MOBILE AGENT ARCHITECTURE FOR NOMADIC COMPUTING

Andrzej Duda, Stéphane Perret
LSR-IMAG,
BP 72,
38402 Saint Martin d'Hères Cedex, France
e-mail: Andrzej.Duda@imag.fr, Stephane.Perret@imag.fr

ABSTRACT

Our goal is to define a communication architecture for nomadic applications. We notice that the mobile agent paradigm is particularly suitable for nomadic applications. Although many mobile agent systems are being proposed, few of them raise the problem of the interface between applications and mobile agents. We propose to use the MAP (*Mobile Assistant Programming*) architecture and add a middleware layer that adapts MAP to nomadic environments. The nomadic adaptation layer provides enhanced services to nomadic applications: result collector is managed as a predictif cache, partial or approximate results may be delivered, and network connections are optimized. An application controls the layer by defining politics to adopt for connection management and result delivery.

1 Introduction

Main issues in nomadic computing include mobility, communication performance, and application support.

To achieve nomadicity, a nomadic host should be able to move from place to place while operating as effectively as if it were connected to a fixed network. Mobility is the distinctive feature of nomadic hosts that may either use wireless connections to communicate while moving or connect intermittently to different wired networks. The desirable characteristics for nomadicity include independence of location, of motion, of platform along with widespread presence of access to services [13]. This goal can be achieved at the network layer, for example by means of *mobile IP* that allows a mobile host to send and receive packets addressed with its home IP address regardless of its current point of attachment.

A mobile host may use various communication supports:

- wireless low bandwidth connections over long distances (cellular - 2.4 Kbit/s, GSM - 9.6 Kbit/s),
- wireless medium bandwidth connections over small distances (waveLAN - 2 Mbit/s),
- low bandwidth modem connections (28.8 Kbit/s - 33.6 Kbit/s),
- Internet connections over long distances (variable bandwidth between 10 bit/s to 10 Kbit/s),
- medium bandwidth local connections (wired LANs - 10 to 100 Mbit/s),
- high bandwidth local connections (ATM - 155 Mbit/s to Gbit/s).

The connections have different bandwidth, latency, cost, and quality of service (error rate, jitter) and for some of them the parameters may vary over time. It is important for a nomadic host to view communication support in a integrated way that adjusts to varying characteristics, optimizes the cost of its usage, and treats disconnections or link failures in a transparent way.

Communication performance is crucial for nomadic hosts. Even if we are able to integrate different types of connections and optimize their usage, for many nomadic applications low bandwidth network connections may become a bottleneck. For example, a browser accessing WWW documents over GSM 9.6 Kbit/s connection will be strongly limited by this bandwidth. Experiments have shown that even using a specialized protocol, the useful application bandwidth is at most 8.7 Kbit/s [10]. The only way to increase performance is to change the computing paradigm at the application level.

A communication architecture for nomadic hosts must take into account application requirements. As the WWW has become an operational prototype of the Information Infrastructure, information access is currently one of the most important applications on nomadic hosts. Such applications rely on information filtering: searching for relevant information, resource discovery, querying databases and may benefit from new paradigms such as programming by delegation.

We are interested in communication and application support for nomadic computing. We propose an approach based on mobile agents to the design of a communication architecture for nomadic applications. This paradigm makes use of mobile agents that a nomadic application activates to execute tasks on remote nodes. Mobile agents move from node to node, perform useful operations and report results. The paradigm is intrinsically asynchronous—an application can disconnect after agent activation and retrieve results later on. Asynchronous mode can also provide important performance gains. Agents and results are persistent in the sense that they are not lost because of node crashes or link failures.

We have defined and implemented an architecture for mobile agents called *Mobile Assistant Programming* (MAP) [22]. The implementation is based on the WWW framework and the Scheme programming language [21]. Initially, MAP has been designed for information access applications for large scale networks [23]. *MAP assistants* are high-level interpreted programs that can move between nodes, create clones and report results. Their execution is asynchronous and persistent to allow client disconnections and survival of node failures. Moving, cloning, and reporting results are atomic actions having transactional semantics. Many other systems supporting mobile agents appear (see Section 5). One of their motivations is nomadic computing. However, providing mobile agents is not enough—there is a gap between nomadic applications and mobile agent systems that should be filled by an intermediate layer.

In this paper, we propose to use the MAP architecture for nomadic applications. We define a middleware layer that adapts MAP to nomadic environment. The nomadic adaptation layer provides application support by means of mobile agents and takes care of optimized network connection management. The layer acts as a proxy of a application for agent activation and result retrieving. It makes results visible to an application after processing them according to application requirements. An application controls the layer by defining politics to adopt for connection management. For example, an application may specify the acceptable quality loss of some types of data or the function for optimization of connection cost. The application may also define how results must be interpreted to provide approximate or partial results to the user.

In the remainder of this paper, we present the Mobile Assistant Programming architecture (Section 2), discuss issues related to the design of an architecture for nomadic applications (Section 3), define the nomadic adaptation layer (Section 4), discuss related work (Section 5), and outline conclusions (Section 6).

2 Mobile Assistant Programming

We consider a large scale network composed of nodes connected via communication links. Nodes are virtual processors with memory and secondary storage. The MAP architecture is based on mobile agents that we call *assistants*. Figure 1 presents the principles of the MAP programming model. An application *activates* an assistant program on a remote node to accomplish a complex task. An interpreter interprets the assistant program and its execution state can be *saved* to persistent storage (this action is called a *checkpoint*). In case of a node failure, assistants are *restored* from persistent storage automatically. To accomplish its task, an assistant *moves* to a remote node, *clones* other assistants and *reports* results. The results are collected in a *result collector*, a persistent object located on any node in the system and unique for an activation. An application can *get results* from the collector.

Figure 2 presents an example of an application that runs on a nomadic host and several WWW servers. The user activates an assistant in Paris, goes to Boston while the assistants execute in the network and requests the results on arrival in Boston.

Figure 3 presents the functional architecture of the model. It is composed of four layers: *application front-end* layer that provides user interface, activates assistants, and checks for results, *assistants* layer that performs work for the application and is programmed using MAP primitives, *MAP primitives* layer that provides basic MAP operations, and *system support* layer that takes care of checkpointing and communication.

2.1 Assistant Primitives

The assistant layer provides primitives for controlling an execution:

- `activate` *source-code activate-node result-node*
  activate primitive initializes the execution of an assistant on a given node. An application provides *source code* to interpret and specifies a *result node* where a collector object is created. A capability for the collector is returned to the user for later retrieval of results.

- `get-results` *collector-capability*
  get-results primitive uses the collector capability to retrieve the results of an execution. The prim-

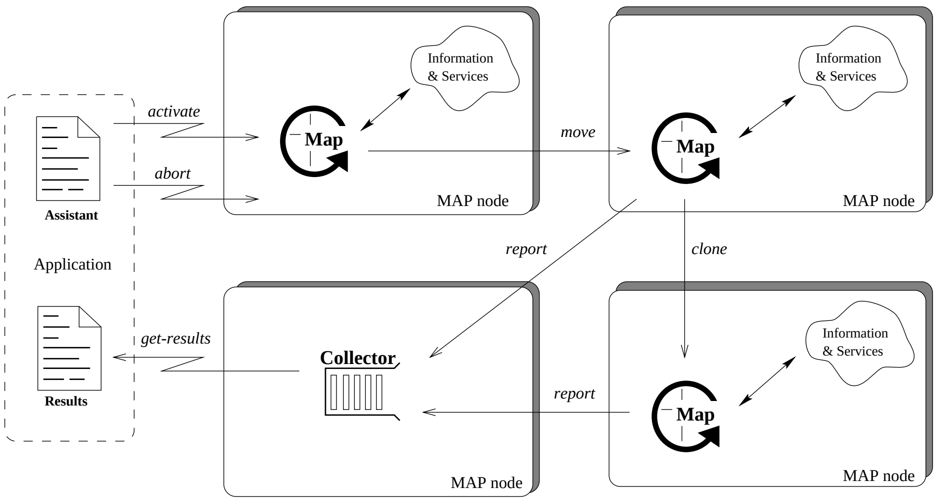

Figure 1: Principles of MAP

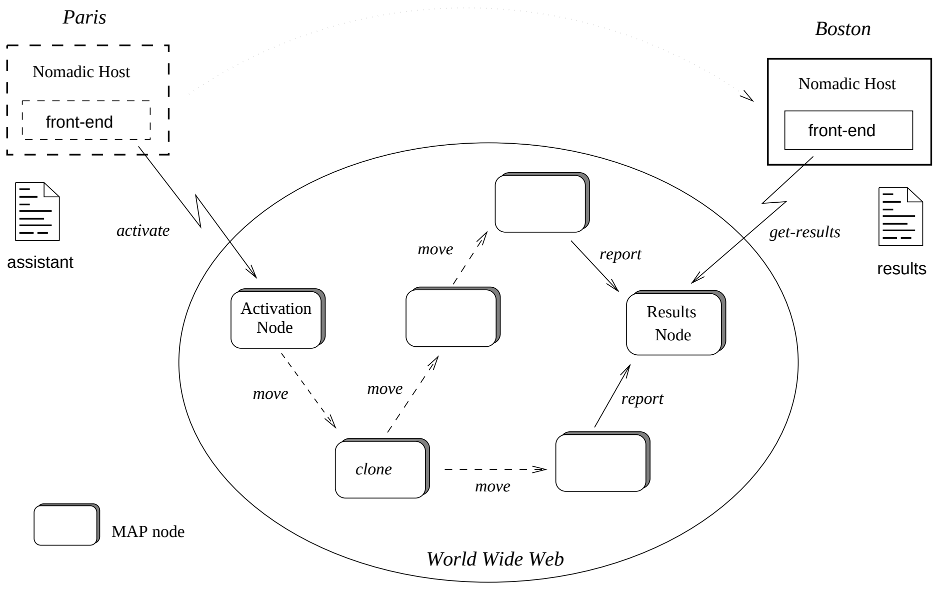

Figure 2: An example application

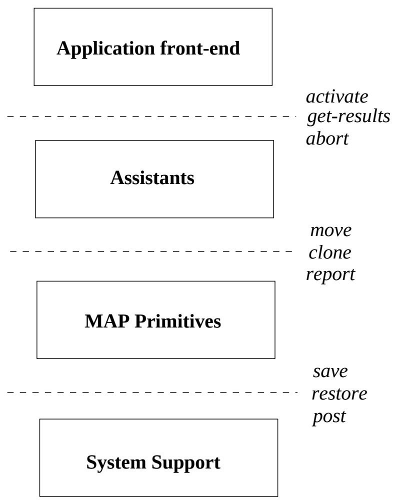

Figure 3: Functional architecture of MAP

itive detects the termination of all assistants associated with the given collector. The termination detection is based on the graph of assistant cloning constructed from the information provided by each assistant when it exits: the identity of its creator and the number of clones it has created. If all the results are not yet available, the primitive returns partial results and status *not-terminated*. If all the reports are available, the primitive returns the results and status *terminated*.

- `abort` *collector-capability*
  abort primitive uses the collector capability to terminate the execution of an assistant. The primitive detects the termination of all assistants associated with the given collector. If the execution is not terminated, it forces all the assistants to terminate.

2.2 MAP primitives

Assistants can use MAP primitives during the execution. The MAP primitives are as follows:

- `move` *node*
  move primitive transfers an assistant to a remote node. The execution state of the interpreter is saved to persistent storage, moved to the node using the *post* system operation and restored at the target node. The operation uses an authentication scheme to verify if the assistant is allowed to execute on the target node. If for any reason the assistant cannot move to the target node (for example the access is denied or *post* fails), the primitive returns appropriate status. A checkpoint is created on the target node and the assistant resumes execution. After the primitive, the current state of the assistant is confined to the target node. The primitive executes as an atomic action.

- `clone` *id node*
  clone primitive creates a copy of an assistant with a given identifier on a given node. Both the creator and the clone assistants are checkpointed. The clone assistant begins its execution independently from its creator. The creator keeps track of all its clones.

- `report` *msg*
  report primitive reports a message to the collector object using the *post* system operation. A checkpoint is taken after reporting.

- `break` *exception*
  break primitive terminates the execution of an assistant. The information about all clones created by the assistant is reported to the result collector object. It returns the exception status.

- `node`
  node primitive returns the identifier of the node where an assistant executes.

- `identity`
  identity primitive returns the identifier of an assistant.

- `exit`
  exit primitive stops the execution of an assistant. The information about all clones created by the assistant is reported to the result collector object, so that the termination of all assistants can be detected.

2.3 System support operations

The MAP primitives require some system support for checkpointing and communication. They are used by the MAP primitives, but they are not accessible by the programmer. The system support operations are as follows:

- `save`
  save operation creates a checkpoint by saving the execution state of the interpreter to persistent storage.

- `restore`
  restore operation resumes execution by restoring the state of the interpreter from persistent storage. This operation is also done automatically after a node failure for all active assistants - they are restored from the last checkpoint.

- `post`
  post operation transfers data to a remote node. The operation takes into account node failures or disconnections and it tries to deliver the data even in such cases. It also implements an authentication scheme based on digital signatures.

2.4 Discussion

In the MAP model, assistants are asynchronous independent entities. They execute in parallel, independently of one another. There is no communication between assistants and each assistant reports results directly to the collector. Reporting is also asynchronous—each assistant can generate as many reports as it wishes at any time. Persistent execution of assistants is based on *checkpointing*—the execution state of an assistant is saved to persistent storage. When an assistant executes a primitive that modifies its external environment such as *move, clone*, or *report*, a checkpoint is taken in the primitive. The primitive and the checkpoint are executed as one transactional atomic action, so that an assistant can be restored after a node failure in a consistent state and continue its execution. After a failure, a node restores all the assistants that were active before a failure. This mechanism allow assistants to progress in computation in spite of node failures.

2.5 Implementation

We have implemented the Mobile Assistant Programming model using the WWW framework and the Scheme programming language [21]. As we were interested in applications that access data distributed over the Internet, WWW was an obvious choice for the first prototype. The implementation is based on the HTTP POST method and CGI scripts. The POST method provides a means for transferring information to a WWW server and a CGI script provide support for executing MAP primitives. Scheme has many advantages: it is a fully fledged programming language with first-class procedures. The source code for its interpreter is widely available and it can run on many heterogeneous platforms [28].

A *MAP node* is a WWW server that runs the *httpd* daemon and provides *MapServer*—a CGI script with the following components: the modified Scheme interpreter, functions implementing MAP primitives and a local service interface. The implementation is based on three elements:

- *interpretation support*: to interpret assistant programs, we provide a modified Scheme interpreter. We have removed some Scheme functions that are *unsafe*, such as interactive primitives (`reset`) and primitives to access operating system facilities (`open, read, write`). We have added all MAP primitives as well as a set of primitives for accessing and processing HTML documents.

- *communication support*: to transfer data between nodes we use the standard WWW HTTP protocol. The data may be either a source code, the encoded execution state of an assistant, a result report, or a control message. The data are signed digitally and sent via the HTTP POST to a remote MapServer CGI script. The *httpd* daemon on the remote node starts the script and passes the data to the MapServer in an environment variable.

- *distributed execution support*: the MapServer CGI script implements the assistant primitives: *activate, get-results, abort.*

3 Architecture for Nomadic Applications

The initial goal of the MAP architecture was to provide flexible mechanisms for programming applications accessing distributed data in large scale networks such as Internet. The characteristics of the networks greatly influenced our design:

- to allow disconnected operation, an assistant is an asynchronous process: the user *activates* an assistant and *gets results* at some later time;

- to reduce network traffic, an assistant *moves* to a remote node and executes operations on data located on that node.

- to accommodate large scale, an assistant can *clone* other assistants that execute in parallel as autonomous and independent entities;

- to cope with node failures, an assistant is a persistent process: its execution state is *saved* and can be *restored* in the case of a node failure;

- to deal with heterogeneity, an assistant is a high-level interpreted program.

Nomadicity raises additional issues that must be taken into account:

- *mobility support*
  A communication architecture for nomadic hosts must provide independence of location, of motion, of platform along with widespread presence of access to services.

- *varying network parameters*
  Nomadic hosts communicate using different support networks with variable parameters of bandwidth, latency, error rate, and cost. A nomadic host must adjust automatically to varying conditions in a transparent and integrated fashion.

- *communication performance*
  Network resources are critical for nomadic hosts, because their bandwidth may be narrow (especially for wireless connections) and must be optimized.

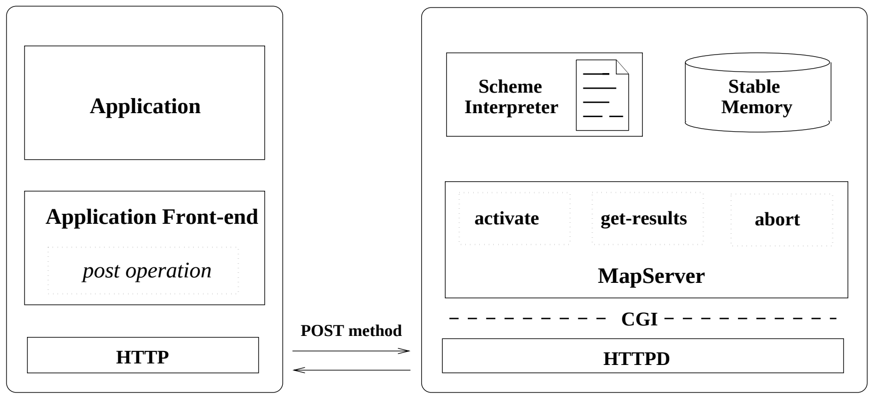

Figure 4: Implementation of MAP

- *connection management*
  A nomadic host must treat disconnections or link failures in a transparent way.
- *application requirements*
  Applications need some system support to deal with nomadicity. The type of support depends on application requirements. We can notice that information access becomes one of the most important application on nomadic hosts.

Mobility is a major issue in nomadic communication. It should be integrated in a transparent way independently of location. This goal can be achieved at the network layer, for example by means of *mobile IP* [19, 5]. This solution allows a mobile host to send and receive packets addressed with its home IP address regardless of its current point of attachment. It maintains communication association such as TCP connections even if the point of attachment changes during their lifetime. Mobile IP is based on tunneling of IP packets between a home agent connected to the home network and a mobile host.

Communication when moving can be achieved using wireless networks such as GSM or wireless LAN. Transmission quality is variable depending on the placement of a mobile host. Communication cost is usually important and should optimized.

To deal with mobility and intermittent connections, a mobile host needs a middleware layer that provides a uniform view of underlying networks and optimizes its usage. Kleinrock has proposed a layered architecture for nomadic communications [12, 13]:

- Open Data Network layer that provides access to different communication substrates,
- Transport Services layer that provides different end-to-end transport services,
- Middleware layer that provides services for nomadic applications,
- Application layer.

The Middleware layer proposed by Kleinrock presents a MIMI interface (*Middleware/Middleware Interface*) that provides services specific to nomadic applications such as:

- *autonomous agents* for execution of tasks in the network,
- *predictive cache* for prefetching data to a nomadic host,
- *approximator* to provide approximative or partial results and to adapt the quality of service to available network resources,
- *disconnected execution queuer* to manage connections and disconnections,
- *security services* to authenticate and authorize access,
- *performance management* to inform applications about resource consumption and advise on their optimal usage,
- *location management* to help to localize network services.

The MIMI interface supposes that nomadic applications become aware of mobility and of some problems related to communication. For example, an application may want to negotiate its quality of service according to available connections or even ask for incomplete or approximative results if complete results are unavailable or too expensive to obtain. In this way, a nomadic application can work in an adaptive manner—adjust its behavior according to varying conditions of network resources.

A communication architecture for nomadic hosts should also take into account different types of applications and provide support for them. The most important applications nowadays and in the future are the following [15]:

- *filtering of information*: information access applications must be able to filter abundant information and identify relevant information,

- *filtering of asynchronous communications*: we are submerged by asynchronous communications such as e-mail; it becomes crucial to be able to filter messages, assign priorities, and process messages according to the priorities.

- *scheduling of synchronous interactions*: synchronous communications are sometimes more efficient than messages (for example a video-conference session) and it is important in this case to schedule them.

Our goal is to consider these types of applications in a nomadic environment. We can notice that the mobile agent paradigm is particularly suitable for a nomadic communication architecture presenting the MIMI interface. First of all, communication performance can be increased—since the size of results is usually smaller than data themselves, less data need to be transmitted over the network. A nomadic host is usually *resource-poor* relative to static hosts. Delegating work to a more powerful server may improve application performance. Moreover, network connections can be managed in an advantageous way, for example, we can process results before transmission to a nomadic host to match available bandwidth: the results can be compressed or adjusted to desired quality (e.g. image resolution that fits the screen on a nomadic host). We can also group data to minimize connection time. The asynchronous interaction between an application and agents allows a nomadic host to deal with disconnections in an efficient and robust manner. The asynchronous mode can also increase performances perceived by the user. The user activates actions in the network and access results later on when they are stored locally on its nomadic host. Local access is much faster than the remote one, so the response time is much better.

Based on the mobile agent paradigm, we can design the functions corresponding to services needed by nomadic applications. Hence, we propose to add a middleware layer to the MAP architecture. The layer provides enhanced services to nomadic applications: result collector is managed as a predictif cache, partial or approximate results may be delivered, and network connections are optimized. We suppose that at the network layer the architecture is supported by a low-level mobility support such as mobile IP.

4 Nomadic Adaptation Layer for MAP

The MAP architecture provides support for mobile agents: assistant activation and result retrieving. To adapt it to a nomadic environment, we add functions for connection management and optimization.

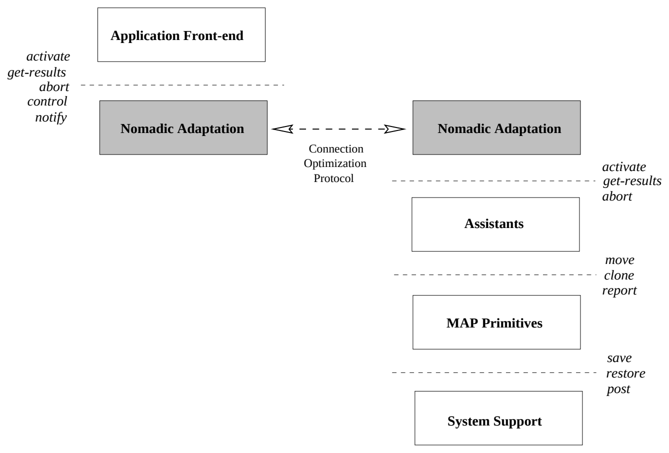

Figure 5: Nomadic Architecture

We have considered two ways of how to adapt MAP for nomadic hosts. The first one consists of considering a nomadic host as a MAP node (i.e. the node where an assistant can execute) and add connection management and optimization to the *post* system support operation. An application could activate assistants on a nomadic host and transfers to other MAP nodes would be optimized. This solution is based on a uniform, homogeneous views of all the nodes—the optimization code would be executed for all transfers, even if some connections do not need such a functionality.

The second way is to consider a nomadic host as a special case. A nomadic host communicates with a stationary MAP node using a connection optimization protocol. This approach places functions exactly where they are needed and does not add any supplementary cost to the transfers inside the network of stationary MAP nodes. For these reasons, we have chosen the second approach.

Adding new functions to the interface for applications means that the applications become *"nomadicity aware"*. For example, they will be able to manage connections at a high-level and to receive notifications related to the state of network resources and adapt to varying conditions.

Figure 5 presents the MAP architecture enhanced with the nomadic adaptation layer. On a nomadic host, the layer offers the interface for assistant activation, result retrieving, and connection management. Figure 6 illustrates the principles of the nomadic architecture. There are three types of entities: a nomadic host, a MAP base, and a MAP node. A nomadic host cooperates with a MAP base by using a connection optimization protocol. For applications on a nomadic host, the nomadic adaptation layer acts as a proxy for assistant activation and result retrieving. It makes results visible to an application after processing them according to application requirements. An application controls the layer by defining politics to adopt for connection management or for result processing. The layer notifies applications on connection states.

In addition to the functions presented in Section 2, the MAP interface offers the following primitives:

- `control` *attribute profile code*
  the `control` primitive passes to the connection manager a profile defining a *connection attribute*. The application may provide some *code* to process data before the transfer.

- `notify` *attribute value*
  the `notify` primitive returns the value of a given connection attribute.

The connection manager cooperates with the MAP base by means of a connection optimization protocol. The protocol maintains information about connection attributes and acts according to the control specification expressed by an application. The protocol can process data on a nomadic host or on the MAP base before the transfer: compression, quality degradation, or any other operation requested by the application.

Connection profiles define politics to adopt for connection optimization and specify They specify cost criteria or operations to perform on data. A profile may specify how the connection attribute should be optimized. For example, an application may define that the time of a GSM connection must be minimized by compressing and grouping data to be transfered. An application may also define how results are to be processed before the transfer to the nomadic host. For example, it may specify that data of `image/jpeg` MIME type should be degraded in resolution to fit the screen of a nomadic host. It may also specify how to provide approximate results, or how to obtain partial results.

The connection manager implements a predictif cache of the result collector with the help of the connection optimization protocol. It manages the transfer of results to the nomadic host while optimizing network connections. The transfer is a series of atomic actions having transactional semantics—communication progresses despite host crashes or connection failures.

Notifications allow applications to supervise connection states, the level of network resource consumption, and to know what part of results is already available on a nomadic host.

4.1 Current status

The MAP architecture has been implemented and the prototype on the WWW is operational (`http://-fidji.imag.fr/map/access.html`). Currently, we are implementing the nomadic adaptation layer using a set of laptops connected via GSM connections as a platform. A MAP base is a workstation connected to a wired LAN. Efficient WWW access is our pilot nomadic application that we use to validate the architecture: the user may specify a set of URLs and a query containing keywords, document update times, and the deepness of hyper-text links. The application activates assistants and may disconnect. The assistants follow hyper-text links starting from the initial URLs and gather all the relevant documents up to the specified deepness of links. When the user connects to the MAP base, the results are transferred to the nomadic host in a optimized way and the user access the documents efficiently. The user may specify additional result processing, for example, the quality of large volume data types such as images or video may be degraded.

We also explore how mobile agents can be used to schedule interactive multimedia interactions. For example, an application may activate agents to organize a video-conference, prepare video streams, and when all participants are ready, start video streams. An application may specify the quality of service parameters of multimedia streams that match current connection bandwidth of a nomadic host.

5 Related Work

Related work can be divided into several categories: communication protocols for nomadic hosts, support for nomadic applications, and mobile agents.

Much work has been done in the area of communication protocols for nomadic hosts. The first problem that the protocols must solve is mobility. As mentioned previously, existing solutions for mobility appear mainly at the network layer: *mobile IP* [19, 5] or IPv6 [20, 26]. The second problem is performance. Many projects try to optimize performance over low-bandwidth wireless connections [2] or propose specific protocols [10, 3]. However, the bandwidth of such connections limits application performance regardless of the optimized or specific protocols. Important performance improvement can only be done based on a different computing paradigm.

Nomadic hosts need not only specific communication support, but also support for applications. Coda is a file system that allows disconnected operation [11]. It uses optimistic replication to provide continuous service to the users on a nomadic host. Bayou provides support for sharing data among mobile users [6]. Rover that combines *relocatable dynamic objects* and *queued remote procedure calls* to provide a uniform distributed object architecture for code shipping, object caching, and asynchronous object invocation [9]. Travler focuses on developing models, prototyping systems software, and running experiments in support of those capabilities needed by a mobile user [14]. Ward is a model for replication in mobile environments [24].

A lot of work is being done in the area of mobile agents. The mobile agent paradigm offers many advan-

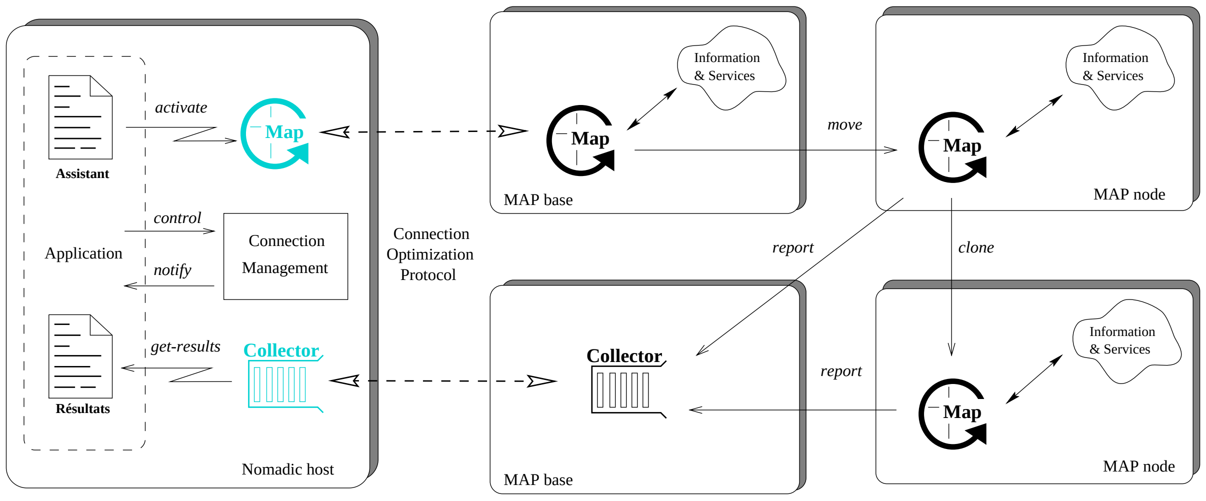

Figure 6: Principles of the Nomadic Architecture

tages with respect to traditional distributed programming models such as *message passing*, *remote procedure call*, *object invocation*, and *remote evaluation*. A programming agent is a concept that appears frequently in the context of artificial intelligence [8]. It denotes an active entity with a well-defined goal that communicates with its peers by exchanging messages in an expressive agent communication language. An agent communication language can be either declarative or procedural. The declarative approach frequently used in AI is based on the idea that communication can be best modeled as the exchange of declarative statements. In the procedural approach, communication is seen as the exchange of procedural directives. Scripting languages such as *TCL* [16], *Apple Events*, and *Telescript* [29] are based on this approach.

Several recent works explore Java as a platform for mobile agents. Java is an interpreted portable object-oriented language that allows a client to dynamically download classes and execute them within its address space [1]. Mobile agent systems based on Java include *Aglets* [4], Concordia [30], JAE [17], Ara [18], and Mole [27]. All these systems have similar features as MAP, however MAP provides persistence and failure resilience based on transactional semantics. Other interesting work include the OMG proposal [4] and a system for WWW access over wireless links [7].

Research on mobile agents is very active. Many research groups work on new mobile agent systems. However, the question of how a given system is used by application has not been raised (none of the presentations at a recent workshop has tackled the problem [25]). Once we have such a support, we realize the need for a middleware layer that provides to nomadic applications management and control functions over agents.

6 Conclusions

The objective of this paper is to define a middleware architecture for nomadic applications. Nomadic applications may benefit from the mobile agent paradigm. However, we must add an intermediate layer between applications and mobile agents. Having gained experience with a mobile agent system called MAP, we have proposed to add an adaptation layer that provides enhanced services to nomadic applications: result collector is managed as a predictif cache, partial or approximate results may be delivered, and network connections are optimized. An application controls the layer by defining politics to adopt for connection management and result delivery.

REFERENCES

[1] The Java language: A white paper, URL: `http://java.sun.com/` , 1994.

[2] H. Balakrishnan et al. A comparison of mechanisms for improving TCP performance over wireless links. In *Proc. ACM SIGCOMM Conference*, Stanford, 1996.

[3] K Brown and S. Singh. M-UDP: UDP for mobile cellular newtroks. *ACM SIGCOMM Computer Communication Review*, 1996.

[4] D.T. Chang and S. Covaci. The OMG mobile agent facility. In *1st Int. Workshop on Mobile Agents*, pages 62–73, Berlin, 1997.

[5] S. Cheshire and M. Baker. Internet mobility 4x4. *ACM Computer Communication Review*, 26(4), 1996.

[6] A. Demers et al. The Bayou architecture: Support for data sharing among mobile users. In *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., 1994.

[7] W.-S. E.Chen and Y.-N. Len. Intelligent messaging for mobile computing over the World-Wide Web. In *2nd Workshop on Mobile Computing*, 1996.

[8] M.R. Genesereth and S.P. Ketchpel. Software agents. *Communications of the ACM*, 37(7).

[9] A.D. Joseph et al. Rover: A toolkit for mobile information access. In *Proc. Fifteenth Symposium on Operating Systems Principles*, 1995.

[10] J. Kiiskinen, et al. Data channel service for wireless telephone links. *IEEE Bulletin on Operating Systems and Application Environments*, 8(1):3–12, 1996.

[11] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10:3–25, 1992.

[12] L. Kleinrock. ARPA PI meeting presentation, URL: `http://millennium.cs.ucla.edu-/LK/lkpimtgfla795/index.html`. 1995.

[13] L. Kleinrock. Nomadic computing—an opportunity. *ACM Computer Communication Review*, 25(1):36–40, 1995.

[14] L. Kleinrock, et al. Travler: System support for nomadic computing,URL: `http://ficus-www.cs.-ucla.edu/travler/`. 1996.

[15] D.G. Messerschmitt. The convergence of communications and computing: What are the implications today? *IEEE Proceedings*, august 1996.

[16] J.K. Ousterhout. An X11 toolkit based on the TCL language. In *Proc. USENIX Association 1991 Winter Conference*, pages 105–115.

[17] A. Park and S. Leuker. A multi-agent architecture supporting services access. In *1st Int. Workshop on Mobile Agents*, pages 62–73, Berlin, 1997.

[18] H. Peine and T. Stolpmann. The architecture of the Ara platform for mobile agents. In *1st Int. Workshop on Mobile Agents*, pages 50–61, Berlin, 1997.

[19] C.E. Perkins. IP mobility support, `draft-ietf--mobileip-protocol-16.txt`. 1995.

[20] C.E. Perkins and D. Johnson. Mobility support in IPv6, `draft-perkins-ipv6-mobility-sup.stxt`. 1996.

[21] S. Perret and A. Duda. Implementation of MAP: A system for mobile assistant programming. In *IEEE International Conference on Parallel and Distributed Systems*, Tokyo, 1996.

[22] S. Perret and A. Duda. MAP: Mobile assistant programming for large scale communication networks. In *IEEE International Communications Conference 96*, Dallas, 1996.

[23] S. Perret and A. Duda. Mobile assistant programming for efficient information access on the WWW. In *Proc. Fifth International World-Wide Web Conference*, Paris, 1996.

[24] P. Reiher et al. Peer-to-peer reconciliation based replication for mobile computers. In *2nd ECOOP Workshop on Mobility and Replication*, 1996.

[25] K. Rothermel and R. Popescu-Zeletin. Mobile Agents, *Proceedings of the 1st International Workshop*. Berlin, 1997.

[26] W. Simpson. IPng mobility considerations. *RFC 1688*, 1994.

[27] M. Straser et al. Mole: a Java based mobile agent system. In *2nd ECOOP Workshop on Mobile Object Systems*, Linz, 1996.

[28] W. Clinger and J. Rees. Revised report on the algorithmic language Scheme. *ACM SIGPLAN Notices*, 21(12):37–79, 1986.

[29] J. E. White. Telescript technology: The foundation for the electronic marketplace. Technical white paper, General Magic Inc., 1994.

[30] D. Wong et al. Concordia: An infrastructure for collaborating mobile agents. In *1st Int. Workshop on Mobile Agents*, pages 86–97, Berlin, 1997.