Middleware for Nomadic Access to Enterprise Information Systems

*Ravi Jain**
Applied Research, Bellcore

Abstract. Future users of enterprise information systems will be increasingly nomadic, i.e., they will be mobile, will use different communication media at different times, and different equipment at different times. This paper focuses on arguing that rather than viewing nomadicity software as a specialized add-on for supporting nomadic users, every user should be regarded as (potentially) nomadic, and the features required to support nomadic users should be part of the core of the enterprise information system.

We review the development of nomadicity software and the key applications that are typically supported, such as messaging, personal information services, mobile sales, and file access. We thus identify the key common functions required, such as profile management, media translation and database query management. We also observe that large enterprise information systems are increasingly being built using commercial middleware platforms, such as CORBA, DCE, OLE etc. The functions required for supporting nomadic users should thus be built into the middleware, either as a nomadicity middleware layer on top of the base middleware platform, or as extensions to the core services these platforms provide. We very briefly describe a high-level functional view of a middleware architecture for for nomadic access to enterprise information systems that could be built on top of a platform such as CORBA.

1 Introduction

The development of low-cost, low-power, and portable computing devices, coupled with the availability of new radio spectrum allocations, has opened the possibility of new models and methods of communications, information access, and computing. It is now recognized that communications and information systems must serve the needs of what we call nomadic users: users who may be mobile and who, at different times, may be communicating using wireless or wired media of different characteristics, from different (local or remote) locations, and using communication or computing devices of different types and capabilities. While this development raises new possibilities in terms of the services that nomadic users can hope to enjoy, it also raises significant challenges in the design and development of communications and information systems.

One possible response to these challenges is to develop specialized subsystems which serve nomadic users, and which provide the tuned applications, specialized system software, and protocol functionality required to deal with the discontinuities introduced by mobility and the idiosyncrasies of wireless media. It is then necessary to connect these specialized subsystems to the "main" system, which serves stationary users over wired links, via inter-working modules which perform protocol conversion, data buffering, address translation, etc.

However, a more comprehensive and integrated approach may be preferable in the long term. Instead of regarding nomadic users as a specialized sub-group of the user community, every user may be regarded as a (potentially) nomadic user. In future, a majority or significant fraction of users may become nomadic. We see a variety of factors encouraging this trend, such as out-sourcing, off-shoring, and decentralization of business operations; increased telecommuting (partially motivated by Clean Air Act provisions, for example) and distance learning; formation of virtual business units, etc. The functions required to support nomadic users should then be integrated into the core facilities and design of the overall system.

In parallel with the growth of nomadic computing and communications is a growing interest, especially among designers and administrators of enterprise computing and networking systems, in "middleware" [4, 5]. Middleware is a somewhat vague term which encompasses the common software functions and building blocks required by a wide range of applications, and which are technology-independent, i.e., reside at a layer above device and protocol drivers, operating systems, and other system software. Examples of functions which can be included in middleware

*Address correspondence to Ravi Jain, Applied Research, Bellcore, 445 South St., Morristown, NJ 07960. E-mail: rjain@bellcore.com.

are security, authentication, and billing modules which offer common facilities to numerous applications. Roughly speaking, if the tasks performed by system software are analogous to "house-keeping" or "book-keeping" chores (memory allocation, process coordination, etc.), those performed by middleware are analogous to "accounting" or "professional" services. One of the chief motivations for the interest in middleware is its potential for software reusability; instead of billing modules being written for each application, for instance, one common module is written.

With this background, we consider a logical place for incorporating the software required to support nomadic users as middleware, thus making it accessible to a wide variety of applications and making it a peer of other core functions such as billing. We will call this middleware building block "nomadicity" software.

This paper concentrates on raising issues and reviewing software approaches for supporting nomadic users, and does not attempt to provide detailed solutions or designs for doing so. In section 2 we give an overview of the recent approaches to nomadicity software. Note that we do not attempt to evaluate or review specific products (both because of their increasing variety and because they are rapidly changing); instead we try to generalize and present the kinds of approaches that have been taken. It appears that while most nomadicity software has indeed been middleware, in the sense of residing between the applications and system software layers, it has tended to be narrowly focused on supporting a single class of applications. In section 3 we will very briefly describe a general high-level functional view of the requirements for the next generation of nomadicity software. In section 4 we end with concluding remarks.

2 A brief overview of nomadicity software

Mobile users are, of course, not a new phenomenon for system designers and administrators to deal with. Before the 1990s, mobile users who needed access to their files, electronic mail, etc. did so typically via modem and dialed-up phone lines. This form of access still remains widespread. However, this decade has given rise to a new class of users, whom we term "nomadic" in order to distinguish them from mobile users [27]. A nomadic user is one who may be:

1. *Mobile.* We distinguish between two types of mobility: *discrete mobility* is the ability to connect to the communications or computing system from a location, perform desired activities, disconnect from the system, and reconnect at a different location. *Continuous mobility* is the ability to remain connected to the system even while moving from one location to another.
2. *Using different communications media.* Thus a nomadic user may use wired access and wireless access at different times; even when using the same medium, he or she may have different characteristics (bandwidth, latency, etc.)
3. *Using different terminal equipment.* A nomadic user may use, for example, a laptop at one location, and a desktop at another.

At the same time nomadic users want seamless access to a variety of communication and computing applications without having to carry multiple terminals, use disparate enterprise systems, or learn multiple user interfaces.

2.1 Nomadic messaging middleware

Among the earliest type of software developed for serving nomadic users was that for media-independent messaging. Examples include AT&T's EasyLink [1], and Bellcore's Electronic Receptionist [29] and AirBoss^{TM} [30, 2] products. Messaging (also called "unified messaging") applications typically aim to provide the nomadic user the equivalent of single-number service, but across multiple media. For example, a user may have several logical addresses for receiving messages: home and office telephone numbers, a fax number, an electronic mail address, a pager number, and a cellular telephone number. A messaging application replaces these logical addresses with a single address (e.g., a single NPA-NXX-XXXX number in place of all the different telephone and fax numbers, and a derived electronic mail address, say `NPA-NXX-XXXX@message_service.com`.) The messaging application then performs the following functions:

1. *Message greeting.* The application can intercept messages or calls to the user, and for voice calls, optionally play a message ("Please hold while I try to locate Joan Smith".)
2. *Message routing.* The application routes messages to the user via different media using some stored criteria. For example, a user can specify a profile ("From 6 am - 7 am, route messages to me at home; from 7am - 8 am, to my cellular phone; from 8 am - 5 pm, use my office phone, etc.") and/or allow the system to try different routes (simultaneously or in sequence).
3. *Message filtering.* The application can try to interpret the content or origin of messages and take different actions depending

upon the users profile. For example, calls from a certain number (e.g., the user's supervisor's office phone) can be handled differently (e.g., try all media simultaneously) from other calls. As another example, the application can scan electronic mail subject lines for the keyword "Urgent", or ask the user to press a telephone key if the message is urgent, and handle the message differently depending upon the outcome, etc.

4. *Cross-media translation.* The application can convert textual electronic mail to fax or (summarized) pager messages or synthesized speech for delivery over voice channels.
5. *Storage.* The application can act as a central voice, electronic, or fax mail box for the user, and allow sophisticated functions for accessing these mailboxes (e.g., non-sequential access, ability to view headers, etc.)
6. *Other functions.* Numerous other functions can be provided. For example, the application can provide a notification service, paging a user if an electronic mail message with "Urgent" in the subject header arrives. For nomadic users with laptops, the application can allow the user to specify how an incoming call is to be handled (e.g. accept the call, route to voice mail, or play an announcement); such a capability (along with others) is provided, for example, by Bellcore's Call Manager system [24].

It is clear that these functions are best combined into a middleware module available to other applications. An example application which can use messaging middleware is Personalized Information Delivery (PID). PID delivers important information to nomadic users in accordance with their interests and needs. An example is real-time road traffic information, as delivered by the Bellcore SCOUT system. SCOUT is a software prototype which delivers information about road conditions and vehicle traffic. The type of information delivered is similar in nature to that obtained from commercial radio station broadcasts and traffic updates. SCOUT differs from commercial radio broadcasts in that

1. it offers personalized information, allowing users to choose to receive information tailored to their interests (e.g. only the key roads, bridges and tunnels on a user's commute route, including roads which may not be covered in typical radio broadcasts),
2. at the time when it is needed (so users need not wait for a traffic update or be interrupted needlessly when the information is not needed), and
3. using the communication media the user wishes to use. Users can choose fixed or cellular telephone, electronic mail, fax, alphanumeric pager, voice mail, or a combination thereof.

Although this example is for road traffic, one can easily extend this to other PID applications which deliver sports, stocks, financial and other information. The personalization of the information content, time of delivery, and medium is done by allowing the user to set up profiles. Thus PID can take advantage not only of the single-number addressing, routing, filtering, translation, and storage functions needed for nomadic messaging, but also the profile management functions required for it. This is the approach taken in the AirBoss^{TM} product line middleware [17, 2].

2.2 Nomadic access to vertical applications

A more recent development in nomadic software products has been increased proliferation of software for access to vertical applications such as field service, field sales, factory operations, car rental, transportation, etc. While wireless communications has been used in some of these industries for many years, and in some cases decades, there has been a recent surge of activity which differs from simply using wireless radios for voice communications. An increased emphasis on data communications and hand-held computing devices (for order entry, inventory scanning, obtaining price quotes, etc.) which allow personnel to be mobile, and the proliferation of multiple communications media (fax, pager, electronic mail) throughout the enterprise has meant that users of these vertical applications can truly be considered nomadic by our definition.

Consider a mobile sales scenario [25, 26, 14]. The user is a salesperson who is provided mobile access to a corporate database containing customer and product information. Since the actual face-to-face interaction time that the salesperson has with a potential buyer is very small (often only a few minutes in some industries), and in any case is generally very precious, it is important that the salesperson be able to obtain any product information (specifications, availability, etc.) as well as check status of previous orders instantly. The salesperson would store on his PDA or other mobile terminal information indicating the profile of his customers (their orders, history, etc.). The fixed servers in the corporate database store relations containing global information about product inventory and sales status. The database as a whole can be therefore be viewed as having a fixed

component, which resides on the fixed servers, and a mobile component which consists of database fragments that reside on salesperson's PDAs. Typically, for availability and reliability reasons, any database fragment which resides on a PDA would also have at least one copy which resides on a fixed server; the fixed server copy may be less up-to-date than the PDA copy, as the latter contains the latest information entered by the salesperson.

There has been a lot of specialized software developed for individual applications in each industry (for example, the transportation industry [32] or the manufacturing industry [33].) The key to developing and successfully deploying a vertical application lies in understanding the details of the application, the environment it is used in, and the specific needs of the users. However, it is also true that while these specifics differ from one application to the next, there is a tremendous amount of commonality among applications, which can be captured in terms of re-usable middleware software modules.

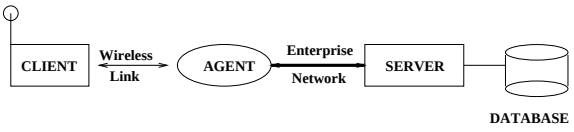

Figure 1: The Client_Agent_Server model

2.2.1 Mobile database access

One of the central features of vertical applications for nomadic users is the ability to access and update corporate databases. We believe this forms a core building block for other vertical applications for nomadic users, and discuss the technical challenges it presents.

One of the key constraints imposed by nomadic users is that they may be using wireless media to communicate with the database. The nature of wireless media is that it typically has low bandwidth, unreliable connectivity (such that even a stationary terminal can fade in and out of coverage) and high cost. On the other hand, in an enterprise environment, fixed users typically communicate with corporate database and information systems over high-speed, reliable, and low-cost links. Rather than modify the corporate database system itself, it is less disruptive to introduce a client-agent-server architecture. An example of such an architecture is shown in Fig. 1. Such an architecture has been used, for example, in Oracle's Oracle Mobile Agents middleware [31, 21], as well as our experimental AirSQL prototype nomadic database access middleware.

The client-agent-server model modifies the typical client-server model for organizing database access in a wired network enterprise environment. The client-agent pair play the role of the traditional client while the agent-server pair play the role of the traditional server. The client in this model focuses on the user interface and navigation while the agent focuses on managing the data flow from the client to the database server and vice-versa. One can think of the agent playing the role of a spooler, not in its traditional sense of supporting I/O requests but at a higher level in terms of supporting database requests and responses. The agent plays the role of a client to the server, and the role of a server to the client.

Clients forward their requests to the agent. The agent takes over the submission of the requests to the tethered server. The agent passes the responses from the server to the client when they arrive. Thus the relatively high-bandwidth, low-latency request-response traffic between clients and servers in a wired environment is carried between the agent and the server, and the agent can conserve the wireless bandwidth by caching and aggregating queries and responses, etc. Similarly, if clients are not accessible (poor coverage, powered down, etc.), responses may be held at the agent till the clients are ready, thus freeing the server from having to cope with the wireless clients that may fade in and out of coverage. In our work on the experimental AirSQL prototype we have considered several other functions which it is desirable that the agent support.

The architecture of the prototype is shown in Fig. 2. The software layers of the client and agent in our prototype implementation are shown. The client is connected to a wired network by means of a BellSouth Wireless Data (formerly RAM Mobile Data) [3, 19] wireless link, and the agent as well as the database server are also connected to a wired network.

We implemented the client software using Visual C++ 1.5x running on a Windows 3.1 platform. The highest layer of the client software controls the graphical user-interface (GUI). Below the GUI layer is the application-communication interface layer that actually makes all the Winsock API calls and manages the flow of data.

We use Dynamic SQL (DSQL) to allow the agent the flexibility to construct SQL statements originating from the client during runtime. Without DSQL, the agent would be limited to queries, or query templates, that were specified at compile time. With DSQL, the client can formulate and

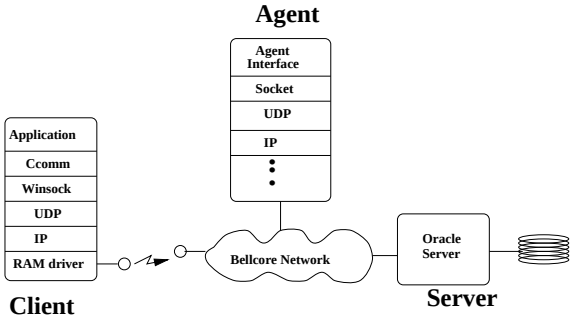

Figure 2: AirSQL prototype architecture

submit any valid SQL query, previously unknown to the agent, during runtime.

2.3 Nomadic computing and communications

Most recently, there has been an increased interest in more general access to computing and communications facilities by nomadic users. That is, instead of simply nomadic messaging or nomadic access to specialized vertical applications (sales, inventory, etc.) by specific sub-groups of the user community, more and more users require access to horizontal applications, such as text processing and spreadsheets, where (at least some of) the information must continue to reside in the enterprise information system. For example, consider the following scenario:

1. A user is running a spreadsheet application and operating upon a large spreadsheet while connected via wired links to a server.

2. At some point, the user needs to become mobile, and so downloads a fragment of the spreadsheet to his or her mobile client, and temporarily disconnects from the server. The client machine has a copy of the spreadsheet application program.

3. While disconnected the user edits the spreadsheet fragment, and then wishes to rerun the spreadsheet calculation.

4. The user reconnects with the server via a wireless link, and issues the recompute command.

The goal of the system designer is to make the steps associated with this scenario easy for the user, while at the same time minimizing the consumption of precious resources like wireless bandwidth and battery power at the portable computing device.

A survey [6] has found that 48% of respondents desired location-independent information access. In a sense, this trend can be seen as similar to the pre-1990s paradigm of remote access to central computers via fixed dial-up modem lines, but with the key elements of nomadicity: mobility (discrete or continuous), multiple communication media, and multiple terminal equipment types.

There has been a fair amount of research on nomadic access to horizontal applications, especially file access. Examples are the Coda file system [16] and the file system developed by Tait and Duchamp [28] which explicitly consider mobile clients who may frequently (voluntarily or involuntarily) be disconnected from the server. The issues addressed include how to ensure that files cached on the client and the server remain consistent, investigating different types of file semantics, etc., while minimizing usage of client resources and wireless bandwidth. Similar approaches have been taken to support nomadic Web access (e.g. [9].)

There has been somewhat less work done on support for other horizontal applications. Returning to our spreadsheet example earlier, when the nomadic user re-connects with the fixed server over a wireless link, there are several possible strategies for performing the spreadsheet re-calculations:

1. Processing at the server. The client sends its spreadsheet fragment to the server, along with a control message. The server computes the desired result and sends it to the client.
2. Processing at the client. The client sends a control message to the server, which responds by sending the required portion of the spreadsheet. The client then computes the desired result locally.
3. Processing at both client and server. The client performs some local computation (e.g., summing columns in the local fragment) and sends the partial result to the server, along with a control message. The server uses the partial results with the rest of the spreadsheet, computes the result, and returns it to the client.

Each of the information access strategies above is feasible, but will incur a different cost in terms of wireless bandwidth and battery power consumption [13, 10]. Since similar considerations are likely to arise for many applications (file access, text processing, database access, etc.), grouping the functions which evaluate these costs and make decisions into middleware building blocks is desirable.

3 Nomadicity middleware for enterprise networks

While software for nomadic messaging, vertical applications, and horizontal applications is useful,

an integrated, systematic view is desirable to provide nomadic users with messaging, and horizontal and vertical applications in a seamless manner. We describe the desired middleware features for nomadic user support, existing middleware for enterprise systems (focusing on CORBA), and a brief high-level architectural approach to nomadicity middleware for providing the desired features.

3.1 Desired middleware features for nomadic user support

Obviously the list of desirable middleware features will grow and be modified as actual experience with providing such services to growing numbers of nomadic users is accumulated. However, it is possible to point out some functions which will be useful:

1. *Nomadic messaging.* The middleware required includes building blocks for single-number addressing (also sometimes called "universal in-box"), message routing, filtering, translation, and storage, as well as profile storage and management.
2. *Nomadic database access.* The middleware required includes building blocks for query and response caching, stored SQL procedures, storing queries and responses to deal with temporary loss of coverage, etc.
3. *Nomadic access to horizontal applications.* The middleware required includes building blocks for consistency (i.e., synchronization of objects like files and spreadsheets cached on the portable computing devices with those on fixed servers), cost analysis (i.e., to determine which synchronization strategies would minimize wireless bandwidth and battery usage, hoarding, etc.)
4. *Address translation.* While nomadic messaging provides single- number service for incoming messages (from different media such as fax, phone, etc.), we see a broader issue of address translation when the nomadic user moves from one service provider's network to another, for the same medium. For example, currently each client machine on the Internet is assigned a distinct IP address. If the client is mobile, then protocols such as IETF's Mobile IP [7, 12] have been designed for delivering packets to the client, by assigning it a temporary IP address as it attaches to different places in the network and maintaining a mapping from the client's permanent IP address to its temporary IP address. Two of the unresolved issues with such a solution are:
   (a) How can this mapping be maintained efficiently, particularly if the underlying transport does not support efficient broadcasting (e.g. for ATM backbones)?
   (b) How can addresses be made provider-independent? With increasing competition, nomadic users will wish to be able to change their Internet Service Provider (ISP) without having to change their IP address. This problem is analogous to the problems of local number portability and 800 number portability in telephony [18, 11], and while it is not exclusively a consequence of nomadicity, nomadicity does make it more likely to occur.
5. *Object mobility.* Many commercially available middleware environments provide location-independence for objects, e.g. in CORBA, a client object need not know the physical location of a server object: it simply issues a request, which is mediated by the Object Request Broker and delivered to the server object. Middleware environments like CORBA also provide objects with logical name mobility, in the sense that an object's position in a logical namespace can be changed. However, with increasing user nomadicity, we see a need for physical object mobility, in the sense of moving the object from one machine or network to another, either autonomously or under the direction of other (user or system administrator) objects. This will allow a user's information (e.g. file or spreadsheet objects, personal agents, service profile objects, or mailbox objects) to follow the user as the user moves, thus improving the performance as seen by the user by reducing communication latency, decreasing communication cost, and conserving system resources. The integration of Java with CORBA potentially offers this object mobility [22].

3.2 Existing middleware for enterprise computing environments

We digress to give some background on the types of middleware facilities currently available for distributed computing environments. For concreteness we focus on the CORBA environment as an example; other available environments include the Distributed Computing Environment (DCE) from Open Software Foundation (OSF), and Microsoft's Component Object Model (COM) for its Object Linking and Embedding (OLE) framework. Other products which can be regarded as middleware include Lotus Notes (see [4] for a discussion) and ANSA's ANSAware [20].

The Common Object Request Broker Architecture (CORBA) is an object-oriented middleware software specification endorsed by the members of the Object Management Group (OMG), currently numbering over 700 international companies and organizations. (For a readable and detailed discussion of CORBA, see [23].) Central to CORBA is the Object Request Broker (ORB) which lets client objects locate and invoke methods upon server objects without having to know where the latter are located, how to communicate with them, or how they are implemented. CORBA also specifies how ORBs, possibly provided by different vendors, can communicate. Sixteen additional services are being specified in a phased manner. These range from a Naming service (which allows objects to manage the name space and locate other objects), an Event service (allows objects to indicate that they should be notified of certain events on other objects) to a Concurrency control, a Transaction and a Trader service [23].

This impressive list of services contains many which are also provided by other middleware platforms (e.g., DCE provides a time service as well as a security service which has mechanisms for authentication, secure communications and access control.) However, one of the attractive features of CORBA is that not only does it provide this extensive, consistent set of services in an integrated fashion, it does so within an object-oriented framework. In addition, CORBA allows the specification of higher-level services which build upon the ORB Core and Object Services listed above, by means of its Common Facilities specification.

A well-known concern with CORBA is that of scalability and efficiency, since CORBA implementations can often be rather heavy-weight. We believe this does not fundamentally affect our argument for middleware to support nomadic users, and there are numerous techniques being developed to improve the performance of CORBA implementations (e.g. [8]). Nonetheless, this is an important issue and deserves further study.

3.3 A middleware approach to nomadic user support

We briefly discuss additional features required for supporting nomadicity, and touch upon a possible architecture for providing them by augmenting or building on top of CORBA.

From our point of view, one of the features currently not present in CORBA is explicit support for nomadicity. In particular, it does not intrinsically contain building blocks for the services we described in the section: nomadic messaging (single-number addressing, message filtering, routing, translation, profile management, etc.), nomadic access to vertical and horizontal applications (in particular, building blocks for handling the bandwidth, connectivity and coverage limitations of wireless media, and conserving resources of portable devices) address translation and object mobility in response to user motion. In general, this observation also holds for other middleware platforms such as DCE. In the specific case of CORBA, some of these facilities can be provided as higher-level functions (possibly, Common Facilities), while some may need to be integrated into the basic CORBA model.

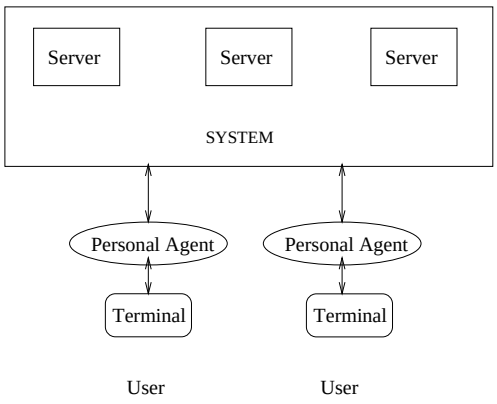

Figure 3: The user's model of the system

Consider a scenario where an enterprise information system needs to support nomadic users. The information system is relatively large and may be physically dispersed, and a distributed computing environment is necessary. We assume that the information system has a client-server computing model, with a high-speed network providing reliable communications at the physical level, and for concreteness we assume that CORBA is the middleware platform used.

Fig. 3 shows the desirable user-level model for nomadicity services, a "user-agent-system" model. Every user is assumed to be a (potentially) nomadic user, and each user has a distinct personal agent. From the user's point of view, all interactions with the system are mediated by the user's personal agent, who maintains all the information required to provide seamless, integrated services, even if the user is mobile, uses different media or different terminal equipment.

Although the user is provided the view that his or her personal agent is logically a single, centralized entity, for efficiency and other reasons the personal agent is not implemented as such. Instead, the personal agent is implemented as a number of distributed functions, which we call "Managers".

Each Manager provides a subset of the functionality required by the personal agent, and each manager provides this functionality for all the personal agents in the system. To ensure scalability as the number of users (and hence personal agents) increases, each Manager itself may be implemented as a number of distributed, objects running on multiple machines.

Each of the functions that we have shown in the "Nomadicity middleware" section consists of functions which can be shared by numerous applications. Thus, as an example, the profile manager may manage the user's "media profile", which specifies how the user can be reached at various times of the day (e.g., between 7 am - 8 am by home phone, 8 am - 5 pm by pager and voice mail, etc.) This information can be used by the nomadic messaging application to deliver phone calls, as well as by the personal information services application to deliver information about stocks that the user is interested in.

We now describe each of the Managers in Fig. 4 in terms of the data they maintain and the functions they perform.

The Profile Manager maintains the user's profile information; any references or changes to the user's profile must be made via Profile Manager objects. Three profile examples are:

1. *Address Profile.* The address profile contains the name and corresponding user-level addresses for the various media that the user employs, i.e., home phone number, fax number, email address, etc.
2. *Media Profile.* The media profile specifies what medium is to be used to deliver information to the user for each time of day (or day of week, etc.) The medium may be one where the user is not immediately disturbed (e.g., voice mail.)
3. *Personal Information Service Profile.* This profile specifies, for each information service that the user is interested in, what items of information the user is interested in. For example, if the user has subscribed to a traffic information service, the profile contains the user's commute route, i.e., the names of roads and bridges which the user commutes on (e.g. "New Jersey Turnpike Exit 11 to Exit 14, Holland Tunnel, Brooklyn Bridge, ...").

Along with these profiles, the Profile Manager will contain objects and methods to create and modify the profiles. These objects can be invoked by applications running on the user's behalf (e.g., the Nomadic Messaging or Personal Information Service application) or the system administrator's behalf.

The Personal Information Service Profile could reside in the relevant Personal Information Service application (e.g., the stocks profile with the stocks application, the traffic profile with the traffic application, etc.), since each such application needs this information in order to function. However, it is preferable that the profile itself be maintained by the Profile Manager, since it specifies the criteria by which personalized information is to be sent to the user, but not how the filtering is to be done, i.e., the filtering engine.

In the case of the traffic information service example, suppose information about traffic conditions arrives to the Personal Information Service Application as a continuous stream of records, where each record is a short paragraph of English text describing traffic conditions at some location. One could imagine a range of filtering engines of different sophistications that could be used to filter the information, from an engine which uses simple textual pattern matching (e.g. search for the pattern "Holland Tunnel", and deliver any item containing that pattern to the user) to advanced artificial intelligence techniques. The filtering engine for any particular application should be part of the application, while the objects used to manipulate the profile should be part of middleware. This allows the filtering engine to be changed without impacting the basic profile manipulation operations.

Now consider the Mailbox Manager. Some of the data which can be maintained by the Mailbox manager includes the actual mailboxes for voice, email, and fax messages. In addition, the facilities the Mailbox manager provides include objects and methods for inserting, deleting and copying messages for each of the mailboxes.

The functions performed by the Mailbox Manager could also be taken out of the nomadicity middleware and placed in the Nomadic Messaging application itself. However, these functions may be useful for other applications also. For instance, a user may be able to request the Sales Database Access application to send him or her a voice mail message if the inventory for a particular item falls below a certain level; the user can then access this information while travelling, using any telephone, without having access to a laptop or having to invoke the sales application. In general, our decision to place the Mailbox functionality at the disposal of all applications reflects the view that all users should be considered potentially nomadic, and mailboxes may be an effective way for applications to deliver the information to them.

The Media Translation Manager provides objects and methods for cross-media translation; in addition to the text conversion facilities

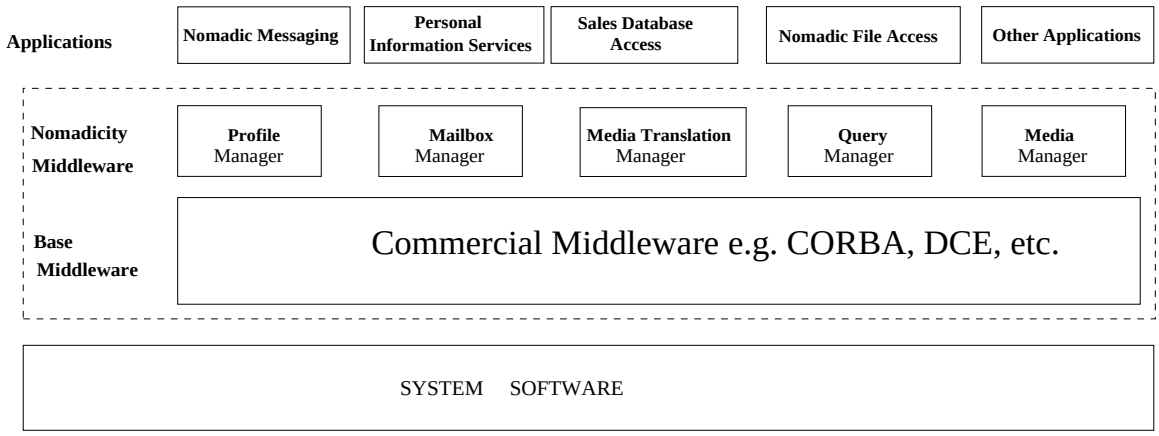

Figure 4: Nomadicity middleware architecture

(text-to-speech, text-to-fax and text-to-pager), it might also provide Optical Character Recognition (OCR), image scanning, or voice recognition facilities if these are useful for a range of applications.

The Query Manager provides a variety of facilities for mobile database access, such as caching of queries and responses and methods to manage canned SQL procedures, to minimize usage of wireless bandwidth. Finally, the Media Manager provides objects and methods for physically transmitting information via several media, such as fax, pager, e-mail, voice, etc. The wireless data media could include transmission over different systems and technologies, such as CDPD, RAM, ARDIS, etc [19]. It is clear that the facilities provided by both the Query Manager and the Media Manager are useful to a variety of possible applications.

4 Summary and Conclusion

We have argued that the functions required for supporting nomadic users should be built into the middleware of the information system, either as nomadicity middleware layer on top of the base middleware platform (CORBA, DCE, etc.), or as extensions to the core services these platforms provide. We have also very briefly touched upon a possible middleware architecture for access to enterprise information systems by nomadic users.

A general concern that arises with the use of platforms such as CORBA is that of scalability and efficiency. While these concerns are very important, they do not directly affect the argument for building common middleware for supporting nomadic users. In addition, there is research underway to address these performance issues e.g. [8]. Other current open issues include middleware facilities for address portability and for object mobility. Address portability allows a user to maintain the same public address (e.g., phone number, IP address, or Internet domain name address) even if the user moves geographically, uses a different communication protocol, or changes service providers. We have presented middleware solutions for PCS phone number portability [11] and are currently studying IP address portability.

Object mobility allows objects in the enterprise information system to physically move from one location to another, either autonomously or under the direction of other (user or system administrator) objects. This will allow a user's information to follow the user as he or she moves, thus providing better response time, reducing communication costs and improving system resource utilization. Object mobility has been studied previously, e.g. in the Emerald distributed object system [15] and in the ANSAware system [20], but primarily from the point of view of improving system load balance, availability etc, where object mobility is relatively infrequent and a system administration function. Recent work on integrating Java with CORBA [22] also offers potential solutions. Further work specifically related to nomadicity needs to be carried out.

Acknowledgements. Thanks are due to several colleagues at Bellcore: Rich Hovey, Paolo Missier, David Pepper and Chien-Chung Shen for reviewing a draft of this paper, and to R. Sekar as well as Arthur Yeo for helpful discussions. Thanks are also due to anonymous referees for their comments.

References

[1] AT&T. AT&T EasyLink Services. *http://www.att.com/work-net/easylink*, 1998.

[2] Bellcore. Bellcore's AirBoss Products. *http://www.bellcore.com*, 1996.

[3] BellSouth. BellSouth wireless data. *http://http://www.ram-wireless.com*, 1998.

[4] P. Bernstein. Middleware: A model for distributed system services. *Comm. ACM*, 39(2):86–98, 1996.

[5] Gael Core. Middleware pans out (eureka!). *LAN Times*, 8 Apr. 1998.

[6] M. Durr. Users clamor for connections. *Comm. Week*, pages S2–S3, December 4 1995.

[7] C. Perkins (ed.). IP mobility support. Technical Report RFC 2002, IETF, Oct. 1996. Available from ftp://ftp.ietf.org/internet-drafts.

[8] A. Gokhale and D. Schmidt. Optimizing the performance of the CORBA Internet Inter-ORB protocol over ATM. *Proc. Hawaii Intl. Conf. Sys. Sci.*, 1997. See http://www.cs.wustl.edu/ schmidt.

[9] B. Housel and D. B. Lindquist. WebExpress: A system for optimizing browsing in a wireless environment. In *Proc. MobiCom*, pages 108–116, Nov. 1996.

[10] R. Jain and N. Krishnakumar. *An asymmetric cost model for query processing in mobile computing environments.* Kluwer Academic Publishers, 1996.

[11] R. Jain, S. Rajagopalan, and L.-F. Chang. Phone number portability for PCS systems with atm backbones using distributed dynamic hashing. *IEEE J. Sel. Areas Comm.*, 15(1):96–105, 1997. Special Issue on Wireless ATM.

[12] R. Jain, T. Raleigh, C. Graff, and M. Bereschinsky. Mobile Internet access and QoS guarantees using Mobile IP and RSVP with location registers. In *ICC*, June 1998.

[13] Ravi Jain and N. Krishnakumar. Asymmetric costs and dynamic query processing in mobile computing environments. In *Proc. WINLAB Workshop, Rutgers Univ.*, Apr. 1995.

[14] A. Joch. Sales with no strings attached. *Byte*, Feb. 1998.

[15] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-grained mobility in the Emerald system. *ACM Trans. Comp. Sys.*, 6(1):109–133, 1988.

[16] J. T. Kisler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Trans. Comp. Sys.*, pages 3–25, Feb. 1992.

[17] Michael Kramer. Technology for data on the go. *Bellcore Exchange*, 12(1):12–17, 1996.

[18] A. Lindstrom. Making the switch to local number portability. *Telephony*, pages 48–49, July 10 1995.

[19] N. J. Muller. *Wireless Data Networking.* Artech House, 1995.

[20] M. H. Olsem. Experimenting with relocation and migration in distributed systems. Technical Report APM.1654, Architecture Projects Management, Poseidon House, Castle Park, Cambridge CB3 0RD, United Kingdom, Nov. 1995.

[21] Oracle. Oracle mobile agents design and white paper. 1997. http://www.uk.oracle.com/europe/sweden/sunexpo/mobag_wp.htm.

[22] R. Orfali and D. Harkey. *Client/server programming with JAVA and CORBA.* John Wiley& Sons, 1997.

[23] R. Orfali, D. Harkey, and J. Edwards. *The Essential Distributed Objects Survival Guide.* John Wiley& Sons, 1996.

[24] D. Pepper, S. Singhal, and S. Soper. Bellcore's Call Manager system. In *Proc. Interactive Voice Tech. for Telecom. App. (IVTTA)*, Sep 1996.

[25] J. Pepper. Globetrotters get access. *Comm. Week*, page S9, Dec. 4, 1995.

[26] J. Pepper. Deploying a remote sales staff. *Comm. Week*, pages 23–24, Jan. 15, 1996.

[27] J. Schwartz. Mobile remote access options grow. *Comm. Week*, pages 50–52, Sep. 25, 1995.

[28] C. Tait and D. Duchamp. An efficient variable-consistency replicated file service. In *Proc. USENIX File System Workshop*, May 1992.

[29] David Turock and Richard Wolff. Making telecommunications nomadic. *Bellcore Exchange*, 9(1):2–7, 1993.

[30] V. Vittore. Bellcore's AirBoss brings single number to data. *America's NETWORK*, page 24, June 1 1995.

[31] P. Wayner. Oracle hits the road. *Byte*, pages 207–208, June 1995.

[32] Special section: Wireless in the transportation industry. *Wireless*, Sep./Oct. 1995.

[33] Special section: Manufacturing and warehousing. *Wireless & Mobility*, Mar. 1998.