Framework of an Application-Aware Adaptation Scheme for Disconnected Operations

Umar Kalim, Hassan Jameel, Ali Sajjad, Sang Man Han, Sungyoung Lee and Young-Koo Lee

Department of Computer Engineering, Kyung Hee University
Sochen-ri, Giheung-eup, Yongin-si, Gyeonggi-do, 449-701, South Korea
{umar, hassan, ali, i30000, sylee}@oslab.khu.ac.kr, yklee@khu.ac.kr

Abstract. The complex software development scenarios for mobile/hand-held devices operating in wireless environments require adaptation to the variations in the environment (such as fluctuating bandwidth). This translates to maintenance of service availability in preferably all circumstances. In this paper we propose that a mobile computing system (for hand-held, wireless devices) must be based on the combination of reflection, remote evaluation and code mobility mechanisms such that the communication framework[1] allows developers to design disconnection-aware applications which maintain service availability in case of varying circumstances by automatically redeploying essential components to appropriate locations. This not only allows the application to continue executing in varying conditions, but also in entirely disconnected modes.

1 Introduction

The complexity, size and distribution of software today is rapidly increasing. This has complemented the ubiquity of hand-held devices and promoted the growth of distributed systems and applications. One can thus think of numerous, complex software development scenarios which utilize a large number of hand-held devices, such as in environment monitoring and surveying, postal services, patient monitoring etc. Such scenarios present intricate technical challenges for middleware technologies [1]. In particular, the middleware must adapt to the variations in the environment (e.g. fluctuating bandwidth) of the mobile device and service availability must be maintained in (preferably) all circumstances [1].

The conventional middleware being heavy, and relatively inflexible, fails to properly address such requirements. The fundamental reason is that traditional middleware systems have been designed adhering to the principle of transparency. Despite the fact that this design [2] [3] has proved successful for traditional distributed systems, this concept has limitations when considered in a mobile environment, where it is neither possible, nor preferred, to hide the implementation details from the user. Also, applications may posses information

[1] This research work has been partially supported by KOSEF, Korea, for which Professor Sungyoung Lee is the corresponding author

that could facilitate the middleware to perform efficiently. Thus to cope with such limitations, numerous research efforts have been made [4] [5] on designing middleware systems suitable for such environments. However the solutions developed to date do not fully support the necessary level of middleware configurability and reconfigurability that is required to facilitate mobile computing and disconnected operations. Thus in our opinion a simple but dynamic solution is the need of the hour.

We propose that a mobile computing system for hand-held devices must be based on the combination of *reflection* [6], *remote evaluation* [7] and *code mobility* [8]. The remote evaluation paradigm not only enables the clients to out-source resource intensive tasks, but also allows more client-side applications (because of their smaller footprint). Reflection is a fundamental technique that supports both introspection and adaptation. In order to maintain service availability in a distributed system, in case of varying circumstances the middleware can utilize code mobility and reflection to automatically redeploy essential components to appropriate locations defined by the application policy. Moreover, such systems can be implemented optimally if the adaptation scheme is application aware, i.e. the framework allows the developer to determine the policies, such as the constraints the components may have, dependency among components, restriction on collocation of components, how the system should react to different circumstances etc.

1.1 Code Mobility and Autonomy of Components

Under favourable circumstances, remote evaluation is one of the most appropriate solutions for mobile, hand-held devices. However in unfavourable conditions, code mobility overcomes the limits of fluctuating bandwidth and disconnections as it allows for specifying complex computations to move across a network - from the server to the client end. This way, the services that need to be executed at a platform residing in a portion of the network reachable only through an unreliable link could be relocated and hence maintain Quality of Service. In particular, it would not need any connection with the node that sent it, except for the transmission of the final results of its computation.

Also, autonomy of application components brings improved fault tolerance as a side-effect. In conventional client-server systems, the state of the computation is distributed between the client and the server. A client program is made of statements that are executed in the local environment, interleaved with statements that invoke remote services on the server. The server contains (copies of) data/code that belongs to the environment of the client program, and will eventually return a result that has to be inserted into the same environment. This structure leads to well-known problems in presence of partial failures, because it is very difficult to determine where and how to intervene to reconstruct a consistent state. The action of migrating code, and possibly sending back the results, is not immune from this problem. In order to determine whether the code has been received and avoid duplicated or lost mobile code, an appropriate protocol must be in place. However, the action of executing code that embodies a set of

interactions that should otherwise take place across the network is actually immune from partial failure and can permit execution even in the face of absolute disconnection. An autonomous component encapsulates all the state involving a distributed computation, and can be easily traced, check-pointed, and possibly recovered locally, without any need for knowledge of the global state.

Thus to introduce the capability of dynamic reconfiguration to achieve the above mentioned objectives, the system must posses certain characteristics, such as, it should be based on a distributed object framework, the system must be able to redeploy/replace[2] components[3], it should be able to recover gracefully in the case of faults and there should be a procedure to reconcile components upon reconnection.

2 Framework

In order to narrow down the scope of the problem, we distinguish between voluntary and involuntary disconnection. The former refers to a user-initiated event that enables the system to prepare for disconnection, and the latter to an unplanned disconnection (e.g., due to network failure). The difference between the two cases is that in involuntary disconnection the system needs to be able to detect disconnection and reconnection, and it needs to be pessimistically prepared for disconnection at any moment, hence requiring to proactively reserve and obtain redundant resources (if any) at the client. Here we are only focusing on voluntary disconnections and defer the task of predicting and dealing with involuntary disconnections as future work. Note that the steps for the remedy will be the same, whether the disconnection is voluntary or involuntary.

2.1 Characteristics of the Components and their Classification

As the system comprises of components as the building blocks we propose that the primary components participating in the reconfiguration must be serializable and they must implement the DisconnectionManagement interface as shown in figure 1. This interface advertises two primary methods; `disconnect` and `reconnect`. These methods are invoked by the framework on disconnection and reconnection. The first requirement facilitates the components in relocating themselves while maintaining their state. The second requirement enables the application to prepare for disconnection. The use of `disconnect` is to compile the component state and transfer it in marshaled form, over the network along, with the code to be executed locally at the client. Similarly, `reconnect` is used to perform the process of reconciliation among components upon reconnection, details of which are explained in [9]. The components are classified with respect

[2] The use of a component with reduced (or similar) functionality but the same interface, as the substitute to a (remote) component with reference functionality

[3] Analogous to an object, here a component is referred to as a self contained entity that comprises of both data and procedures. Also data access has not been considered separately because data is always encapsulated in a component

to disconnection (*Log*, *Substitute* and *Replica*) and reconnection (*Latest*, *Revoke*, *Prime* and *Merge*), details of which are specified in [9].

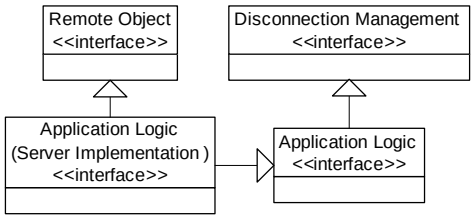

Fig. 1. Hierarchy of interfaces for *disconnection-aware* components

3 Disconnection Management

When it comes to maintaining service availability in the face of a disconnection, there is a need to relocate the required server code (partially or completely) to the client, in order to make local processing possible.

3.1 Working

The state-transition diagram in figure 2 summarizes the working [9] of the framework. The mechanisms of Reflection, dynamic class loading and linking and serialization (provided by Java [10]) are employed to achieve code mobility. Once a disconnection event is fired, the framework propagates the event to all disconnection-aware references. These references then invoke the `disconnect` method. This method prepares the reference for the disconnection. Using the mechanisms of introspection each component and each of its contained objects are traversed recursively and a list of references to be relocated is prepared. This list is prioritized with respect to the policy determined by the application designer and each reference is treated as per its classification [9]. The framework maintains a sufficient state of each reference in order to restore the system to the state before disconnection.

4 Related Work

A substantial debt is owed to Coda [4]. The authors were among the first to demonstrate that client resources could be effectively used to insulate users and applications from the hurdles of mobile information access. Coda treats disconnection as a special case of network partitioning where the client may continue to use the data present in its cache, even if its disconnected. Odyssey [5], inspired by Coda [4], proposed the concept of application-aware adaptation. The essence

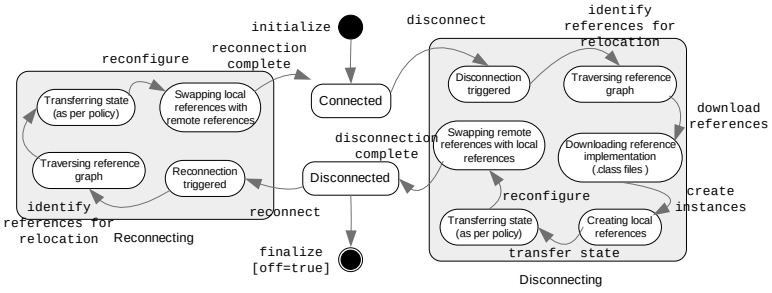

Fig. 2. State-transition diagram from disconnection/reconnection management

of this approach is to have a collaborative partnership between the application and the system, with a clear separation of concerns.

FarGo-DA [11], an extension of FarGo, a mobile component framework for distributed applications proposes a programming model with support for designing the behaviour of applications under frequent disconnection conditions. The programming model enables designers to augment their applications with disconnection-aware semantics that are tightly coupled with the architecture, and are automatically carried out upon disconnection.

5 Prototype Implementation

The framework is developed using J2SE [10] as the fundamental platform for the application; both the client and server components. Java RMI [12] is used for remote evaluation, where as the Reflection classes are used for introspection and reference management when objects are relocated (from the server to the client or vice versa) and references are swapped. Components are notified about disconnection or reconnection via event-notification mechanism.

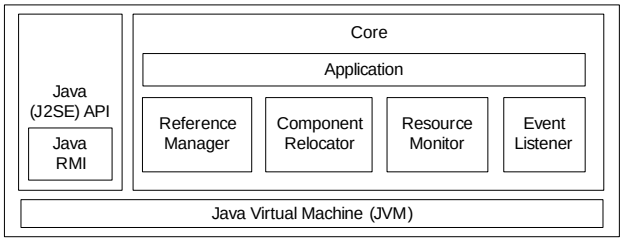

Fig. 3. Module layout of the prototype implementation

We have implemented a prototype application for patient monitoring and diagnosis service along with the framework libraries in order to verify the feasibility of our proposal. This implementation [9] is part of our ongoing research

[13]. The module layout of the framework along with the application is shown in figure 3. It may be noted that the framework comprises of two sub-systems; one operating at the server end and the other at the client end.

Unlike [14], our approach being simple and discreet avoids the computational overhead required to determine the component distribution in different circumstances. This is primarily due to the application aware approach, which allows the developer to determine the application policies

6 Conclusion

In this paper we proposed a mobile computing middleware-framework for handheld devices which is based on the combination of reflection [6], remote evaluation [7] and code mobility [8]. We have implemented a prototype application [9] along with the framework libraries in order to demonstrate the feasibility of the approach. The results reflect that significant benefits may be obtained by maintaining service availability even in the face of a disconnection.

References

1. Satyanarayanan, M.: Pervasive computing: Vision and challenges. IEEE Personal Communications (2001) 10–17
2. Reference-model: Iso 10746-1 - open distributed processing. International Standardization Organization (1998)
3. Emmerich, W.: Engineering Distributed Objects. John Wiley and Sons (2000)
4. Kistler, J. Satyanarayanan, M.: Disconnected operation in the coda file system. In: 13th ACM symposium on Operating Systems Principles, ACM (1991) 213–225
5. Noble, B. Satyanarayanan, M.: Agile application-aware adaptation for mobility. In: 16th ACM Symposium on Operating Systems Principles, ACM (1997)
6. Maes, P.: Concepts and experiments in computational reflection. (In: 2nd Conference on Object Oriented Programming Systems, Languages and Applications)
7. Stamos, J. Gifford, D.: Remote evaluation. In: Transactions on Programming Languages and Systems, ACM (1990) 537–564
8. Fuggetta, A.: Understanding code mobility. In: Transactions on Software Engineering. Volume 24., (IEEE) 342–361
9. Kalim, U.: Technical report: Framework of an application-aware adaptation scheme for disconnected operations. (http://oslab.khu.ac.kr/xims/mgrid/techreport-disconn-umar.pdf)
10. Sun-Microsystems: Java. (http://java.sun.com/j2se/)
11. Weinsberg, Y. Israel, H.: A programming model and system support for disconnected-aware applications on resource-constrained devices. In: 24th International Conference on Software Engineering. (2002) 374–384
12. Sun-Microsystems: Java rmi. (http://java.sun.com/products/jdk/rmi/)
13. Kalim, U. Jameel, H.: Mobile-to-grid middleware: An approach for breaching the divide between mobile and grid environments. In: 4th International Conference on Networking, Springer Verlag (2005) 1–8
14. Marija, M.: Improving availability in large, distributed, component-based systems via redeployment. Technical Report USC-CSE-2003-515, Center for Software Engineering, University of Southern California (2003)