Context-Aware Leasing for Mobile Ad hoc Networks

Elisa Gonzalez Boix, Jorge Vallejos, Tom Van Cutsem [⋆], Jessie Dedecker, and Wolfgang De Meuter

Programming Technology Lab
Vrije Universiteit Brussel, Belgium
{egonzale,jvallejo,tvcutsem,jededeck,wdmeuter}@vub.ac.be

Abstract. Distributed memory management is substantially complicated in mobile ad hoc networks due to the fact that nodes in the network only have intermittent connectivity and often lack any kind of centralized coordination facility. *Leasing* provides a robust mechanism to manage reclamation of remote objects in mobile ad hoc networks. However, leasing techniques limits the lifetime of remote objects based on timeouts. In mobile networks, we also observe that devices need to continuously adapt to changes in their *context*. In this position paper, we argue that changes in context not only require adaptation in the behaviour of the application but also permeate to distributed memory management, leading to the concept of *context-aware leasing*.

1 Introduction

The recent advances in the field of Ambient Intelligence (AmI) have set new challenges to the development of a new type of distributed applications with sophisticated characteristics. AmI applications are distributed among mobile devices interconnected by wireless communication media that allow them to interact spontaneously with other devices in the environment forming *mobile ad hoc networks*. Example mobile ad hoc networking applications range from modest, already commonplace applications like collaborative text-editors, to more futuristic pervasive and ubiquitous computing [11] scenarios. Such scenarios also introduce new opportunities to build applications that can sense and deal with changes in their physical and computational *context*. Typically, these context changes require adaptation in the behaviour of the application.

In previous work, we have explored the impact of the hardware phenomena of mobile networks on distributed memory management and proposed the use of leasing [4] as a robust technique to reclaim remote objects in such network topology [3]. This paper focuses on the repercussions of context information on leasing and subsequently proposes the integration of context events directly into distributed memory management resulting in the concept of *context-aware leasing*.

2 Context-awareness and Leasing

Before discussing the repercussions of context-dependent adaptations on distributed memory management, we first introduce some terminology and concepts from the area of context-awareness and leasing.

[⋆] Research Assistant of the Fund for Scientific Research Flanders, Belgium (F.W.O.)

Leasing. Leases were originally introduced as a fault-tolerant approach in the context of distributed file cache consistency [4]. A *lease* denotes the right to access a resource for a limited amount of time. In distributed memory management, remote references play the role of the lease and the objects they refer to play the role of the resource. In other words, client objects from other machines can reference remote objects by means of *leased object references*. Therefore, a leased object reference is a remote object reference that grants access to a remote object only for a limited period of time. When a client first references a remote object, a leased object reference is created and associated to the remote object. From that moment on, the client accesses the remote object transparently via the leased reference until it expires. A leased reference can be renewed or revoked before its lease time expires. When the time interval has elapsed, the access to the remote object is revoked, i.e. the lease expires and thus the client can no longer access the remote object. Once all leases for a remote object have expired, the object can be garbage collected when no local references refer to it.

Leases are a robust technique for distributed garbage collection in mobile ad hoc networks such that remote objects can be reclaimed in face of both transient and permanent disconnections. In previous work, we described the above sketched leased object references as time-decoupled remote object references with built-in leasing semantics [3]. Throughout this paper, we assume such a leasing approach as the basis for our reasoning and examples. In what follows, we introduce specific features necessary for this paper but further details are available in a technical report [3].

As in contemporary leasing approaches (e.g. in Java RMI [9], Jini [10] and .NET Remoting [6])), the lifetime of remote objects is determined by means of timeouts. Our leasing approach also incorporates two variants of leased references which transparently adapt their lease time under certain circumstances. The first variant is a *renew-on-call* leased reference that automatically prolongs the lease upon each method call received by the remote object. The second variant is a *single-call* leased reference that automatically revokes the lease upon performing a method call on the remote object. Such leases are useful for objects which adhere to a *single call* pattern such as the callbacks objects that are often passed along with messages in asynchronous message passing schemes to return computed values. Our leasing approach provides the following language support to create a basic leased reference and the two above mention variations:

```
lease: timeout for: object
renewOnCallLease: timeout for: object
singleCallLease: timeout for: object
```

A leased reference created with the **lease** construct only lasts for the given time unless a renewal or revocation is explicitly issued. The **renewOnCallLease** construct creates a lease that is automatically prolonged on every message invocation with `timeout` value. Finally, **singleCallLease** construct creates a lease that expires either after the remote object receives a single message or when the `timeout` expires if no messages have been received. Other language support is provided to explicitly manipulate the lifetime of a leased reference (i.e. **renew** and **revoke** language constructs).

Context-awareness. We adopt the definition of *context information* as proposed in [5]: context is any piece of information which is computationally accessible. Examples of such information include not only information that can be automatically extracted from the surroundings by means of sensors (e.g. location or temperature), but also information from the computation environment (e.g. when devices disjoin and join the network) or user preferences. For the sake of this paper, we assume the existence of a mechanism to extract meaningful context information, e.g. ContextToolkit [8], and the use of a common standard ontology for context information that allows applications to understand this information [7], i.e. all applications use the same terminology for context events.

In the remainder of this section, we discuss the effect of such context information on leasing and illustrate why we need extensions to leasing to deal with it.

2.1 Leasing based on time is not enough

Leasing allow developers to guide the distributed garbage collector by determining the lifetime of remote objects. However, this places burden on developers. Determining the proper lease period is not straightforward and may even depend on system parameters such as the number of clients. Typically, leasing only allows developers to express the validity of a lease based on timeouts [9, 10, 6]. In mobile ad hoc networks, we observe that the validity of the leased reference can also depend on changing context parameters which monitor different types of events such as hardware events (e.g. disconnections of devices) or physical events (e.g. location of devices). As a concrete example, consider a remote reference which should remain valid only while the battery level of the device hosting the remote object is above an acceptable limit. A first approximation to implement this scenario in our leasing approach is to extend the **lease** language construct as follows:

```
renewOnCallLease: minutes(10) renewalConditions: {if: { batterylevel > 10%}}
  for: object
```

The above code excerpt illustrates the use of a boolean condition in a renew-on-call lease so that the leased reference is automatically renewed in relation to a context event, i.e. battery consumption.

Such renewal conditions can also be applied to other context events. Consider an example of a user attending a conference who wants to print one of their PDA files on a printer located at the conference building. Typically, such users will have restricted access to the resources available during the time that the conference is held, e.g. their internet access is limited to the conference building or they cannot print more than 100 pages. This example could be modelled in our leasing approach by extending the **lease** language construct to express the lease time in terms of boolean conditions. For example, restricting the number of pages that a user can print could be expressed as follows:

```
lease: getTimeLeft(days(3)) revokedOn: {if: { printedPages > MAXIMUM}} for: (
  object: {
     printingQueue : Queue.new();
     def print(doc){
       queue.add(doc);
     }
  }
)
```

As shown in the code above, a leased reference to a remote object offering a printing service will thus expire either when the conference finishes, i.e. in 3 days, or when the user exceeds its printing quota.

Another example of the impact of context information on leasing is related to the connection volatility phenomenon featured in mobile ad hoc networks. Transient disconnections should not affect an application, allowing both parties to continue their collaboration where they left off. Often, a remote reference may only be useful while the devices are in each others communication range. Note that a disconnection event is not related to time: it can happen at any point in time in mobile ad hoc networks due to the mobility of devices together with the limited communication range of devices. As a concrete example, consider an application to visualize the map and request sightseeing information of a site as the user moves about. The application will interact with devices embedded in different locations (e.g. buildings or touristic information points) to visualize the map and get information about a particular location. As the user moves out of range, the leased reference established between devices will become disconnected. However, such references can be immediately revoked since the application will search another suitable service in the environment to rebind the reference.

All these examples demonstrate that applications running on mobile networks need more semantic means to express the validity of leased references than merely timeouts. A leasing mechanism needs to incorporate context information in their semantics to provide developers more expressive language constructs to create and manage leased object references.

2.2 Adaptive Leasing: Changes on the state of a leased reference

In the previous section, we use renew-on-call leases which are automatically prolonged upon each method call. Although in those examples we assumed that the renewal time is the time interval specified when a leased reference is created, our leasing mechanism also allows developers to specify a renewal time. Renewing a lease means to change the state of the leased reference which is extended with a certain time interval.

Determining the proper renewal time is another issue to consider in any leasing mechanism. In particular, questions arise regarding how long this renewal time should be since it may depend on dynamic parameters of the system such as the number of clients. In order to abstract away as much as possible such low-level leasing management details, a leasing approach where the renewal time is dynamically adapted seems much more suitable. We observe that such adaptations are also based on context information. For example, objects exported in the context of a transaction, i.e., banking payment, could be renewed with a longer time interval in order to increase the level of robustness in the presence of transient failures so omnipresent in mobile ad hoc networks. On the other hand, leased references to remote objects that can be reconstructed with persistent data from a database could be renewed with a smaller time interval. Such examples illustrate that changes on the state of a leased reference performed by renewals may depend on context information.

In the context of service-discovery protocols, Bowers et al. have proposed self-adaptive algorithms for varying lease periods in response to the system size [1]. Although the authors specifically focus on restricting the lease time to guarantee minimum average responsiveness in a Jini system, this technique can be interpreted as another instance of how a low-level parameter in the computation context, i.e. responsiveness of the system, influences the frequency of renewal of leased object references. We argue in favour of a generalization of such adaptive techniques based on context information.

2.3 Distributed Garbage Collection Policies: Changes on the behaviour of a leased reference

We observe that changes in context not only affect the state of a leased reference as argued in the previous section, but also its complete behaviour, i.e, they introduce changes on the way how remote objects are collected. Recall the example of a user in a conference who wants to print one of their PDA files. The leased reference between the user and the printer located at conference eventually expired when the conference terminates. Consider now the same interaction at the user's home. In that case, the leased reference can be permanently kept since the user will return home eventually and print other files. This is a naive example but it illustrates that depending on the physical environment where devices are, different distributed garbage collection strategies can be applied. For example, a leased reference that never expires, i.e. a strong reference, can be applied if the user is at home or a leased reference for a concrete time interval if the interaction happens at a conference. We claim that there is no single strategy to reclaim objects in mobile ad hoc networks. However, a leasing mechanism should still provide means to adapt its behaviour in relation to changes on the context of the application.

3 Position Statement

Applications running on such mobile networks must adapt their behavior to different context events such as frequent disconnections of devices or location of mobile devices. Leasing provides a robust mechanism to reclaim remote objects in such sophisticated network topology. However, in current leasing approaches the validity of a leased reference is entirely based on timeouts. We have demonstrated by means of concrete examples that more expressiveness is required to determine the period of validity of a leased reference and how context information permeates to distributed memory management. As a result, we argue that leasing should to be augmented with context information: distribution memory management should be *aware* of the changes on the context of application to properly reclaim objects in mobile ad hoc networks. We thus propose *context-aware leasing* as a generalization of leasing which in response to context information provides automatic adaptation of the renewal time of leased references and dynamic adaptation of different garbage collection strategies.

We are currently implementing the extensions to leasing that we describe in this paper in AmbientTalk[2], a programming language especially designed for pervasive computing. We have already experimented with different combinations of the three types of leases supported and explored the integration of leasing with other language constructs such as futures. In future work, we want to explore a number of open issues such as how to deal with combinations of contexts events or performance considerations.

References

1. BOWERS, K., MILLS, K., AND ROSE, S. Self-adaptive leasing for jini. In *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications* (Washington, DC, USA, 2003), IEEE Computer Society, p. 539.
2. DEDECKER, J., VAN CUTSEM, T., MOSTINCKX, S., D'HONDT, T., AND DE MEUTER, W. Ambient-oriented Programming in Ambienttalk. In *Proceedings of the 20th European Conference on Object-oriented Programming (ECOOP)* (2006), vol. 4067, Springer, pp. 230–254.
3. GONZALEZ BOIX, E., VAN CUTSEM, T., DEDECKER, J., AND DE MEUTER, W. Language support for leasing in mobile ad hoc networks. Technical Report VUB-PROG-TR-07-08, Vrije Universiteit Brussel.
4. GRAY, C., AND CHERITON, D. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In *SOSP '89: Proceedings of the twelfth ACM symposium on Operating systems principles* (New York, NY, USA, 1989), ACM Press, pp. 202–210.
5. HIRSCHFELD, R., COSTANZA, P., AND NIERSTRASZ, O. Context-oriented programming. To appear in the Journal of Object Technology (2007), `http://www.jot.fm`.
6. MCLEAN, S., WILLIAMS, K., AND NAFTEL, J. *Microsoft .Net Remoting*. Microsoft Press, Redmond, WA, USA, 2002.
7. PREUVENEERS, D., VAN den BERGH, J., WAGELAAR, D., GEORGES, A., RIGOLE, P., CLERCKX, T., BERBERS, Y., CONINX, K., JONCKERS, V., AND DE BOSSCHERE, K. Towards an extensible context ontology for ambient intelligence. In *EUSAI* (2004), pp. 148–159.
8. SALBER, D., DEY, A. K., AND ABOWD, G. D. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 1999), ACM Press, pp. 434–441.
9. SUN MICROSYSTEMS. Java RMI specification, 1998. `http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html`.
10. WALDO, J. The Jini Architecture for Network-centric Computing. *Commun. ACM 42*, 7 (1999), 76–82.
11. WEISER, M. The computer for the twenty-first century. *SIGMOBILE Mob. Comput. Commun. Rev. 3*, 3 (september 1991), 94–100.