

Managing Software Components with Teamcenter Enterprise and UML-based Model-Driven Development

Jim McElroy

I-Logix, Inc.

Jimm@ilogix.com

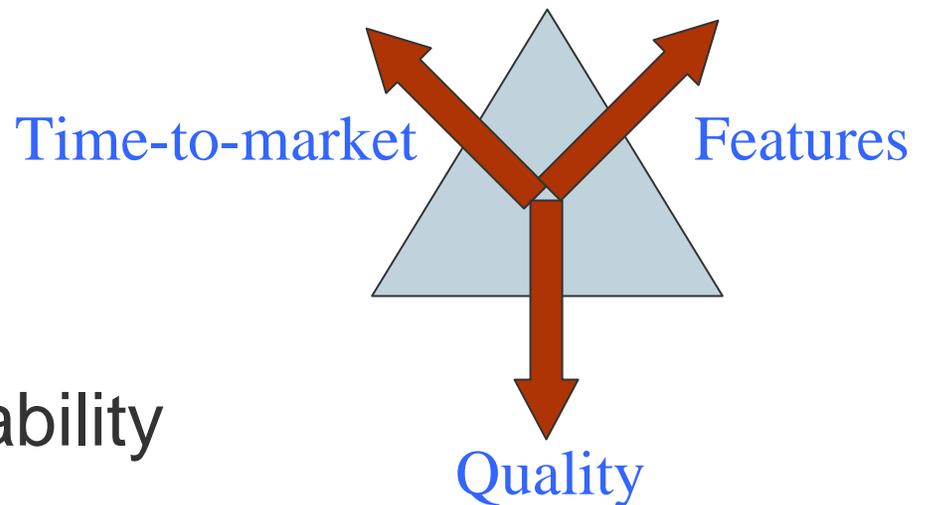
978-645-3026

Our Goal

Present a practical cohesive methodology and infrastructure for both developing and managing “Software Parts” with the mechatronic systems they control

Primary Issues in Today's Global Software Development Market

- Complexity
- Communication and Collaboration
- Productivity
- Time-to-market
- Quality
- Safety, Security, Reliability



Strategies and Technologies to Solve these Global Issues

- Model-Driven Development
 - Abstraction
 - Automation (Code generation, testing, documentation)
 - Reuse (through models and CBD)
 - Communication and collaboration
- Software Configuration Management
- Product Life-cycle Management
 - Functional Decomposition
 - Traceability of Requirements, into and throughout the Physical Design
 - Linkage and Navigation to/from the BOM

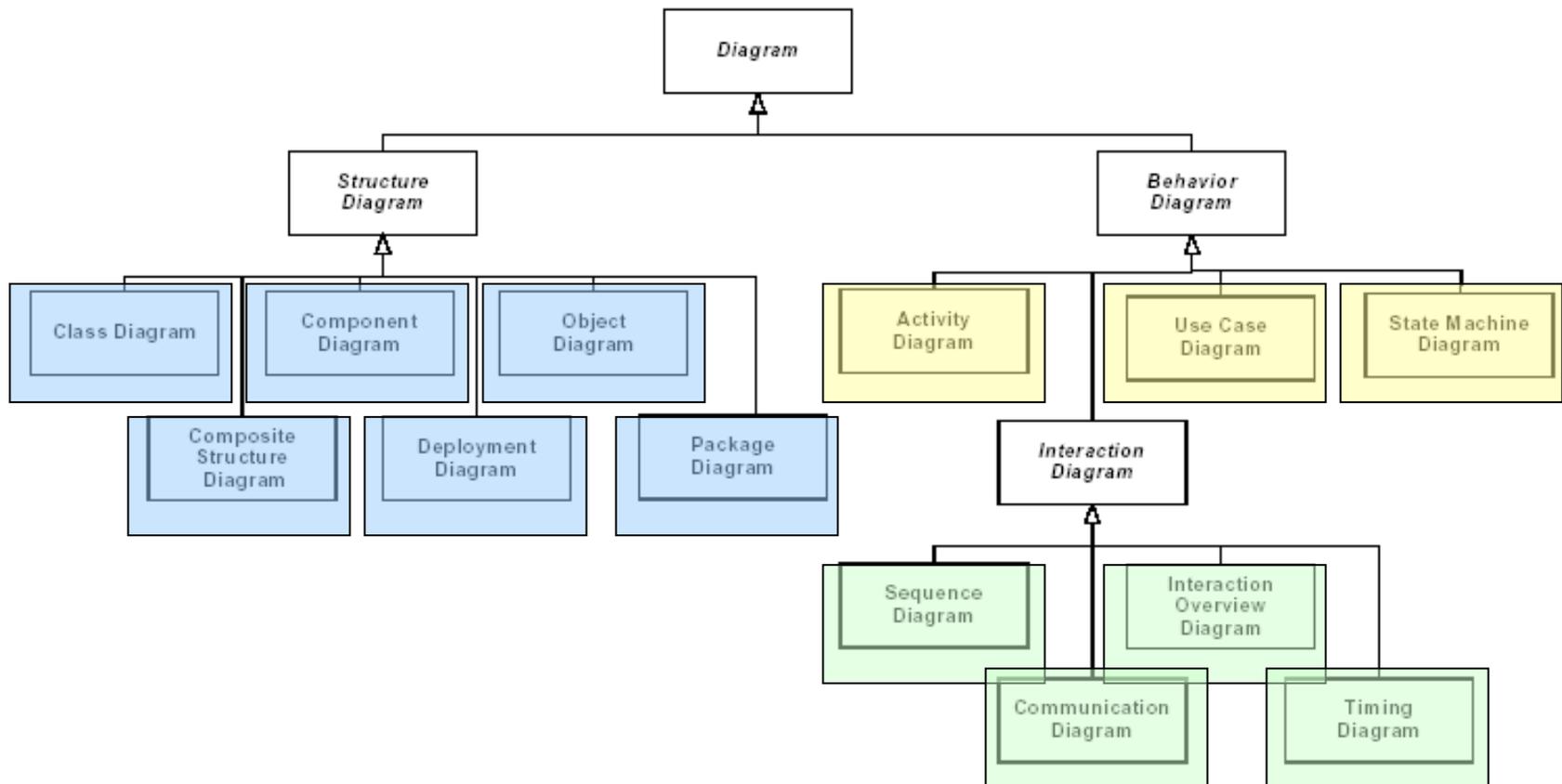
Model-driven Development

A design and development methodology which uses models as the basis for analyzing requirements, developing the design, implementing, testing, and deploying the application

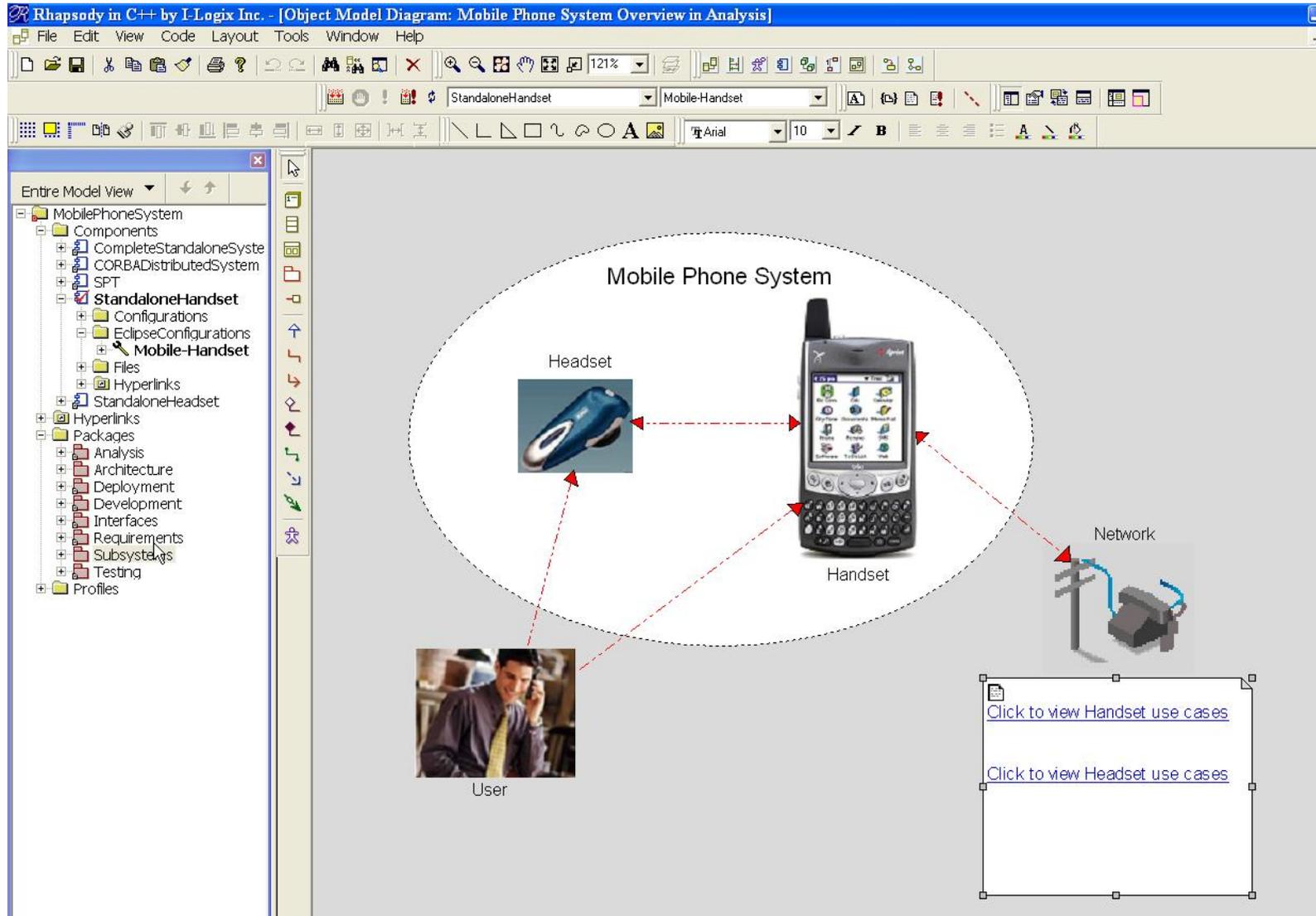
Modeling Requires a Language

- UML 2.0 – Standards Compliance
 - Architecture
 - Behavior
 - Collaboration
- SysML
 - A specialized profile of UML 2.0 for systems engineering. Both reuses and extends UML 2.0.
- DoDAF
 - An architectural framework providing a common communication mechanism for operational, system, and technical architectural views.

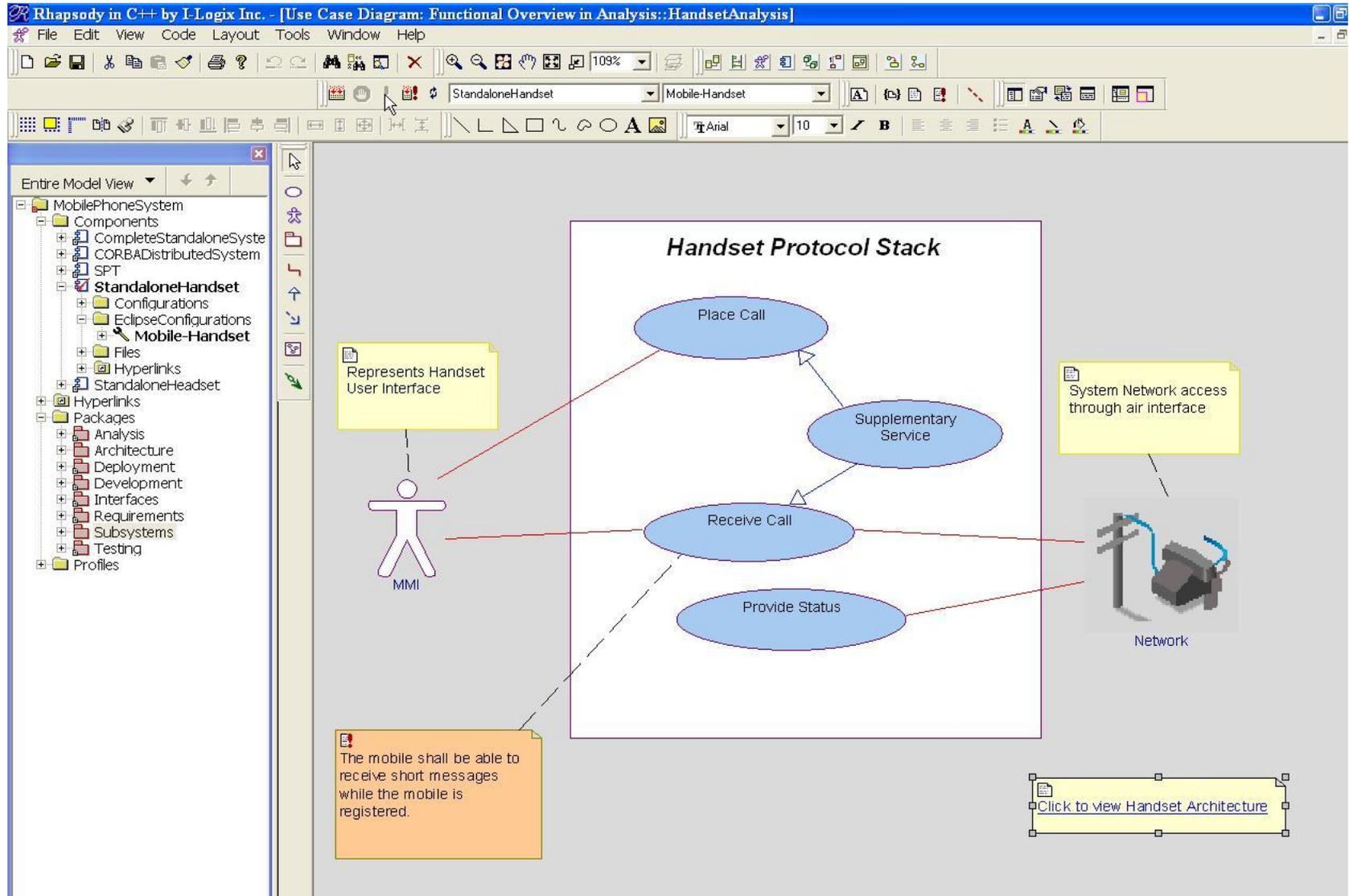
UML 2.0 Diagrams



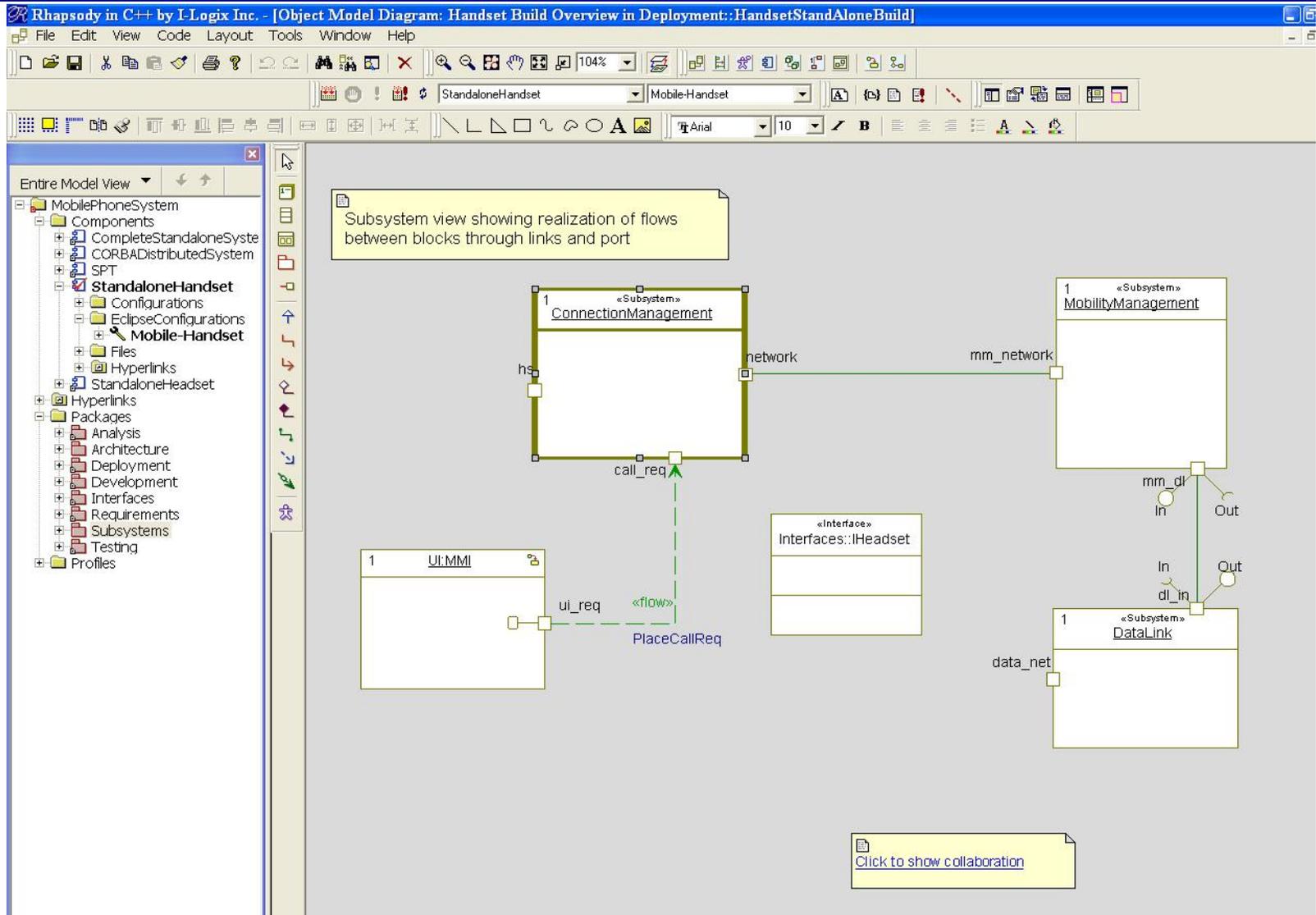
MDD by Example



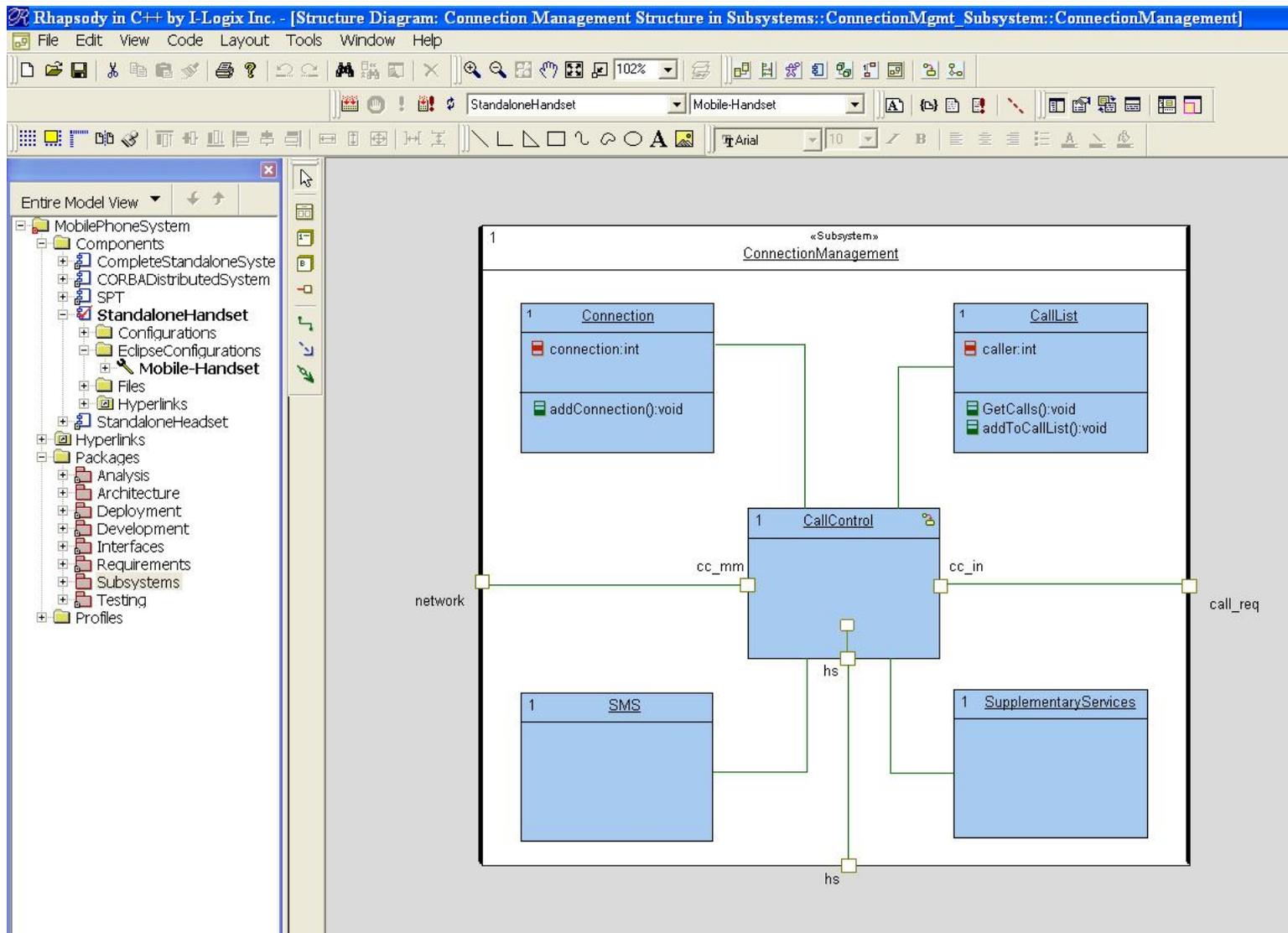
MDD – Functional Requirements View



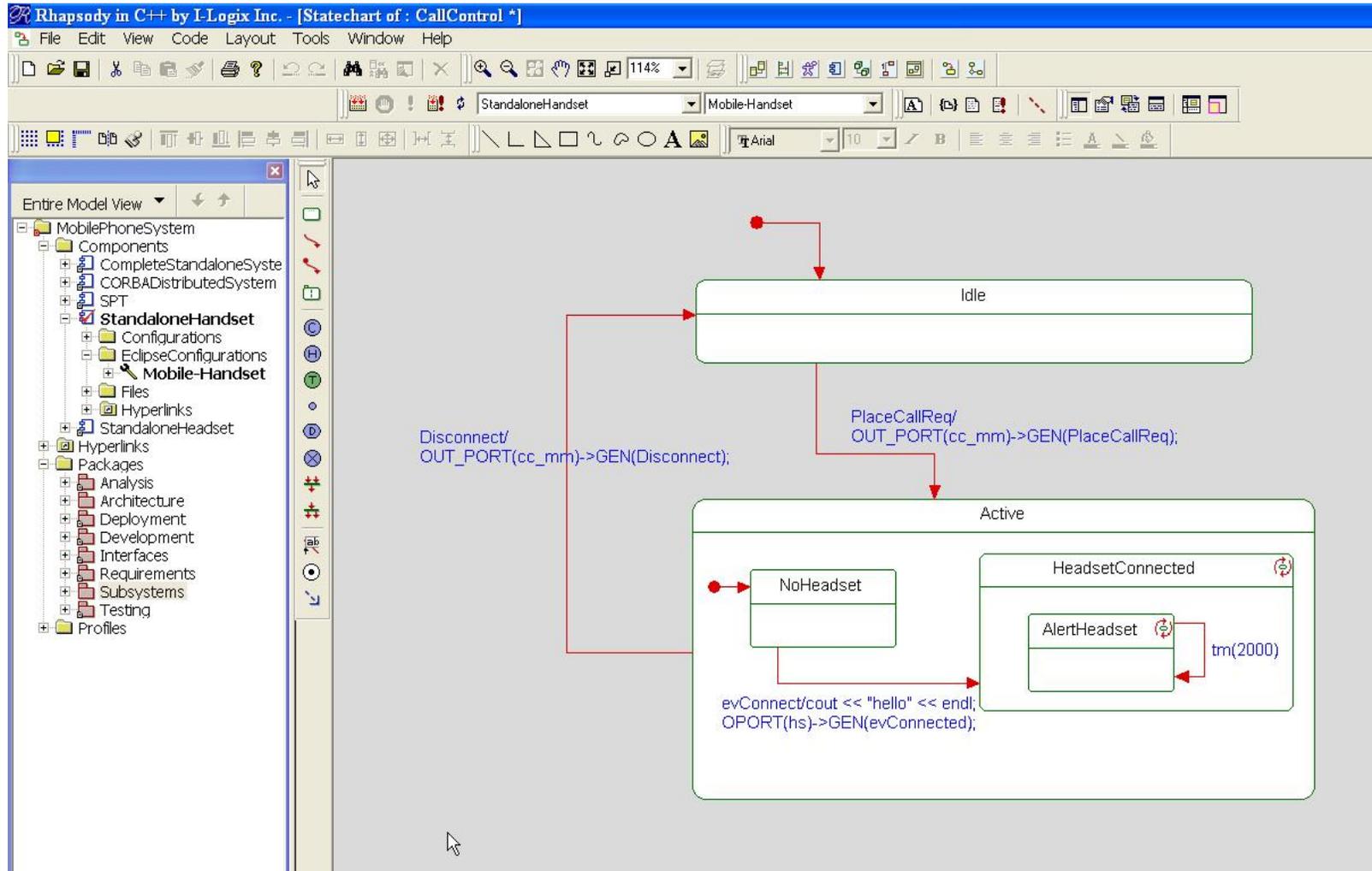
MDD – Architecture Example



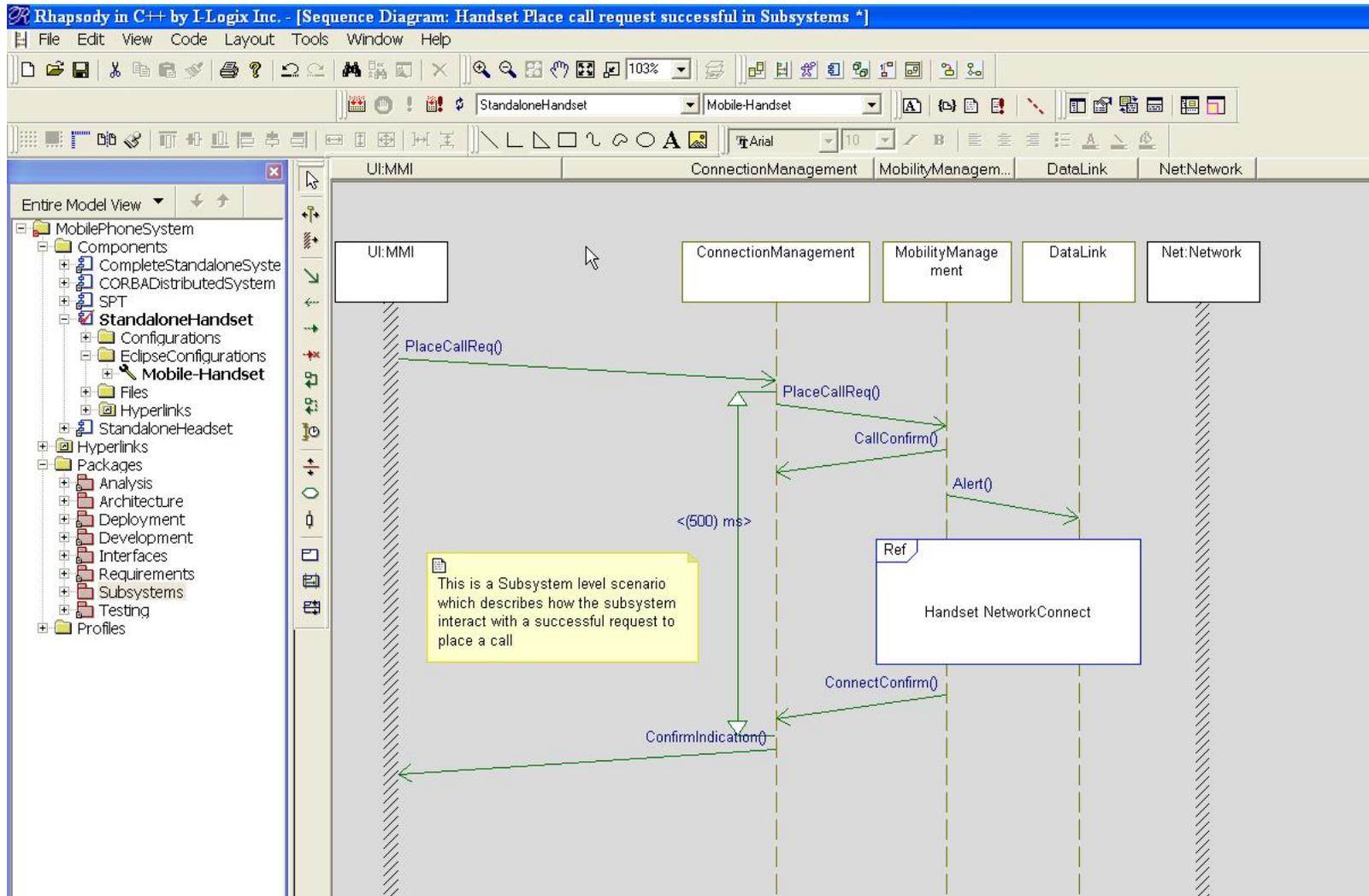
MDD – Architecture Level 2



MDD – Behavior Example



MDD – Collaboration Example



Benefits of Modeling

- Modeling gives us the ability to visualize the system clearly
- Simplify the problem through abstraction
- Executable models further enhance the visualization, understanding, clarification of intended functionality and behavior

Key Differentiators for MDD

- **Modeling (UML 2.0 PLUS)**
 - Benefit: Designing systems at a higher level of abstraction in order to easily deal with complexity
- **Code Generation**
 - Benefit: Get to the final product quicker. Enables designers to work at a higher level of abstraction to deal with system complexity
- **Model/Code Associativity**
 - Benefit: Freedom to work at the model level or source code level, and ensure views of the system are always synchronized.
- **Real Time Framework**
 - Benefit: Allows you to create a deployable application, with the generated code, onto a real time operating system
- **Reverse Engineering**
 - Benefit: Allows you to reuse your IP, as well as coexist with ongoing hand coding activities
- **Design for Testability**
 - Benefit: Automate the testing process through using the *design requirements* to validate and to completely cover all system scenarios

Crossing the PLM Chasm

- Historically, versioning and configuration management have been major bottlenecks to system development
- Hardware and physical elements follow their own life-cycles separate from the software
- Need to connect the software development life-cycle and the artifacts that are produced with the actual mechatronic systems they control
- Save development time and energy while at the same time improving system quality

Integrating MDD with PLM

- Software Component Development and Management with MDD
- Elevate Components to “Software Parts”
- Managing and Controlling “Software Parts” within the PLM System and the Vault

What is a Software Component?

- An executable?
- A library?
- Collection of source code and header files?

A generic component definition...

A **component** is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces

Component Life-cycle and Management

Required Capabilities

- Creation
- Validate
- Modification
- Documenting
- Publishing

Utility Capabilities

- Storing
- Finding
- Downloading
- Testing
- Submit and monitor
Defect and
Enhancement
Requests
- Metrics collection

The Software Part

Software Part

Meta Data

Author, Date of Creation, Type

Links to Defects/Enhs

PLM
Links

Design Documentation

UML Design Files

Links to Reqts Files

Test Documentation

Test Plans and Procedures

Test Results

Component

Include Files

Source Files

Binary Files

Libraries

Executables

Teamcenter - Management and Distribution of the Component

- Storing the component for reuse
- Determining system-wide dependencies
- Publishing the component
- Finding components in the gallery
- Keeping metrics and data pertaining to component
- Software Part becomes a part within the Teamcenter Assembly
 - Single-source configuration control

MDD, CM, and Vault Workflow

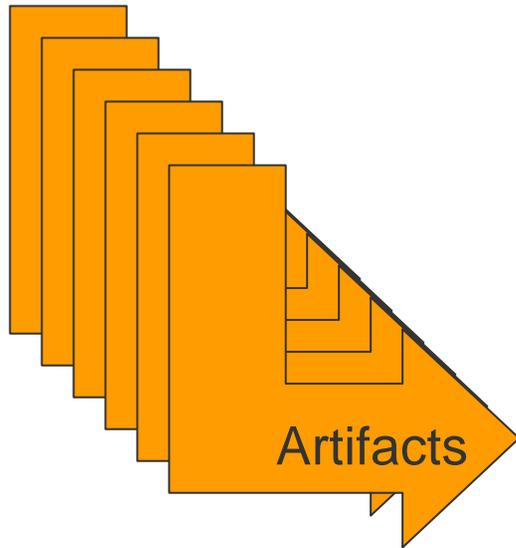
MDD

Artifacts

Models

Code

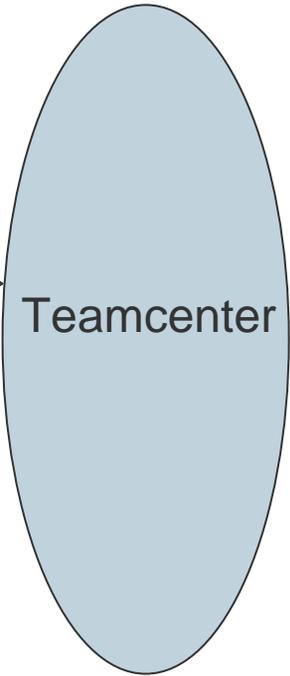
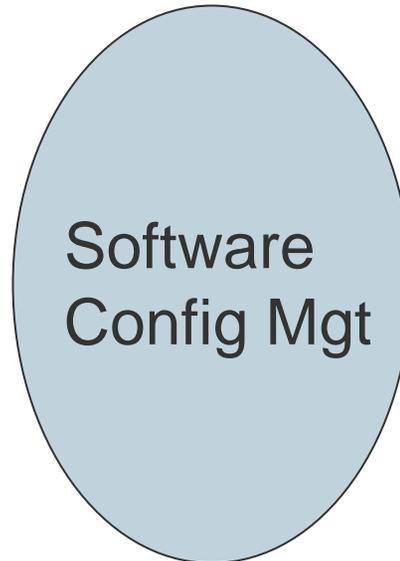
Documentation



Developer - Daily/Weekly



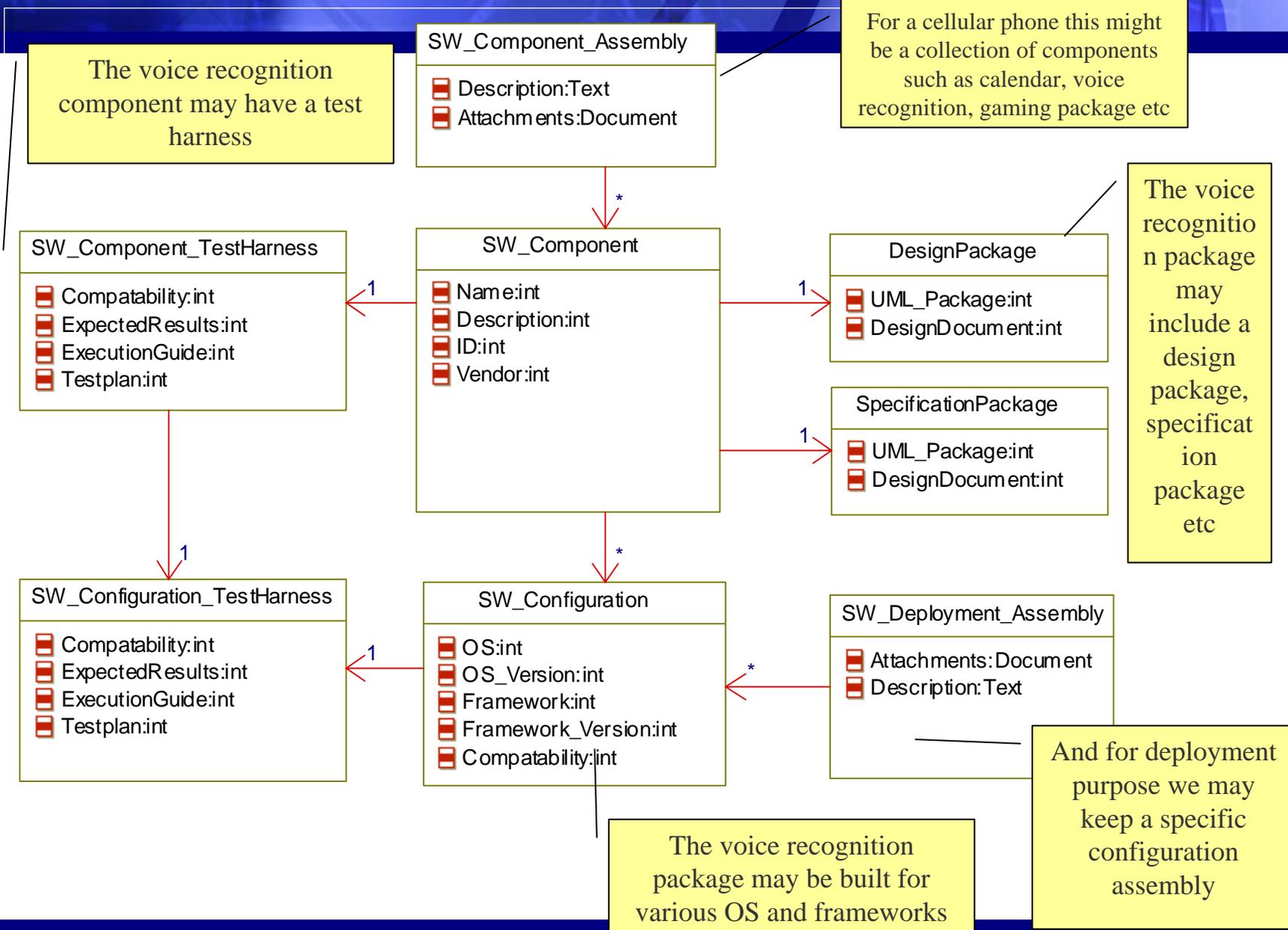
Release Mgr - As required



Integrated Approach

- MDD will enable the creation, validation, and executable visualization of Software Components
- Teamcenter will provide the infrastructure for elevating Software Components to Software Parts and then managing and controlling those parts within the assembly
- Synchronizes and couples the “Software Parts” with the mechatronic parts they control improving workflow, productivity, and quality

Just an example of a S/W Part!



Pulling all the Value Together

- Synchronized MDD, SCM, and PLM
- Single source for configuration control ensures seamless workflow and productivity gains
- Made software developers more productive, release managers more effective and efficient
- Strengthened our joint position in developing highly scalable software solutions