Graceful disconnection

Christian Queinnec
UPMC – LIP6
E-mail: Christian.Queinnec@lip6.fr

Luc Moreau
University of Southampton
E-mail: L.Moreau@ecs.soton.ac.uk

A distributed object system allows objects to be communicated from site to site disregarding their physical locations. Communicating objects often leaves a trail homing to the site that owns the original object. Short-cutting these trails reduces the number of "zombies" i.e., sites that are part of the trail but do not need the object for themselves. This paper proposes an algorithm that allows a site to disconnect gracefully that is, without global network synchronization. This algorithm focuses on the proper treatment of zombies.

1 The Intention

Distributed object systems offer, under various names, a notion of *remote pointer* that allows one object from one site to refer to any object of any other site. Although similar in intention to regular in-memory pointers, the nature of distributed computing confer remote pointers some peculiarities that must be coped with. First, the failure of a site may prevent a remote pointer to be dereferenced into the pointed object. Second, dereferencing a pointer is not an instantaneous operation – to read/write the content of or, – to send a message to a remote object, takes a variable time. Third, managing the memory of a distributed therefore multi-tasks application may only be achieved with garbage collection (GC).

When used by multiple simultaneous users, distributed systems allow sites to join or disconnect from the common distributed object space. The goal of this paper is to explain a technique allowing a graceful disconnection. By graceful, we mean that the technique requires the collaboration of some sites to ensure the safe disconnection of sites but does not require a global synchronization.

Most of the time, an object is owned i.e., managed by a single site, its *owner site*, most often its birth site. When an object is created and communicated to other sites, it leaves a "trail"[Piq91] leading to its owner site. This trail is made of the data required to manage these remote pointers, more precisely these are the entry/exit tables[LQP92] or, the stub/scion pairs[SDP92] or for the present paper, the send/receive tables[Mor98b]. For a given object, these trails form a diffusion tree, the root of which is the owner site. This diffusion tree is used for GC with the following rules: *(i)* when a leaf is recycled, the branch that leads to it is cut, *(ii)* when a node has no branches, it becomes a leaf, *(iii)* when the root becomes a leaf, the original object is no longer reachable from other sites.

While the diffusion tree is a sound idea, it creates the possibility of "zombies" where a site only serves to maintain the diffusion tree even if it has no use of the remote object itself. A zombie is defined as a node, which is not a leaf nor the root of the diffusion tree and which has no use of the object (except for GC purpose). Zombies may form long chains within diffusion trees whose fringe only is active. Short-cutting these trails to reduce zombies was first addressed by Plainfossé[SDP92] but really and elegantly solved by Moreau[Mor98b]. While removing zombies seems desirable, this is not always so since this augments the branching factor of owner sites (if every trail is shortcut all leaves are directly attached to their owner site) which may not have the necessary resources (number of TCP connections or memory space). Another problem is the presence of fire-walls that prevent short-cutting since they impose diffusion trees to pass through them. Slow connections are a third problem since short-cutting may augment their number and perturb caching policies. In all these cases, the diffusion "trees" are not simple "shrubs" and may contain zombies.

When a site wants to disconnect from the net of active sites, it has to migrate all the objects it owns (a migration protocol such as Moreau's[Mor98a] may be used) so that the other sites may still use them. It has to delete all the references to remote objects it no longer uses since it is disconnecting. Eventually it has to take care of all its zombies. This paper addresses this single point and proposes a solution that involves a limited number of sites.

The essence of the algorithm is quite simple, see Figure 1. When a site S wants to disconnect, it sends to the owners of its zombies a disconnect message and then immediately disconnects itself: that's all for it! Upon reception of a disconnect message, the owner sends a disconnected message to inform the parent of the disconnecting site S of that disconnection.

Meanwhile, the children sites of S are orphan and their diffusion trees are no longer valid since they pass through the disconnected site. In order to readjust their diffusion trees, the children sites send reconnect messages to the owner which then become their new parent. The initial disconnect message told how many reconnect messages the owner will receive.

An object becomes useless when its diffusion tree is reduced to the owner however there is an additional condition coming from disconnecting sites: disconnect messages must be balanced by reconnect messages. The next section will detail this algorithm more thoroughly.

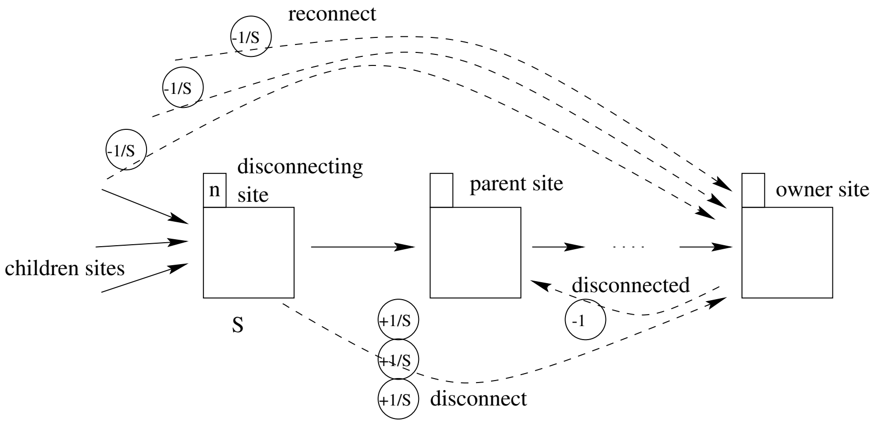

Figure 1: The disconnection of S

2 The Algorithm

This Section describes the algorithm with the conventions of Moreau[Mor98a] i.e., as transitions of a state machine. With this terminology, inherited from Nexus[FKT96], reference to objects are named *global pointers* aka GP.

All sites have a *send table* where the count of copies sent to other sites are kept. There is also a *receive table* where the GPs received from other sites are kept. A GP may occur at most once in a receive table.

Every object is owned by a site. Every object that appears in the send table of its owner site is associated to a vector (indexed by sites) of *disconnection counters*. Disconnection counters hold positive or negative integers and are essential for our algorithm since they hold the balance between disconnect and reconnect messages.

Messages from one site to another site are ordered and processed in the order of their emission.

Following Moreau's formalism[Mor98a] the distributed object system is characterized, with respect to the GC, by a configuration made of the send tables, the receive tables, the diffusion trees, the disconnected sites, the disconnection counters and the queues of in-transit messages: see Fig. 2 for the precise representations.

Diffusion trees are created or extended when the reference of an object (a GP) is copied from one site to another. This copying action and its consequences on the configuration are represented by the two transitions make_copy and receive_copy, see Figure 3. make_copy sends a copy message while receive_copy receives it.

When a reference becomes useless on a site i.e., when a local GC recycles a GP, the delete and receive_dec transitions describe the evolution of the configuration, see Figure 3. The delete transition sends a dec message to the parent site which receives it with receive_dec. This, in turn, may lead this parent site to also recycle the GP.

When a site s wants to disconnect, it has to migrate the objects it owns on another living site then it recycles the GP it holds as leaves. These aspects are not described here, we concentrate on zombies i.e., GPs unneeded by s whose diffusion tree pass through s. For all these GPs, see the disconnect transition of Figure 4, s sends a disconnect message to the owner of the GP telling *(i)* how many reconnect messages the owner should expect (this is the current counter of the send table of s concerning this GP), *(ii)* who was the parent of s (this is to help the owner to prepare its disconnected message, see below).

Receiving a disconnect message is described by the receive_disconnect transition of Figure 4. With respect to a given GP, when an owner site s_{owner}

$$
\begin{array}{rcll}
\mathcal{S} & = & \{s_0, s_1, \ldots, s_{n_s}\} & \text{(Set of Sites)} \\
\mathcal{G} & = & \{gp_0, gp_1, \ldots, gp_{n_g}\} & \text{(Set of Global Pointers)} \\
\mathcal{M} & = & \mathsf{copy} : \mathcal{G} \rightarrow \mathcal{M} \;\mid\; \mathsf{dec} : \mathcal{G} \rightarrow \mathcal{M} & \text{(Set of Messages)} \\
 & & \quad \mid\; \mathsf{disconnect} : \mathcal{G} \times \mathcal{S} \times \mathbb{N} \rightarrow \mathcal{M} & \\
 & & \quad \mid\; \mathsf{disconnected} : \mathcal{G} \times \mathcal{S} \rightarrow \mathcal{M} & \\
 & & \quad \mid\; \mathsf{reconnect} : \mathcal{G} \times \mathcal{S} \rightarrow \mathcal{M} & \\
 & & \quad \mid\; \mathsf{reconnected} : \mathcal{G} \times \mathcal{S} \rightarrow \mathcal{M} & \\
\mathcal{K} & = & \mathcal{S} \times \mathcal{S} \rightarrow Queue(\mathcal{M}) & \text{(Set of Message Queues)} \\
\mathcal{ST} & = & \mathcal{S} \times \mathcal{G} \rightarrow \mathbb{N} & \text{(Set of Send Tables)} \\
\mathcal{RT} & = & \mathcal{S} \times \mathcal{G} \rightarrow Bool & \text{(Set of Receive Tables)} \\
\mathcal{DT} & = & \mathcal{S} \rightarrow Bool & \text{(Set of Disconnected Sites)} \\
\mathcal{PT} & = & \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{S} & \text{(Set of Diffusion Trees)} \\
\mathcal{DC} & = & \mathcal{G} \times \mathcal{S} \rightarrow \mathbf{Z} & \text{(Set of Disconnection Counters)}
\end{array}
$$

Characteristic variables:

$$
s \in \mathcal{S},\quad GP \in \mathcal{G},\quad m \in \mathcal{M},\quad k \in \mathcal{K},\quad send\_T \in \mathcal{ST},\quad rec\_T \in \mathcal{RT},
$$
$$
parent \in \mathcal{PT},\quad disconnectp \in \mathcal{DT},\quad dc \in \mathcal{DC}.
$$

Figure 2: Configurations

$\mathsf{make\_copy}(s_1, s_2, GP)$:

$$
\begin{array}{l}
\;\; s_1 \neq s_2 \;\wedge\; rec\_T(s_1, GP) \;\wedge\; \neg disconnectp(s_1) \\
\;\; \rightarrow \;\{ \quad send\_T(s_1, GP) := send\_T(s_1, GP) + 1 \\
\qquad\quad\; post(s_1, s_2, \mathsf{copy}(GP)) \quad \}
\end{array}
$$

$\mathsf{receive\_copy}(s_1, s_2, GP)$:

$$
\begin{array}{l}
\;\; first(k(s_1, s_2)) = \mathsf{copy}(GP) \;\wedge\; \neg disconnectp(s_2) \\
\;\; \rightarrow \;\{ \quad receive(s_1, s_2) \\
\qquad\quad\; \text{if } rec\_T(s_2, GP) \;\; \text{then} \\
\qquad\qquad \{ \; post(s_2, s_1, \mathsf{dec}(GP)) \; \} \\
\qquad\quad\; \text{else} \\
\qquad\qquad \{ rec\_T(s_2, GP) := true \\
\qquad\qquad\; parent(S_2, GP) := s_1 \\
\qquad\qquad\; send\_T(s_2, GP) := 0 \; \} \quad \}
\end{array}
$$

$\mathsf{delete}(s, GP)$:

$$
\begin{array}{l}
\;\; send\_T(s, GP) = 0 \;\wedge\; rec\_T(s, GP) \\
\wedge\; owner(GP) \neq s \;\wedge\; \neg disconnectp(s) \\
\;\; \rightarrow \;\{ \quad rec\_T(s, GP) := false \\
\qquad\quad\; post(s, parent(s, GP), \mathsf{dec}(GP)) \quad \}
\end{array}
$$

$\mathsf{receive\_dec}(s_1, s_2, GP)$:

$$
\begin{array}{l}
\;\; first(k(s_1, s_2)) = \mathsf{dec}(GP) \;\wedge\; \neg disconnectp(s_2) \\
\;\; \rightarrow \;\{ \quad receive(s_1, s_2) \\
\qquad\quad\; send\_T(s_2, GP) := send\_T(s_2, GP) - 1 \quad \}
\end{array}
$$

Figure 3: Regular transitions

$\mathsf{disconnect}(s, GP)$:

$$s \neq owner(GP) \;\wedge\; \neg disconnectp(s)$$
$$\to \{ \quad disconnectp(s) = true$$
$$post(s, owner(GP), \mathsf{disconnect}(GP, parent(s, GP), send\_T(s, GP))) \quad \}$$

$\mathsf{receive\_disconnect}(s_1, s_2, GP, s_3, n)$:

$$first(k(s_1, s_2)) = \mathsf{disconnect}(GP, s_3, n) \quad \wedge \quad s_2 = owner(GP)$$
$$\to \{ \quad receive(s_1, s_2)$$
$$dc(GP, s_3) := dc(GP, s_3) + n$$
$$post(s_2, s_3, \mathsf{disconnected}(GP, s_1))\}$$

$\mathsf{receive\_disconnected}(s_1, s_2, GP, s_3)$:

$$first(k(s_1, s_2)) = \mathsf{disconnected}(GP, s) \quad \wedge \quad s_1 = owner(GP)$$
$$\to \{ \quad receive(s_1, s_2)$$
$$send\_T(s_2, GP) := send\_T(s_2, GP) - 1 \quad \}$$

Figure 4: Disconnection related transitions

$$
\begin{array}{l}
\mathsf{reconnect}(s_1, s_2, GP) : \\
\quad s_1 \neq owner(GP) \ \wedge \ parent(s_1, GP) = s_2 \ \wedge s_2 \neq owner(GP) \\
\quad \rightarrow \ \{ \quad post(s_1, owner(GP), \mathsf{reconnect}(GP, s_2)) \\
\qquad\qquad parent(s_1, GP) = owner(GP) \ \ \}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{receive\_reconnect}(s_1, s_2, GP, s_3) : \\
\quad first(k(s_1, s_2)) = \mathsf{reconnect}(GP, s_3) \ \wedge \ s_2 = owner(GP) \\
\quad \rightarrow \ \{ \quad receive(s_1, s_2) \\
\qquad\qquad dc(GP, s_3) := dc(GP, s_3) - 1 \\
\qquad\qquad send\_T(s_2, GP) := send\_T(s_2, GP) + 1 \ \ \}
\end{array}
$$

Figure 5: Reconnection related transitions

receives a disconnect message, it increments the disconnection counter related to the disconnecting site s with the former content of the send table of s: this number represents how many orphans s creates. Then, s_{owner} sends a disconnected message to s_{parent}, the parent of s whose identity appears in the disconnect message, to inform s_{parent} of the disconnection of s.

At the reception of the disconnected message, the parent of the now disconnected site s adjusts its send table by removing s from its children. The previous behavior makes sense in a reference-listing GC, here, in the case of a reference-counting GC, the receive_disconnected transition (see Figure 4) shows that we only decrement the send table.

The disconnect and disconnected messages are somewhat similar to the inc_dec and dec messages of Moreau's protocol[Mor98a]. However the disconnect conveys an additional information that will be used when handling reconnect messages.

The second part of our protocol is concerned with the orphans that have become orphans. Their diffusion trees are incorrect since they pass through the now disconnected site s. In this situation and since they know the owners of their GPs, they trigger the reconnect transition whose goal is to graft their diffusion trees to the owner, see Figure 5.

With respect to a given GP, when a site s_{child} discovers that its parent site s is disconnected, it sends a reconnect message to the site that owns the GP in order to make this site the new parent site of s_{child}. The reconnect message tells that s, the parent of s_{child}, is disconnected so the owner site may update the disconnection counter of s which now has one orphan less. No orphans (for that GP) means that the disconnection of s is now completely repaired.

The disconnect and reconnect messages are harmlessly concurrent. Only one disconnect message may be emitted (since the emitting site is now disconnected for ever (if it ever comes back, it will have to bear another different identity). The effect of disconnect message is annihilated when the appropriate number of reconnect messages is received: at that time, the consequences of the disconnection are completely propagated since all the children of the now disconnected site are grafted onto the owner site. Meanwhile, the reconnect messages may be freely intermixed with or even arrive before the single disconnect message. The disconnection counter associated to s becomes null iff s is disconnected and all its children sites are reconnected.

3 The Variant

Disconnection counters may easily be compressed since it is only necessary to record non null values. Moreover non null values only appear while sites are disconnecting therefore the size of the disconnection counters is proportional to the number of currently disconnecting sites and not to the number of living sites. Once a disconnection counter reaches zero, its associated site is completely disconnected and all its orphans are re-rooted: the disconnection counter is no longer useful and may be recycled. In Scheme-ish parlance, disconnection counters may be implemented by a kind of association list.

4 The Related Work

Piquer and Visconti[PV98] present indirect reference listing a variant of indirect reference counting that maintains the list of sites where pointers are copied in addition to the number of their copies. Using this information, they present the skeleton of a protocol to shutdown hosts. In essence, the node that wishes to disconnect communicates the list of its children to its parent p, which in turn informs each child that its new parent is p. Their algorithm maintains the structure of the diffusion tree, but it forces communications between the children and the parent of the disconnecting node. On the other hand, our algorithm does not require reference listing and is more lazy, because only the disconnecting node and the owner are initially required to communicate; children are involved in the disconnection protocol only when they have to communicate with their parent (when sending a dec message for instance).

If communicating with the owner turns up to be too centralized, one can adopt a hierarchical reorganization[Mor98b]. Sites are grouped by domains, organized in a hierarchical manner. In each domain, a site is assigned the role of a *gateway*: a gateway acts as a representative of the outside world within a domain, and symmetrically it represents the domain in the outside world. Therefore, instead of sending the disconnect message to the owner, it would be sufficient to send such a message to the local gateway.

5 The Conclusions

We believe that there are two main results in this paper.

The first one is technical: this is the idea of disconnection counters, a kind of vector-like reference counter. Moreover, numbers may be negative to balance not yet received messages.

The second result is the concept of graceful disconnection where a site wants to release its share of the distributed memory it was part of. We propose an algorithm that looks like Moreau's^{Mor98a} inc-dec protocol except that the disconnecting site (the short-cuttee) is the one that initiates the algorithm.

6 Acknowledgements

This research was partially funded by EPSRC grant GR/M84077 and Alliance 98096.

[FKT96] 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.

[LQP92] Bernard Lang, Christian Queinnec, and José Piquer. Garbage collecting the world. In *POPL '92 – Nineteenth Annual ACM symposium on Principles of Programming Languages*, pages 39–50, Albuquerque (New Mexico, USA), January 1992.

[Mor98a] Luc Moreau. A Distributed Garbage Collector with Diffusion Tree Reorganisation and Object Mobility. In *Proceedings of the Third International Conference of Functional Programming (ICFP'98)*, pages 204–215, September 1998.

[Mor98b] Luc Moreau. Hierarchical Distributed Reference Counting. In *Proceedings of the First ACM SIGPLAN International Symposium on Memory Management (ISMM'98)*, pages 57–67, Vancouver, BC, Canada, October 1998.

[Piq91] José Miguel Piquer. Indirect reference counting: A distributed garbage collection algorithm. In *PARLE '91 – Parallel Architectures and Languages Europe*, pages 150–165. Lecture Notes in Computer Science 505, Springer-Verlag, June 1991.

[PV98] José M. Piquer and Ivana Visconti. Indirect reference listing: A robust distributed gc. In *Parallel Processing (EuroPar'98)*, number 1470 in Lecture Notes in Computer Science, pages 610–619, Southampton (UK), September 1998.

[SDP92] Marc Shapiro, Peter Dickman, and David Plainfossé. Robust, distributed references and acyclic garbage collection. In *Symp. on Principles of Distributed Computing*, pages 135–146, Vancouver (Canada), August 1992. ACM.