, dynamic load balancing on computational grids as proposed in [BSH02] and [GS01]. Applications with a similar structure could also benefit from the mfold pattern.
4. **Forces**
   Consider the mfold pattern when:
   - The application domain is characterised by decentralised resources, or the system can be thought of a set of co-operating components. You want to take this distributed data structure and apply a function to each of its components.
   - The mobile computation can be autonomous in the sense that it requires no user intervention at each resource location and can autonomously migrate to each location in its itinerary.
   - You only want relevant information returned from your distributed processing task. You want the mobile code to remotely process the results from each resource location, for example by merging or filtering, before migrating to the next resource location, thus reducing bandwidth consumption and the amount of data to be transmitted through the network. If the amount of data to be transmitted is large, this is not the best pattern and the user should refer to the mmap pattern.
   - Using a client server model, the server would be overloaded with information. Using the mfold pattern eliminates server overload and possible

bottlenecks by distributing computation load. It also reduces the number of messages sent through the network as the mobile code carries intermediate results with it as it migrates from location to location. Using the traditional client server model would also restrict your distributed processing tasks to those already predefined at the remote locations.
– You want the flexibility to perform any processing task on the locations, i.e. a plug and play application.
– The user may want to use resources on a network with relatively narrow bandwidth or on devices with intermitted access to a network. The user may also have concerns with network reliability.
– You are not concerned with the security of the mobile code or the results it obtains. If security is an issue, the user is directed to the secure mfold pattern, to be presented in a companion paper.
– The remote locations are known in advance and all locations are known to be stable. If not, the reliable mfold pattern, to be presented in a future companion paper, should be used.

5. **Solution**

Mobile agents have been chosen as the mobile design paradigm most appropriate for the mfold mobility pattern and subsequent skeleton due to their inherent advantages over the other mobile code paradigms (REV and COD). Not only can the customizable and flexible mobile agent autonomously migrate to the remote location of a required resource but they can also carry intermediate data as they autonomously migrate to the next location, conserving bandwidth and overcoming latency. These interactions can also continue if the network connection goes down temporarily, increasing reliability.

The following platform independent models provide an abstract view of the mfold pattern, including the entities, their relationships and the operations that must be implemented in general.

The diagram in figure 1 illustrates the behaviour of the mfold pattern. The mobile agent is created at host 1. It then migrates to host 2, where it resumes execution and performs a task with the local resources. The mechanisms for resuming execution of the agent at a new location differ slightly with each mobile agent framework, although in object-orientated mobile languages they typically work on *entry-point migration* strategies based on *call-back* methods which are invoked transparently as the result of arriving at a new location. Once execution is complete, the mobile agent merges or filters the results and then carries these intermediate results with it as it migrates to the next location. Once again, it resumes execution and performs a task with the local resources. It continues with this pattern until it reaches the last host, where it performs the task for the last time and returns the result to the initiating program residing on host 1.

The class diagram in figure 2 shows the simple structure of the mfold pattern and the participants. In practice, the mfold class is initiated with a list of lo-

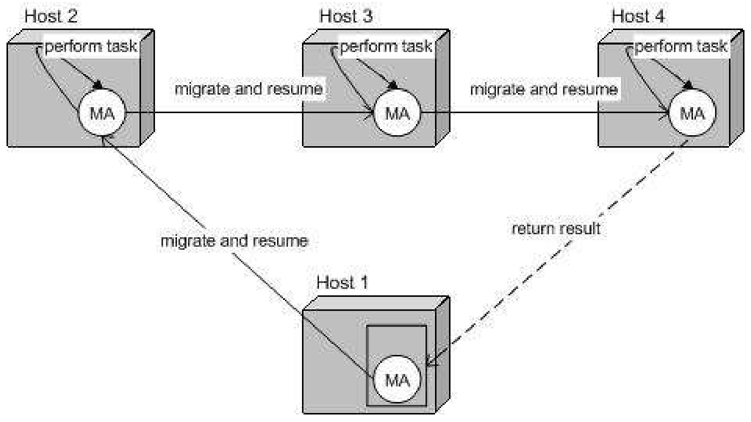


Fig. 1. Behaviour of mfold

cations and an ObjectTomfold, which encapsulates the task to be performed at each location and the helper method to process the results.

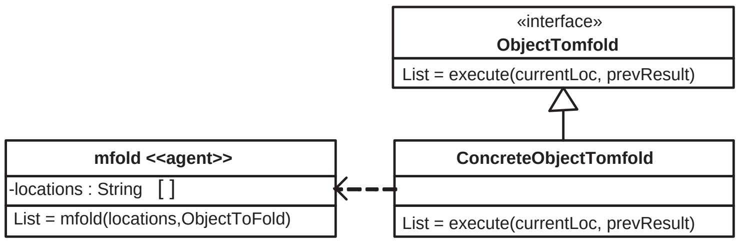

«interface»
ObjectTomfold
List = execute(currentLoc, prevResult)
mfold <<agent>>
-locations : String [ ]
List = mfold(locations,ObjectToFold)
ConcreteObjectTomfold
List = execute(currentLoc, prevResult)

Fig. 2. UML class diagram for mfold pattern

Participants

– **mfold** The mfold class contains the implementation for,
  - the sequential coordination algorithm responsible for the agents traversal along a list of locations,
  - invoking a polymorphic `List = execute(currentLoc, prevRes)`, whereby each call to this method is performed on the current location with the results from the previous location.
  - handling the storage of the intermediate results within the mobile agent.
  - returning the result to the initiating program when the task has been performed at the last location in the given list.

– **ObjectTomfold**
The ObjectTomfold interface, which must be implemented to use the pattern, forces the programmer to implement a `List = execute(currentLoc,`

`prevRes)` method that conforms to the `List = execute(currentLoc, prevRes)` used by the mfold class to invoke the polymorphic method at each location.

– **ConcreteObjectTomfold**
  This class contains the implementation for the `List = execute(currentLoc, prevRes)` method to be performed at each location. It should contain the implementation for the distributed information processing task and helper methods that defines how newly obtained results should be combined with the results from the previous location (i.e. how the results should be reduced).

The UML sequence diagram in figure 3 further illustrates the mfold pattern by showing the sequence of interactions involved. It shows how, after the instantiation of an mfold agent with a list of locations and an ObjectTomfold at location 1, the agent migrates to the first location in the list. It then invokes the `List = execute(currentLoc, prevRes)`, contained within the ObjectTomfold and handles the reduction and storage of the intermediate results before migrating to the next location. This continues until the last location where the result is returned to the initiating program.

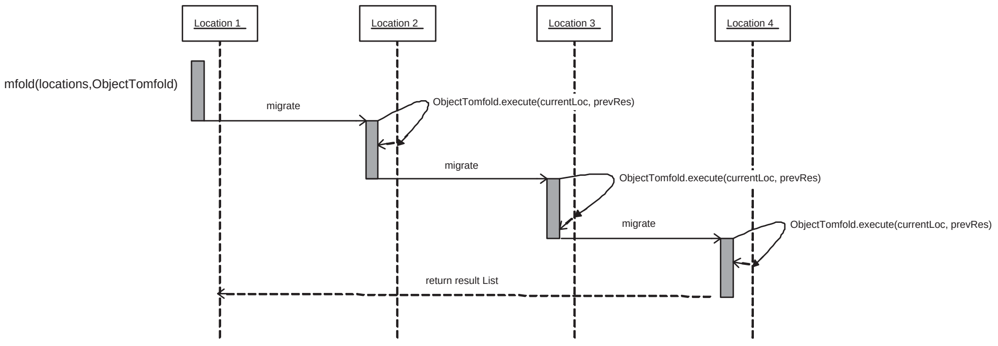

Location 1
Location 2
Location 3
Location 4
mfold(locations,ObjectTomfold)
migrate
ObjectTomfold.execute(currentLoc, prevRes)
migrate
ObjectTomfold.execute(currentLoc, prevRes)
migrate
ObjectTomfold.execute(currentLoc, prevRes)
return result List

Fig. 3. Platform Independent Sequence Diagram for mfold Pattern

6. **mfold Mobility Skeleton implemented in Voyager**
   The mobility patterns described in this paper are equipped with corresponding mobility skeletons. Although these mobility skeletons have been implemented in Voyager, Aglets and mHaskell, the following section describes the specific implementation details for the mfold skeleton implemented in Voyager.
   The mobility skeleton for the mfold pattern using the mobile agent paradigm and implemented in Voyager has the method signature,

```
List = mfold(ObjectTomfold obj, String[] lo)
```

and takes as its parameters, an ObjectTomfold `obj` that contains the distributed information processing and result reducing methods and a list of remote locations `lo`. In a call to mfold, the `obj` is converted into a agent by Voyager's *dynamic aggregation.* It then migrates through the list of locations `lo` performing a task, `execute(currentLoc, prevRes)` at each location. The results from the task performed at each location are processed using the helper method contained within the ObjectTomfold, with the final results returned when all locations have been visited.
In practice, a callback method is responsible for invoking the `execute(currentLoc, prevRes)` method when the agent arrives at each location, which subsequently reduces the results and initiates the agents movement to the next location in the list `lo`(the itinerary). Incidentally, the use of a reflexive callback method, that is automatically called by the destination voyager daemon, is the only means for coordinating Voyager agents and is hard-coded into Voyager. When the agent has reached the last location in the list `lo`, it returns home with the result.
The class diagram in figure 4 shows how the Voyager implementation requires the additional interface classes `IAgent` that is provided by Voyager and `Imfold`. The `IAgent` class contains the methods for creating mobile agents (obtaining an agent *facet*), moving them to new locations and invoking the callback method. The `Imfold` class is required in Voyager as the framework forces the use of interfaces for invoking the methods of mobile agents.

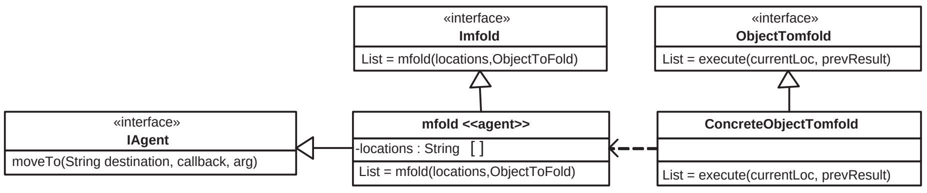

«interface»
Imfold
List = mfold(locations,ObjectToFold)
«interface»
ObjectTomfold
List = execute(currentLoc, prevResult)
«interface»
IAgent
moveTo(String destination, callback, arg)
mfold <<agent>>
-locations : String [ ]
List = mfold(locations,ObjectToFold)
ConcreteObjectTomfold
List = execute(currentLoc, prevResult)

Fig. 4. UML class diagram for mfold pattern in Voyager

7. **Consequences**
   – Advantages
      • The benefit of using the mfold pattern is that once you have encapsulated the required coordination behaviour of your mobile agent within the mfold object, you can attach any arbitrary OjectTomfold object to it that contains the correct `execute()` method and format. You can then effectively plug the functionality required from a set of predefined task objects, or simply create them as required.
      • The mobile agent can invoke resource operations locally, increasing performance through locality. These results are also remotely processed (i.e. reduced), which serves to distribute computation load.

    - The mobile agent can carry results with it as it travels through the network, eliminating the transfer of intermediate data.
    - The mobile agent can continue even if network links go down.
  - Disadvantages
    - Information transmitted from location to location is practically insecure. Although machines can be protected from foreign and migrating objects, migrating agents and the data they carry are typically unprotected from potentially malicious hosts [JK00]. This issue will be addressed in the secure form of the mfold pattern and skeleton (to be presented in a companion paper).
    - Reliability and fault-tolerance is an issue. If the mobile agent gets lost, for example, as a result of an infrastructure failure, the information retrieved is also lost. This issue will be addressed in the reliable form of the mfold pattern and skeleton (to be presented in a companion paper).
    - It may be possible that when the environment is stable and network links are reliable, a distributed system using mobile code may not perform as efficiently as traditional RMI or RPC approaches. This however depends on the mobile code platform being used whereby performance is subject to performance refinements.

8. **Known Uses** One such common example is when mobile agents are used in e-commerce to visit a set of potential product sources searching for prices. However, rather than returning with a list of all prices, the agent could use the mfold pattern and return only the best price. Mobile agents are currently used in e-commerce and are commonly referred to as shopping or e-commerce *bots* (for a list visit www.botspot.com/search/s-shop.htm) however the migration pattern of these mobile agents is unclear.
9. **Related Patterns and Frameworks**
   - Mobile Agent Itinerary Pattern [TOH99]
     The mfold is similar to the itinerary pattern present in [TOH99] where a single agent is used to itinerate through the destination locations performing a task at each location. However, the mfold pattern differs from this simple pattern as it also addresses the processing of the information returned from performing a task at each of the locations, providing yet a higher level of abstraction.
   - mmap
     The mmap pattern can be composed neatly with the mfold pattern. The mfold pattern can initially be used to locate a result common to all locations in the itinerary and the mmap to use these results (see section 4).

3.2 Mobile Map

1. **Name** mobile map (mmap)
   The name for the mmap pattern is inspired by the `map` higher-order function in functional languages that applies a function to every element in a list and

returns the resulting list. In an mmap invocation, a list is returned that is the result of executing a function on every location within the list of remote locations.

2. **Problem**
   – How can a customizable (mobile) computation be used to perform a task at a set of locations, while immediately returning the results from each location?
   – How can a customizable (mobile) computation be multicast to a set of remote locations, while hiding the migration details from the user?
3. **Motivating Scenario**
   You are designing a distributed information system. You need to implement an algorithm that will migrate a customizable mobile computation to each location within a list, perform a task and return the result. The system will be run on reliable/high bandwidth connections where the amount of data to be transmitted after each location visit is potentially relatively large.
   The distributed information processing task most suitable for this mobility pattern is distributed information retrieval of large amounts of unfiltered information. In contrast to the mfold pattern, the mmap returns the full results from each location immediately and does not carry the intermediate results from one location to the next. The main benefit of mmap mobility pattern is that the user can perform any task on the remote locations and is not restricted to predefined operations, for example in traditional client/server technologies such as remote method invocation (RMI) or remote procedure call (RPC). The user can also delegate the migration of the mobile code to the algorithm, which returns the results from all locations once complete.
   This pattern can also be used for distributed information dispersal of the same task to a set of locations, as observed in systems such as grid computing.
4. **Forces**
   Consider the mmap pattern when:
   – The application domain is characterised by decentralised resources, or the system can be thought of a set of co-operating components.
   – You want to multicast a customisable mobile computation either synchronously or asynchronously to a set of remote locations.
   – You want the flexibility to perform any processing task on the remote locations. The client/server model is too restricted in terms of services available as in this traditional paradigm, the server only offers a predefined set of services that may or may not accept code fragments as parameters.
   – The amount of data to be returned from performing a task at the remote locations may be relatively large and using the mfold pattern (and mobile agents) would under perform in relation to the amount of data to be transmitted through the network.
   – The user does not require any remote preprocessing of the results returned from performing a task at each location.
   – The system may be used on devices with relatively high processing capabilities.

   - Autonomy is required by the user, wherein they wish to delegate some task to a mobile computation.
   - Repetitive time consuming tasks can be delegated to a mobile computation to perform a task on behalf of the user and require only minimal intervention.
   - The user wants to control what task the mobile computation performs.
5. **Solution**
   Remote evaluation (REV) [AF98] has been chosen as the mobile paradigm most appropriate for the mmap mobility pattern and subsequent mobility skeleton. With remote evaluation, one location has the code to perform a task but does not have the required resources, which are located at a remote location. The location with the code migrates the code to the location with the resources, which then performs the task locally, using the resources as instructed and returning the results. A direct interaction between the source location and remote location exists with each interaction, as the code sent by the source always returns the data directly back to the source (see figure 5). Therefore, in essence the context of execution of remote evaluation is fundamentally limited to a single host location. This pattern can therefore be applied in situations where you require flexibility to multicast a task to a set of remote locations and the amount of data to be transmitted may be relatively large.

   The following platform independent models provide an abstract view of the mmap pattern, including the entities, their relationships and the operations that must be implemented in general.

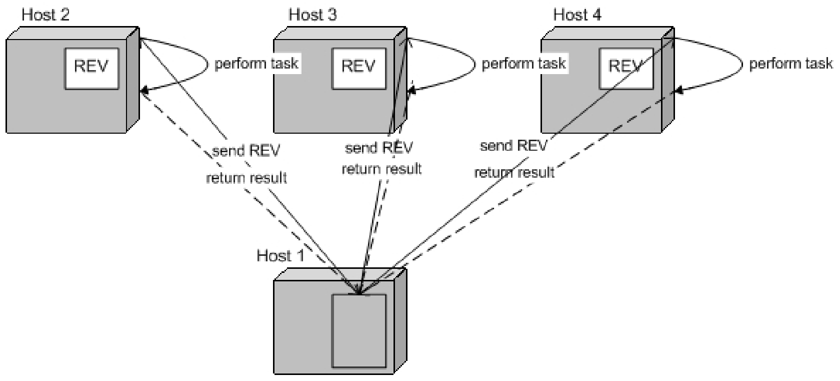


Fig. 5. Behaviour of mmap

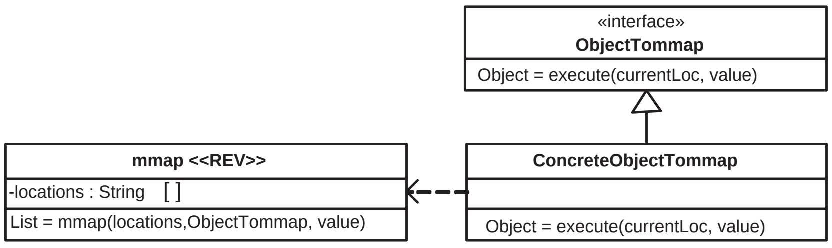
   The diagram in figure 5 illustrates the behaviour of the mmap pattern. The remote evaluation unit is created at host 1. It then migrates to host 2, performs the task with the resources locally and returns the result. This pattern

continues until all locations have been visited. Unlike the mfold pattern, the results from each location are stored at host 1 and no intermediate processing is performed.
The class diagram in figure 6 shows the simple structure of the mmap pattern and the participants. In practice, the mmap class is initiated with a list of locations and an ObjectTommap, which encapsulates the task to be performed at each location. An additional value can also be supplied at compile time, which is to be used by the ObjectTommap object i.e. the ObjectToMap holds the method to update a file and the value is passed as a parameter.

– **mmap** The mmap class contains the implementation for,
  - the sequential coordination algorithm responsible for migrating the remote evaluation unit to each location in the list,
  - invoking a polymorphic `Object = execute(currentLoc, value)`, whereby each call to this method is performed on the current location with the optional value.

– **ObjectTommap**
  The ObjectTomfmap interface, which must be implemented to use the pattern, forces the programmer to implement an `Object = execute(currentLoc, value)` method that conforms to the `Object = execute(currentLoc, value)` used by the mmap class to invoke the polymorphic method at each location.

– **ConcreteObjectTommap**
  This class contains the implementation for the `Object = execute(currentLoc, value)` method to be performed at each location.



«interface»
ObjectTommap
Object = execute(currentLoc, value)
mmap <<REV>>
-locations : String [ ]
List = mmap(locations,ObjectTommap, value)
ConcreteObjectTommap
Object = execute(currentLoc, value)

Fig. 6. UML class diagram for mmap pattern

The UML sequence diagram in figure 7 further illustrates the mmap pattern by showing the sequence of interactions involved. It shows how, after the instantiation of the remote evaluation unit with a list of locations and an ObjectTommap at location 1, it migrates to the first location in the list. It then invokes the `Object = execute(currentLoc, value)`, contained within the ObjectTommap. The result is then returned to location 1. This pattern continues until all locations have been visited. It should be noted here that you can perform these interactions either synchronously (as shown in figure 7 or asynchronously.

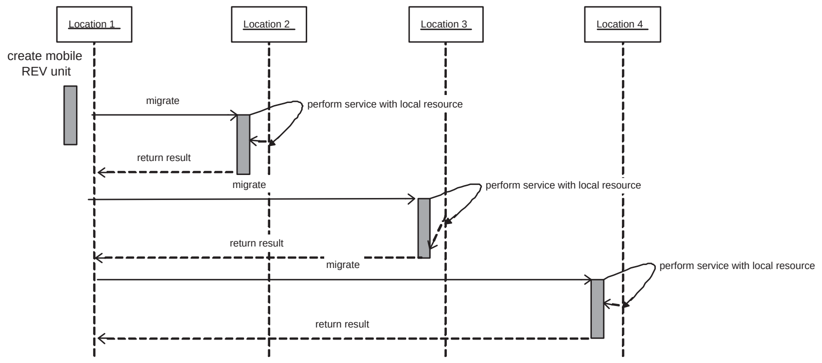


Fig. 7. Platform Independent Sequence Diagram for mmap Pattern

6. **mmap Mobility Skeleton implemented in Voyager**
The mobility skeleton for the mmap pattern using the remote evaluation paradigm and implemented in Voyager has the method signature,

```
List = mmap(ObjectTommap obj, String[] lo, Object value)
```

and takes as its parameters, an object `obj` which is converted into a mobile object by Voyager's *dynamic aggregation*. It then migrates to the first location in the list `lo`, passed as the second argument. A task is then invoked, `execute(currentLoc, value` using the `value` as the parameter to that method. The result is then returned. This continues until the remote evaluation has been performed at all locations. In contrast to the mfold skeleton, the mmap skeleton stores the results within the initiating program. When all results have been returned, a `list` of all results is returned to the user.
The UML class diagram in figure 8 shows how the Voyager implementation of the mmap pattern/skeleton requires an additional `IMobility` class, provided by Voyager, which contains the methods for creating a mobile object (obtaining a mobile *facet*) and transparently moving objects to new locations.

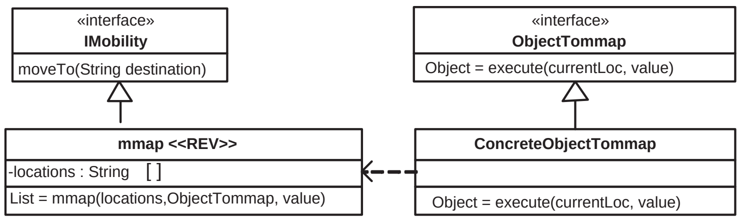

«interface»
IMobility
moveTo(String destination)
«interface»
ObjectTommap
Object = execute(currentLoc, value)
mmap <<REV>>
-locations : String [ ]
List = mmap(locations,ObjectTommap, value)
ConcreteObjectTommap
Object = execute(currentLoc, value)

Fig. 8. UML class diagram for mmap pattern in Voyager

7. **Consequences**
   - Advantages
     - The benefit of using the mmap pattern is that once you have encapsulated the required coordination behaviour of your remote evaluation within the mmap object, you can attach any arbitrary OjectTommap object to it that contains the correct `execute()` method and format. You can then effectively plug the functionality required from a set of predefined task objects, or simply create them as required.
     - The remote evaluation can invoke customizable resource operations locally, increasing performance through locality.
     - The remote evaluation can continue even if network links go down, with result returned when the network link is regained.
     - The security of the results obtained from each location becomes less of an issue as compared to the mfold pattern, as intermediate results are not carried from location to location.
   - Disadvantages
     - The results from the distributed information processing task may still need to be processed on the initiating host, in contrast to the mfold pattern where the results are preprocessed and reduced at the remote locations. In extreme circumstances, this could lead to server overload.
     - Reliability and fault-tolerance is an issue. If an infrastructure failure occurs on the remote location, the remote evaluation, and thus results are lost. This issue will be addressed in the reliable form of the mmap pattern and skeleton (to be presented in a companion paper).
8. **Known Uses**
   This type of pattern can be observed in branching and merging patterns [TOH99] that are used in software development (version control tools). Branching can be likened to the Unix fork that creates a new thread of execution, which incidentally corresponds to the rfork function used in the mHaskell implementation of the mmap skeleton. [DBTL05]. Another more prominent example of such a pattern being observed in distributed computing is the use of remote evaluation is grid computing. The mmap pattern would simplify the programmers task when performing the same computation on a set of remote computers in the grid.

4 Executable Mobility Patterns in Practice: An Example

4.1 Mobile Meeting Scheduler

A mobile, automatic meeting scheduler is designed that exploits code mobility by distributing computation load and reducing network interactions. Users request a meeting and the scheduler automatically checks the availability of peers by sequentially migrating to their respective calendar locations performing availability assessments through local interactions (*use the mfold pattern here!*). When the first common time is identified for all peers, a meeting allocation will be broadcast that automatically updates their calendars (*and the mmap here!*).

This application can be built by composing the mfold and mmap patterns, wherein the mfold is used to perform the calendar assessments, carry the intermittent results and finally return the results to the initiating program. The mmap pattern is then used to multicast the results.

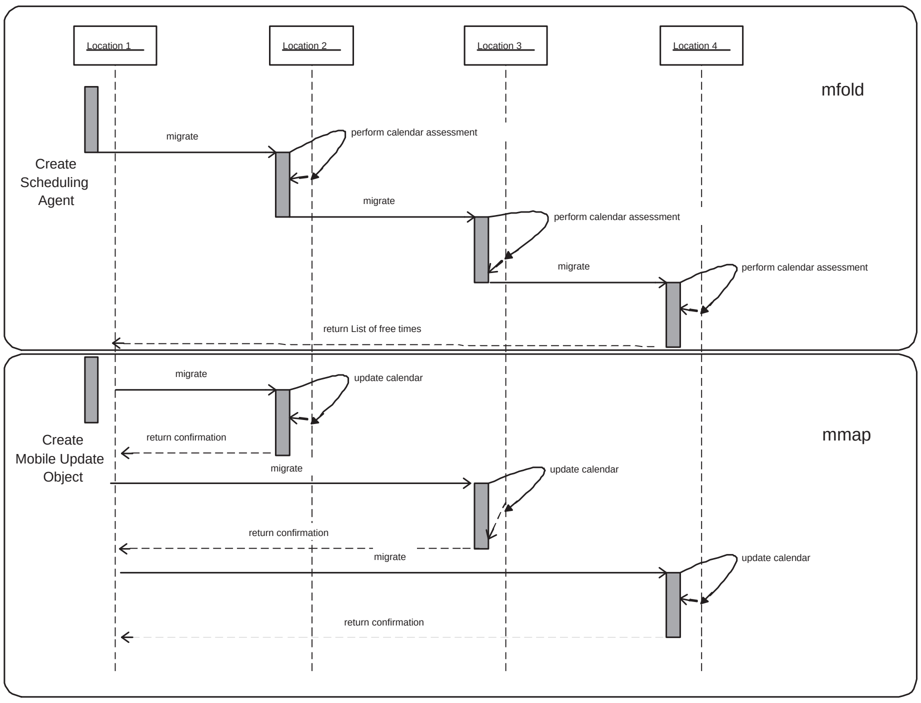


Fig. 9. Sequence diagram for mobile meeting scheduler

The agents mfold interactions are depicted in figure 9. Upon creation, the agent accesses the local calendar for a list of free times available. These intermediate free times are then carried with the agent to the next location and intersected with the free times at the new location. The agent traverses the list of locations performing an intersection between the newly acquired list of free times and the result of the previous intersection, whereby the final result of the distributed and systematic intersections is a list of common free times to all peers. At the last location, the intersection is performed for the last time and the first free time from the resulting list is returned to the initiating program.

Figure 9 then shows how the initiating program, after receiving the result, multi-casts it to all peer calendars. A synchronous mmap method provides confirmation that the calendars have been updated.

The following implementation code, in Voyager, shows how mfold and mmap patterns are composed to form a multifunctional and flexible distributed information system.

```
 public class meetingScheduler{
    public static void main (String[] args)
    {
    String[] s = new String[3];
    s[0] = "//linux25:8000";
    s[1] = "//linux26:8000";
    s[2] = "//linux29:8000";
    s[3] = "//linux30:8000";
    List results;
     try{
        Voyager.startup("8000");
            }catch(Exception e){}
       IObjectToFold b = new ObjectToFold(); // Object used for the fold
       IObjectToMap c = new ObjectToMap(); // Object used for the map
        try{
        mfold f = new mfold();
        results = f.mobilefold(b,s);  // mfold skeleton
        if(results.size()==0){ // no result
         Voyager.shutdown();
        }
        mmap m = new mmap();
        List l = m.map(c,s,results.get(0).toString()); // mmap skeleton
        for(int i = 0 ; i< l.size(); i++){
            System.out.println(l.get(i).toString());
        }
        Voyager.shutdown();
        }catch(Exception e){}
    }
}
```

5 Conclusion and Further Work

The patterns and corresponding skeletons are designed to improve the communication and comprehension of the mobility concepts they describe, thus providing more support for the development of distributed information system applications that may benefit from code mobility.

The consequence of defining such patterns and corresponding skeletons is that the developer can use the patterns and skeletons to create a number of reusable modules (classes) that coordinate mobile code using a variety of mobile paradigms. A set of objects can then be created that perform alternative distributed information processing tasks. In effect, the programmer will have a dynamic and very flexible set of predefined tools suitable for any distributed information processing task.

We will extend this work by formalising additional distributed information processing mobility patterns. These patterns, and more significantly the skeletons, will be extended to ensure the reliability and fault-tolerance of the mobile computations.

References

[AF98] Giovanni Vigna Alfonso Fuggetta, Gian Pietro Picco. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, may 1998.

[AL98] Yariv Aridor and Danny B. Lange. Agent design patterns: elements of agent application design. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, pages 108–115. ACM Press, 1998.

[BGM$^+$99] Brian Brewington, Robert Gray, Katsuhiro Moizumi, David Kotz, George Cybenko, and Daniela Rus. Mobile agents in distributed information retrieval. In Matthias Klusch, editor, *Intelligent Information Agents*. Springer-Verlag: Heidelberg, Germany, 1999.

[BGP97] Mario Baldi, Silvano Gai, and Gian Pietro Picco. Exploiting code mobility in decentralized and flexible network management. In *Proceedings of the First International Workshop on Mobile Agents*, pages 13–26, Berlin, Germany, 1997.

[BP98] Mario Baldi and Gian Pietro Picco. Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications. In R. Kemmerer, editor, *Proceedings of the 20th International Conference on Software Engineering*, pages 146–155. IEEE Computer Society Press, 1998.

[BSH02] W. Binder, G. Scrugendo, and J. Hulaas. Towards a secure and efficient model for grid computing using mobile code, 2002.

[CH02] Wen-Shyen E. Chen and Chih-Lin Hu. A mobile agent-based active network architecture for intelligent network control. *Inf. Sci. Inf. Comput. Sci.*, 141(1-2):3–35, 2002.

[CPV97] Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna. Designing distributed applications with a mobile code paradigm. In *Proceedings of the 19th International Conference on Software Engineering*, Boston, MA, USA, 1997.

[CTZ02] Paolo Ciancarini, Robert Tolksdorf, and Franco Zambonelli. Coordination middleware for xml-centric applications. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 336–343, New York, NY, USA, 2002. ACM Press.

[DBTL05] André Rauber Du Bois, Phil Trinder, and Hans-Wolfgang Loidl. Towards Mobility Skeletons. *Parallel Processing Letters*, 15(3):273–288, 2005.

[EGV95] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. *Design Patterns - Elements of Reusable Object-Orientated Software*. Addison-Wesley, 1995.

[FFMS] Cedric Fournet, Fabrice Le Fessant, Luc Maranget, and Alan Schmitt. Jocaml: a language for concurrent distributed and mobile programming.

[GS01] J. Gomoluch and M. Schroeder. Information agents on the move: A survey on loadbalancing with mobile agents, 2001.

[HfL02] Minghua He and Ho fung Leung. Agents in e-commerce: state of the art. *Knowl. Inf. Syst.*, 4(3):257–282, 2002.

[JK00] W. Jansen and T. Karygiannis. Nist special publication 800-19 - mobile agent security, 2000.

[KA03] Ryszard Kowalczyk and Leila Alem. Supporting mobility and negotiation in agent-based e-commerce. pages 226–244, 2003.

[KBD02] Hairong Kuang, Lubomir F. Bic, and Michael B. Dillencourt. Iterative grid-based computing using mobile agents. In Tarek S. Abdelrahman, editor, *Proceedings of the 2002 Intternational Conference on Parallel Processing*, pages 109–117, Los Alamitos, Calif., aug 2002. The International Association for Computers and Communications (IACC), IEEE Computer Society.

[ken97] *The Layered Agent Pattern Language*, 1997.

[KGR02] David Kotz, Robert S. Gray, and Daniela Rus. Future Directions for Mobile-Agent Research. Technical Report TR2002-415, Hanover, NH, 2002.

[KKPS98] Elizabeth A. Kendall, P. V. Murali Krishna, Chirag V. Pathak, and C. B. Suresh. Patterns of intelligent and mobile agents. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 92–99, New York, 9–13, 1998. ACM Press.

[Kue] Deugo Oppacher Kuester. Ic-ai'99 502sa patterns as a means for intelligent software engineering.

[Lan98] Danny B. Lange. Mobile objects and mobile agents: The future of distributed computing? *Lecture Notes in Computer Science*, 1445, 1998.

[LMSF04] Emerson F. A. Lima, Patricia D. L. Machado, Flavio R. Sampaio, and Jorge C. A. Figueiredo. An approach to modelling and applying mobile agent design patterns. *SIGSOFT Softw. Eng. Notes*, 29(3):1–8, 2004.

[LO99] D. Lange and M. Oshima. *Programming And Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1999.

[MR04] Beniamino Di Martino and Omer F. Rana. Grid performance and resource management using mobile agents. pages 251–263, 2004.

[MRK96] T. Magedanz, K. Rothermel, and S. Krause. Intelligent agents: An emerging technology for next generation telecommunications? In *INFOCOM'96*, San Francisco, CA, USA, 24–28 1996.

[SD98] R. Silva and A. Delgado. The agent pattern: A design pattern for dynamic and distributed applications, 1998.

[STO99] J. W. Shepherdson, S. G. Thompson, and B. R. Odgers. Decentralised workflows and software agents. *BT Technology Journal*, 17(4):65–71, 1999.

[TKI$^+$97] Hiroyuki Tarumi, Koji Kida, Yoshihide Ishiguro, Kenji Yoshifu, and Takayoshi Asakura. Workweb system multi workflow management with a multi-agent system. In *GROUP '97: Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 299–308, New York, NY, USA, 1997. ACM Press.

[TOH99] Yasuyuki Tahara, Akihiko Ohsuga, and Shinichi Honiden. Agent system development method based on agent patterns. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 356–367. IEEE Computer Society Press, 1999.

[Wei03] Michael Weiss. A pattern language for motivating the use of agents. In *AOIS*, pages 142–157, 2003.

[Whe05] Thomas Wheeler. Voyager architecture best practices. March 2005.

[Whi94] Jim White. Mobile agents white paper. 1994.

[WT02] Christian Wagner and Efraim Turban. Are intelligent e-commerce agents partners or predators? *Commun. ACM*, 45(5):84–90, 2002.