Control System Design Using LabVIEW Object Oriented Programming (LVOOP)

D. Beck and H. Brand
GSI, Planckstr. 1, D-64291 Darmstadt, Germany

Introduction

There are fundamental differences between LabVIEW and text-based languages: Concept of "Dataflow".

- Data type characterized by type of wire
- Data "flow" from source to drain
- Data have no defined life time
- Data are always "by value"
- "classical" VARIABLES DON'T EXIST!
- "wire fork" duplicates the data

The value on each wire is independent from every other wire - natural parallelism.

LabVIEW Object Oriented Programming (LVOOP) strictly follows the paradigm of "Dataflow".

- A class in LVOOP is only a type of data together with methods operating on the data/objects.
- In other words: LVOOP provides "object oriented wire types" including inheritance.
- An object in LVOOP is nothing more than data flowing through a wire - no agents.

- Objects can never be addressed "by reference".
- Member attributes can only be accessed/modified by its class methods - strict encapsulation.

⇒ Well known OO design patterns must be (re-)invented!

Reference Pattern

Problem: How can an object be accessed from different threads or computer nodes?

Solution: An object flows into a unique resource like a "message queue" or a TCP/IP connection.

Implementation: "Reference Pattern": Only one thread can use the object, the other thread has to wait...

Factory Pattern

Problem: How to dynamically create an object of any class without a constructor?

Implementation: "Factory Pattern".

Solution: "Factory Pattern".

Design Study CSOOMM

Main Requirements

- Create objects dynamically ("Factory Pattern" + object management).
- Multi-threaded applications ("Reference Pattern")
- Support distributed control systems (DIM as communication layer).

Object Management

Dynamically Create Objects

Class Independent Thread for Handling DIM Commands

Conclusion

Pros: Memory management within LVOOP is well done, very fast access to even large (> 10MB) of data. "Class independent" wires and override methods allow for programming simple but efficient code. Creating more than thousands of device objects (corresponds to thousands of physical devices per PC) is no problem.

Cons: Need to (re)invent design patterns for "Dataflow" paradigm. Override methods can't be executed in parallel (solved in LabVIEW 8.5). Danger, if objects contain references! Is LVOOP suitable for a large fraction of the LabVIEW community?

Features (Pros or Cons?): No multiple inheritance. Override methods must have exactly the same parameters as the base class. LVOOP is a consequent application of OO techniques to the "Dataflow" paradigm.