OSISOFT USERS CONFERENCE 2004

DISCOVER YOUR PORTAL TO PERFORMANCE

Solution Development Using Visual Studio.NET and the MDB

Bob Knox

George Muller

Conectiv Energy

- Merchant Generation company in PJM
- Merchant provides
  - Fuel & fuel arbitrage
  - Power marketing
  - Generation dispatch & management

PI at Conectiv Energy

- The Early Years - Plants
  - PI at each plant to capture operational data
  - Used for plant engrng, operation & performance
- Next Phase – Merchant
  - Time-series market data
  - Market interfaces & information
  - Built by External Consulting Firm on Excel/VBA platform

PI at Conectiv Energy

- Current Phase
  - Merchant / Generation Integration, Generation Desk, Plant management & metrics
  - New and re-architected applications
  - Built on Visual Studio.NET, MDB, PI-SDK, and ACE
  - Allows team to more economically meet growing business needs
    - Leverage code across units with varying characteristics
    - Reduces development and maintenance costs

Value Proposition
(how to sell this to your management)

- First – understand your business model:

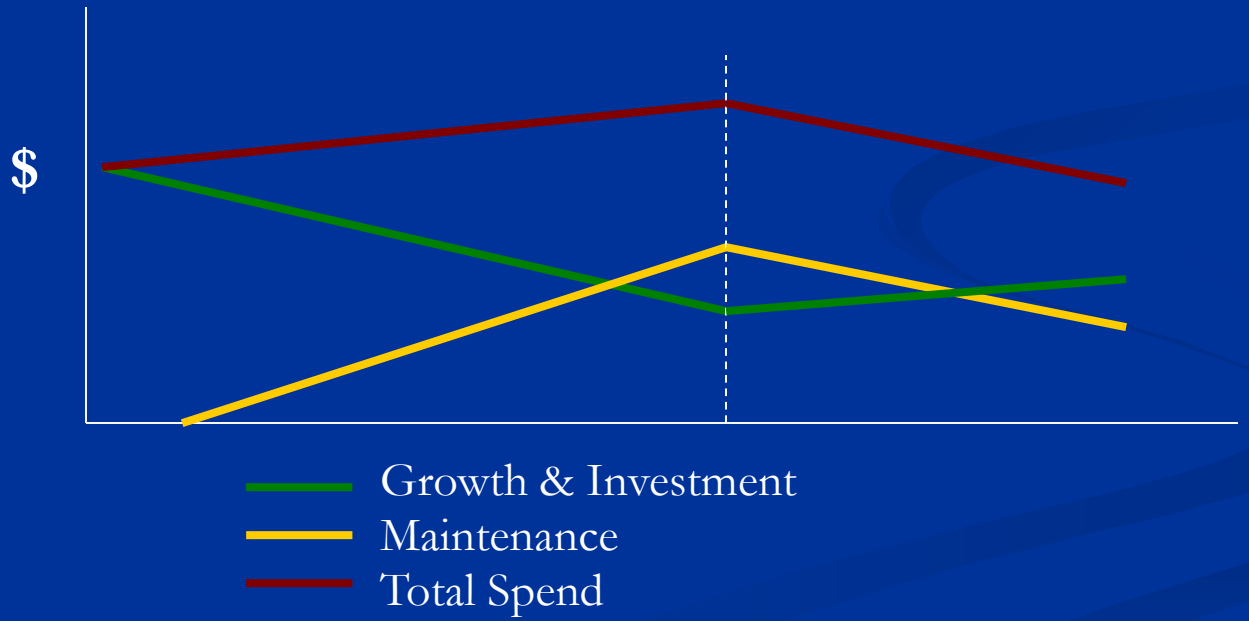
  - Information = Time + Decisions = $
  - Information important in each segment
  - Growing demand for information & applications
  - Growing pressure to control escalating costs

Value Proposition

- Understand how to add value
  - Where do you spend application dollars?
    - Core – sustain the business, maintenance
    - Growth – support business growth through enhancement
    - Venture – allow new business opportunities
  - Core spending grows over time, and with limited resources limits your ability to support Growth & Innovation
  - Challenge – how to maximize the value of your application dollar

Value Proposition
Shifting the Spending

- MDB to categorize and commoditize assets
- ACE for context-sensitive automated processing
- VS.NET components provide reusable function



Solution Development Essentials

- Platform and Architecture Identification (use ACE?, Web Application?, Windows Application? Other Relational Databases?)
- Module Database Factoring and Configuration
- Software Modeling Using an Object Oriented Approach

Module Database Factoring

- Primary Objective: Model the Physical Design of the Plant and Equipment
- Secondary Objective: Model the Application Design, by Meeting the Software and Business Requirements

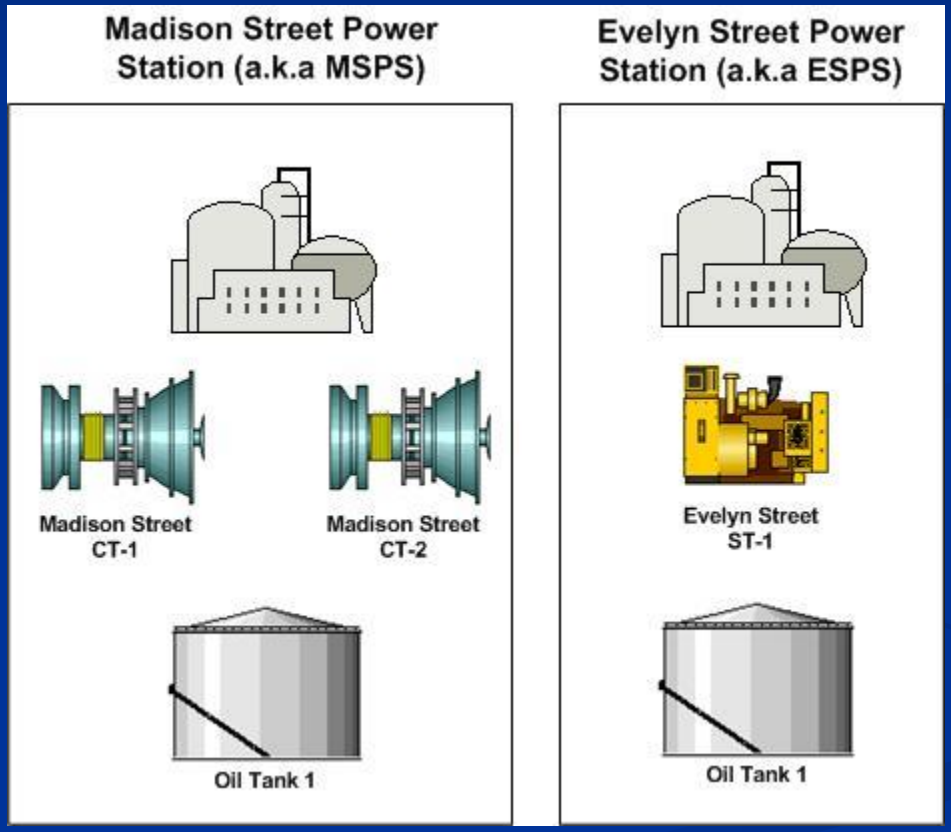
Module Database - Logical Module Design

- Create a Parent Module named "logical" to contain the Plant Assets and Equipment
- Model the Assets of the Plant and Equipment in Child Modules
  - Match static data and descriptions against Module Database Properties
  - Identify and categorize each PI point to a Module Database Alias

Module Database – Logical Design Suggestions

- Keep the Design as 'Flat' as Possible
  - Minimize Module and Property hierarchy
  - Use Concatenation (i.e. "ParentProp:ChildProp") of Aliases and Properties
- This design will be the basis for creating a reusable software library, to be shared by other applications

Module Database Example: Plant and Equipment Summary



Module Database Example: "logical" Configuration

Live Illustration!

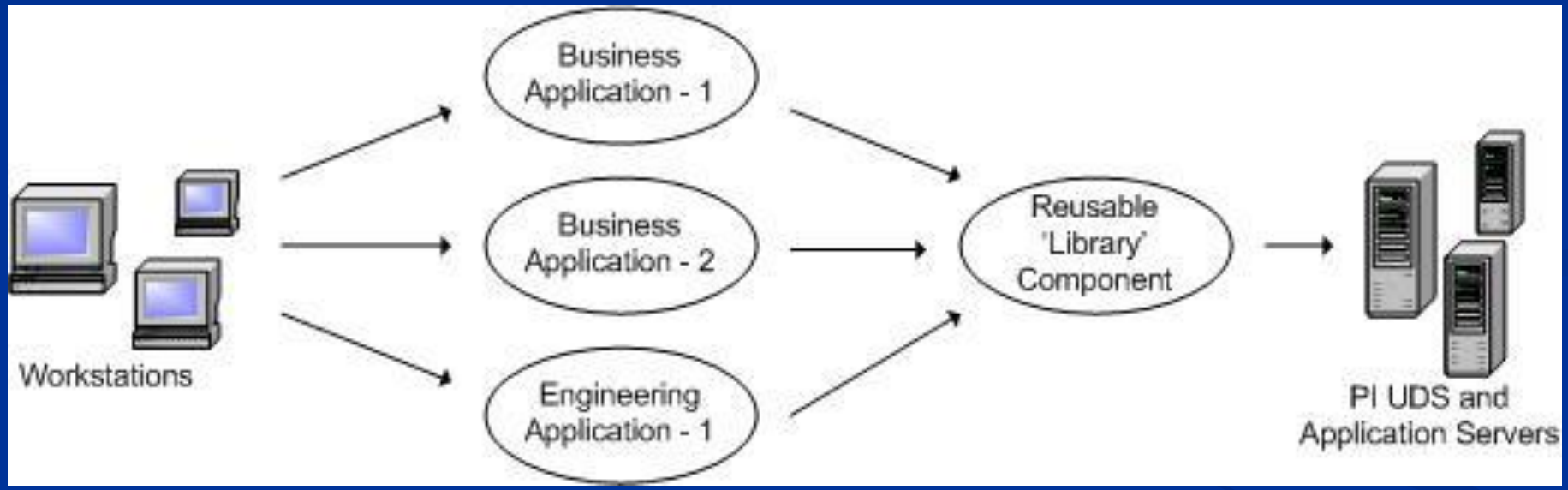
Module Database – Application Module Design

- Create a Parent Module named “apps” for each Application Created
  - Store Security Settings
  - Store “Win Registry” Settings to Minimize Deployment Issues
  - Store Application Layout and Personalization Settings (i.e. Column Layouts, Color Schemes, etc.)
- Link “logical” Modules to “apps” Modules, if necessary

VS.NET Development – Software Modeling

- Create a Reusable Software Library (a.k.a. Library) to interact with the Module Database "logical" Structure (n-Tier)
- Create new applications which reuse the Library to read / write data from the PI database (User Interface Tier)
- Use the PI-SDK vs. PI-API for component development (also, perhaps the PI-OLEDB, dependent upon the platform)

VS.NET Development – Architectural Overview



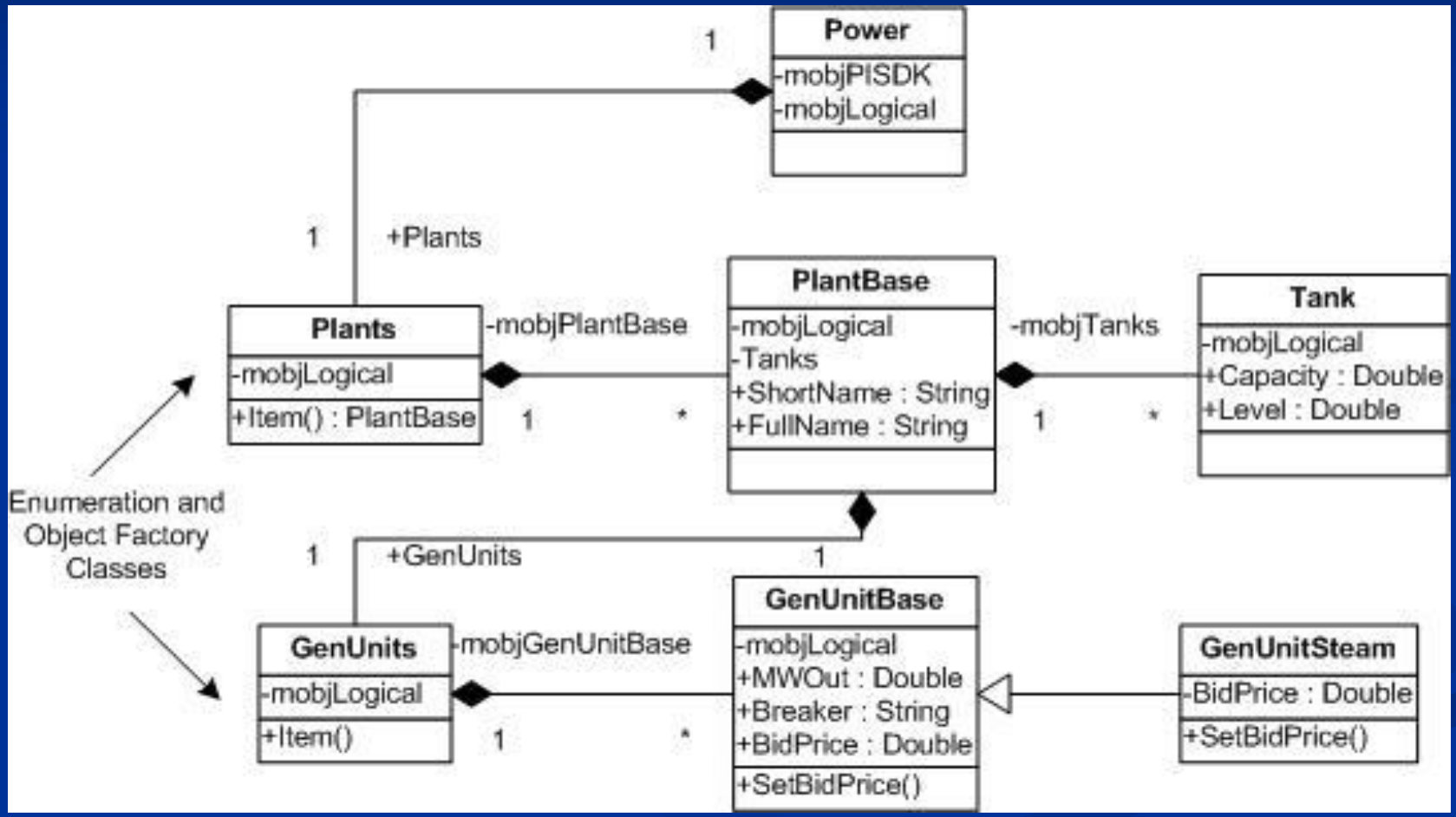
VS.NET Development – "Library" Design Basics

- Mirror Class Composition to MDB Hierarchy
  - Start with a Parent class that creates PI-SDK and OLE-DB connections, validates initial security, and instantiates other required objects (i.e. class named "Power")
  - Create additional classes in order of depth in the Module database "logical" design
  - Results in a composite structure friendly to programmers (i.e. Power.Plants("MSPS").Units("MSCT01").Property)

VS.NET Development – "Library" Design Basics (cont'd…)

- Consider creating classes that support enumeration (through IEnumerable)
  - Results in support of For Each… (i.e. For Each objUnit in Power.Plants("MSPS").Units)
  - Validate security to each child object within the enumeration class
- Consider creating Base Class and Derived Classes to allow for the different characteristics and attributes of each plant and equipment

VS.NET Development – "Library" Class Diagram



VS.NET Development

Sample Project Review –
Live Illustration!

Thank You!

Bob Knox
George Muller
Conectiv Energy
Newark, DE