A Distributed Garbage Collector for NeXeme

Luc Moreau* **and David De Roure***

Abstract

The remote service request, a form of remote procedure call, and the global pointer, a global naming mechanism, are two features at the heart of Nexus, a library to build distributed systems. NeXeme is an extension of Scheme that fully integrates both concepts in a mostly-functional framework. This short paper describes the distributed garbage collector that we implemented in NeXeme.

Introduction

Scheme [7] is a mostly-functional language, i.e. it is a fully functional language, which also supports imperative notions like assignments and continuations, for efficiency and expressivity reasons. We believe that a distributed extension of such a language requires a mechanism to invoke functions remotely, so that distribution becomes part of the most fundamental operation of the language.

Nexus [1], a library for building distributed systems, has two salient features: a *remote service request* is a form of remote procedure call, and *global pointers* provide for global naming in a distributed environment. By offering a functionality close to remote function invocation, Nexus is a suitable building block for our distributed language. Furthermore, when designing a distributed version of Scheme, our concerns were portability and potential use of high-performance hardware or protocols (e.g. supercomputers, ATM, UDP). Nexus also addresses these concerns as it runs on a variety of platforms and protocols.

NeXeme integrates the Nexus approach, with its remote service requests and global pointers, in a mostly functional language. NeXeme is a novel approach of distribution in functional languages. It offers expressivity, development ease, and automatic memory management (via a distributed garbage collector). Not only does it provide very powerful abstractions to control distribution, but also it remains efficient. We believe that NeXeme is an excellent medium to implement other forms of parallelism like communication channels or futures [3,4]. It is also an ideal platform to develop distributed symbolic applications, based for instance on distributed mobile agents.

In this paper, we describe the distributed garbage collector that we implemented in NeXeme. An extended version of the paper describes the semantics of remote service requests and gives details about NeXeme[5].

The Nexus Architecture

Nexus [1] is structured in terms of five basic abstractions: nodes, contexts, threads, global pointers, and remote service requests. A computation executes on a set of *nodes* and consists of a set of *threads*, each executing in an address space called a *context*. (For the purposes of this article, it suffices to assume that a context is equivalent to a process.) An individual thread executes a sequential program, which may read and write data shared with other threads executing in the same context.

The *global pointer* (GP) provides a global name space for objects, while the *remote service request* (RSR) is used to initiate communication and invoke remote computation. A GP represents a communication endpoint: that is, it specifies a destination to which a communication operation can be directed by an RSR. GPs can be created dynamically; once created, a GP can be communicated between nodes by including it in an RSR. A GP can be thought of as a capability granting rights to operate on the associated endpoint.

Practically, an RSR is specified by providing a global pointer, a handler identifier, and a data buffer, in which data are serialised. Issuing an RSR causes the data buffer to be transfered to the context designated by the global pointer, after which the routine specified by the handler is executed, potentially in a new thread of control. Both the data buffer and pointed specific data are available to the RSR handler.

Distributed Garbage Collection

NeXeme has a distributed garbage collector which takes care of memory management automatically. Our working hypotheses, provided by Nexus threaded handlers, are a reliable message-passing and a FIFO ordering of messages between two sites.

Each site relies on a thread safe, conservative, mark and sweep garbage collector [2]; conservativeness is required as Scheme data are passed to Nexus, written in C, and are pointed by Nexus data structures. In addition, NeXeme maintains two tables for each site. The *exit table* associates each global pointer with the number of distinct remote copies of this pointer originating from the site. The *entry table* contains all global pointers received by a site, except those that point at itself. The exit table, but not the entry table, is a root of the local garbage collector. The role of the distributed collector it to update counters in a safe and consistent way. To this end, it relies on two types of *control* messages, called "$decrement(gp)$" and "$increment\text{-}decrement(gp, s)$", as described below.

Reference counters are updated according to the *diffusion tree* [6] of global pointers. The first time a global pointer gp is serialised, an entry is added in the exit table of the current site with a counter set to 1; afterwards, for every serialisation, this counter is incremented. Symmetrically, the first time a global pointer gp is deserialised, it is added to the entry table (if it points at a remote host); if it is already present, NeXeme sends a decrement message for this global pointer, "$decrement(gp)$" to the site that sent the remote service request. When a site receives a message "$decrement(gp)$", the counter of gp in the exit table is decremented. Once the counter reaches zero, the global pointer may be removed from the table; only then, the pointer itself and the local data may be reclaimed, if no longer accessible.

In Nexus, a global pointer contains the site it is pointing at, and not the site it is arriving from. Therefore, once a global pointer gp pointing at a site s_1 becomes inaccessible on a site s_2, s_2 sends a message "$decrement(gp)$" to s_1. This naive implementation of the reference counter technique is sound if causality is preserved in the system [6]. This is ensured

*This research was supported in part by EPSRC grant GR/K30773. Authors' address: Department of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ. United Kingdom. E-mail: (L.Moreau,dder)@ecs.soton.ac.uk.

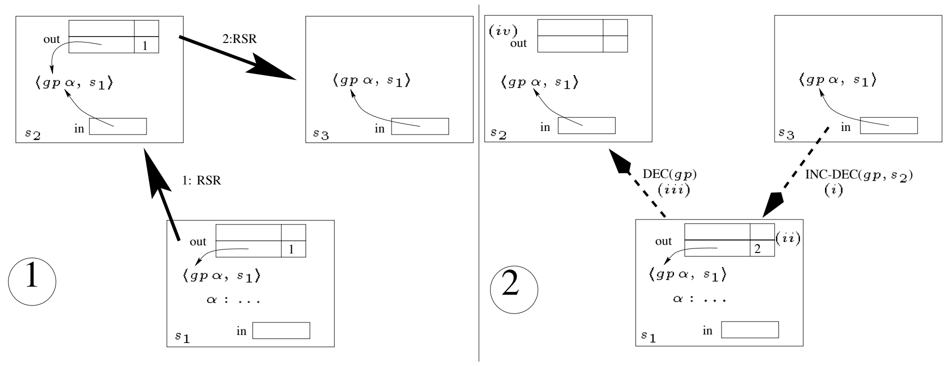

Figure 1: Distributed Reference Counters (1) after Remote Service Requests (2) after Control Messages

by a reorganisation of the diffusion tree, as explained in the following scenario illustrated in Figure 1.

Let us consider that gp, a global pointer pointing to address α in s_1, is migrated to s_2, and then migrated to s_3. Figure 1 part 1 shows that the reference counters in s_1 and s_2 are equal to 1, meaning that there are $1+1=2$ active remote references of gp. Once s_3 deserialises gp, the global pointer received from s_2, a reorganisation can be initiated, as illustrated in Figure 1 part 2. *(i)* Site s_3 sends an "$increment\text{-}decrement(gp, s_2)$" message to s_1; *(ii)* when the message is received by s_1, the counter for gp is incremented on s_1; *(iii)* afterwards, a "$decrement(gp)$" message is sent to s_2; *(iv)* when the message is received, the counter for gp is decremented on s_2.

Race conditions are avoided between s_1 and s_3 by giving priority to "increment-decrement" over "decrement" messages.

The simplicity and portability of the solution is unfortunately counter-balanced by its inability to collect distributed cycles. Our approach differs from "Indirect Reference Counting" [6] because it can reclaim "*zombie*" pointers, i.e. gp can be freed on s_2 even though it remains active on s_3. Several optimisations are possible. First, control messages of the same type may be grouped in a single message. Second, an "$increment\text{-}decrement(gp, s_2)$" to be sent to s_1, followed by a "$decrement(gp)$" to the same destination, may be replaced by a "$decrement(gp)$" to s_2.

The entry table should be designed carefully. If a global pointer entered in an entry table remains accessible to the gc, it will never be collected, nor the object on the origin host. Therefore, entries of global pointers in an entry table should be *masked* so that their inaccessibility can be detected. Once such a global pointer becomes inaccessible, it must also be removed from the entry table. Inaccessibility is detected after a local collection by installing *finalizers* on global pointers. A finalizer is a procedure called by the gc on an object once it is detected to be inaccessible. In NeXeme, such finalizers remove global pointers from the entry table and prepare a "increment-decrement" message. The message itself cannot be sent at garbage-collection time because such an operation requires memory not necessarily available at that moment: instead, inaccessible global pointers are queued (without allocation) by finalizers, and only after the end of the garbage collection, messages are sent to their destination sites.

Conclusion

This paper presents the distributed garbage collector of NeXeme, a distributed dialect of Scheme, based on remote service requests and global pointers provided by the library for distribution Nexus. The functional interface to remote service requests of NeXeme is a perfect abstraction to build a Scheme with futures [4]; in addition, NeXeme is used for programming multimedia distributed applications.

References

1 Ian Foster, Carl Kesselman, and Steven Tuecke. The Nexus Approach to Integrating Multithreading and Communication. *Journal of Parallel and Distributed Computing*, 37:70–82, 1996.

2 H.-J.Boehm and M. Weiser. Garbage Collection in an Uncooperative Environment. *Software – Practice and Experience*, 18(9):807–820, 1988.

3 Robert H. Halstead, Jr. Parallel Symbolic Computing. *IEEE Computer*, pages 35–43, August 1986.

4 Luc Moreau. Correctness of a Distributed-Memory Model for Scheme. In *Second International Europar Conference (EURO-PAR'96)*, number 1123 in Lecture Notes in Computer Science, pages 615–624, Lyon, France, August 1996. Springer-Verlag.

5 Luc Moreau, David DeRoure, and Ian Foster. NeXeme: a Distributed Scheme Based on Nexus. Technical report, University of Southampton, 1997.

6 José M. Piquer. Indirect Distributed Garbage Collection: Handling Object Migration. *ACM Transactions on Programming Languages and Systems*, 18(5):615–647, September 1996.

7 Jonathan Rees and William Clinger, editors. Revised[4] Report on the Algorithmic Language Scheme. *Lisp Pointers*, 4(3):1–55, July-September 1991.