Visualizing Calling Context Profiles with Ring Charts

Philippe Moret  Walter Binder  Danilo Ansaloni  Alex Villazón
Faculty of Informatics, University of Lugano, Switzerland
firstname.lastname@usi.ch

Abstract

Calling context profiling is an important technique for analysing the performance of object-oriented software with complex inter-procedural control flow. A common data structure is the Calling Context Tree (CCT), which stores dynamic metrics, such as CPU time, separately for each calling context. As CCTs may comprise millions of nodes, there is need for a condensed visualization that eases the location of performance bottlenecks. In this paper, we introduce Calling Context Ring Charts, a new compact visualization for CCTs, where callee methods are represented in ring segments surrounding the caller’s ring segment. In order to reveal hot methods, their callers, and callees, the ring segments can be sized according to a chosen dynamic metric.

1. Introduction

Calling context profiling is a common technique to explore the dynamic behavior of programs and to find the reasons for performance problems. Calling context profiling yields dynamic metrics separately for each calling context, such as the number of method invocations or the CPU time spent in a calling context. A calling context is a sequence of methods that were invoked but have not yet completed; that is, a calling context corresponds to the methods represented on the call stack at some moment during program execution.

Calling context profiling helps analyse the dynamic inter-procedural control flow of applications. This technique is particularly important for understanding and optimizing object-oriented software, where polymorphism and dynamic binding often hinder static analyses. Typically, object-oriented applications make use of many short methods such that the inter-procedural control flow can become very complex.

The Calling Context Tree (CCT) [1] is a prevailing datastructure for representing calling context profiles. Each node in the CCT corresponds to a calling context and stores the measured dynamic metrics for that calling context. CCTs are often huge trees, sometimes comprising millions of nodes. Furthermore, the depth of CCTs can be high; CCTs with 50–400 levels are common in practice. Hence, there is need for a compact representation of CCTs helping the developer analyse dynamic program behavior. Pre vailing tools supporting calling context profiling typically present the CCT as an expandable tree. However, exploring large and deep CCTs, such as for locating invocations of a particular method in various calling contexts, is cumbersome with an expandable tree representation.

In this paper, we introduce Calling Context Ring Charts (CCRCs), a new way of visualizing and analysing CCTs. In a CCRC, the CCT root is represented as a circle in the center. Callee methods are represented by ring segments surrounding the caller’s ring segment. With CCRCs it is possible to display all calling contexts of a CCT in a single chart, preserving the caller/callee relationships conveyed in the CCT. For a detailed analysis of certain calling contexts, CCT subtrees can be selected to be visualized separately and the tree depth can be limited.

2. Background: The Calling Context Tree

The CCT was first introduced by Ammons et al. in [1] as runtime data structure for calling context profiling. Each node in the CCT represents a calling context and stores the measured dynamic metrics for that calling context; it also refers to a unique identifier of the method in which the metrics were collected. The parent of a CCT node represents the caller’s context, while the children nodes correspond to the callee methods. The CCT does not restrict which dynamic metrics are stored in the tree nodes. Common metrics include the number of method invocations, the CPU time, the number of cache misses, the number of object allocations, or the amount of allocated memory.

For some metrics \( M \), it is useful to consider also the aggregated metrics \( ag(M) \) for subtrees of the CCT, in order to explore the overall costs of method executions (i.e., including the costs incurred by all direct and indirect callees). For each CCT node \( n \), \( ag(M_n) \) is computed as the sum of the metric values \( M_x \) of all nodes \( x \) in the CCT subtree rooted at node \( n \). For instance, for a given CCT node \( n \), the metric
void main (String[] args) {
    for (int j=0; j<20; j++) {
        f(j);
        g(j);
        for (int k=1; k<j/2; k++) {
            h(k);
        }
    }
}

void f(int n) {
    int n tk=g ( n ) ;
    k=h(k)*k;
}

int g(int n) {
    if (n%2==0)
        return h(n/2);
    else
        return g(n+1);
}

void i(int n) {
    n=n*n;
}

int h(int n) {
    i(n);
    return n-1;
}

(a) Example code

(b) Generated CCT (conceptual representation)

Figure 1. Sample Java code and the CCT generated for one execution of method main(String[])

CPUₙ shows the CPU time spent executing the instructions within the body of the method mₙ represented by the node n, excluding the CPU time spent in callees of mₙ. In contrast, the metric ag(CPUₙ) gives the overall CPU time spent in the calling context corresponding to node n and in all callees.

The example in Figure 1(b) shows a conceptual representation of the CCT resulting from executing the Java code sample in Figure 1(a). The illustrated CCT represents one invocation of method main(String[]). For instance, such a CCT may be created with the profiler described in [2, 3]. In this example, two dynamic metrics are collected for each calling context, the number of method invocations and the number of executed bytecodes (in a Java Virtual Machine). Regarding the latter metric, which is a rather platform-independent alternative to the common CPU time metric, we are also computing the aggregated number of executed bytecodes ag(bytecodes) for subtrees of the CCT.

3. CCT Visualization

In this section we present three ways of visualizing CCTs as Calling Context Ring Charts. All three visualizations use an onion-like structure with circular layers, each layer corresponding to a level in the CCT. Nodes are ring segments, and children nodes (callees) are represented on the outer ring of their parent (caller).

a) Ring Segments of Equal Length: Figure 2 illustrates a ring segment s and its n children nodes c₁, ..., cₙ. In this first visualization, given α the angle covered by s, the angle θᵢ covered by its ith child cᵢ is computed as θᵢ = α/n.

Figure 3(a) shows this first visualization for our example code from Figure 1. As a consequence of the definition of θᵢ, the children completely surround their parent. For instance, in the second layer the ring segments representing the callees of main(String[]) (i.e., f(int), g(int), and h(int)) have the same length and completely surround the ring segment of main(String[]). The root node of the CCT is represented by the central circle.

This visualization gives a condensed view of the overall CCT and eases the analysis of caller/callee relationships. However, the visualization does not convey any dynamic metric stored in the CCT.
b) Ring Segment Length Proportional to a Selected Aggregated Metric: In order to ease locating performance problems, we support a second visualization, where each ring segment is sized proportionally to an aggregated metric $ag(M)$ chosen by the user. In this visualization, $\theta_i$ is computed as $\theta_i = \alpha(\frac{ag(M_c)}{ag(M_e)})$.

In this equation, the sum of all $\theta_i$ can be less than $\alpha$; that is, the children ring segments do not entirely surround the parent ring segment. The remaining portion of the parent ring segment represents the metrics contribution of the parent excluding its callees.

Figure 3(b) presents a ring chart where ring segments are sized according to the aggregated metric $ag$ (bytecodes). In this example, only about 67% of the ring segment of `main(String[])` are surrounded by the callees' ring segments, and the remaining 33% illustrates the contribution of method `main(String[])` to the overall bytecode execution.

This visualization helps locate hot calling contexts, since the length of ring segments is proportional to an aggregated metric. However, ring segments representing calling contexts with a low metric contribution can be very small such that the user may not be able to see all callees of a method. Hence, our different visualizations are complementary, and the user can switch between the different views.

c) Ring Segment Area Proportional to a Selected Aggregated Metric: In the previous visualization, we considered sizing ring segments with respect to the angle, which can be misleading: A node close to the center (i.e., with a low depth in the CCT) will appear smaller in terms of area than a node with the same aggregated metric contribution but in a deeper CCT layer. The third visualization presented below compensates for this bias by reducing the width of outer layers such that any two nodes with the same aggregated metric contribution will be represented by ring segments of the same area.

We call $r_i$ the radius of the $i$th concentric circle, $r_0$ being the radius of the central circle. In the two previous visualizations, we implicitly defined $r_i = r_{i-1} + c$, where $c$ is the constant width of ring segments. Since the area of a circle sector is $(\alpha/2)r^2$ (where $r$ is the radius and $\alpha$ is measured in radians), the area $A_i$ of the ring segment $s$ in layer $i$ is given by the equation $A_i = (\alpha/2)(r_i^2 - r_{i-1}^2)$.

For an area-proportional visualization of a calling context according to its aggregated metric contribution, but independently of its depth in the CCT, the term $(r_i^2 - r_{i-1}^2)$ in the equation of $A_i$ must be constant. This is achieved by redefining $r_i = \sqrt{r_{i-1}^2 + K/\pi}$, where $K$ is a constant corresponding to the desired overall area of a ring (layer) in the chart.

We illustrate this visualization in Figure 3(c). While this visualization better supports locating hotspots, in a deep tree outer rings may become too thin to be easily visible. Thus, it is important to let the user toggle between the different views.

In order to help exploring large calling context profiles, our visualizations are complemented with tree manipulation operations, such as subtree selection (see Figure 4(a)), depth limitation (see Figure 4(b)), and metrics aggregation for each method occurring in the CCT (see Figure 4(c)).

4. Conclusion

Calling context profiling is an important technique for exploring dynamic program behavior. The resulting profiles, usually represented as CCTs, are often huge, and prevailing visualizations, such as expandable trees used in sev-
eral profilers, are too verbose and do not make good use of space.

In order to ease handling large calling context profiles, we introduced Calling Context Ring Charts (CCRCs). A CCRC visualizes a CCT in a compact way, correctly representing all caller/callee relationships. Each calling context is displayed as a ring segment, surrounded by the ring segments representing the callees. Ring segments can be sized according to a chosen dynamic metric in order to ease the location of hotspots.

Acknowledgements: The work presented in this paper has been supported by the Swiss National Science Foundation.

References