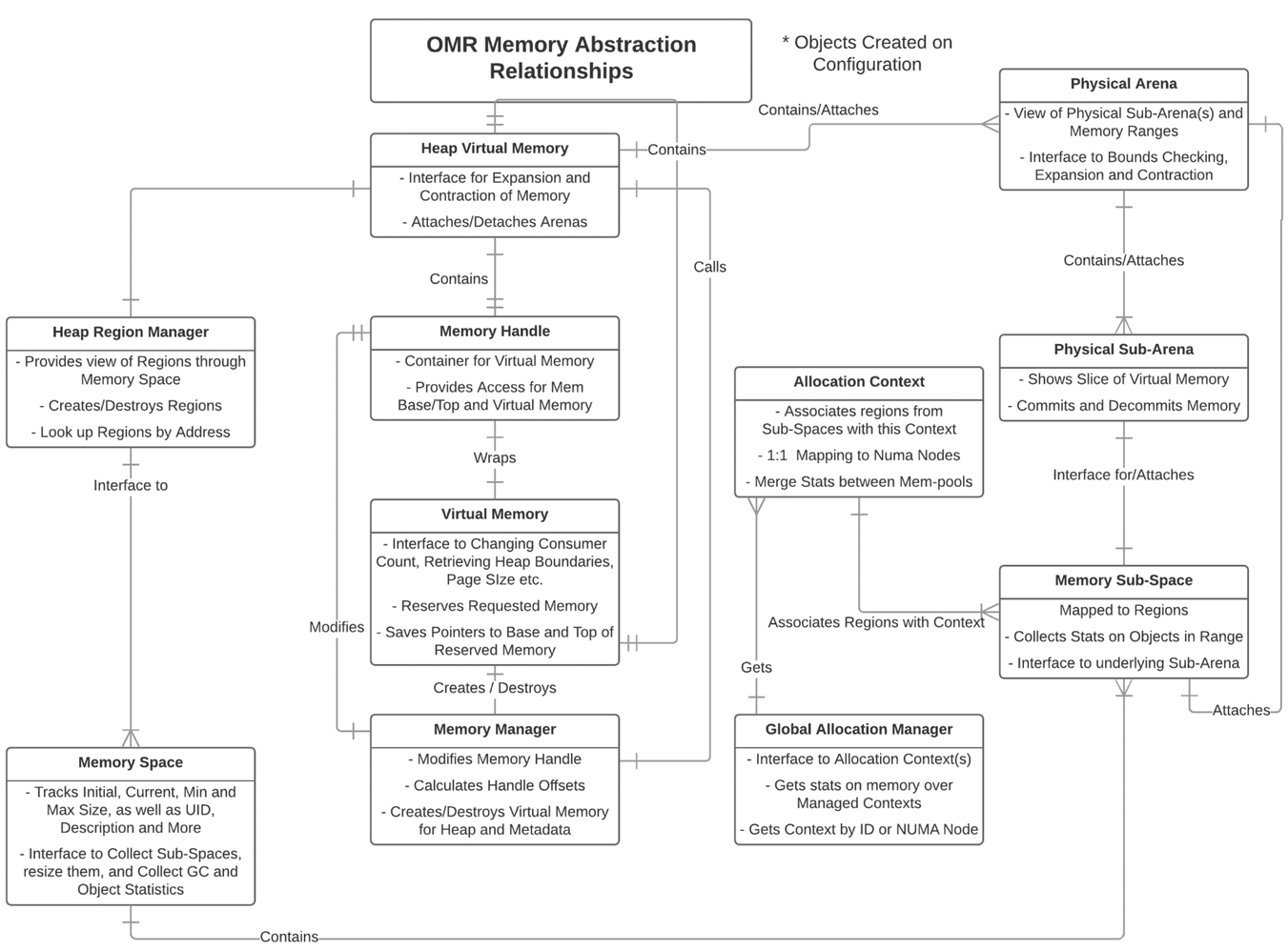
Implementing a Heap over Multiple Virtual Memory Areas in OMR

Michael Flawn, Scott Young, Kenneth B. Kent, Gerhard Dueck
University of New Brunswick, Faculty of Computer Science
Charlie Gracie
IBM Canada
{mflawn, scott.young, ken, gdueck}@unb.ca, charlie_gracie@ca.ibm.com

OMR Memory Abstraction

In the normal operation of OMR there is only one instance of Virtual Memory. We aim to separate virtual memory into two categories if requested, which for clarity will be designated ‘primary’ and ‘secondary’ virtual memory. These can be described as separate heaps but exist as contiguous memory in one heap object.

Creating two different types of virtual memory in OMR requires changes at all levels of the OMR memory abstraction; Virtual Memory exists as the lowest level of this abstraction.



Creating Two Virtual Memory Instances

Though the primary and secondary Virtual Memory instances are differentiated primarily by having different address ranges within the heap, the change from one Virtual Memory object to two presents several challenges.

The 1:1 relationships from Heap and Memory Manager to Virtual Memory must be changed. This involves additional consideration of both primary and secondary memory tops, bases, the alignment between memory object boundaries, as well as the true ceiling of the heap.

Care must be taken to avoid overlapping allocations, and to make sure that all components of the OMR memory abstraction are using the correct heap ceiling and size. The heap is still one object and needs to be treated as such.

Preserving Normal OMR Operating Parameters

It is necessary to make modifications that do not change the normal behavior of OMR’s Memory Abstraction, such that if no secondary memory is requested on configuration, then there is only one virtual memory object. This requires checks to see if secondary memory has a nonzero size. It also necessitates that heap’s initialization uses the primary memory size and boundary addresses to define the heap’s dimensions until the secondary memory’s existence is verified.

Distinguishing Virtual Memory Instances

Many of the functions that perform operations on memory for the heap (e.g. expanding, contracting, reserving, committing etc.) are responsible for memory operations outside of the heap’s address range. Non-heap operations must not trigger anything related to secondary memory, the same is true for primary memory. Additionally both Virtual Memory instances share a memory handle. Thus it is necessary to know not only that secondary memory exists but also whether the memory addresses used as parameters on function calls refer to secondary address space.