International Journal of Computer and Communication Technology

Volume 2 | Issue 2 Article 1

April 2011

Next Generation Middleware Technology for Mobile Computing

B. Darsana
Sr. Lecturer, Dept of ISE, The Oxford College of Engineering, Bangalore – 560068, Karnataka,
darsana@gmail.com

Karabi Konar
Sr. Lecturer, Dept of ISE, The Oxford College of Engineering, Bangalore – 560068, Karnataka,
karabi@gmail.com

Next Generation Middleware Technology for Mobile Computing

B. Darsana, Karabi Konar
Sr. Lecturer, Dept of ISE,
The Oxford College of Engineering,
Bangalore – 560068, Karnataka

Abstract-*Current advances in portable devices, wireless technologies, and distributed systems have created a mobile computing environment that is characterized by a large scale of dynamism. Diversities in network connectivity, platform capability, and resource availability can significantly affect the application performance. Traditional middleware systems are not prepared to offer proper support for addressing the dynamic aspects of mobile systems. Modern distributed applications need a middleware that is capable of adapting to environment changes and that supports the required level of quality of service.*

This paper represents the experience of several research projects related to next generation middleware systems. We first indicate the major challenges in mobile computing systems and try to identify the main requirements for mobile middleware systems. The different categories of mobile middleware technologies are reviewed and their strength and weakness are analyzed.

Key Words: dynamism, platform capability, quality of service, resource availability, network connectivity.

1. Introduction

The availability of lightweight, portable computers and wireless technologies has created a new class of applications called mobile applications. These applications often run on scarce resource platforms such as personal digital assistants, notebooks, and mobile phones, each of which have limited CPU power, memory, and battery life. They are usually connected to wireless links, which are characterized by lower bandwidths, higher error rates, and more frequent disconnections.

Most distributed applications and services were designed with the assumption that the terminals were powerful, stationary and connected to fixed networks. Conventional middleware technologies thus have focused on masking out the problems of heterogeneity and distribution to facilitate the development of distributed systems. They allow the application developers to focus on application functionality rather than on dealing explicitly with distribution issues.

Under the highly variable computing environment conditions that characterize mobile platforms, it is believed that existing traditional middleware systems are not capable of providing adequate support for the mobile wireless computing environment. There is a great demand for designing modern middleware systems that can support new requirements imposed by mobility. This paper provides a most relevant mobile middleware systems and goals that still need to be achieved.

2. Mobile Architectural Requirements

Middleware is an enabling layer of software that resides between the application program and the networked layer of heterogeneous platforms and protocols. It decouples applications from any dependencies on the plumbing layer that consists of heterogeneous operating systems, hardware platforms and communication protocols

Middleware plays a vital role in hiding the complexity of distributed applications. These applications typically operate in an environment that may include heterogeneous computer architectures, operating systems, network protocols, and databases. It is unpleasant for an application developer to deal with such heterogeneous "plumbing".

Middleware's primary role is to conceal this complexity from developers by deploying an isolated layer of APIs[6]. This layer bridges the gap between

application program and platform dependency. Middleware is defined as follows by Linthicum.

2.1 The Limitations of Mobile Computing

There are at least three common factors that affect the design of the middleware infrastructure required for mobile computing: mobile devices, network connection, and mobility, which vary from one to another in term of resource availability. Devices like laptops can offer fast CPUs and large amount of RAM and disk space while others like pocket PCs and phones usually have scarce resources. Hence, middleware should be designed to achieve optimal resource utilization. Network connections in mobile scenarios is characterized by limited bandwidth, high error rate, higher cost, and frequent disconnections due to power limitations, available spectrum, and mobility.

Due to these limitations, conventional middleware technologies designed for fixed distributed systems are not prepared to support mobile systems. They target a static execution platform where the host location is fixed, the network bandwidth does not fluctuate, and services are well defined. We next identify a number of important requirements that must be provided by middleware for mobile computing.

2.2 Analyzing the Requirements for Mobile Computing

During the system lifetime, the application behavior may need to be altered due to dynamic changes in infrastructure facilities, such as the availability of particular services.

Dynamic reconfiguration is thus required and can be achieved by adding a new behavior or changing an existing one at system runtime. Dynamic changes in system behavior and operating context at runtime can trigger re-evaluation and reallocation of resources. Middleware supporting dynamic reconfiguration needs to detect changes in available resources and either reallocate resources, or notify the application to adapt to the changes.

Adaptability[9] is also part of the new requirements that allows applications to run efficiently and predictably under a broader range of conditions. Through adaptation a system can adapt its behavior instead of providing a uniform interface in all situations. The middleware needs to monitor the resource supply/demand, compute adaptation decisions, and notify applications about changes.

Asynchronous interaction tackles the problems of high latency and disconnected operations that can arise with other interaction models. A client using asynchronous communication primitives issues a request and continues operating and then collects the result at any appropriate time. The client and server components do not need to be running concurrently to communicate with each other. A client may issue a request for a service, disconnect from the network, and collect the result later on. This type of interaction style reduces the network bandwidth consumption, achieves decoupling of client and server, and elevates system scalability.

Context-awareness is an important requirement to build an effective and efficient adaptive system. The context of a mobile unit is usually determined by its current location which, in turn, defines the environment where the computation associated with the unit is performed. The context may include device characteristics, user's activities, services, as well as other resources of the system. Context-awareness is used by several systems; however, few systems sense execution context other than location. The system performance can be increased when execution context is disclosed to the upper layer that assists middleware in making the right decision.

Lightweight middleware needs to be considered when constructing middleware for mobile computing. Current middleware platforms like CORBA[7] are too heavy to run on devices with limited resources. By default, they contain a wide range of optional features and all possible functionalities, many of which will be unused by most applications. For example, invoking a method on a remote object involves only client side functionality and either Dynamic or Static Invocation Interface. Most of the existing ORB implementations provide either a single or two separate libraries for the client and server sides that contains all functionality. This forces the client program to be glued with the entire functionality without having a choice to select a specific subset of this functionality.

3. Mobile Middleware Technologies

This section sheds some light on the different types of mobile middleware technologies. We start by introducing a classification that allows us to contrast and evaluate the different categories. Among the middleware systems we reviewed, we have identified four categories of middleware. Each category aims to support at least one of the above requirements imposed by mobility. These categories are reflective middleware, tuple space, context-aware middleware, and event-based middleware, each of which attempts to address the previous requirements using different approaches. The following table illustrates how various requirements are met by the different categories.

Table: 3.1. Requirements Vs Categories

Requirements	Reflective	Tuple Space	Context Aware	Event Based
Synchronous/ connection based	X		X	
Asynchronous/ connectionless		X		X
Re-configuration	X			
Adaptation	X		X	
Awareness	X		X	
Light weight				X

The above table shows the relation between Requirements and Categories. For synchronous /connection based Reflective and Context Aware is applicable. For asynchronous/connection Event based and Tuple space is applicable. For Light weight requirement Event based is the best approach. For Awareness and Adaption Reflective and context Aware meet the requirements.

3.1 Reflective Middleware

The reflection technique was initially used in the field of programming languages to support the design of more open and extensible languages. Reflection is also applied in other fields including operating systems and more recently distributed systems. The principle of reflection enables a program to access, reason about and change its own behavior.

Smith defined the concept of reflection in the following quote: “In as much as a computational process can be constructed to reason about an external world in virtue of comprising an ingredient process (interpreter) formally manipulating representations of that world, so too a computational process could be made to reason about itself in virtue of comprising an ingredient process (interpreter) formally manipulating representations of its own operations and structures”.

A reflective system consists of two levels referred to as meta-level and base-level[11]. The former performs computation on the objects residing in the lower levels. The latter performs computation on the application domain entities. The reflection approach supports the inspection and adaptation of the underlying implementation (the base-level) at run time. A reflective system provides a meta-object protocol (meta-interface) to define the services available at the meta-level. The meta-level can be accessed via a concept of reification. Reification means exposing some hidden aspect of the internal representation and hence they can be accessed by the application (the base-level). The implementation openness offers a straightforward mechanism to insert some behavior to monitor or alter the internal behavior of the platform. This enables the application to be in charge of inspecting and adapting the middleware behavior based on its own needs. Thus, a lightweight middleware with a minimal set of functionality is achieved to run on mobile systems.

The main motivation of this approach is to make the middleware more adaptable to its environment and better able to cope with changes. Examples of middleware systems that adopted the concept of reflection are OpenCorba, Open-ORB(Object request Broaker), DynamicTAO , FlexiNet , and Globe.

3.2 Tuple Space Middleware

Communication in a wireless environment is characterized by frequent disconnections and limited bandwidth. Communication models such as message passing, RPC, or RMI[6] all have the drawback of tight coupling. This means that a sender has to know the exact identity and address of a receiver. Also, the sender has to wait for the receiver to be ready for exchanging information (synchronization paradigm). In distributed open systems this tends to be too restrictive. A decoupled and opportunistic style of computing is thus required. Computing is expected to proceed even in the presence of disconnection and to exploit connectivity whenever it becomes available.

One solution is the concept of tuple space, which was initially introduced by Gelernter in as part of the Linda coordination language. Tuple Space systems[10] have proved their ability for facilitating communication in wireless settings. In general, a tuple space is a globally shared, associatively addressed memory space that is used by processes to communicate. A tuple space system can be realized as a repository of tuples, which are basically a vector of typed values or fields. Client processes create tuples and place them in the tuple space using a write operation. Also, they can concurrently access tuples using read or take operations. Most tuple space systems support both versions of the tuple retrieval operations, blocking and non-blocking.

A template, which is similar to a tuple, is used to match the contents of tuples in the tuple space during the retrieval operations. A template matches a tuple if they have an equal number of fields and each template field matches the corresponding tuple field. This form of communication fits well in mobile setting where logical and physical mobility is involved.

The Tuple space communication model, such as the one used in Linda,provides great flexibility for modeling concurrent process. This approach has also been extended with distributed tuple space.

3.3 Context-Aware Middleware

Mobile systems run in an extremely dynamic environment. The execution context changes frequently due to the user's mobility. Mobile hosts often roam around different areas, and services that are available before disconnecting may not be available after reconnecting. Also, the bandwidth and connectivity quality may quickly alter based on the mobile host movements and their locations.

The application developers cannot predict all the possible execution contexts that allow the application to know how to react in every scenario. The middleware has to expose the context information to the application to make it aware of the dynamic changes in execution environment.

The application then instructs the middleware on how to adapt its own behavior in order to achieve the best quality of service. Many research groups gave special attention in particular to location awareness. For example, location information was exploited to provide travelers directional guidance, to discover neighboring services, and to broadcast messages to users in a specific area. Most location-aware systems depend on the underlying network operating system to obtain location information and generate a suitable format to be used by the system.

The heterogeneity of coordination information is not supported and hence different positioning systems are required to deal with different sensor technologies, such as the Global Positioning System (GPS) outdoors, and infrared and radio frequency indoors.

MobiPADS is a middleware system for mobile environment. The principal entity is Mobilets, Which are entities that provide a service, and which can be migrated between different MobiPADS environment.

3.4 Event-Based Middleware

Invocation-based middleware systems such as CORBA(Common object request Broaker Architecture) or Java (Remote Method Invocation)[7] are useful abstractions for building distributed systems. The communication model for these platforms is based on a request/reply pattern: an object remains passive until a principle performs an operation on it. This kind of model is adequate for a local area network (LAN) with a small number of clients and servers, but it does not scale well to large networks like the Internet.

The main reason is that the request/reply model only supports one-to-one communication and imposes a tight coupling between the involved participants because of the synchronous paradigm. This model is also unsuitable for unreliable and dynamic environment.

The event-based communication paradigm is a possible alternative for dealing with large-scale systems. Event notification is the basic communication paradigm that is used by event-based middleware systems. Events contain data that describes a request or message. They are propagated from the sending components to the receiver components. In order to receive events, clients (subscribers) have to express (subscribe) their interest in receiving particular events. Once clients have subscribed, servers (publishers) publish events, which will be sent to all interested subscribers.

This paradigm hence offers a decoupled, many-to-many communication model between clients and servers. Asynchronous notification of events is also supported naturally. There are several examples of middleware based on the event-based systems, but not limited to, Hermes, CEA, STEAM, JEDI and ToPSS.

3.5 Other Middleware Solutions

There are many other middleware solutions that have been proposed particularly to target mobility aspects. Unpredictable disconnections are one of the major mobility issues that have been addressed by several systems. Systems like Coda, its successor Odyssey], Bayou, and xmiddle have used data replication to increase data availability to mobile users. This allows users access to replicas and to continue their tasks whenever the disconnection operations take place.

Each system uses different mechanisms to guarantee the ultimate consistency among the replicas. These mechanisms include the support for discovery of inconsistent data as well as data reconciliation. Services discovery is another well-know problem introduced by user mobility. In a static environment, new services can be easily discovered by asking service providers to register with a well-known location service. In a mobile computing environment, the situation is different since mobile hosts often roam around various areas.

Services that were present before disconnecting from the network may not exist after reconnecting. Jini and Ninja Service Discovery Service (SDS) are examples of systems that support dynamic service discovery, Bayou is

the system which support s disconnected operations and Jini is the system which support discovery of services.

4. Analysis of next-generation middleware

This section summarizes the previous discussion on next-generation middleware with an emphasis on lessons learned from investigating the proposed solutions presented in the previous section. We particularly aim to highlight in which extent these solutions are suitable for mobile settings.

It is a major challenge to solve all problems of mobile distributed systems. This is true due to the high degree of dynamism in mobile environments. Current middleware platforms like CORBA cannot successfully run in such an environment. Hence, there is an urgent need for new solutions that support particular application requirements such as dynamic reconfiguration, context-awareness, and adaptation. We believe that the reflective approach provides a solid base for building next generation middleware platforms and overcomes the limitations of the current middleware technologies.

More specifically, the architecture follows a white box philosophy that provides principled and comprehensive access to internal details. It can also decrease problems of maintaining integrity since each object/interface is attached to a single meta-object at a time. Therefore, any modification to a meta-object can only affect a single object.

Some reflective systems support higher level of reflection since they can add or remove methods from objects and classes dynamically and even alter the class of an object at run time. In contrast, others concentrate on a simpler reflective paradigm to achieve a better performance. Their reflective mechanisms are not part of the usual flow of control and only invoked when required. Reflective middleware like FlexiNet and DynamicTAO are built around the concept of object-oriented and component frameworks respectively.

Component Frameworks (CFs) were initially defined by Szyperski as “collection of rules and interfaces that govern the interaction of a set of components plugged into them” There are several advantages of using CFs over the object-oriented approach. The uses of CFs are not limited to a particular programming language and there is no inheritance relation between components and framework.

Hence, components and CFs can be developed independently, distributed in binary form, and combined at run time. We have noticed that the issue of consistent dynamic reconfiguration is still under research. There is some work in this area that has focused on developing reconfiguration models and algorithms that enforce well-defined consistency rules while minimizing system disturbance.

Performance is another issue that remains a matter for further investigation. All of the reflective systems presented previously impose a heavy computational load that would cause significant performance degradation on mobile devices. Tuple-space systems exploit the decoupled nature of tuple spaces for supporting disconnected operations in a natural manner. By default they offer an asynchronous interaction paradigm that appears to be more appropriate for dealing with intermittent connection of mobile devices, as is often the case when a server is not in reach or a mobile client requires to voluntary disconnect to save battery and bandwidth.

By using a tuple-space approach, we can decouple the client and server components in time and space. In other words, they do not need to be connected at the same time and in the same place. Tuple-space systems support the concept of a space of spaces that offers the ability to join objects into appropriate spaces for ease of access. This opens up the possibility of constructing a dynamic super space environment to allow participating spaces to join or leave at arbitrary time. The ability to use multiple spaces will elevate the overall throughput of the system.

Throughout our study, we have noticed that JaveSpaces and TSpaces typically require at least 60Mbytes of RAM. This is not affordable by most handheld devices available on the market nowadays.

Context-Aware systems provide mobile applications with the necessary knowledge about the execution context in order to allow applications to adapt to dynamic changes in mobile host and network condition. The execution context includes but is not limited to: mobile user location, mobile device characteristics, network condition, and user activity (i.e., driving or sitting in a room). The context information is typically disclosed in a convenient format to the applications that instruct the middleware system to apply a certain adaptation policy. To our knowledge, most context-aware applications are only focusing on a user’s location while other things of interest are also mobile and changing.

We believe that a reflective approach may improve the development of context-aware services and applications. In general, a reflective system provides mobile applications with context information that they need to optimize middleware and their own behaviors. One reflection solution has suggested the use of metadata and reflection to support context-aware applications.

Traditional, invocation-based middleware like CORBA follow a request/reply communication style, which does not scale well to large networks like the Internet.

Event-based paradigms present an interesting style that supports the development of large-scale distributed systems. In such a system, clients first announce their

interest in receiving specific events and then servers broadcast events to all interested clients. Hence, the event-based model achieves a highly decoupled system and many-to-many interaction style between clients and servers.

We believe that not a lot of work has managed to merge the publish/subscribe communication approach with event-based middleware systems. Most existing systems do not combine traditional middleware functionality (i.e., security, QoS, transactions, reliability, access control, etc.) with the event-based paradigm. We feel that event-based middleware can be more successful if such functionality is provided in the future.

Event-based systems also do not integrate well with object-oriented programming languages due to the major mismatch between the concept of objects and events. Events are viewed as untyped collection of data (attribute/value pairs) whereas current programming languages only support typed objects. Hence, events should support data typing in order to be treated as objects. In addition, the developers are responsible for handling the low-level event transmission issues. Current publish/subscribe systems are restricted to certain application scenarios such as instant messaging and stock quote dissemination. This indicates that such systems are not designed as general middleware platforms. From this discussion, we can realize that until this moment there is no middleware system that can fully support the requirements for mobile applications. Several solutions have considered one aspect or another; however, the door for further research is still wide open.

5. Conclusion

The proliferation and development of wireless technologies and portable appliances have paved the way for a new computing paradigm called mobile computing. Mobile computing software is expected to operate in environments that are highly dynamic with respect to resource availability and network connectivity. Traditional middleware products, like CORBA and Java RMI, are based on the assumptions that applications in distributed systems will run in a static environment; hence, they fail to provide the appropriate support for mobile applications. This gives a strong incentive to many researchers to develop modern middleware that supports and facilitates the implementation of mobile applications.

We discussed the state-of-the-art of middleware for mobile computing. We presented common characteristics and a set of requirements for mobile computing middleware, which allows us to better understand the relationship between the existing bodies of work on next-generation middleware. We explained the reasons behind the failure of traditional middleware systems for supporting mobile settings. We also identified, illustrated, and comparatively discussed four middleware classes: reflective middleware, tuple space, context-aware middleware, and event-based middleware. Beside these four categories, a pool of other middleware solutions has been developed to address specific mobility issues. However, none of these middleware systems support all the requirements. We concluded each category with a simple qualitative evaluation and made a number of observations related to some issues that need further investigations.

6. References

[1] Gordon S. Blair, Geoff Coulson, Anders Andersen, Lynne Blair, Michael Clarke, F´abio Costa, Hector Duran Limon, Tom Fitzpatrick, Lee Johnston, Rui Moreira, Nikos Parlavantzas, and Katia Saikoski, “The Design and Implementation of Open ORB 2”, IEEE Distributed Systems Online, 2(6), 2009.

[2] R. Meier and V. Cahill, "STEAM: Event-Based Middleware for Wireless Ad Hoc Networks", in Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02). Vienna, Austria, 2009, pp. 639-644.

[3] Antonio Carzaniga and Alexander L. Wolf, “Content-Based Networking: A New Communication Infrastructure”, In NSF Workshop on an Infrastructure for Mobile and Wireless Systems, Scottsdale, AZ, October 2008.

[4] Antony Rowstron and Peter Druschel, “Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-Peer Systems”, In Proc. of Middleware 2009, November 2009.

[5] G. Ashayer, H. K. Y. Leung, and H.-A. Jacobsen, “Predicate Matching and Subscription Matching in Publish/Subscribe Systems,” in Proceedings of the Workshop on Distributed Event-based Systems, 22nd International Conference on Distributed Computing Systems, (Vienna, Austria), IEEE Computer Society Press, July 2008.

[6]International Journal of Ad Hoc and Ubiquitous Computing (2007)
Volume: 2, Issue: 4, Publisher: Inderscience Publishers, Pages: 263
ISSN: 17438225

[7] The impact of research on middleware technology
Volume 41 , Issue 1 (January 2007
Pages: 89 - 112 Year of Publication: 2007 ISSN:0163-5980

[8]W. Andreas. HyperDesk's response to the ORB RFP. OMG TC Document 91.1.6, Object Management Group, 492 Old Connecticut Path, Framingham, MA, USA, Jan. 1991.

[9] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana ANSA.

[10] The Advanced Network Systems Architecture (ANSA). Reference manual, Architecture Project Management, Castle Hill, Cambridge, UK, 1989.

[11] H. E. Bal. *The Shared Data Object Model as a Paradigm for Programming Distributed Systems*. PhD thesis, Dept. of Computer Science, Vrije Universiteit Amsterdam, The Netherlands, 1989.