NetMorph - an intuitive mobile object system

Masashi Umezawa, Kazuhiro Abe, Satoshi Nishihara, Tetsuya Kurihara
IPA Exploratory Software Project FY2002
ume@mamezou.com, abee.abe@nifty.ne.jp, nishis@zephyr.dti.ne.jp, kuri-t@mamezou.com

Abstract

NetMorph is a mobile object system that provides a seamless integration of network and desktop. We introduce the notion of network location to desktop objects. By combining desktop 2D coordinates with network locations, NetMorph provides free intuitive cyber spaces. It helps computer beginners to acquire skills of network computing in a very natural way.

1 Introduction

Currently, computers are rarely run by stand-alone. Almost all of them are connected by networks. Computers become more and more powerful and they are getting synergy effects by interacting each other. The rapid growth of the Internet is symbolically showing that power. The Internet has drastically changed human way of collaboration and has become a basic infrastructure of our daily life. Nowadays understanding the notion of distributed computing is the essential part of computer literacy.

There are some interesting systems that support beginners to understand computing in interactive, intuitive ways [1]. However, these environments basically lack the support of learning distributed computing.

Therefore, we propose a new visualized mobile object system for seamless distributed computing. It saves beginners from worrying about unnecessary details of network architectures. By using the system, even beginners can develop distributed applications in a highly intuitive way.

1.1 Problems of the present networking

Most networking systems do not provide proper intuitive interfaces for us. Users have to acquire specific knowledge depending on various protocols.

This is a real burden for computer beginners. Nowadays, we can see many ‘wizards’ to hide these complexities, but it does not solve real problems.

In many cases, users can only use prepared network applications. Developing distributed programs by themselves is tremendously difficult.

1.2 Visible mobile objects

In Squeak [2], there is a scripting environment called SqueakToys [3] [4]. It enables us to manipulate objects interactively and intuitively. Users can freely edit actions and attributes of a visual object (called Morph) by attaching tile phrases. Even children can write interesting applications in the visual scripting environment.

All objects are visualized. They return vivid feedbacks to users. This feature greatly helps beginners to acquire the skills of writing application programs.

For example, Morph has two-dimensional coordinates (x, y values). A user can inspect these values by a viewer. If the user attaches a tile script that changes these values, the viewer shows the updated values instantly, while Morph simply moves around according to these values.

However, Morph is so-called “a bird in a cage”. The range that Morph can move around is strictly restricted within a narrow desktop of the local computer.

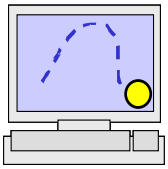

Figure 1.The movable space of Morph is limited to one narrow desktop.

We would like to extend this interactive environment to networks. The new system is called NetMorph. By equally treating the movement of objects in a desktop and the migration of them in network nodes, the system

Proceedings of the First Conference on Creating, Connecting and Collaborating through Computing (C5’03)
0-7695-1975-X/03 $17.00 © 2003 IEEE

drastically reduces the difficulties of distributed computing. Users can change their local application programs into network-enabled ones with no special efforts.

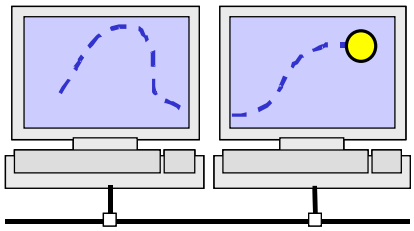

Figure 2. NetMorph system provides logical broad space, where objects can move seamlessly

1.3 Differences from related systems

NetMorph is not one of the so-called "mobile agent" environments [5] [6]. Such environments are not so intuitive, because they do not support visual representations of mobile objects. Mobile objects basically serve as invisible 'daemon'. These environments do not provide the notion that network world can be integrated with the desktop world.

Also, there are some environments that support interactive cooperative work among remote users by using desktop metaphors [7] [8]. However, these systems are based upon the concept of shared desktop and they do not have the notion of mobile objects.

1.4 Basic concept

In this system, we add the third logical network dimension to morphs. Each morph has its own "network location" as well as its desktop position. This system broadens the notion of object movable space to networks.

The location of Morph is represented by the combination of 2D coordinates and logical network address. Morph can move to other nodes, simply changing its location values.

The actual computer where Morph lives can be detected by referring to a "Map". The map manages the sequence of desktop relations.

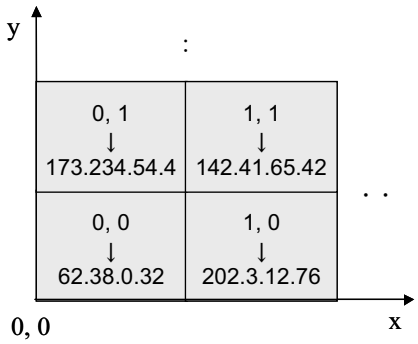

Figure 3. Map manages mapping of logical Morph locations and physical computers

When a morph object reaches a boundary of a desktop, it does not bounce and simply begins to migrate to another desktop, which is logically next to the original desktop. From users' point of view, there is no big difference between moving objects on the screen and distributing them to the network.

For example, if 4 host machines are mapped to a 2D rectangular plot, users can program a network patrol morph that visits all nodes and returns to the original node by simply writing a "draw a circle" script.

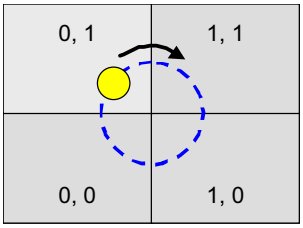

Figure 4. A "draw a circle" scripted morph can be changed into a network patrol morph

1.5 Applications

As we have seen, NetMorph concept is very simple. However, we can think many application usages.

Suppose an e-mail-like messaging system. When developing such system, there is no need to define or investigate primitive network protocol such as SMTP, POP, etc. A user can simply create a "mail" object and then drag and pitch it to the target location.

Suppose some collaborative work. We do not have to mind shared network file directories or FTP. It is sufficient by exchanging the document objects that we are working on.

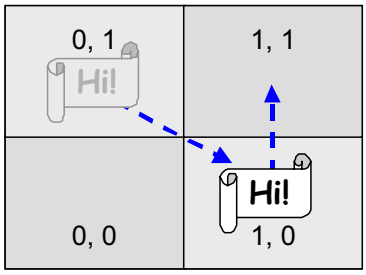

Figure 5. A simple circular notice application built on NetMorph system

NetMorph can be considered as "LOGO in network age". In LOGO, simple recursive movements of turtle often generate an unexpected, beautiful drawing [9]. In NetMorph, combining simple mobile objects can create very sophisticated collaborative work that we have never seen.

For example, if someone writes a car morph that follows a path on the screen and then if another person writes a pen morph that draws a line on desktops, it would be possible to create a network car race game by using these morphs.

As another example, if we add the property of "computation load" to each World, and a morph is programmed to move toward a lower-load location, an optimized distributed computation program can be developed.

2 Design and implementation

We have developed NetMorph system in Squeak 3.2. In this section, we'll discuss its design and implementation details.

2.1 Migration ('warp drive') mechanism

First, we added new methods to the existing Morph class in order to support network migration (we call the migration feature 'warp drive').

The `#warp` method checks whether a Morph is beyond the boundary of the local desktop (World bounds). If so, that Morph will move out to another desktop. The Morph's next desktop is detected by referring to a WorldMap, because WorldMap maintains the desktop locations described in URL format (NmUrl). The x y values of Morph in the new desktop can be computed according to the original desktop position. (The system supposes that incoming screen size is the same as the outgoing ones. When the desktop sizes are different, the system will automatically adjust x y values).

```
Morph>>warp
  ...
myBox ← self bounds.
ownerBox ← ActiveWorld bounds.
myBox right < ownerBox left
  ifTrue:
    [destination ← NmWorldMap
      leftOf: NmUrl local.
    destination notNil
      ifTrue:
        [self warpTo: destination
        x: ownerBox width
        y: myBox top]].
  ...
```

In the `#warp` method, `#warpTo:x:y:` is called. Actual migration occurs as follows:

At the source desktop:

1. A Morph remembers a current desktop size for the later x y adjustments.
2. The Morph serializes itself.
3. The Morph sends the serialized data to the destination desktop by NMP (NetMorph Protocol).
4. The System invokes a '`warpedOut`' event handling method that is usable for additional user defined behaviors.

```
Morph>>warpTo: destination x: x y: y
...
[self formerWorldBounds: ActiveWorld bounds.
morphByteArray ← self toWarpByteArray.
connector ← NmRemoteConnector
  connect: destination hostOrIpAddress
  port: destination port.
connector callCatchAndGo: morphByteArray
  x: x
  y: y.
renderedMorph warpedOut: destination]
  on: Error
  do: [:ex | self handleWarpError: ex]
```

At the destination desktop:

5. A migration server (called Catcher) receives the serialized data and deserializes it to the Morph .
6. The Catcher adjusts the Morph position by comparing the source desktop size and the destination desktop size.
7. The system attaches the Morph to the destination desktop. Typically, the Morph is serialized in motion, so at this time it immediately begins to move in the new desktop.
8. The System invokes a 'warpedIn' event handling method for additional user defined behaviors.

```
NmCatcher>>catchAndGo: morphByteArray x: x y: y
  ...
read ← RWBinaryOrTextStream
  with: (NmUtil
    gunzippedFrom: morphByteArray) asString.
Cursor read
  showWhile: [newMorph ←
    Morph readFrom: read].
self relocate: newMorph x: x y: y.
ActiveWorld addMorph: newMorph.
newMorph renderedMorph
  warpedIn: NmUrl local.
↑ nil.
```

2.2 Serialization in warp

When #warp is invoked by SqueakToys tile scripting system, the Morph must pack its 'player' class definitions as well as its instance data. This is because Player has script implementations of the related Morph. If the Player class definition is not correctly imported to the destination desktop, the Morph will lose its scripted behaviors.

To avoid the problem, we use SmartRefStream for serialization, which is a built-in object serializer in Squeak. SmartRefStream can automatically serialize class definitions as well as instance data. It greatly simplifies the class-loading problem seen in some other environments (like Java).

The shortcoming is its performance. SmartRefStream is a general serializer. And it does not do any optimizations for Morph and Player classes. We may develop a customized serializer for NetMorph.

2.3 Network communication between Morphs

After a morph goes out, there are situations that we still want to control the warped morph from the original desktop. For example, if we write scripts so that a joystick morph controls a car morph, and the car morph goes out to a neighboring desktop, we expect that the car morph is still controllable remotely from the local desktop.

Coping with such cases, we have implemented network communication ability in Morph.

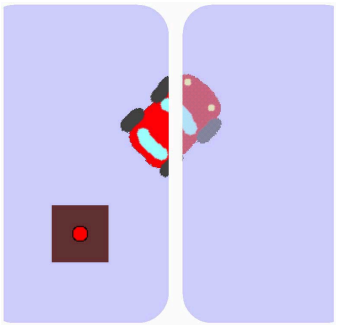

Figure 6. A car morph should be controllable remotely

2.4 Finding a morph in networks

To realize the communication, first we had to detect a morph in networks. Therefore, we added an object identifier attribute to Morph class. To identify a morph in networks, identifier value should be universally unique. Fortunately, UUID class is available in Squeak 3.2. Therefore, simply we were able to use it. When the Morph's #identifier method is first called, each Morph has its own identifier by lazy initialization.

We also added a class method (#fromUUID:) to Morph class so that we can specify an identifier to get the same Morph. A basic search algorithm is as follows:

1. First, ActiveWorld (local) is searched.
2. If a Morph is found, the Morph is returned to the caller. If not, local cache is searched.
3. If found in the cache, a cached Morph proxy is returned. If not, remote images that are registered in WorldMap are then searched.
4. If found, a Morph proxy is returned to the caller. If it is not found, nil is returned.

```
Morph>>fromUUID: anUUID
  ...
detection ← ActiveWorld allMorphs
  detect: [:morph |
    morph identifier = anUUID]
  ifNone: [].
detection isNil ifFalse: [↑ detection].
netMorphProxy ← NmUtil netMorphProxyCache
  at: anUUID
  ifAbsent: [].
netMorphProxy isNil
  ifFalse: [↑ netMorphProxy].
morphProxy ← NmUtil
  callFindBy: anUUID
  retry: 3.
morphProxy isNil
  ifFalse: [NmUtil netMorphProxyCache
    at: anUUID
    put: morphProxy.
  ↑ morphProxy].
↑ nil
```

At the step 3, remote servers are searched by calling 'Finder' services.

Finders are servers running on each NetMorph images. When a Finder receives a request, it tries to search its own local environment.

As a remote search result, we should not return a morph itself, because copied morphs will be created in both client and server images. Therefore, a MorphProxy is returned instead.

MorphProxy is a typical proxy class, described in GoF book [10].

MorphProxy's superclass is ProtoObject. ProtoObject implements almost no methods. Messages sent to the ProtoObject are handled in a `#doesNotUnderstand:` hook mechanism. In a subclass, we can implement a hook to delegate messages to others. This is a well-known Smalltalk idiom to implement a generic proxy.

2.5 Filtering the targets of the finder

It is desirable to achieve a fast response in finding a remote morph, because finding requests can occur in a very short term. At the step 3 of the previous section, we can omit the remote server entries that are not available for a while.

A utility class, AvailabilityChecker periodically checks if the registered entries of WorldMap can respond to a 'ping' message. Finder does not refer to WorldMap entries directly but uses AvailabilityChecker's available entry list for a remote morph search. It avoids useless timeouts and improves the 'find' performance much.

2.6 Finding a player

In order to support the tile scripting, Player should also be identified like Morph. In a tile script, a morph's methods are called by a related player. Basically there is a one to one relationship between Morph and Player. Therefore, we can simply identify a Player by sending `#player` method to the identified Morph.

We changed a script code generator so that Players can always refer to other Players by identifier.

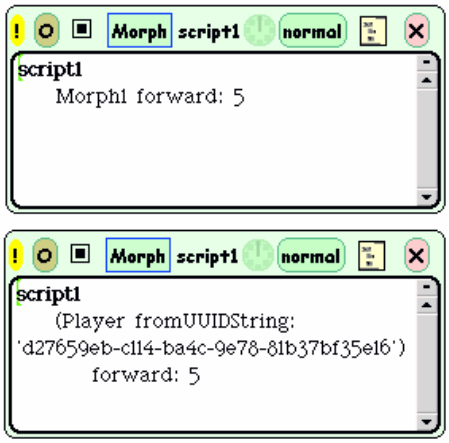

Figure 7. A new script uses UUID for referring to a player

In a new generated code, when a player needs to be referred to, a constructor method `#fromUUIDString:` is always called. In that method, a player is searched by identifier. The method implementation is as follows:

```
Player>>fromUUIDString:anUUIDString
  ...
morph ← Morph fromUUIDString: anUUIDString.
morph isNil ifTrue: [↑ nil].
↑ morph player
```

2.7 Remote message sending between morphs

After a morph is found in networks, we get a proxy object as a return value. By sending messages to the proxy, we can communicate to the remote morph.

MorphProxy has an original morph's identifier and an NmUrl, which specifies the hostname and port of the original morph. By using this information, the MorphProxy can send remote methods to an appropriate image where the actual morph exists. MorphProxy overrides #doesNotUnderstand: to do the job, but actual delegations are performed in private helper methods.

```
NmMorphProxy>>
xxxCallWithArgumentsDispatch: aMessage
target: targetString
  ...
arguments ← aMessage arguments.
Arguments withIndexDo: [:arg :idx | arg
  shouldPassAsProxy
    ifTrue:
      [arguments at: idx put: arg asProxy]].
connector ← NmRemoteConnector
  connect: url hostOrIpAddress
  port: url port.
↑ connector
  callDispatch: uuid asString
  to: targetString
  selector: aMessage selector
  withArguments: aMessage arguments
```

We cannot expect what kind of message is sent to MorphProxy. Therefore, we prefixed 'xxx' to the helper methods in order to avoid conflicts with the real methods in Morph. Also note that the method checks if the arguments should be passed as proxy by sending #shouldPassAsProxy. It is important to prevent from sending unneeded argument copies to the destination image. Currently Morph and Player implement the message to return true.

The requested message is delegated to the destination host by a RemoteConnector, which uses NMP(NetMorph Protocol) as a transport protocol.

In a server side, a Dispatcher receives the delegated request and dispatches its message to the actual morph. The morph performs the message and returns a value.

The Dispatch method implementation is as follows:

```
NmDispatcher>>dispatch: uuidString to: target
selector: selector withArguments: arguments
  ...
morph ← NmFinder findMorphBy: uuidString
  ifNone: [NmNoSuchMorph
    signalWithUuidString: uuidString].
returnObject ←
  [(self performer: target for: morph)
    perform: selector asSymbol
    withArguments: arguments]
    on: MessageNotUnderstood
    do: [:ex | ...].
↑ returnObject shouldPassAsProxy
  ifTrue: [returnObject asProxy]
  ifFalse: [returnObject]
```

Like the arguments, the return value is checked whether it should be returned as a proxy.

Actually there is also another proxy, PlayerProxy. However, we will omit the explanation because the basic mechanism is the same.

2.8 WorldMap management

WorldMap performs a very important role in NetMorph. Its data is checked when Morph warps out and when Morph is found for remote method invocations.

In order that these features work correctly, Map data must be managed consistently among NetMorph images.

At least one MapServer is needed in NetMorph system. MapServer provides the latest map data for NetMorph images. WorldMap is basically a simple dictionary. The dictionary's key is x y coordinates and its values is NmUrl. Therefore, the map data can be converted as a simple CSV file.

```
0@0,nm://host1:37458/
1@0,nm://host2:37458/
```

Each NetMorph image periodically retrieves the CSV map data from the MapServer. The MapServer is implemented as SOAP services, which are convenient to passing through firewalls and providing a setting tool for a web browser [11]. Map data will be browsed from a

normal web browser. However, currently this feature has not been implemented yet. From a NetMorph image, we can edit a map data graphically by using a WorldMapMorph.

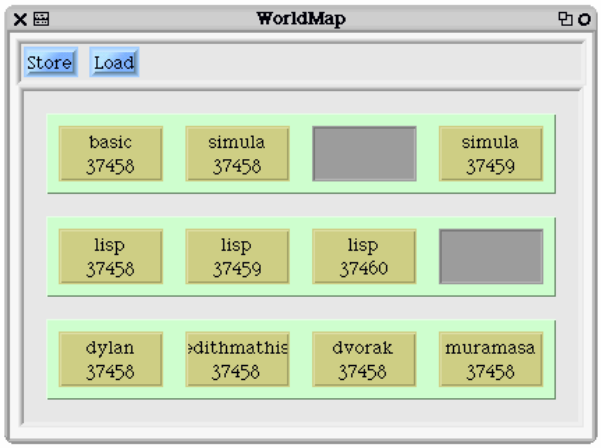

Figure 8. WorldMapMorph

2.9 IP address management

To resolve an IP address from a hostname, Squeak has a utility class, NetNameResolver.

However, NetNameResolver does not provide enough performance for us. In the highly interactive NetMorph applications built by scripts, the remote operations `#warp`, `#findBy:`, and `#dispatch:` are called in a tremendously short interval , because Players have many scripts whose ticking rate are 8 times per a second. We need to resolve target IP addresses to invoke these remote methods, but NetNameResolver does not have a cache and takes rather long time to resolve addresses.

Therefore, we decided to implement a faster name resolver mechanism.

In NetMorph, NmNameResolver resolves names instead. It has a simple dictionary cache; its key is a hostname and its value is an IP address.

The NetMorph clients periodically pull the cache from a master NmNameResolver via SOAP. Typically, a master NmNameResolver runs on a remote, well-maintained machine. The clients also put their own name and IP associations to the master NmNameResolver at regular intervals.

2.10 NetMorph protocol (NMP)

NetMorph uses a proprietary transport protocol for supporting the warp drive and the remote messaging. To gain a good performance, we have developed a rather primitive protocol.

First, NMP has a notion of connection. There is a connection pool in client side. The pool maintains the connections per remote server NetMorph image. The NMP connection pool reuses connections when possible.

Secondly, NMP is a binary protocol so that its payload would be smaller than other text-based transfer protocols such as HTTP.

Its message header format is as follows:

```
'NMP<version>'  '<semantic>'  '<command>'
'<encoder>' *'<size>'!'
```

<**version**> is a NMP version. Currently '01' is used.

<**semantic**> indicates the message semantics. In NMP, #ONEWAY means the message wants no return.

<**command**> indicates specific commands to be performed in a server. Warp command (`#warp:x:y:`), find command (`#findBy:`), and remote message commands (`#dispatch:to:selector:withArguments:`, `#dispatch:to:selector:`) are supported. There are also #RETURN and #ERROR commands for sending back normal returns and errors respectively.

<**encoder**> indicates the encoder that is used for encoding message body.

<**size**> indicates the byte sizes of the following bytes of the body.

The body is mere bytes array that is encoded in <**encoder**> format.

A NetMorph initial prototype used SOAP over HTTP for warp and remote messaging. Compared with that, the current version that uses NMP shows remarkable performance gain because of its connection pooling and small payloads features.

3 Conclusions and future work

NetMorph development is still under way. We will continue the work until February 2003. The implementation is found at NetMorph portal site (http://swikis.ddo.jp/NetMorph). In order to be easily installable, NetMorph will also be available on SqueakMap (http://map2.squeakfoundation.org/sm).

We have already demonstrated NetMorph several times. The impressions were generally good. However, we feel more steps are needed in order that NetMorph will achieve a desirable easiness for all end-uses. We will conclude this paper by pointing out the future plans.

3.1 Field tests

We performed the first field test of NetMorph in CSK Okawa-center (CAMP) on 2nd February 2003. Although we do not describe it closely in this paper, we were able to have the feeling that some children found a new pleasure in controlling networked computers. Sometimes bugs prevented children from developing NetMorph projects. We think these are quite serious problems that should be fixed in the next release.

3.2 More sample applications

To show the applicability of NetMorph as a new collaboration tool, currently we have these ideas of sample applications:

- NetHelper

A helper morph that collects Q&A from various users, drifting in networks.

- Camera NetMorph

A remote control camera morph that moves to a target desktop and takes a screenshot.

- Tour Band NetMorph

A band morph that plays on someone's desktop (sometimes performs a jam session with other bands).

- Treasure Hunter NetMorph

A network collaboration game to compete to find a treasure fast.

- FlyingTools

Movable editors for team development (DistributedWorkspace, DistributedInspector, etc.)

3.3 More map topologies and NetMorph biosphere

It is also interesting to change mapping scheme of network nodes. Mapping shapes are not limited to 2D. If we define a ring, cube or sphere-shaped map, it is possible to write a NetMorph that would automatically travel around all the nodes.

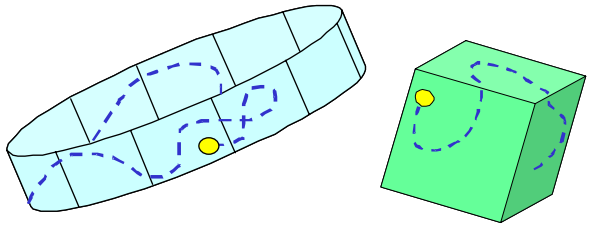

Figure 9. 3D map can be defined

It is even possible to add interesting properties to a WorldMap itself. For example, adding the notion such as temperature, wind and ocean currents can make the NetMorph environment a distributed virtual biosphere.

4 References

[1] Ken Kahn, "ToonTalk - Making programming child's play", http://www.toontalk.com, 2002

[2] D. Ingalls, T. Kaehler, J. Maloney, S. Wallace, and A. Kay, "Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself", OOPSLA '97, 1997.

[3] Alan Kay, "Etoys and Simstories in Squeak", http://www.squeakland.org/author/etoys.html

[4] John Maloney and Randall Smith, "Directness and Liveness in the Morphic User Interface Construction Environment", UIST '95, 1995.

[5] IBM Tokyo Research Laboratory, "Aglets", http://www.trl.ibm.com/aglets/index_e.htm, 2002

[6] Recursion Software, Inc., "Voyager", http://www.recursionsw.com/products/voyager/, 2002

[7] Sun Microsystems, Inc., "The Kansas Project A Vast, 2D Space for Realtime Collaboration", http://research.sun.com/ics/kansas.html

[8] Lex Spoon and Bob Arning, "Nebraska", http://minnow.cc.gatech.edu/squeak/1356

[9] Seymour Papert, *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, New York, 1980.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Massachusetts, 1995

[11] World Wide Web Consortium, "SOAP Version 1.2", http://www.w3.org/TR/soap12-part0/, 2003