Semi-Automatic Garbage Collection for Mobile Networks

Elisa Gonzalez Boix, Tom Van Cutsem[†], Stijn Mostinckx[*],
Jessie Dedecker[†], Wolfgang De Meuter, and Theo D'Hondt
Programming Technology Lab
Vrije Universiteit Brussel
Pleinlaan 2 - 1050 Brussels - Belgium
{egonzale,tvcutsem,smostinc,jededeck,wdmeuter,tjdhondt}@vub.ac.be

ABSTRACT

Mobile networks pose new issues in the field of distributed garbage collection. Garbage collection must deal with volatile connections that may break remote object references unexpectedly for an unpredictable amount of time. As a result, no automatic distributed garbage collection satisfies the new hardware phenomena. A semantic-based approach called *semi-automatic garbage collection* is proposed as a new strategy for distributed garbage collection where the collector will be steered by the developer to decide whether remote objects can be reclaimed. We investigate how to transmit the needs of the application to the garbage collection process.

1. INTRODUCTION

In recent years remarkable progress has been made in the fields of mobile hardware and wireless network technologies. The widespread adoption of small and multi-purpose devices such as SmartPhones or GPS systems has provided users with mobile devices equipped with wireless connection media - e.g. Bluetooth or WIFI. Devices communicate by means of such wireless infrastructure with other devices in their environment in ad hoc way spontaneously creating networks. These *mobile networks* make applications deployed on the devices become smart applications which interact with their environment. However, developing applications for such devices is very complex due to the lack of support in current programming languages to deal with the specific properties that distinguish mobile networks from the traditional distributed systems. Our research is focused on providing programming language support to alleviate the complexity encountered when developing such applications. In previous work, we have identified the following phenomena intrinsic to mobile networks [4]:

Connection Volatility. Devices typically interact with their environment by means of wireless technology. Due to the limited communication range of these technologies combined with the fact that devices roam with their users, devices can move out of range unexpectedly. Although devices cannot assume reliable connections, applications should accomplish their tasks in presence of frequent disconnections.

Ambient Resources. As the user moves about, services offered by proximate devices may become available. At a software level, it is impractical to encode beforehand which services will be available on a certain location. On the contrary, a discovery mechanism is required to dynamically locate resources in the environment.

Autonomy. Devices must be able to provide services to devices in the environment. Every mobile device should act as an autonomous computing unit that cooperates with other devices without relying on a predetermined infastructure - i.e a fixed server - which may not be available when two devices meet in an ad hoc network.

The repercussions of these hardware phenomena on the design of programming languages have also been examined in previous work [4]. The *Ambient-Oriented Programming* paradigm was postulated as a new computing paradigm which incorporates these hardware phenomena at the heart of its programming model in order to ease the development of applications for mobile networks. In this paper we discuss the issues of distributed garbage collection for mobile networks and subsequently introduce a new family of distributed garbage collection mechanisms to cope with them.

2. GARBAGE COLLECTION IN MOBILE NETWORKS

This section illustrates the repercussions of the above hardware phenomena on the process of distributed garbage collection. In order to discuss the need for a novel distributed garbage collection approach, we first introduce an overview of existing distributed garbage collection techniques along with some necessary terminology.

2.1 Distributed Garbage Collection

The aim of garbage collection is to automatically reclaim objects in memory which will be no longer used. This problem has already been successfully tackled for centralized systems [7, 15] - this paper assumes some familiarity with it. Objects are considered garbage if they are not referenced directly or indirectly from a *root* object -i.e. a set of objects that are always considered non-garbage. Therefore, the problem of garbage collection comes down to identify and remove objects which are proved not to be reachable from

[*]Funded by a doctoral scholarship of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen), Belgium.
[†]Research Assistant of the Fund for Scientific Research Flanders, Belgium (F.W.O.)

the root set - i.e. garbage objects. In a distributed system, objects can also reference objects residing on other machines via *remote object references*. In such context, the object graph spans different machines and communication between them is necessary to decide when an object is unreachable.

Much research has been carried out on distributed garbage collection (DGC) [2, 11]. Most of the DGC algorithms can be classified in one of the two well-known families derived from centralized systems, namely *tracing* and *reference counting*. Tracing-based algorithms perform different passes to examine the whole object graph and identify garbage objects. In a distributed setting, these algorithms consider memory as being logically shared although it is physically distributed across different machines. Garbage collection runs in parallel in all the machines, but global synchronization is required at certain points. This assumption proves to be the weakness of such algorithms for mobile networks since it requires every device to be accessible to complete garbage collection which can no longer be assumed due to the volatile connections.

Distributed reference counting is also an extension to the centralized algorithm where every object tracks how many references are pointing to it by objects on other machines. Once the object is no externally referenced, it can be reclaimed by the local garbage collector. These approaches improve scalability since not all devices must cooperate to reclaim objects. In the past years much research has been focused to make these algorithms scalable [1, 10, 5, 3, 14] and complete [8, 6]. However, communication is still required between several devices to determine if a remote object can be reclaimed.

2.2 Terminology

We conceive mobile applications as suites of active objects deployed on autonomous devices. Each device is said to *host* a set of active objects which communicate with each other by means of asynchronous message passing. Two active objects can get acquainted via a built-in service discovery mechanism. This mechanism allows objects residing on different devices to get to know each other through an external description which denotes a *service*. These active objects can provide services - e.g. a printer offering a printing service to the proximity - or request services - e.g. a PDA text editor that needs to print a file. We call the active objects which provide a service to the environment *service providers*, and the ones which request a service *clients* of a service provider. Note that an active object can be both service provider and client of another service at once. Imagine the case of an instant messenger running on a mobile device that spontaneously discovers other "instant messenger" services appearing in its proximity to exchange text messages or files. Upon creation, each messenger will broadcast its presence in the environment by providing an instant messenger service whose description is based on a tag that identifies the user -e.g. their nickname. When the application establishes a conversation with another discovered "buddy", it also becomes a client of the buddy instant messenger service.

Once two objects get to know each other through the service discovery mechanism, a remote reference is allocated and they can transparently communicate with each other. Figure 1 illustrates a graphical representation of an allocated remote reference. Conceptually, an active object `a` can 'directly' refer to another active object `b` that resides on a different machine via a remote reference. This is represented via a dotted line in the figure 1. A remote reference is a unidirectional reference from a client, namely the *source* of the reference, to a remote service provider known as the *target* of the reference.

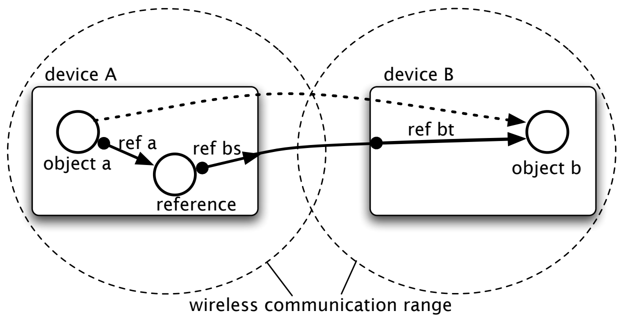

Figure 1: A remote reference

From an implementation point of view, a remote reference consists of an ensemble of object references as figure 1 also shows. First, on the client side, a remote reference actually points to a *reference* object which is the local representative of a remote service provider. The reference object acts as a communication channel between client and service provider which delivers messages sent to the service provider. When the application establishes a remote reference, the system will create a reference object and allocate the reference $\texttt{ref}_a$ from the object `a` -i.e. the actual client of the service provider - to the reference object. The reference object is then responsible to discover the service requested by the client. Upon discovery of a suitable service, the reference object will then be *bound* to the remote service provider via the reference $\texttt{ref}_{bs}$. To the client object, the reference object is thus nothing but a proxy object to the service provider. Note that the application does not know the concept of proxies at the programming language level. Secondly, on the service side, object `b` will be pointed to by a number of references $\texttt{ref}_{bt}$ which represent the clients referencing it. A reference $\texttt{ref}_{bt}$ can be considered as the target of $\texttt{ref}_{bs}$.

2.3 Problem Statement

Current DGC mechanisms determine the reachability of the remote objects using communication between the nodes involved in the interaction. Although latency may be indeterministic in a distributed system, some DGC approaches alleviate the problem by reducing communication overhead and coping with the partial order of the messages received [10, 5, 3, 14]. However, they still depend on network connectivity of the nodes to detect garbage objects. In mobile networks, the object graph is distributed across several devices which can be temporarily unavailable. Due to the volatile connection, it is not possible to foresee when a device will be inaccessible. Moreover, devices roam disappearing and joining again the network at different locations. A direct consequence is that remote object references will be frequently broken becoming *inaccessible object references* as shown in figure 2. Consider then figure 1 again where device `A` holds a remote reference to device `B`. When communication between the two devices breaks because they move out of communication range, device `B` no longer knows whether object `b` can be reclaimed. Note that the reclamation of the source of the remote reference - i.e. the reference $\texttt{ref}_{bs}$ - does not pose new problems since the reference object in the device `A` can be reclaimed by the local collector once it is no longer referenced by object `a`. On the contrary, the disconnection of two devices will keep the reference $\texttt{ref}_{bt}$ dangling while it should at some point in time be collected. To this end, device `B` requires communication with device `A` to find out if the reference is still in use. However, the system cannot determine if an inaccessible object reference is only temporarily lost and will become accessible again by a repaired connection - e.g. a buddy of a instant message application may disappear during a few seconds because they moved out of the WIFI earshot - or if it will no longer be accessible because the two devices never encounter again.

In short, the problem comes down to knowing when the communication will be restored in order to ascertain if the reference $\texttt{ref}_{bt}$ is still accessible. However, this is an application-dependent issue since applications can react differently to disconnections. In contrast to traditional distributed systems where node or network failures are considered as errors which break the remote references, inaccessible references should not immediately be considered as permanently broken. An application may wait for the connection to be repaired to resume its task. Other applications may continue their task with a substitute service available in the proximity. For example, a user that wants to print one of their PDA files will search the printing service in the proximity. Once a printing service is discovered, their PDA will establish a remote object reference to the service provider. Considering that the interaction occurs at the user's home, the PDA application could keep the remote reference to the service provider after the disconnection of the devices since there is a high chance that the user will come back eventually and request a printing service again. Considering the same type of interaction between a user attending a conference and a printer located at the conference building, the remote reference could be reclaimed after the termination of the conference once the user has left. These examples illustrates that there is information in the context - i.e. the semantics of the application and the role of the object reference in the network - necessary to decide whether an inaccessible object reference can be cleared. Therefore, the garbage collection process depends not only on the object graph but also on the context in which these objects are themselves.

As a result, DGC approaches based on tracing are no longer applicable for mobile networks since all the devices may not be connected to cooperate in the collection. However, distributed reference counting approaches no longer satisfy the new hardware phenomena because there is application-dependent information necessary to ascertain whether a remote object can be cleared. *Our position statement is that automatic transparent distributed garbage collection is irreconcilable with the hardware phenomena of mobile networks*. On the other hand, manual reclamation is not a desirable solution; it has proven to be complex and error-prone since it is difficult to keep track of how many references have been handed out to clients. However, an unbounded amount of inaccessible object references will be accumulated as the devices move about and they may be kept indefinitely because devices may never meet again. Despite being inaccessible, some references are conceptually garbage since they are of no concern to the application so that the remote object they point to could be reclaimed. Thus, the responsibility of garbage collection must be shared between the collector and the developer which has a semantic knowledge of the object graph and the way references are used. Developers should be able to install different garbage collection strategies on the remote references in order to help the collector to determine the reachability of the remote objects they point to. To this end, we propose a novel family of distributed garbage collection mechanisms called *semi-automatic garbage collection* based on the collaboration between the developer and the garbage collector which will be guided by the developer to ascertain whether remote objects can be reclaimed.

3. SEMI-AUTOMATIC GARBAGE COLLECTION

Semi-automatic garbage collection is a hybrid approach which relies on an underlying local garbage collector and proposes a non-transparent distributed garbage collection based on a reference counting scheme augmented with additional semantic information. The rationale behind this approach is to provide the developer with support to steer the garbage collection process and accommodate the requirements of the application. Annotations to the code could be used to describe garbage collection strategies based on the needs of the application and the contexts where references are used. Therefore, the annotation system is in essence a meta object protocol to express semantic information and give hints to the collector.

3.1 Remote object references as a two-party contract

In current distributed applications, the developer still has the responsibility of defining which objects are relevant to the application. Typically, this information is hand encoded as part of the application by means of maintaining a collection of useful objects from which the no longer relevant ones can be removed. When the removal is not conscientiously done, it is possible that conceptually useless objects cannot be reclaimed [9]. Hand encoding such decisions is no longer possible within the context of mobile networks because the information may not reach all parties due to the limited connectivity as a consequence of the volatile connections. In such a setting, when a remote reference is first allocated, both devices ought to establish a contract which describes under which circumstances the reference is meaningful in order to help the collector to ascertain if the remote object pointed to by the reference can be reclaimed (since devices may at some time not be able to communicate anymore). Since the application has a knowledge of both parties in the interaction and which behaviour it can expect of the other party, references could be annotated with this additional information and hence transmitted to the device of the object pointed to. Therefore, if one device becomes inaccessible due to a broken connection, the other device is aware of the conditions in which the inaccessible object references are still valuable.

As explained in section 2.2, remote references are unidirectional and hence service providers are unaware of the clients pointing to it and have no direct way to contact them. However, clients cannot unilaterally decide the circumstances which determine the importance of the remote reference. Instead, services providers should be able to reply to the intentionality of the client since there will be cases where the demand cannot be fulfilled. Reconsider the example of a printing service, although the user may be willing to use a printing service during two hours to print numerous documents, the printing service may be only available for a shorter time slot. The service should then be able to answer the client intention of establishing a two-hour reference with the actual time that the service will be available. If the conditions do not live up to the client's expectations, the client could search for a more suitable printing service. This example demonstrates that it does not suffice that one side of the interaction transmit its intention, garbage collection requires also service providers to know their clients in order to negotiate a collection policy for each remote reference pointed to it. Therefore, we talk about remote object references as *a two-party contract* between both objects involved in the interaction.

Since the contract must be acknowledged by both parties, a handshake is hence necessary during the binding process of a reference to properly determine the validity of the reference. In practice that means clients will exchange control messages with the device hosting the service to transmit the semantic information of the reference and eventually agree on a contract defining the garbage collection strategy applicable to the reference. Once the contract is signed, both devices are aware of the strategy to check in order to reclaim objects without requiring later communication with the other device.

A first analysis of the field reveals that two-party contracts already exist at the moment. A known example is *leased references* such as the ones provided in Jini [13] and Java RMI [12]. In such approaches, an object can obtain a lease on another object, known as lease *holder* and *grantor* respectively, for a certain period of time that is negotiated by the two objects when the access to the grantor object is first requested. Once the lease expires, the reference becomes invalid unless it was swiftly renewed by the lease holder. Although, both partners know indeed what they can expect of each other, these forms of leasing are insufficiently powerful for mobile networks. Rather than solving the problem, the developer still has to specify which time stamp is effectively necessary for an object or in other words, under which conditions the object can be cleared. This certainly implies that the code responsible for renewing the lease is interwoven with the functional part of the application. However, as applications employed on mobile networks become more complex, describing when a object can be reclaimed cannot continue being resolved in ad hoc way. Instead, more structured control over the garbage collection process should be provided to developers. To this end, leasing should be integrated at the language level. Note that in contrast to traditional leasing techniques, leased references that we are proposing never break when there is a connection between devices since the system is responsible for the renewal of the lease transparently to the client.

3.2 Referencing strategies

Since devices may not be able to communicate with each other at some point in time, we have argued that a two-party contract should be established to describe under which circumstances a reference is useful to the application. At programming language level, this implies that remote references should be tangible in order to allow the developer to change their behaviour in terms of garbage collection. We propose *referencing strategies* as the support given at language level to the developer to apply a collection strategy to the remote reference. A referencing strategy expresses thus the collection policy that will be applied to both source and target of the reference upon a disconnection once the remote reference becomes inaccessible. The grayish surface shown in figure 2 illustrates the graphical representation where referencing strategies are applied. To be precise, referencing strategies specify the contract that applies to both reference $\texttt{ref}_{bs}$ and $\texttt{ref}_{bt}$ which corresponds to the source and target of the conceptual remote reference, respectively. Based on the analysis of the referencing strategies of the remote references pointing to a service provider, the DGC will be able to ascertain the reachability of the remote object and thus decide under which conditions it can be collected.

We consider the support provided by referencing strategies for DGC in mobile networks to be analogous to what weak pointers provide at local garbage collection level. Weak pointers denote a reference to an object which cannot prevent the object referenced from being garbage collected. This means that if an object is only pointed

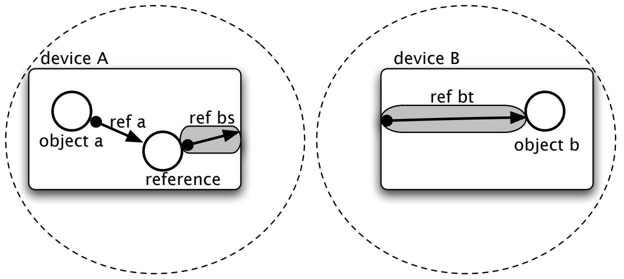

Figure 2: An inaccessible reference

by weak references then the garbage collector can actually reclaim it. Similarly to our referencing strategies, weak pointers provide a means to express the disposability of a reference to the local collector. However, while weak pointers apply a strict sense of the disposability (since the semantics transmitted to the collector refers only to a 'can-be-collected' property), we envisage different types of referencing strategies.

3.3 A tentative classification of referencing strategies

In order to come up with different types of referencing strategies that the annotation system should provide, we investigate the disposability of the remote references. The disposability of a reference is the condition that must be satisfied in order for the collector to clear the reference. We have considered the following aspects of the disposability of the remote references: how they react to disconnections and the kind of semantic information that they express.

Temporal Disposability

As already mentioned, devices interact with their surroundings to discover available services and if a suitable one is detected, a remote reference is installed to it. Applications can react differently whenever the device hosting the objects pointed to by the reference moves out of communication range. Temporal disposability is reminiscent of the leasing techniques since the disposability of the remote references after a disconnection is expressed based on a time period.

Weak References. Some applications are only interested in a particular service as long as the service is in the close proximity. This means that the reference will be rebound in case of disconnection and thus, the service provider pointed to by the reference can change over the time. At garbage collection level this implies that both source and target of the remote reference can be discarded after a disconnection since the reference is conceptually useless to the application and the target of the reference will be rebound to a new service provider. For example, imagine a futuristic application that interacts with devices embedded in the physical environment to visualize the map of a building as the user moves about to a certain place. The application will request the information to the nearest device to the user location providing the service. As the user moves physically to other buildings, the reference will be broken and the application will search another suitable information service to rebind the reference. To sum up, *weak references* are said to be disposable once a disconnection between the client and service provider is detected.

Temporal References. The disposability of these references is based on a time interval that specifies how long the reference will last upon a disconnection. After the time interval, the collector can reclaim the reference since it is no longer relevant to the application. Unlike the weak reference strategy, a temporal reference is disposable only if the disconnection outlasts the time interval. For instance consider another futuristic scenario where the attenders to a conference receive several electronic meal tickets which will be used by their PDAs to pay the included meals. Typically, these electronic meal tickets can be used only one day and thus, they are useless once the day has passed.

Strong References. This strategy denotes the references which are never disposable. This category is comparable to the traditional definition of a remote object reference and the only way to reclaim them will involve further communication between the devices. Upon a disconnection, they keep on referring to a service provider indefinitely unless both parties are connected and there is an explicit intervention to change the strategy or delete them. As a concrete example, recall the printing service again. Imagine that the client would like to send several documents to his home printer. Since it is desirable that all documents are printed by the same printer, a strong reference as follows expresses the application intentionality (presuming that `Printer` denotes the description of the service requested).

```
// client strategy for outbound references
aPrinter = discover PrinterBlaBla with strategy {strong};
foreach document in batch {
    aPrinter<-print(document);
}
```

The printing service will expect that the user will send more documents to print in the future. Therefore, strong references also express the default behaviour expected by the printer about the references to its printing service. Consider that a declaration of a service named `Printer` implies the broadcast of the availability of this service in the environment automatically.

```
// server strategy for incoming references
service Printer strategy {strong}{
   queue: new Queue();
   method print(doc) {
     (...)
     queue.add(doc);
   }
}
```

Domain-specific Disposability

The temporal disposability classification corresponds to the integration of leasing techniques into our collection strategies. Notice that leasing techniques are entirely based on a certain time interval: weak, temporal and strong references conceptually express a zero, a certain time and an infinite lease, respectively. However, in mobile networks part of the intention of the application depends on the *context* where objects are. Sometimes the developer would need to transmit the state of the environment to the collector since the geographic location of the devices may be relevant to reclaim certain remote objects. Recall the example of the printing service requested when the user was at home and at a conference. Depending on the physical environment where devices are themselves, different collection strategies may be agreed - i.e. a strong reference if the user is at home and a temporal one if the interaction happens at a conference. Furthermore, the state of the application can also determine different collection techniques. That is the case of transactional-oriented and event-based applications. As an example, consider a ubiquitous shopping application where products are tagged with information which can be accessible by customer devices. At some point in time, a customer device will establish a reference to an electronic payment service to pay the products in their shopping basket. Whether this remote reference can be removed depends on the sequence of messages exchanged between the devices rather than a concrete time interval. In the presence of a disconnection, the reference should be kept only if that happens at the precise moment in time when the order was being sent to the bank. Otherwise, the reference is disposable to the collector - e.g the disconnection of devices is detected while exchanging the user data or the sum to be paid. The pseudo-code below shows the implementation of this example. For simplicity, we assume that the client is interested in discovering an available payment service to pay for the products, but the kind of remote reference that is established is irrelevant to them. A `server` strategy as follows expresses the willingness of the client to accept whatever referencing strategy that the service provider offers.

```
aCashier = discover Payment with strategy {server};
aCashier<-processPayment(customerId);
```

The payment service uses a conditional disposable strategy to express that the reference can be cleared unless it has placed a bank order for the customer bill -i.e. state of the application is `TransmitOrder`. A conditional disposable strategy models the disposability of a reference with a boolean condition to express the circumstances under which the reference can be cleared. Consider that the **when** construct allows one to specify what code should be executed upon reception of the result of an asynchronous message without blocking the computation of the service provider.

```
service Payment strategy {
  disposable if( not(state.Equals("TransmitOrder"))){
   state: new State("Ready");
   method processPayment(customerId){
     (...)
     state.set("PrepareCheckOut");
     // compute the customer bill
     when(customer<-getAccountInfo(), {
       state.set("TransmitOrder");
       this.transmitBankOrder(clientAccount);
     });
   }
}
```

These examples illustrate that referencing strategies require more expressiveness than numerical timeouts. In short, strategies based on domain-specific information, such as the geographical location of the device or the state of a transaction, expresses the disposability of certain references where this intention cannot be captured with a time-based strategy such as leasing.

The disposability aspects identified have resulted in different classes of collection strategies which should be provided by the annotation system to transmit the needs of the application to the collector. These garbage collection strategies have been exhibit from our relatively restricted expertise in writing applications for mobile networks and the analysis of the hardware phenomena. One important goal of our research is to develop concrete examples using the semantic annotation system to assist DGC. We believe that the study of these examples will allow the identification of more garbage collection strategies and come up with a definition of the required language concepts to support them.

4. OPEN ISSUES AND FUTURE WORK

So far we have focused on the motivation and rationale behind semi-automatic garbage collection. What follows now is the discussion of a number of open issues which are not properly addressed yet by our approach.

- Since an object can be pointed to by different clients, the contract agreed for each object reference may be different. From the service provider point of view, this means that different kinds of strategies should be considered to conclude if a remote object reference can be cleared. A default strategy should be determined to resolve conflicts between the different kinds of collection strategies. For the moment we consider that the service provider has priority over the client during the reconciliation of a referencing strategy since its resources are the ones which have to be reclaimed.
- Renewal of policies must be carefully designed. In some cases it may be useful that the parties will be able to renegotiate the garbage collection strategy applied to a remote reference. Due to the connection volatility, extra attention should be paid to design a handshake protocol which ensures that the renewal information reaches both sides in the communication in order to avoid unsound practices such as reclaiming an object which is still in use.
- There are different restrictions to be considered to implement domain-specific strategies. Firstly, it is not the aim of a system which assists the garbage collection to be recursive; this would lead to allocate an unbounded quantity of memory. The quantity of domain-specific information specified should be thus limited so that it will always remain locally calculable. However, developers do not need to have a knowledge of the internal functioning of the garbage collector to express their custom strategies. Moreover, the annotation system must be also sufficiently expressive so that group of objects could be annotated at the same time and allow to make group annotations which are dependent of each other- i.e. the meaning of an annotation may also depend on the 'annotation context' in which the annotation is itself.

5. CONCLUSION

In this paper we have investigated how the hardware phenomena of mobile networks affect the distributed garbage collection process. Current DGC mechanisms implicitly assume network connectivity of nodes to determine the reachability of objects. We have argued that this assumption is no longer held in mobile networks since devices may not be able to cooperate at a certain moment in time to collect garbage due to the limited connectivity. Moreover, there is application-dependent information necessary to ascertain if a remote object can be reclaimed. We have claimed that automatic distributed garbage collection is incompatible with the characteristics of mobile networks. Mobile networks thus require a new generation of DGC schemes where the semantics of the application are considered so as to ascertain whether a remote object can be removed. We have subsequently proposed semi-automatic garbage collection where the collector is assisted by the developer who has semantic knowledge of the object graph. Devices will first negotiate and agree on a collection strategy for a remote reference in order to help the collector to ascertain whether a remote object can be recaimed. We have also identified different kinds of collection strategies that should be integrated to help the developer to transmit the needs of the application to the collector.

6. REFERENCES

[1] A. BIRELL, D. EVERS, G. N. S. O., AND WOBBER, E. Distributed garbage collection for network objects. Tech. Rep. 116, Digital Equipment Corp. Research Center, 1993.

[2] ABDULLAHI, S. E., AND RINGWOOD, G. A. Garbage collecting the internet: A survey of distributed garbage collection. In *ACM Computing Surveys* (1998), vol. 30, pp. 330–373.

[3] BEVAN, D. I. Distributed garbage collection using reference counting. In *Parlallel Architectures and Languages Europe* (1987), Springer-Verlag, pp. 176–187.

[4] 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), LNCS, Springer. To Appear.

[5] GOLDBERG, B. Generational reference counting: A reduced communication distributed storage reclamation scheme. In *Programming Languages Design and Implementation* (1989), A. SIGPLAN, Ed., vol. 24, pp. 313–321.

[6] JONES, R., AND LINS, R. Cyclic weighted reference counting without delay. In *Proceedings of Parlallel Architectures and Languages Europe* (1993), Springer-Verlang, pp. 712–715.

[7] JONES, R., AND LINS, R. *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*. Wiley, 1996.

[8] MAHESHWARI, U., AND LISKOV, B. Collecting cyclic distributed garbage by controlled migration. In *Proceedings of PODC'95 Principles of Distributed Computing* (1995).

[9] NYLUND, J. Memory leaks in java programs. Tech. Rep. 11, Java Report, 1999.

[10] PIQUER, J. M. Indirect reference counting: A distributed garbage collection algorithm. In *Proceedings of the Conference on Parallel Architectures and Languages Europe* (1991), vol. 505 of *LNCS*, Springer-Verlag.

[11] PLAINFOSSÉ, D., AND SHAPIRO, M. A survey of distributed garbage collection techniques. In *Proceedings of International Workshop on Memory Management* (1995).

[12] SUN MICROSYSTEMS. Java RMI specification, 1998. `http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html`.

[13] WALDO, J. The Jini Architecture for Network-centric Computing. *Commun. ACM 42*, 7 (1999), 76–82.

[14] WATSON, P., AND I.WATSON. An efficient garbage collection scheme for parallel computer architecture. In *Parlallel Architectures and Languages Europe* (1987), Springer-Verlang, pp. 432–443.

[15] WILSON, P. R. Uniprocessor garbage collection techniques. In *Proc. Int. Workshop on Memory Management* (Saint-Malo (France), 1992), no. 637, Springer-Verlag.