Object Oriented Programming in Control System Software Engineering

Sean Leonard, B.Sc Eng. CompE, Msc (CompE), MBA
OPC Product Manager Matrikon, Inc.
**Introduction**

While esoteric in its infancy, Object Oriented Programming (OOP) crept into mainstream Information Technology (IT) applications by the mid 1980's. A decade later, OOP permeated Control System Software Engineering applications with many commercially available libraries and applications. The very popular incarnation of these object-based process control applications comes in the form of OPC client/server technology. These applications use OPC as the standards-based communication mechanism. OOP methodologies played a key role in the rapid adoption of the OPC technology. With widespread adoption, OPC facilitates the abstraction of control systems components such as DCSs (Honeywell, Emerson, Yokogawa, etc), PLCs (Rockwell, Siemens, Triconex, etc), process historians (AspenTech, Microsoft, OSI, etc), and many others.

**Procedural Programming Methodology**

Historically, building a control system software application involved developing algorithms and data structures that solved a specific problem. The problem could be solved in a series of steps (or procedures). This approach of programming is referred to as imperative or procedural programming. This approach to software development works extremely well if the problem only needs to be solved once and if the size of the problem is relatively small.

As the size of the problems being solved by control system software applications increased, the complexity of the problems also increased. With this increase in complexity, procedural programming became more cumbersome. Software developed using the procedural approach is often difficult to describe, maintain, and extend.

From an economic perspective, an obvious way to profit from developing software is to reduce the amount of software code that needs to be developed. Code re-use is one of the easiest ways to achieve this objective. Typically the amount of code reuse that is achievable using the procedural methodologies is relatively small. Using the procedural approach to develop software eventually became less attractive.

**Object Oriented Programming Methodology**

A fundamental breakthrough in software design was the concept of data abstraction using objects. Instead of developing software by modeling program flow, the software is modeled using objects. Objects are the nouns of the system. An object is a unit of structural and behavioral modularity that has properties inside code. Not only does an object encapsulate design decisions, it also encapsulates behavior, identity, state, and even business rules of the system. The process of representing a problem as a set of cooperating objects, and the relationships between the objects is Object Oriented Programming.

Using OOP design techniques, the software architect can abstract the problem domain with a set of objects. Each object fully encapsulates a portion of the software problem. For example, if a vendor wanted to develop a software application, like the Matrikon OPC server for Bailey Infi90, that abstracts a Distributed Control System, each function block object could have several attribute objects. You can reuse these objects to build a software application that abstracts a separate DCS or even a PLC like Matrikon’s OPC Server for Triconex Tricon. Once the problem has been solved, the objects can be re-used to solve other software problems.

OOP techniques represent entities in the problem domain with objects in code. This abstraction can aid in program conceptualization and design communication between developers. One of the benefits of choosing OOP is the ability to use computer-aided software engineering (CASE) tools in the design and extension process. Typically CASE tools...
tools enable the developer to rapidly design and capture the architecture of a product using the standard modeling language “the Unified Modeling Language (UML)".

Having a standardized system of describing software designs enables software engineers from anywhere in the world to share designs without having to first explain what the modeling conventions are.

**Benefits of OOP**

The three tenets of OOP design are encapsulation, inheritance, and polymorphism; all three are pillars of code re-use. Encapsulation is a modeling and implementation technique that separates external aspects of the object from the internal, implementation details. Encapsulation enables extensibility by enforcing locality of change to the scope of the changed objects. Inheritance allows existing objects to be extended without changing the original working code or the context. Polymorphism (dynamic binding) allows tested algorithms, used with the object, to be reused with the child class implementations without retesting the algorithm. These features combine to provide programming constructs that are aligned with framework design and code extensibility.

Another, very important re-use mechanism offered by OOP development is the re-use of code as a run-time library. A collection of objects can be packaged together in a runtime library or application. If one application expects to interact with a specified set of objects, many vendors can provide different implementations of the same library (thus enabling a best of breed solution).

**OOP and Control System Software**

Academia promotes the OOP approach to software development. Thus, software engineering schools teach OOP techniques, not procedural techniques. Software engineers who design control system applications typically have training in OOP techniques. Companies that develop control system software are more often than not, using OOP techniques to maximize code reuse, and minimize engineer training time. New hires are able to comprehend, maintain, and extend existing legacy systems built using OOP methodologies.

Products that leverage OOP methodology enable customers to easily extend base objects to suit the customer specific need. For example, the Matrikon OPC Genie (Generic Information Exchange) allows the user to create a protocol object that encapsulates a custom protocol to create a custom OPC server. Companies that utilize object-oriented frameworks can rapidly build software leveraging existing OOP code. Leveraging an object-oriented software code base enables companies like Matrikon to create custom software applications quickly and economically.

Inter-vendor solutions are a side effect of object-oriented design methodologies. A technology that has revolutionized the control system software development and deployment is OPC. OPC enables the abstraction of any data-generating source in a control system into a simple application. OPC defines a set of interfaces that describe how control system applications can interact. The OPC foundation is comprised of hundreds of vendors (often competitors). Work of the OPC Foundation members helps evolve the OPC specification. One could argue that OOP training that is prevalent in the member organizations has enabled the foundation to quickly adapt the specifications to the needs of the consumers of the technology. A typical OPC specification meeting involves engineers from many different parts of the world working together using UML and object-oriented technologies to capture and evolve the specifications. The OPC foundation supplies example object architectures that can be used as the basis of OPC development, thus easing the proliferation of the OPC technology.
**OOP in the future**

The software consumer does not know, nor care, which software methodology was used to build the applications that are used every day. Whether the latest HMI was built using procedural, OOP, or using some other less mainstream technique does not have a direct affect on the end result. Many very successful applications have been built using procedural programming techniques. Many applications written using OOP are not successful. The approach to building software is a vendor specific (and often project specific) choice.

OOP techniques have many benefits (as mentioned in this article). Some of the drawbacks of OOP methodologies are that object models can get complex very quickly. Software is becoming more and more complex. Procedural programming was fine for small systems; OOP is arguably good for large systems. What about extremely large systems? At one time 640k of memory seemed like an extremely large amount of memory. The system size is a relative measure. OOP may be seeing its age now that systems are measured in numbers of PCs (as is the case with many OPC based solutions), rather than amount of memory. A Service Oriented software methodology (a topic for another article) could be next. The point is that methodologies will come and go. OOP has solved many problems that the plagued the previous methodologies.

**Conclusion**

OOP has enabled rapid growth in control system software implementation and deployment. The rapid adoption of OPC is a good case study on some of the benefits of an OOP approach. OOP techniques have enabled software engineers to put the word ‘advanced’ into the phrase ‘control applications’. It has also facilitated the rapid evolution of software for process control. Without the economic gains due to creation of re-useable code, vendors would not have been motivated to push the technology envelope as they have. Process Control programmers have done their best to enable the assembly of best-of-breed systems using OPC. Now the focus shifts to the solution integration.

**About the Author**

Sean Leonard, B.Sc Eng, CompE, Msc (CompE), MBA (Technology Commercialization), is the OPC Product Manager for Matrikon.

**Additional Resources:**

Matrikon OPC: www.matrikonopc.com

Multimedia Tutorial: www.matrikon.com/tutorial

Copyright © 2005 Matrikon, Inc. All rights reserved. The information contained in this document is proprietary to Matrikon, Inc. No part of this document may be reproduced, stored in a retrieval system, translated, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission from Matrikon, Inc.