Z2 Computer Solutions
Wayne L. Atchison
1609 Lund Lane
Polson, MT 59860
www.z2cs.com

The Snippet Engine

Personal Comments and History

November 13, 2015
Edited November 29, 2018

During the years 1985 through 1992 the C++ compiler, and the C++ Object Oriented Design (OOD) concepts, began to capture the computer software industry. It was during this time I realized that this popular movement would ultimately take software development into the wrong direction. I understood back then that the foundational concepts behind OOD were wrong. Making this assertion may sound like "blasphemy", but it has been proven correct many times.

It is not that the concept of using class-objects is a wrong direction. I use C++ classes all of the time. Rather, it is the fundamental fact that the C++ class-object, and the OOD methodology, inherently limits the software design in a number of very significant ways. There are many limitations imposed, but the two primary limitations are:

1.) **The C++ class-object cannot automatically take advantage of multiple CPU-Core Threads**. The OOD methodology assumes that each designed-class-object method is executed as a single block of code, compiled together along with all other class-objects. Many "tricks" must be used to force the class-objects to execute using more than one Core-Thread.

2.) **The C++ class-object cannot include external objects**. Other objects running externally, on networked computers, cannot be included as subparts of itself. All class-objects must be compiled as a single unit, forming a single localized Executable file. Thus, all class-objects must be linked together, with all other objects it needs, so that they can "**Call**" methods into execution. The concept of a class-object including external objects as part of itself, **to be "Asserted" into execution**, is nonexistent. Many "tricks" must be used to allow class-objects needing data from external sources to execute.

Even as the new OOD methodology was just emerging in project designs, it was clear to me that this popular movement would ultimately mean that future software development would become overly complicated, and very expensive. Today, I no longer need to defend this vision, as the number of failed large multi-million dollar OOD projects are too numerous to even count.
Search for "IT project failure examples":

http://it-cortex.com/Examples_f.htm
http://calleam.com/WTPF/?tag=examples-of-failed-it-project

An enhancement to the OOD Methodology:

Seeing these significant short falls, I set about to create a way to enhance the C++ class object and the OOD methodology. The result is the creation of a new "Engine" to process these new enhanced Objects. **This new engine is the Snippet Engine, and the enhanced Objects are the Snippet Engine "Nodes"**.

The foundational concept behind the Snippet Engine technology, is to use these new kind of Objects (Nodes) to network together, to do anything. Each **Snippet Engine Object** can talk to, and aggregate (build one accomplishment on top of another) with any other Objects running anywhere in the world.

Simplistically, the Snippet Engine is designed specifically for modern networked computers, each having multiple CPU-Cores. The Snippet Engine Object-Node automatically takes full advantage of massive parallel processing though the multiple CPU-Cores, and allows both local and externally executed Objects to be included as designed subsets in performing its overall task. **Thus, the two most significant limitations of the C++ class-object are overcome**.

The result of removing these two limitations is astronomical !

Snippet Engine Objects (Nodes) are always executed as the lowest-level execution units of a computer, as CPU-Threads. Executing as a Thread, each Snippet Engine Object automatically takes advantage of processing behind a solid system-lock, which means their data is always isolated for concurrency, and they cannot be interrupted until they are done. This means far less overhead in synchronizing "Command and Control" and database changes.

Snippet Engine Objects operate both asynchronously and autonomously. **Each Node behaves as a self-scheduling, self-aware, command-driven "Knowledge Center"**.

Each Snippet Engine Object-Node "**knows how to do something**", and uses other Objects, as subsets of itself, to ensure that it is always in a state of being "done". The Snippet Engine Application performs by having all Object "Knowledge Centers" perform and aggregate their results together.

Essentially, **Snippet Engine Objects (Nodes) have no scope limitations**, and each may be designed to do "anything", large or small, as each may interact directly with any number of other Objects, running anywhere in the world, to perform their designed duties. Each Object is "**always alive**", and **can be thinking ahead** in anticipation with the other Objects on the other networked computers.

Each Snippet Engine Object can be thought of as a "**Cyber Ant**". The **Snippet Engine Network** can create thousands of "Cyber Ants", each working autonomously, each doing what it does, and each telling the others, around the world, of what it has accomplished.

No other software technology enables the developers to create high level Object Oriented Constructs, where each Object is capable of worldwide, cooperative, self-aware interaction, and infinite aggregation of intelligence.

There are no limits to what can be accomplished when aggregating Snippet Engine Objects, as "Cyber Ants".