

Thèse

présentée à

L'université Pierre & Marie Curie – Paris VI

en vue de l'obtention du titre de

Docteur de l'Université Paris VI

spécialité

Informatique

par

DAVID PLAINFOSSÉ

sujet

**Distributed Garbage Collection and
Referencing Management in the Soul Object
Support System**

Soutenue, le 20 Juin 1994, devant le jury composé de :

MM.	BERNARD LORHO	Président
	SACHA KRAKOWIAK	Rapporteurs
	CHRISTIAN QUEINNEC	
	BERNARD BERTHOMIEU	Examineurs
	MARC SHAPIRO	
	PATRICK VALDURIEZ	

Résumé

La programmation d'applications réparties est une tâche difficile. Le programmeur doit utiliser des mécanismes distincts pour désigner, invoquer ou recycler des objets localisés sur des espaces d'adressage disjoints. Ce manque de transparence nécessite la connaissance préalable de la localisation effective des objets afin d'utiliser le mécanisme adéquat. Nous proposons un mécanisme de désignation uniforme d'objets distants : les chaînes de paires souches-scion ou chaînes de PSS. Les chaînes de PSS permettent de désigner et d'invoquer des objets, potentiellement mobiles, en environnement réparti hétérogène et soumis aux pannes de machines. Les chaînes de PSS coopèrent étroitement avec un ramasse-miettes réparti pour assurer la récupération des objets inaccessibles (i.e., miettes).

Le système Soul met en œuvre les chaînes de PSS au dessus d'Unix en C++. Ce système permet désigner, d'invoquer et de recycler des objets C++ de grain fin et potentiellement mobiles. Ce système démontre l'utilité de notre mécanisme de désignation et son intégration dans un système réparti. Les mesures de performances réalisées démontrent que le surcoût introduit par les chaînes de PSS est négligeable au regard du service fourni.

Mots-clés : *Ramasse-miettes Réparti, Désignation d'Objets, Chaînes de PSS, Soul.*

Abstract

Programming distributed applications is still a tedious task. One key reason is that the transparency provided by common operating systems is very poor. Local and remote objects are generally handled differently: A plain pointer may be used when a local object is created; a special handle must be used for remote objects. Programmers must therefore be aware of which kind of object they manipulate be they local, remote or replicated. We propose an uniform reference mechanism, called Stub-Scion Chains, to locate and invoke mobile objects in distributed settings. A fault-tolerant distributed garbage collector is intimately associated with our reference mechanism.

The Soul system implements our reference mechanism above Unix in the C++ programming language. It supports transparent identification, invocation and reclamation of fine-grained C++ objects. This demonstrates the practicality of supplying a uniform reference scheme including distributed garbage collection at the system level. Performance measurements show that these goals can be achieved without sacrificing efficiency and that those mechanisms worth largely the slight overhead.

Keywords: *Distributed Garbage Collection, Object Identification, SSP Chains, Soul.*

Avant-propos

The style *italique* is used to catch the attention of the reader to sentences, expressions, or keywords defined in this dissertation.

The style **sans-serif** is used for fonctions or code exhibited in this dissertation. Function names are always followed by a pair of parenthesis in order to distinguish them from variables.

The expressions in **typeset** style are system functions or libraries.

A glossary appears at the end of this document defines a number of keywords related to garbage collection and the *C++* programming language.

Table des matières

I	Introduction	1
1	Contribution	2
2	Our distributed GC protocols	2
3	Brief Description of SSP Chains	3
4	Structure of this Thesis	4
II	Uniprocessor Garbage Collection	7
1	Principle	7
1.1	Live and Garbage Objects	8
1.2	Automatic Reclamation	9
1.3	Liveness and Safety Properties	10
2	Reference Counting	11
3	Tracing Techniques	12
3.1	Mark & Sweep	13
3.2	Copy Collector	14
4	Enhanced Techniques	14
4.1	Incremental Garbage Collection	15
4.2	Replication-based garbage collector	16
4.3	Generational Scavenging	17
4.4	Conservative Garbage Collection	18
III	Study of Distributed Garbage Collection Techniques	21
1	Garbage Collection in Distributed Systems	21
1.1	Model	22
1.1.1	Spaces	22
1.1.2	Stub and Scions	22
1.1.3	Representation of Remote References	22
1.2	Operations on References	23
2	Distributed Reference Counting Techniques	25
2.1	The Distributed Reference Counting Problem	25
2.2	Weighted Reference Counting	27
2.3	Shortcomings of Weighted Reference Counting	28
2.4	Optimised Weighted Reference Counting	31
2.5	Indirect Reference Counting	31
3	Reference Listing	32
3.1	Garbage Collector for Network Objects	33
4	Hybrid Cyclic Techniques	36
4.1	Complementary Tracing	36

4.2	Object Migration	36
4.3	Trial Deletion	39
4.4	Local Tracing	39
4.5	Discussion	40
5	Tracing-based Distributed Garbage Collectors	40
5.1	The Distributed Tracing Problem	41
5.2	Tracing with Timestamps	42
5.3	Logically Centralised Reference Service	47
5.4	Tracing Within Groups	48
6	Summary	51
7	Conclusion	53
IV	SSP Chains	55
1	Overview and Definitions	56
1.1	SSP Chain Invariants	58
1.2	Structure of this Chapter	58
2	Specification of the Protocol	59
2.1	Data structures	59
2.2	Transport Protocol	61
2.3	Presentation Protocol	62
2.4	Invocations and Short-Cutting SSP Chains	67
2.4.1	Basic Call-Response Invocation Protocol	67
2.4.2	Indirect SSP Chains	68
2.4.3	Short-Cutting an SSP Chain	68
2.4.4	Short-Cutting Strong Chain	69
2.4.5	Short-Cutting Weak Locator	70
2.5	Invocation Pseudocode	71
2.6	Collector Protocol	75
2.6.1	Local Garbage Collection and Create-Create	75
2.6.2	Cleanup Protocol and Create-Delete Race	75
2.7	Termination Recovery	76
2.7.1	Recovering Communication	78
2.7.2	Re-Establishing the Invariants	79
3	Analysis	79
3.1	Failures	80
3.2	Costs	80
3.2.1	Number and Size of Messages	80
3.2.2	CPU Time	81
3.2.3	Memory	81
3.3	Possible Simplifications	82
4	Conclusion	83
V	The Soul System	85
1	System Overview	85
2	Representation of References	87
2.1	Surrogate Data Structure	88
2.2	Scion Data Structure	88
2.3	Transport	89
2.4	Manager object	90

3	Binding of References	92
3.1	Binding Policy	92
3.2	Binding Protocol	93
3.3	Exception Handling	95
4	Maillons	101
4.1	Maillons Principles	101
4.2	Uniform v.s. Combined Maillon Policy	102
4.3	Final v.s. Indirect Maillons	103
4.4	Shortcut of Indirect Maillons	104
4.5	Locking within a Maillons	107
4.6	Using Maillons for Binding	109
4.7	Assessments of our Experiences with Maillons	109
5	Type Conformity in C++	111
6	Interface with the Local Garbage Collector	114
6.1	Finalisation	114
6.2	Weak Pointers	116
6.3	Object Relocation Prohibited	116
6.4	Surrogate Collectable Class	117
6.5	Indirect Roots	117
6.6	Local Garbage Collection Interactions with Maillons	119
6.7	Edelson Collector Assessments	121
7	Bootstrapping	121
8	Performance Measurements	123
8.1	Sun Lightweight Processes	123
8.1.1	Blocking I/Os and Agents	124
8.2	Invocation	124
8.3	Marshalling/Unmarshalling of References	125
8.4	Binding of Reference	125
8.5	Processing Live Messages	126
8.6	Performance Improvements	127
VI	Conclusions	129
1	Goals Achievement	129
2	State of the Prototype	130
2.1	Portability Issues	130
2.1.1	Pre-processing	130
2.1.2	Communications	131
2.1.3	Dynamic Type Checking	131
2.1.4	Local Garbage collector	132
2.1.5	Name Server	132
3	Future trends	132
3.1	Une preuve formelle des Chaînes de PSS	132
3.2	SSP Chains to Support Large Scale Applications	133
3.3	References to Persistent Object	133
4	General Assessments	133