A Reactive Approach to Comprehensive Global Garbage Detection

by

Sylvain R.Y. Louboutin

ISBN: 1-58112-044-3

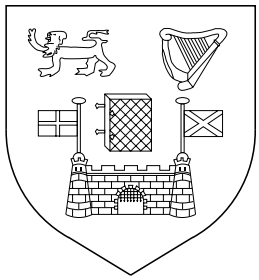
DISSERTATION.COM

1999

ISBN: 1-58112-044-3

Dissertation.com
1999

www.dissertation.com/library/1120443a.htm



A Reactive Approach to Comprehensive Global Garbage Detection

A thesis submitted to
University of Dublin, Trinity College
for the degree of
Doctor of Philosophy

Sylvain René Yves Louboutin
Distributed Systems Group
Department of Computer Science
Trinity College
Dublin 2
Ireland

http://www.dsg.cs.tcd.ie/

May 1997

Abstract

Automatic object management and, more specifically, *global garbage detection,* that makes it possible to automatically recycle the resources allocated to *garbage objects,* i.e., objects that are no longer useful, is a key system component that is necessary for the development of large scale distributed applications. *Comprehensive* global garbage detection in object–oriented distributed systems, i.e., global garbage detection that is intrinsically able to detect distributed cycles of garbage, has mostly been addressed via distributed versions of object graph tracing algorithms. The usual taxonomy of global garbage detection algorithms, which emphasises the "reference counting versus graph tracing" dichotomy, is a legacy of centralised garbage collection and fails to identify alternative approaches.

Distributed object graph tracing algorithms suffer from two flaws that jeopardise their scalability and robustness: the bottleneck associated with having to account for every live object in the system before any resource can actually be reclaimed and the overhead of *eager log–keeping.* As a consequence it is tempting to trade–off comprehensiveness for scalability and robustness under the assumptions that distributed cycles of garbage are rare and that all comprehensive algorithms are necessarily unscalable. In the absence of evidence to the contrary, this thesis contends instead that distributed cycles of garbage are as likely to occur as local cycles and show that a comprehensive alternative to object graph tracing global garbage detection is possible.

This thesis presents a new abstract model of the application processes' and local garbage collectors' computation, the combined effects of which on the global object graph fulfil the rôle of a *global mutator* from the global garbage detection perspective. This new model is based on identifying a subset of events of the global mutator's computation, called *log–keeping events,* that result in modifications to the inter–site paths in the global object graph, which constitute the edges of a *global root graph.* A log–keeping event reflects either the creation, or the destruction, of an edge incident to the object in the global root graph at which the event occurs, and two kinds of log–keeping events are identified: *edge–creation* events and *edge–destruction* events.

The causal history of a log–keeping event corresponds to the set of events responsible for the creation of all the paths ever created that are incident to the object at which the event occurs. The *path history* of this event is defined as a subset of its causal history containing only those events responsible for the creation of the *extant* paths to this object. Knowing the

path histories of log–keeping events makes it possible to identify garbage objects that are not identifiable by means of local garbage collection alone directly from the computation graph of the global mutator rather than from the analysis of its by–product, i.e., the object graph.

This abstract model of the global mutator's computation and the associated *lazy log–keeping* mechanism, which implements this model, constitute the basis of a novel global garbage detection algorithm. This algorithm entails computing *dependency vectors* that characterise the path histories of edge–destruction events by propagating increasingly accurate approximations of these vectors along the paths of the global root graph. In effect, this algorithm reacts to events of the global mutator that may result in the creation of garbage and makes it possible to identify garbage from those objects that are actually affected by these events, without requiring a complete scan of the whole object graph. The reactive nature of the new algorithm presented in this thesis, combined with a lazy log–keeping mechanism that does not require additional control messages to be exchanged by the application processes in support of global garbage detection, addresses both of the aforementioned stumbling blocks of traditional comprehensive global garbage detection algorithms.

Acknowledgements

My deepest gratitude certainly goes first to my parents who have patiently supported and encouraged a seemingly perpetual student, and who were definitely instrumental in the success of this endeavour.

I am certainly indebted to the people who have made it possible for me to join the Distributed Systems Group and who are Neville Harris, who led this group until 1996, my former supervisor Chris Horn, now successful CEO of Iona Technologies, as well as Brendan Tangney who introduced me to the group while I was still studying in UCD.

I am very grateful to Vinny Cahill, for his support and patience in supervising this thesis, especially considering the amount of additional work caused by my suboptimal command of the English language, and who did not seem to mind too much my somewhat unusual schedules.

I definitely appreciated the camaraderie of my fellow postgrads, and in particular Ciaran McHale, particularly in times of self–doubt, which are, I was told, part of the joys of being a postgrad.

During these past few years, I had the privilege of meeting a number of very special people who definitely made the whole experience far more bearable.

I certainly appreciated the long conversations I had with my friend Sue Fry, distinguished historian, about our respective experiences as overseas postgrads in Ireland, and her insights on related cultural issues.

I am particularly grateful to Agnes Lam who encouraged me till the bitter end, instilled the idea of seeking a better life in the Golden State, and was instrumental in getting me a job there.

At last but not least, I am very thankful to Kevin Houlihan, a good friend and incidentally my solicitor, who graciously kept away from me some unscrupulous opportunists so that I could focus on my thesis.

Contents

List of Figures

Chapter 1

Introduction

Automatic recycling of resources, or global garbage collection, is a key component of an object-oriented distributed system and is necessary for the development of genuine large scale distributed applications. However, distribution introduces new challenges and invalidates many of the assumptions and techniques that are a legacy of centralised (uniprocessor) garbage collection. For instance, distributed versions of incremental mark-and-sweep approaches have so far been the only means of performing comprehensive global garbage collection. As will be explained in this chapter, these approaches suffer from two inherent flaws, the *consensus bottleneck* and the overhead of *eager log-keeping,* that jeopardise their scalability. As a consequence, comprehensiveness has often been traded–off for scalability. A new comprehensive approach to global garbage detection, which tackles both of these issues, is therefore proposed in this thesis.

1.1 Object-Oriented Distributed Operating Systems

Modern distributed operating systems arise on one hand from the wider availability of more cost effective and powerful individual workstations, and, on the other hand, from a growing need to support applications that are inherently distributed.

1.1.1 Distributed Operating Systems

Computing power is no longer a scarce resource that must be shared among as many users as possible in order to maximise its utilisation. It has become more cost effective to provide individual users with autonomous computing power. The resulting need to provide common resources (e.g., file servers, printing facilities and processor pools) and services (e.g., system administration and automatic resource management) to a community of interconnected autonomous workstations has contributed to the development of distributed operating systems [10, 6, 4]. Moreover, many real life problem domains are physically distributed in nature (e.g., corporate information systems, environmental data and information systems [183], air

traffic control [86] and Computer Supported Cooperative Work (CSCW) [166]) and demand some support for large scale distributed applications.

1.1.2 Object Orientation

The object-oriented paradigm is particularly well suited to dealing with the complexity of distributed applications because of the high level of abstraction that it provides. The object model, which supports autonomous and self-contained entities interacting via well defined interfaces, is particularly well adapted to a distributed system composed of interconnected autonomous processors communicating via messages. Furthermore, an object-oriented approach makes it possible to support incremental development that adds to the flexibility of a distributed system.

The match between distributed operating systems and object-orientation is reflected in a number of research projects and the emergence of industrial standards [185, 184] for object based distributed application support environments. Object technology fosters greater integration and interoperability across heterogeneous platforms, and increases the applicability of distributed operating systems and the feasibility of large scale distributed applications.

Amadeus, as part of the Comandos project [5], has successfully demonstrated the feasibility of a distributed object–based platform featuring persistence in an heterogeneous environment. Amadeus made it possible to provide support for programming distributed persistent applications, written in a variety of object-oriented languages, thanks to its Generic Run–Time system [81]. Similarly, the CORBA standard, defined by the Object Management Group (OMG) [184], makes it possible to develop and integrate distributed applications across a wide range of heterogeneous platforms.

An object-oriented distributed operating system typically provides support for distribution in the form of global object naming and location services, remote invocation, object sharing, persistence, and a canonical object model facilitating interoperability between languages. Ideally, persistence and distribution are orthogonal and transparent to applications. I contend that automatic object management and, more specifically, automatic recycling of storage resources are crucial to the fulfillment of this goal.

1.2 The Need for Automatic Recycling of Storage Resources

The development of large scale and distributed applications is hampered by the lack of adequate system support for automatic resource management and, in particular, automatic recycling of storage resources, commonly known as garbage collection. Although distributed systems are likely to provide vast amounts of resources, e.g., large object spaces on both primary and secondary storage, which are considerably larger than what is available on centralised systems, these resources are nevertheless finite. By their very nature, physical and geographical distribution, and resilience to partial failure, distributed systems are also more

likely to be expected to provide continuous service availability, i.e., continuous consumption of these resources. Recycling of these resources when no longer used is therefore essential.

1.2.1 Supporting Distribution

Automated garbage collection has often been advertised as a means of obviating the burden and hazard of explicit resource management, i.e., as a lesser evil or expensive convenience, which could nevertheless be avoided altogether under appropriate circumstances. It may be argued that automated garbage collection contributes to enforcing the object-oriented paradigm by reducing interdependencies between the components of an application and hence preserving the encapsulation of the different abstractions. Automated garbage collection may nevertheless remain optional in the context of a centralised system where each thread of control independently manages its own private object graph, i.e., where the visibility, accessibility, and lifespan of objects does not extend beyond the scope of the thread that created them.

However, automated distributed — or global — garbage collection is undoubtedly a necessary and integral component of a distributed operating system featuring persistent and shared objects [125]. These objects may be shared by threads that cannot have a consistent and up-to-date view of the global object graph because this graph spans a number of disjoint address spaces scattered among autonomous processors. These autonomous processors communicate by exchanging messages that are delivered with unpredictable delays precluding any common frame of reference [127].

Additionally, these objects may outlive the thread(s) that created them, making it necessary to define their lifetime independently of the lifetime of these thread(s), e.g., objects may persist as long as they remain transitively reachable from some entry points, known as *roots,* in the system.

1.2.2 Supporting Transparent Persistence

As previously stated, persistence and distribution are ideally orthogonal and transparent to applications. However, this is not possible without automatic global garbage collection.

If persistence is transparent, all objects that are transitively reachable from a persistent object implicitly become persistent as well. Some objects can be made persistent explicitly and act as roots. However, when an object that has not been made persistent explicitly becomes known remotely by virtue of a reference denoting this object having crossed its site boundary, only a global garbage collection mechanism can determine whether or not it has become reachable from some remote persistent object and therefore implicitly become persistent as a result.

Distribution and persistence may be combined in either one of the following ways: Amadeus [5] abides by the principle that there should not be any hidden cost, like implicit persistence, in making an object globally available. An object is implicitly persistent only if it is locally

accessible from a persistent root. Alternatively, all global objects can be conservatively considered persistent. It is then up to the global garbage collection mechanism to determine which global objects should eventually be discarded [163].

The former approach requires that applications participate in explicit (and hazardous) resource management and potentially leads to the occurrence of dangling references and erroneous loss of data. Furthermore, it renders the semantics of persistence inconsistent: only objects that are transitively reachable via a local path from an explicitly persistent object residing on the same site, are implicitly persistent as well. As a consequence, distribution is no longer transparent to applications. On the other hand, the latter approach is only practical with effective automatic global garbage collection. In other words, automatic global garbage collection is necessary for genuinely transparent persistence.

Global garbage collection is therefore a key system support facility enabling the development of large and robust object-oriented distributed applications.

1.3 Escaping the Legacy of Centralised Garbage Collection

Although distribution can be made transparent to some extent, direct adaptation of centralised garbage collection algorithms could lead to unacceptable overhead. Distribution introduces additional costs resulting from unpredictable and unbounded delays in the delivery of messages across site boundaries and a potentially much larger object space. However, it also provides more available resources (that can be dynamically extended) and makes contention for these resources less critical than in centralised systems that lack dynamic extensibility.

1.3.1 Trade–Offs

In a system where communication across site boundaries is significantly more expensive than communication within a site, the message complexity of a distributed algorithm, i.e., the number of messages exchanged across site boundaries, imposes a heavier penalty than its space or computation overhead. The message complexity of a global garbage collection algorithm is therefore more likely to be the determining factor on the overall algorithm performance. As a result, different tradeoffs ought to be made. For instance, longer latency in the detection of garbage objects should be more easily tolerated than it would be in a centralised garbage collection algorithm, as long as this latency remains finite. Similarly, additional space overheads should also be less critical to a global garbage collection algorithm as long as these overheads remain bounded.

1.3.2 Taxonomy

The usual taxonomy of global garbage collection algorithms in object-oriented distributed systems, which emphasises the "reference counting versus graph tracing" dichotomy of centralised garbage collection algorithms, fails to identify alternative approaches which would not

be applicable in a centralised environment. Another legacy of centralised garbage collection techniques is that *comprehensive* global garbage collection, i.e., algorithms that are intrinsically able to detect distributed cycles of garbage, has mostly been addressed via distributed versions of mark-and-sweep graph-tracing algorithms.

The taxonomy introduced in this thesis distinguishes between *proactive* and *reactive* approaches. The former proceed under their own initiative and are not directly subservient to the mutator computation. Proactive global garbage collection is launched periodically, whenever an acute lack of resources is detected, or whenever else is convenient. A proactive algorithm iteratively re–evaluates, i.e., scans, the whole object graph whenever garbage must be collected.

A proactive garbage collection algorithm entails scanning all live objects to identify garbage by default, i.e., as those objects that are not alive, and subsequently scanning all garbage objects thus identified in order to reclaim their resource, therefore scanning the whole object graph in successive iterations. By nature, it therefore requires every site in the system to eventually participate in each of its iterations. The costs of collecting garbage using such a proactive algorithm grows at least as fast as the number of sites in the system. In contrast, a *reactive* algorithm, as will later be explained, does not require a global re–evaluation of the overall object graph and makes it possible to identify garbage on the basis of less information (it does not require all the sites in the system to eventually participate in the identification of garbage).

1.3.3 Garbage Collection Versus Garbage Detection

The distinction between various algorithms, for instance various forms of graph tracing algorithms such as mark-and-sweep *per se* or copying for instance, usually made in the context of centralised garbage collection [150], does not apply to global garbage collection algorithms, because the distinction between these algorithms really lies in how garbage is actually *collected.* The term *mark-and-sweep* is used to emphasise the iterative nature of the approach, i.e., that collection can only occur once a mark phase is completed, even though there might not be any actual "sweeping" performed by the global garbage collection algorithm.

More generally, the problem with global garbage collection really concerns global garbage *detection*, as garbage is actually collected, i.e., the corresponding resource reclaimed, by the local garbage collector at the site where it is located. *Global Garbage Detection* (GGD) is therefore a more appropriate term used henceforth throughout the remainder of this thesis to describe its actual focus. This thesis is not about global garbage collection *per se* but about global garbage detection, i.e., it does not address issues of either resources allocation or reclamation.

1.4 Stumbling Blocks of Proactive Global Garbage Detection

Two phases can usually be identified in proactive global garbage detection schemes. The first phase involves detecting live objects, while the second phase makes sure that the first phase is complete, i.e., is concerned with termination detection.

1.4.1 Detecting Live Objects

Live objects may be either detected directly by colouring the object graph *in situ* [72, 177, 106, 11] or indirectly from some logs maintained by the mutator processes in support of global garbage detection. Log-keeping is usually performed incrementally by the mutator processes in support of global garbage detection and entails keeping track of references that cross site boundaries. Log–keeping may require additional control messages to be exchanged in which case it is referred to as an *eager log–keeping* mechanism (see Chapter 4). The contents of these logs may be used to reconstruct consistent representations of the overall object graph that can be traced locally, either by a conceptually centralised service [120, 115] or by each site participating in global garbage detection [163, 100].

Once it has been determined that all live objects have been accounted for, or that enough information has been collected to reconstruct a consistent representation of the object graph, garbage objects can then be safely identified and their resources eventually reclaimed. Termination detection is often a distinct phase [177, 106, 72, 163, 100], although Tel has shown how the two phases of proactive global garbage detection can be superimposed [11, pp.193–226] albeit at the expense of a loss of concurrency between local garbage collectors. Using a conceptually centralised log-keeping service makes it possible to reconstruct a consistent representation of the overall object graph and obviates an explicit termination detection phase [120, 115].

To increase concurrency, multiple global garbage detection iterations may overlap and proceed concurrently. This can be achieved to some extent via a four colour marking scheme [135], making it possible to have up to three interleaved global garbage detection iterations proceeding concurrently. Approaches using a "cycle number" [177] or time stamps [106] make it possible to interleave any (bounded) number of iterations. However, as pointed out by Juul [177, pp.60], the benefit of this approach is limited. There is very little benefit in initiating subsequent overlapping marking phases without actually collecting the garbage already detected, as these subsequent marking phases would keep identifying the same garbage objects. Once an object has become garbage, it remains garbage, and need be detected only once, making these overlapping phases somewhat redundant.

Proactive global garbage detection does not prevent local autonomous per-site garbage collection from proceeding asynchronously, as local garbage collectors can be made to cooperate on a pair-wise basis, and global garbage detection iterations can be interleaved. However, garbage can only be detected once global garbage detection has accounted for all live objects. Proactive algorithms are therefore better described as *live object detectors* rather than

genuine *garbage detectors* because garbage is characterised as being everything that is not alive. Proactive global garbage detection algorithms require that all sites in the system reach some consensus and eventually participate in completing any given global garbage detection iteration.

Proactive global garbage detection makes it possible to guarantee bounded detection latency, in the sense that garbage is detected in at most, but not sooner than, one global garbage detection iteration. However, as the system grows (in the number of sites), the actual detection latency increases and so does the likelihood of indefinite delays due to partial failures. Proactive approaches are therefore *a priori* inherently unscalable and lack resilience to system failures.

1.4.2 Scalability

Palliative measures have been devised to circumvent the inherent lack of scalability of proactive approaches, however, commonly accepted assumptions, which are the basis of heuristics used by some garbage collection techniques, e.g., empirical observations of object life–cycles, have not been shown satisfactorily to apply to large scale distributed system of objects. For instance, Butler [80] argues that generational schemes, based on the empirical observation that "objects that die, die young" [155, 159] could not be reliably extrapolated to large scale distributed databases.

Baker [74] later demonstrated that a simple application that mimics the natural decay of radioactive isotopes (and in which garbage objects are continuously replaced), may exhibit a high mortality rate among new objects but nevertheless defeat a generational garbage collection approach because the probability of any of its live objects dying remains constant over time. The complete object lifetime probability density function — and not just the infant mortality rate — must be known if any gain is to be obtained from a garbage collection strategy based on the lifetime of objects. There is however no data supporting the hypothesis that the life cycle of the objects in a large scale distributed object store can be extrapolated from that of a centralised system.

Other strategies that rely on some *a priori* assumption regarding the usage pattern of objects, e.g., that lesser used objects are less likely to be used in the future, have not necessarily been shown to apply to large scale distributed object stores either. Such "aging" strategies progressively relegate unused objects to deeper layers of a storage hierarchy; the deeper in this hierarchy, the cheaper the cost of storage, but, presumably, the higher the cost of retrieval. Moreover, aging puts no limit on the amount of information that would have to be ultimately kept and therefore does not address the issue of effectively recycling resources.

1.4.3 Robustness

In summary, two inherent characteristics of proactive global garbage detection approaches jeopardise their scalability: on one hand the bottleneck associated with having to reach a

global consensus before any resource can actually be reclaimed, i.e., the "consensus bottleneck." On the other hand, proactive global garbage detection must either rely on eager log-keeping — see Chapter 4 — in order to benefit from the flexibility and increased parallelism of autonomous per-site garbage collection or must use exhaustive *in situ* global graph tracing.

These characteristics also compromise the robustness of proactive global garbage detection algorithms. Eager log–keeping is not resilient to the loss of messages and the loss of data due to the transient failure of a site could cause an exhaustive *in situ* global graph–tracing based garbage collector to spread damage to sites that did not fail. In other words, such a collector may erroneously collect live objects that were only reachable from live objects lost as a result of the unrecoverable failure of a site, turning a partial failure into an unsalvageable global one. Moreover, no garbage may be detected unless such a failed site eventually recovers (with or without loss of data) because proactive global garbage detection requires all sites in the system to eventually participate in the detection of garbage.

1.5 Reactive Global Garbage Detection

Unlike proactive approaches, reactive global garbage detection algorithms react directly to some events of the mutator computation, i.e., the destruction of some reference. These approaches make it possible to identify a garbage object from information received from objects directly adjacent to it in the global object graph. Collection can therefore proceed without the need for all the sites in the system to participate. Thus, these approaches avoid the consensus bottleneck common to proactive global garbage detection. Reactive approaches are therefore more scalable.

1.5.1 Non–Comprehensive Approaches

Unfortunately, the most typical reactive global garbage detection algorithms are those based on variations of indirect or weighted reference counting [76, 149, 85, 169] or reference listing [186, 190] that are not intrinsically comprehensive, i.e., that are not able to detect cycles of garbage. Hybrid approaches and heuristics have been proposed as a partial remedy.

For instance, a non-comprehensive reactive global garbage detection algorithm such as weighted reference counting, which is assumed to be able to collect most of the garbage, may be used in conjunction with a complementary comprehensive proactive global garbage detection algorithm used parsimoniously to collect the residual garbage, thus alleviating the limitation of the main global garbage detection algorithm [169]. This relies on the underlying assumption that distributed cycles of objects, and thus distributed cycles of garbage, are rare.

1.5.2 Heuristics

Alternatively, a non-comprehensive global garbage detection algorithm can be combined with some heuristics aimed at collocating suspected cycles of objects so that they can be dealt with locally by some comprehensive garbage collection algorithm should they ever become garbage as suggested by Plainfossé [186] and initially proposed by Bishop [161] to deal with partitioned address spaces. Moving objects around may however turn out to be prohibitive, as it may involve moving potentially many objects, most of which may never become garbage. Furthermore, as pointed out by Dickman [169] this may clash with other object management goals such as load balancing.

Thrashing is another potential downside of these heuristics: objects may be continuously moved around without ever managing to actually collocate cycles of objects. To prevent thrashing from occurring, objects belonging to suspected cycles can be moved according to some preestablished site order, although this may potentially overload those sites at the end of the chain. An alternative entails "virtually" moving the boundaries of the sites instead of moving the objects themselves. Within these virtual sites, each spanning several physical nodes, comprehensive proactive global garbage detection techniques can be used. The consensus bottleneck would therefore be confined to each of these subsets of the physical nodes of the system.

One may wonder whether these virtual sites would remain small enough to warrant the complexity of managing them. Cyclic data structures may be distributed among any number of physical sites and this approach does not necessarily solve either the thrashing or overloading problems. In this context, thrashing would result in continuously moving virtual site boundaries and overloading in a virtual site boundary eventually encompassing the whole system.

However, these palliative measures are not satisfactory because they rely *a priori* on assumptions about the overall object graph, e.g., that distributed cycles are rare, which have not necessarily been validated in a large scale object graph, as is also the case for those assumptions about object life cycles mentioned in Section 1.4.2. Furthermore, these approaches implicitly assume — a legacy of centralised garbage collection techniques mentioned in Section 1.3.2 — that only proactive global garbage detection is intrinsically comprehensive.

1.6 An Intrinsically Comprehensive Reactive Approach

This thesis proposes a new abstract model of the application processes and local garbage collectors computation, the combined effects of which on the global object graph fulfil the rôle of a *global mutator* from the global garbage detection perspective, that identifies a subset of events of the global mutator computation, called *log–keeping events.* Log–keeping events result in modifications to the inter–site paths in the global object graph, which constitute the edges of a *global root graph.* A log–keeping event reflects either the creation, or the

destruction, of an edge incident to[1] the object in the global root graph at which the event occurs, and two kinds of log–keeping events are identified: *edge–creation* events and *edge–destruction* events.

The causal history of a log–keeping event corresponds to the set of events responsible for the creation of all the paths ever created that are incident to the object at which the event occurs. The *path history* of this event is defined as a subset of its causal history containing only those events responsible for the creation of the *extant* paths to this object. Knowing the path histories of log–keeping events makes it possible to identify garbage objects that are not identifiable by means of local garbage collection alone directly from the computation graph of the global mutator rather than from the analysis of its by–product, i.e., the object graph.

This abstract model of the global mutator computation and the associated *lazy log–keeping* mechanism, which implements this model, constitute the basis of a novel reactive global garbage detection algorithm, that is nevertheless intrinsically comprehensive. This algorithm entails computing *dependency vectors* that characterise the path histories of edge–destruction events by propagating increasingly accurate approximations of these vectors along the paths of the global root graph. In effect, this algorithm reacts to events of the global mutator that may result in the creation of garbage and makes it possible to identify garbage from those objects that are actually affected by these events, without requiring a complete scan of the whole object graph. The reactive nature of the new algorithm introduced in this thesis, combined with a lazy log–keeping mechanism that does not require additional control messages to be exchanged by the application processes in support of global garbage detection, addresses both of the aforementioned stumbling blocks of traditional comprehensive global garbage detection algorithms.

The approach introduced in this thesis does not guarantee a bounded detection latency. On one hand, the contention on available resources is not as critical as in a centralised system, and I contend that it justifies an unbounded detection latency as long as it remains finite. On the other hand, there is no lower bound on this latency either, because reactive approaches do not suffer from the consensus bottleneck. Furthermore, this thesis shows that computing the dependency vectors necessary to identify garbage is possible without the unbounded space overheads usually associated with dynamically reconstructing vector times of arbitrary events of distributed computations. This can be achieved with a scalable message complexity and space overhead.

Reconstructing the causal history of the events of some distributed computation, using known techniques [97], might require a rather large and unbounded local history of events to be maintained. Tel [12, pp.344] for instance, rules out using snapshot algorithms as a basis for global garbage detection because the space overhead can be potentially prohibitive. This thesis shows however that this can be achieved because log–keeping events are not events of

[1] If an object A holds a reference denoting an object B, i.e., if a directed edge exists from object A pointing to object B, this edge is said to be *incident to* object B, while the same edge is *incident from* object A. Similarly, object A is said to be *adjacent to* object B, while object B is *adjacent from* object A.

an arbitrary computation.

1.7 Contribution and Outline of this Thesis

The contribution of this thesis is threefold:

(i) It proposes a new taxonomy which presents existing global garbage detection algorithms in a new perspective that makes it possible to identify a new direction of research in the area.

(ii) More significantly, it describes a novel approach to global garbage detection, called the *Path Listing Algorithm,* that is reactive but, nevertheless, intrinsically comprehensive. A reactive global garbage detection algorithm reacts only to the actions of mutator processes that might result in the creation of garbage, and its message complexity depends only on the number of edges in the subgraphs that may have become garbage as the result of these actions.

A new abstract model of the mutator's computation is developed, based on those events of its computation that are relevant to global garbage detection, i.e., that result in modifications to the inter-site paths in the global object graph. The global garbage detection algorithm itself entails computing dependency vectors, i.e., a kind of vector times, characterising the path histories of these events.

(iii) Finally, it presents a lazy log–keeping mechanism that implements this abstract mutator model and interfaces this algorithm with Amadeus. The integration of a reactive global garbage detection algorithm with a lazy log–keeping mechanism makes it possible to tackle both of the aforementioned stumbling blocks of traditional comprehensive proactive global garbage detection algorithms and contributes to a genuinely robust and scalable approach to comprehensive global garbage detection.

The remainder of this thesis is organised as follows:

Chapter 2 establishes the *Background* for the study presented in this work and defines the scope and goals of this research. It leads from centralised garbage collection to decentralised global garbage detection, introducing the terminology commonly used in the garbage collection literature with a short overview of centralised garbage collection techniques. The system model under consideration, i.e., the basic assumptions about the underlying infrastructure and the mechanisms used to support the object system, is defined. Global garbage detection is then presented, showing how it can be decoupled from the mutator processes and local garbage collection via the abstraction of the global root graph.

Chapter 3 surveys *Related Work* conducted in the area of global garbage detection via a detailed overview of the most significant existing approaches to global garbage detection.

This survey follows a new taxonomy of global garbage detection that distinguishes between proactive and reactive algorithms.

Reactive global garbage detection algorithms are intrinsically more scalable than proactive global garbage detection algorithms because, unlike proactive algorithms, they do not require all sites in the system to eventually participate in the task of detecting garbage. However, most reactive algorithms are based on some form of reference counting and are not comprehensive. It is argued that this legacy of centralised garbage collection algorithms is an unfortunate coincidence that hindered the identification and development of alternative approaches. Furthermore, reactive, and thus scalable, global garbage detection algorithms can nevertheless be inherently comprehensive.

Chapter 4 *Characterisation of Garbage* describes the theoretical principles that are the essence of the novel approach presented in this thesis. It describes a new abstract model of the mutator processes computation, based on the occurrence of two types of *log–keeping events* that correspond to the creation or destruction of an edge in the global root graph and that does not depend on any assumption about the application support protocols being used.

The *causal histories* of these events characterise all the paths to the object, at which the event occurs, that have ever been created. I define the *path history* of a log-keeping event such that it is the subset of the causal history of that event containing only those events responsible for the creation of the *extant* paths to the object. The analysis of the path history of a log–keeping event makes it possible to characterise garbage and forms the basis for a new comprehensive global garbage detection algorithm presented in the next chapter.

Chapter 5 presents a new *Path Listing Algorithm.* It first introduces the two main log–keeping strategies, i.e., *eager* and *lazy* log–keeping that can be used to implement the abstraction of log–keeping events introduced in the previous chapter. Lazy log–keeping does not require control messages to be exchanged by the application processes in addition to their own messages. Therefore, lazy log–keeping obviates synchronous, or even FIFO, communications between application processes required by eager log–keeping strategies to ensure the consistency of the information that they maintain in support of global garbage detection, and therefore contributes to its scalability and robustness.

A novel approach to global garbage detection that constitutes a *Path Listing Algorithm* is then described. It entails computing the dependency vectors that characterise the path histories of the log–keeping events and, in particular, the path histories of edge–destruction events. It shows how this can be achieved by taking advantage of the non arbitrary nature of the global mutator computation.

Chapter 6 reconciles the description of a lazy log–keeping mechanism introduced in the pre-