Finer Garbage Collection in LINDACAP

Nur Izura Udzir, Universiti Putra Malaysia, Malaysia
Hamidah Ibrahim, Universiti Putra Malaysia, Malaysia
Sileshi Demesie, Bahir Dar University, Ethiopia

ABSTRACT

As open systems persist, garbage collection (GC) can be a vital aspect in managing system resources. Although garbage collection has been proposed for the standard LINDA, it was a rather course-grained mechanism. This finer-grained method is offered in LINDACAP, a capability-based coordination system for open distributed systems. Multicapaibilies in LINDACAP enable tuples to be uniquely referenced, thus providing sufficient information on the usability of tuples (data) within the tuple-space. This paper describes the garbage collection mechanism deployed in LINDACAP, which involves selectively garbage collecting tuples within tuple-spaces. The authors present the approach using reference counting, followed by the tracing (mark-and-sweep) algorithm to garbage collect cyclic structures. A time-to-idle (TTI) technique is also proposed, which allows for garbage collection of multicapability regions that are being referred to by agents but are not used in a specified length of time. The performance results indicate that the incorporation of garbage collection techniques adds little overhead to the overall performance of the system. The difference between the average overhead caused by the mark-and-sweep and reference counting is small, and can be considered insignificant if the benefits brought by the mark-and-sweep is taken into account.

Keywords: Capabilities, Garbage Collection, LINDA Coordination, Multicapabilities, Time-to-Idle

INTRODUCTION

The LINDA coordination model (Gelernter, 1985; Gelernter, 1989) offers an alternative to the conventional point-to-point communication framework with regard to coordinating and synchronising agents’ activities. The shared data space known as tuple-spaces (TSs) provides a medium for communication and facilitates the coordination among the interacting agents—agents communicate with each other via the tuple-space where they write and retrieve data (known as tuples). The clear separation between the coordination and the computation concerns relieves the agents of the messy aspects of communication, leaving them free to concentrate their time and space for other more crucial aspects of computation. This paradigm allows for anonymous interaction between agents separated in time and space: communicating agents need not know each other’s identity, and also the data can be retrieved any time after it has been placed in the tuple-space.

DOI: 10.4018/jitwe.2010070101
In the original LINDA model, there are three primitive operations which enable agents to manipulate the tuple-space: **out** to write a tuple, **rd** to perform read and **in** to read and remove the data. The tuples are retrieved associatively in a non-deterministic fashion: the retrieval operation may return any matching tuple; and, if a number of agents are waiting for a tuple of the same template, a matching tuple, when available, may be given to any one of them. Interacting via the TS where there is no direct communication between them, the communicating agents are decoupled in name, space and time—they need not know each other’s identity, nor co-exist at the same time in order to communicate with each other—providing a flexible coordination mechanism suitable for open, heterogeneous systems. LINDA’s popularity is shown in its commercial variants such as Sun’s JavaSpaces (Freeman, 1999) and IBM’s TSpaces (Wyckoff, 1998).

In open implementations of LINDA, agents or active entities can join and leave the system at anytime: they do not need to be defined at compile time. Agents which are not executing at the same time, can communicate via tuple-spaces. An agent can also store a tuple in a tuple-space to be used by some other agents at any time in the future. In general, agents running on different machines and operating systems can be compiled separately, and written in different languages. On the other hand, in closed implementations, all agents that wish to interact must be known at compile time. The information derived at compile time can be used to control the system and optimize the application in the best possible way, including when to remove which object when it becomes unnecessary. Applications for open systems are, in general, intended to have a longer running time than those for closed systems. In LINDA’s open system, tuples created are never removed from the system as it is difficult to decide if the tuples are still in-use or otherwise. As memory is a finite resource, this may lead to system failure due to memory exhaustion—the system will eventually run out of memory space if unusable objects are not reclaimed. Therefore, the implementations of open LINDA systems need to address the problem of memory management in order to be of practical use. One way of managing memory is by means of garbage collection (GC), which attempts to reclaim memory that is no longer used by an application.

The garbage collection mechanism has been implemented in Ligia (Menezes, 2000) to garbage collect unnecessary tuple-spaces using a graph, where reference information is maintained. However, tuples within the tuple-space cannot be garbage collected selectively: in order to perform garbage collection on tuples, information about their usefulness should be maintained for the garbage collector. This is not possible in the standard LINDA as tuples cannot be individually referenced—for the purpose of maintaining the information related to their usefulness—as they are nameless. In LINDACAP (Udzir, 2006; Udzir, 2007), with the introduction of mult-capabilities, i.e. capabilities for classes of objects, tuples within a tuple-space can now be referenced. Therefore, the information about the accessibility of tuples can be maintained by adopting the graph structure used in Ligia. As a result, garbage collection on tuples has been implemented in order to remove unnecessary tuples with specified template and subsequently reclaim memory space.

This paper describes the deployment of garbage collection in LINDACAP. The implementation is based on reference counting. Unfortunately, this technique does not apply to cyclic garbage, i.e., two or more multicapability regions referring to one another. In order to overcome this, the implementation of garbage collection in LINDACAP is extended using the mark-and-sweep algorithm to allow cyclic garbage collection.

There may also be the case where some agents may still have references to certain groups of tuples (represented by multicapabilities regions) but they no longer use the tuples/groups: the agents may have forgotten about these tuples which are consuming memory space. Based on the graph, the regions will not be considered as garbage as they are agents referring to them. A mechanism should be provided...
Related Content

A Lifecycle Approach for Scenario Driven Decision Systems
www.irma-international.org/chapter/lifecycle-approach-scenario-driven-decision/40432/

Connecting Communication to Curriculum and Pedagogy in Online Environments
www.irma-international.org/chapter/connecting-communication-to-curriculum-and-pedagogy-in-online-environments/137353/

Applying CBR Over an AI Planner for Dynamic Web Service Composition
www.irma-international.org/article/applying-cbr-over-an-ai-planner-for-dynamic-web-service-composition/103165/

Experience Report: A Component-Based Data Management and Knowledge Discovery Framework for Aviation Studies
www.irma-international.org/chapter/experience-report-component-based-data/23997/
Improving Software Agent Communication with Structural Ontology Alignment Methods
www.irma-international.org/article/improving-software-agent-communication-structural/47026/