Polymorphic System Architecture Summary

Jeffery Bryson
(jeffery.e.bryson@lmco.com)
Software Engineer Staff Lockheed Martin Simulation, Training & Support

Categories and Subject Descriptors C.1.4 Computer Systems Organization, PROCESSOR ARCHITECTURES, Parallel Architectures, Distributed architectures ; D.2.11 Software, SOFTWARE ENGINEERING, Software Architectures

General Terms Design.

Keywords *Abstraction, Architecture, Polymorphism, Systems Architecture.*

Abstract

Run-Time polymorphism (RTP) has been used in the software community for two decades to satisfy dynamic reconfiguration, plug-n-play, extensibility, and system redundancy requirements. RTP is also used to construct software systems of systems. System engineers now have the same requirements applied to large-scale system architecture. A Polymorphic System Architecture (PSA) use's the same technology, by applying it to the system architecture. By defining specific polymorphic relationship within the system architecture the system architect can reduce the system complexity and satisfy functional requirement.

It is 'Abstraction' and the specific relationships to the abstraction that allows the architect to define the 'Types' of building blocks that will make up the system. These specific relationships are 'Abstract Aggregation' and 'Abstract Inheritance'. Figure 1 below illustrates these relationships.

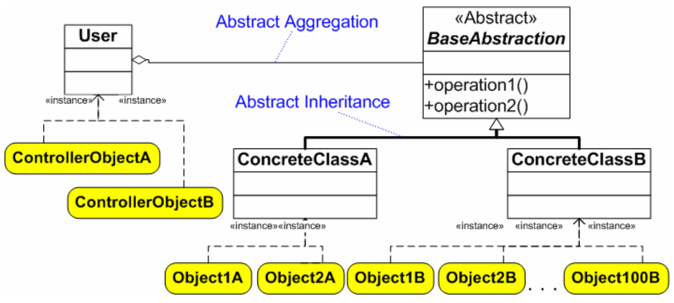

Figure 1 Polymorphic Relationships

The experienced system architect realizes that a PSA does not require software to be implemented. All that is required are formally defined abstract interfaces, objects (or components) that are developed based on the abstract interface, and a polymorphic mediator. A polymorphic mediator allows the system to dynamically connect and reconnect the building blocks of the system. The mediator is one or more applications in the system that allows a polymorphic user to find and connect to a polymorphic provider.

A system architect can create a PSA by defining an abstract interface, having a components or entity use (via aggregation) the interface, and having objects instantiated from subclasses (that inherit) from the abstract interface,

It is the relationships of inheritance and aggregation, to abstract classes that allows the system to be constructed of building blocks. It is the definition of these specific relationships and abstract classes that allow the system architect to create a PSA and satisfy specific functional requirements with the architecture.

Using a PSA will allow the developers to satisfy the following types of requirements:

- Dynamic reconfiguration of system functionality
- Plug-N-Play functionality
- Extensibility
- System Redundancy
- System Architecture Reuse

The system architect needs to understand these technologies, not only to create the system architecture, but to explain how the system architecture can directly satisfy these types of functional requirements.

In software, RTP is dependent on creating 'Abstract' interfaces. This same technology is now being applied to non-software components within the system architectures of Service Oriented Architecture and System of System solutions.

It is easy to translate functional logic to (run-time) polymorphic logic. Structure "Case" logic, (regardless of software or non-software implemented) can be translated as follows:

```
Case value is
   when red => doSomething();
   when blue => doSomethingElse();
   when green => doAnotherthing();
 end case;
```

This can be translated into a PSA as follows and is illustrated in Figure 2:

- The 'value' translates to the abstract interface reference.
- Each enumerated value (red, blue, and green)

becomes a child classes derived from the abstract interface.

- Each child develops its own implementation of ‘doSomething’.
-

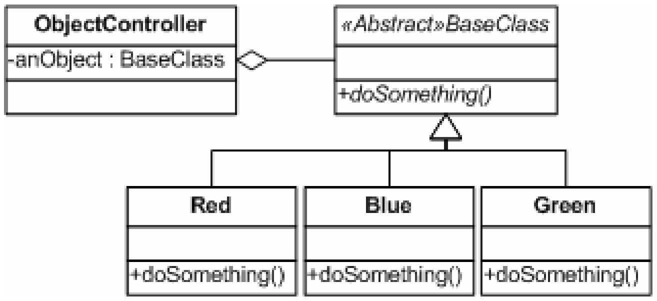

Figure 2 Polymorphic Logic

Every time there is a need to add behavior in the structured logic the control logic must be updated. When Polymorphic logic is used new objects and new classes can be added without changing the control logic.

The value of a PSA can be measured by mapping the specific types of requirements identified above to the ‘Abstract Aggregation’ and ‘Abstract Inheritance’ relationships. This measurement should be taken from the objects of the system not the classes. In Figure 1 there are two controller objects that can control any number of the 102 objects instantiated from the concrete classes.

As projects become more complex, there is a growing need to reuse software, hardware, and system architecture. A PSA allows the system architect to create an extensible and reusable system design. With a PSA satisfying requirements at the architectural level the system complexity can be reduced. The ability to apply polymorphic technology to software and non-software systems is dependent on the architects, customers, and program managements understanding of abstract classes, inheritance, and aggregation relationships and how these technologies are applied within the problem space.

Full paper at
http://www.incose.org/orlando/Attach/200905/Polymorphic System Architecture.pdf