Nate Bachmeier

Week 3: Algorithms and Data Structures

TIM-8101: Principals of Computer Science

Northcentral University

March 24th, 2019

# What are template classes?

MadLibs is a famous word game which requires two participants. The first will come up with a short story and then insert place holders for some of the nouns, verbs and adjectives. Next the second person without knowing the story will provide values for these placeholders. Finally, the story is read a loud and everyone has a good laugh.

Templates are very similar to MadLibs, as the programmer can write the sequence of actions needed for declaring a function or class. Within that sequence are placeholders which can be replaced by the compiler as it generates compiled instances of these templates.

For instance, a class could be written to hold a list of integers and expose public methods add, remove, and get. These functions would then perform the necessary actions to manipulate some private integer array structure.

If the software package also needs to hold a list of strings, then the they would likely want to reuse as much of the list of integers code as possible. The code could be manually copied however this introduces challenges as bug fixes must go into multiple places. Instead they would create one template, and then ask the compiler to emit the different versions at build time.

Instead of using templates the developer could have written a class which holds a list of objects. Since everything in object-oriented programming is an object, the one list could have held either the string or the integer. However, this introduces other challenges as the implementation is essentially disabling the type system and its compile time checks. This has the potential to result in runtime errors, such as invalid cast exceptions, as the list can also contain multiple incompatible data types.

# Where are they used? Include a real-world example.

## Data Structures and Algorithms

Templates are often used in data structures and algorithms scenarios, as it is common for the implementation to behave correctly without change across a set of data types. Consider the stack and queue data structures, which maintain a list of things and then push and pop items to either end. It does not matter if these are strings, sockets, or tacos—they will behave the same. Then look at a sorting algorithm such as quick and merge sort. These manipulations can operate across any data type which implements a comparability interface.

## Metaprogramming

Some implementations of templating allow for metaprogramming during the compilation phase. This allows the generation to perform customized actions based on annotations or constant calculations.

For instance, the developer might create an interface with different methods representing different schematized log messages. Using an interface creates a clear separation between the component and the logging system. A factory class could then be used to emit the concrete log client based on the interface definition. This enforces consistency across all log clients as one central location is emitting the each of them.

An Object Relational Models (ORM) is another consumer of the metadata driven model. First a process will connect to a data store and fetch the schema information, this will be used to template serialization definitions for each of the tables. The output of these templates is then fed into another round of templates to provide create, read, update, and delete (CRUD) operations. This results in strongly typed generic code that can be statically verified as being accurate.

# What are some benefits and drawbacks of using template classes?

## Increased Size

Each instantiation of a template results in the generation of more code which needs to live within the binary. This will increase the size requirements of the of the program. For some embedded systems with limited resources this can introduce challenges.

Languages like C# have partially mitigated the code bloat scenario by exposing generics instead of templates. Generics are a runtime constructs that exploit the type system directly instead of relying on code generation. This provides many of the core use cases but is more limited than templates that run at the compilation level. An example of this can be seen with the inability to inject constant values into a generic.

## Recursive Templating

When a template is used to generate a class, that class part of the type system and treated the same as any other class in the system. This property exposes the ability for templates to operate on templates and then pass them to more templates. That enables general algorithms to expand into very specific concrete implementations.

The simplicity of that model makes it easy for developers to abuse the technology and write unmaintainable code. There are still use cases that require a concrete type instead of a mash up of the typed data structures. Consider the scenario of a dictionary that is keyed on a tuple and valued with list of async function pointers which might be written as Dictionary< Tuple<T,U,V>, IList< Func<Task<X,Y,Z> >>. It can be difficult for future developers to make sense of what T or U represent. Alternatively, a container class and small object model being returned. These classes could then be documented, and the intent made clear.

# What are some alternatives to template classes, and where/when would you use them?

Templating technologies are not limited to the compilation phase and can take occur during other stages of the build process.

## Macros

During the preprocessing stage macros can be applied to perform literal manipulations on the code itself.

For instance, an application might defined a macro called ISOK(expression,message); and liberally use it throughout the system. The preprocessor could expand this snippet into a try-catch block that automatically logs a message on error. This makes the code cleaner and easier to read as the redundancies are not shown to the developer.

## Moq Framework

One of the challenges with writing integration tests is that they often become coupled with live services. This can make them expensive to run or error prone as certain failure cases are difficult to reproduce. An innovative solution to this problem can be seen in the Moq framework for .net.

Developers first create an instance of the factory class Mock<T> where T is the interface to be generated. The setup method can be called multiple time and passed expression trees and callback function pointers. The framework then generates a new assembly in memory and constructs the templated instance. By having this technology execute during the runtime layer, it mitigates the lack of support for templates in the language itself, cleanly hide the generated code, and provides a mechanism to weave another runtime state into the mocked object.

# What research has been conducted using template classes?

## LISP

Clojure is a language that is derived from LISP and has taken a minimalistic stand point toward built-in language constructs (Hickey, 2019). As a LISt Processing language, the primitive into the system is a list with the first item specifying the macro name followed by the arguments. The macros are expanded at runtime and processed until they result in a single data value.

This makes it very trivial to extend the language with new keywords and operators. In fact, nearly all of operators within Clojure are runtime expanded macros (Hickey, 2019). As the developer introduces new operators they can be highly customized to the domain specific problem. Consider a build script that is written in Clojure and added macros for git checkout, make code, push to Artifactory, and execute unit tests.

A clear separation could then exist between the build own and the component owner, so that the neither needs to understand the internals of the other. Since there are central definitions of the macros and the expansion happens at runtime, the cost to maintain multiple build repositories decreases.

## Nemerle

Nemerle is a managed language for the Common Language Runtime and has taken templating to another level. This is accomplished by allowing developers to extend the compilation pipeline and making custom syntax trees first-class citizens.

In the language tutorial the authors describe how to write a custom for-loop syntax tree and then use it anywhere else in the program (Russian Software Developer Network, 2012). Having the ability to extend the language gives the developer unlimited flexibility in the way they template. It also provides a mechanism for adding domain specific operators which can greatly increase the simplicity and readability of software systems.

The tutorial also describes the <[expression]> operator which can is executed during compilation. The expression can directly access the parser and even manipulate the TypeBuilder structure to inspect or alter the inheritance structure and even include additional methods.

# What is some future research you can envision using template classes?

## Templated Solutions

One of the challenges with templated classes is that they stop short at the class level. In many systems the developer has redundancies from component to component that include more than just a single class. For instance, my employer has a service template creation utility, and there are hundreds of instances of applications that have been emitted from it.

However, the mechanism for providing this generation is to duplicate the entire code into a new git repository. After this point the templating system has no control over the lifecycle of that code. When changes are later required it becomes prohibitively expensive as the same two-line fix needs to be placed in hundreds of different locations.

There are existing systems for placing code in redistributable packages, though these solutions fall short managing the complete picture. Along with the code itself is documentation, build scripts, container definitions, and deployment scripts to name a few.

## Better Serialization

Web service programming typically involves passing either JSON or Xml based object representation. The serialization and deserialization are typically often performed by generic runtime parsers which can be slow to operate. In contrast templating could be used to create very specific parsers that are highly optimized for the individual object.

These optimized implementations could save 100s of milliseconds per round trip between the client and service. Multiple that by the number of calls across an Internet scale service such as Facebook or Amazon and that quickly becomes CPU years of saved time.

Binary serializers such Google’s protocol buffers have partially solved this issue (Shuai & Xiaojun, 2017). The challenge with binary serializers is that both the client and service need additional dependencies. Having additional dependencies restricts the potential audience and increases barrier to entry. The system could create a model where the gateway supports textual representations and the backend uses binary. For a 3-tiered web application the benefits would be minimal in contrast to a micro-service architecture.

# Create a small code example template classes in a language of your choice

Please see attached files <blah>