



Developing n-tier JEE Enterprise Applications using MDD

Richard Hensman, Tony Wilkins

scisys



Agenda

1. Context
2. Our MDD Environment
3. Examples of MDD in use
4. MDD Issues and Considerations
5. Advantages of MDD
6. Conclusions



Part 1 - Context

SciSys in Perspective

- Systems Integrator/Software House
 - ↳ Develop bespoke applications
- Develop IT systems for
 - ↳ Space
 - ↳ Defence
 - ↳ Government
 - ↳ Utilities
 - ↳ Broadcasting
- Approximately 450 staff
- Offices in UK and Germany
- Head office Chippenham

The Project

- Programme of work

- ↪ Started late 2005
- ↪ Development started 2006
- ↪ Currently in 5th phase

- Application

- ↪ SOA N-tier JEE workflow driven application
- ↪ Intranet and Internet applications
- ↪ Scalable architecture
- ↪ Frameworks – EJB2, Struts, Spring
- ↪ Languages – Java, JSP, XML, SQL
- ↪ Application mainly concerned with allowing users to:
 - View and enter information (via Internet and screens)
 - Implement business processes using workflow

Why did we use MDD?

- Allow us to develop application in UK at competitive price
- Maximise reuse across programme
 - ↳ E.g. screens and services
- Reduce development effort
 - ↳ Generate code from design models
- Develop JEE Software Factory

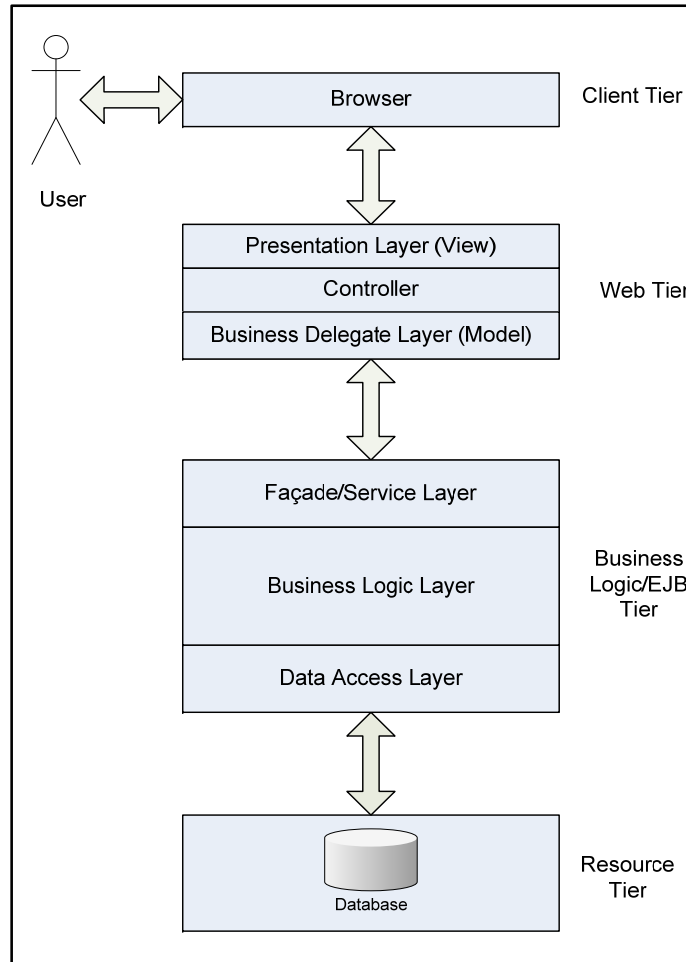


Part 2 – Our MDD Environment

MDD Tool

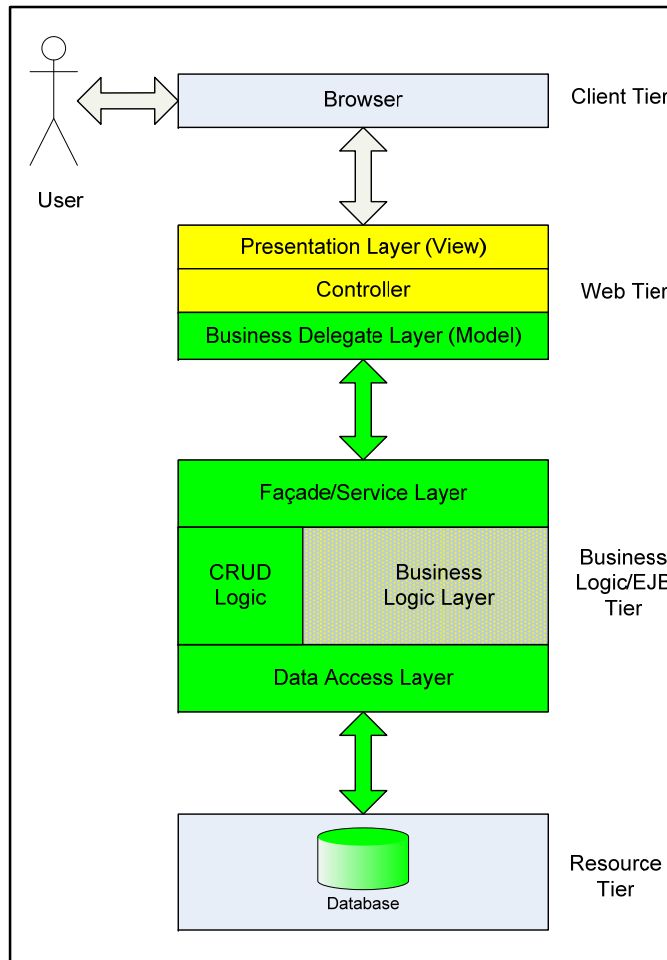
- OptimalJ
 - ↳ Generates out-of-the-box:
 - N-tier JEE or client server applications
 - Java, JSP, Struts, EJB2, JDO, XML, SQL
 - ↳ Model PIM and PSM levels:
 - PIM – Domain Class, Service, Use Case
 - PSM – Screen, Parameter Types, Business Logic, PDM
 - ↳ Extensible – create/modify meta-models and transformations
 - ↳ Allows iterative/incremental generation of models and code
 - ↳ Allows user editing and extension of generated models
 - ↳ Able to embed hand-written code in generated code
- Want to generate code for own SciSys JEE architecture
 - ↳ Used and modified existing meta-models and transformations
 - ↳ Developed new meta-models and transformations
- Examples of meta-models/transformations created:
 - ↳ Meta-models: Use case, Parameter Types, DSL to capture requirements
 - ↳ Transformations: Use Case to Web Flow, Web Flow to Code

JEE Code Generation



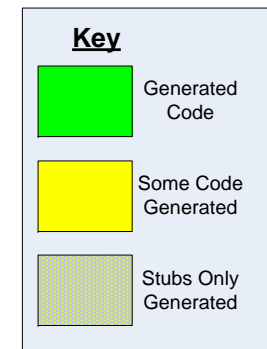
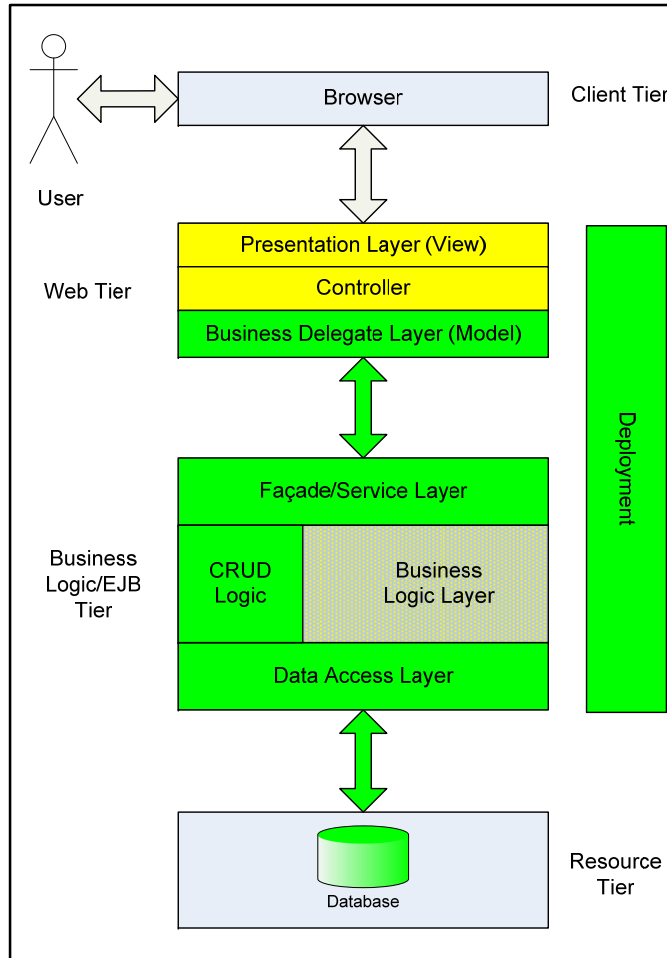
JEE n-tier Architecture

JEE Code Generation



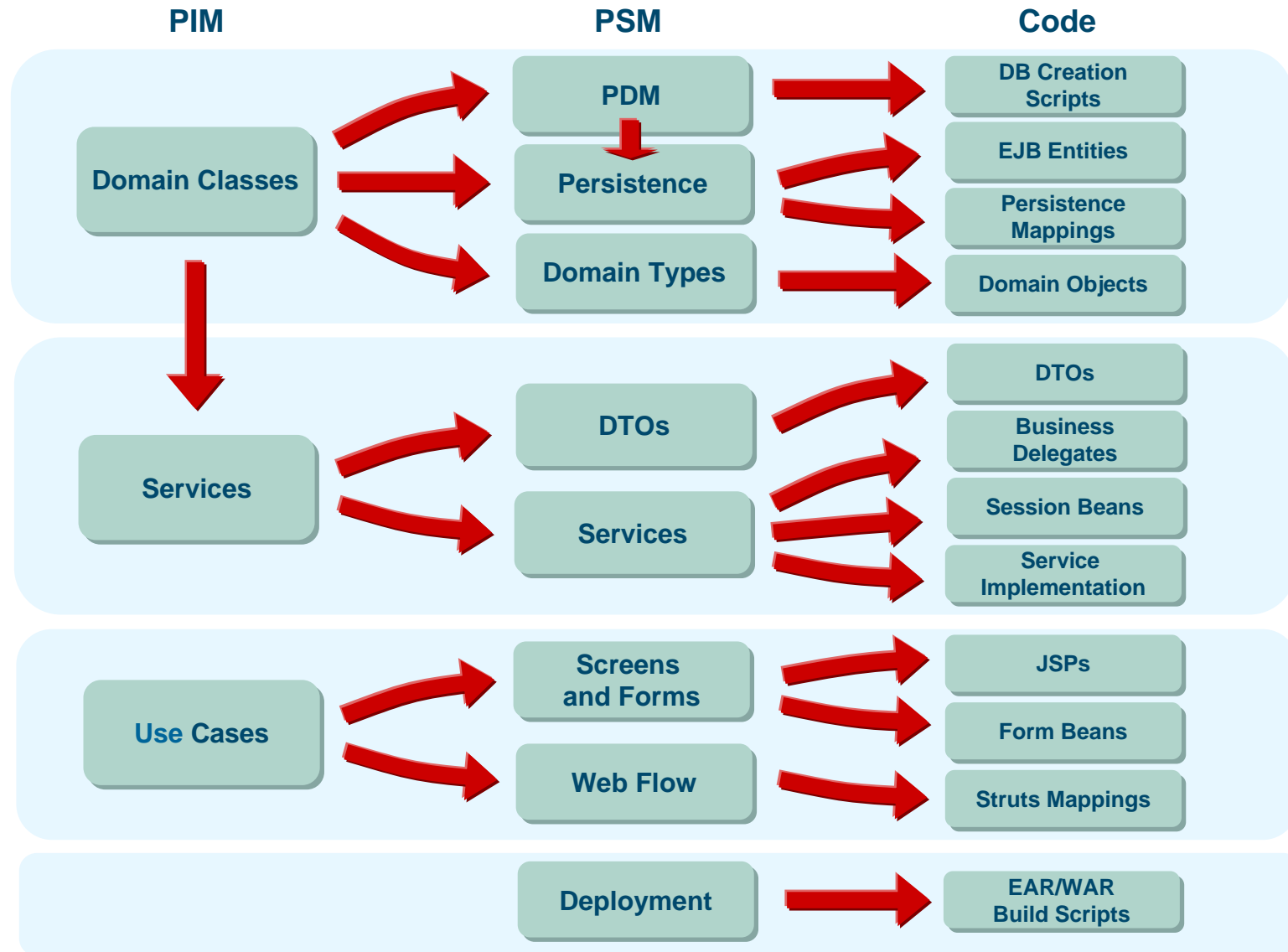
MDD Generated Code

JEE Code Generation



MDD Generated Deployment

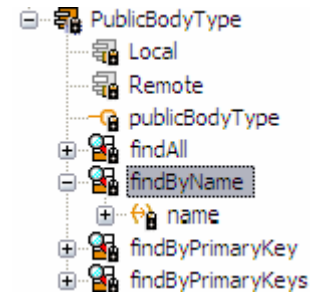
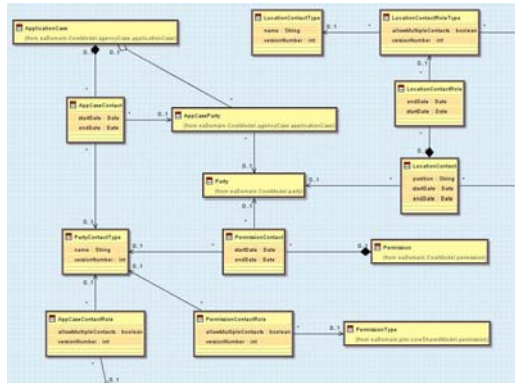
Models and Transformations



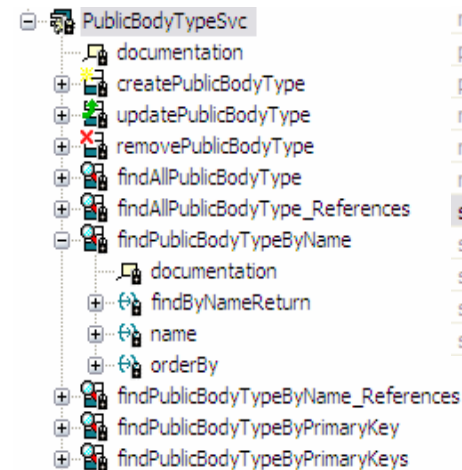
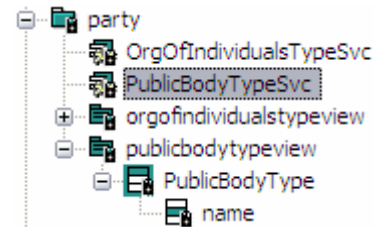


Part 3 - Examples of MDD in use

Entity and Service Generation



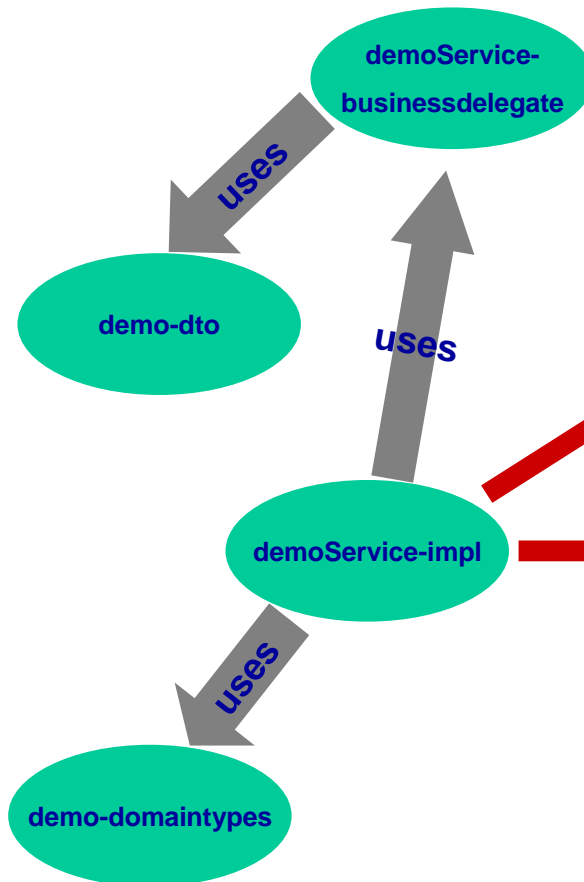
locking	optimistic (CRC)
metaName	EJBEntityComponent
module	coreSharedEntity.p
name	PublicBodyType
producesTo	
published	true



name	PublicBodyTypeSvc
producesTo	
published	true
regenerateProtected	
remoteInterface	coreSharedSvc.p
role	
securityRole	
state	Stateless
supportLazyRetrieve	false
supportPageIterator	false
supportTimerService	false

- Generates EJB2.0
- Models technology-specific points of variation only
- High productivity
 - Lots of code generated
 - Easy to swap technology

Build Generation

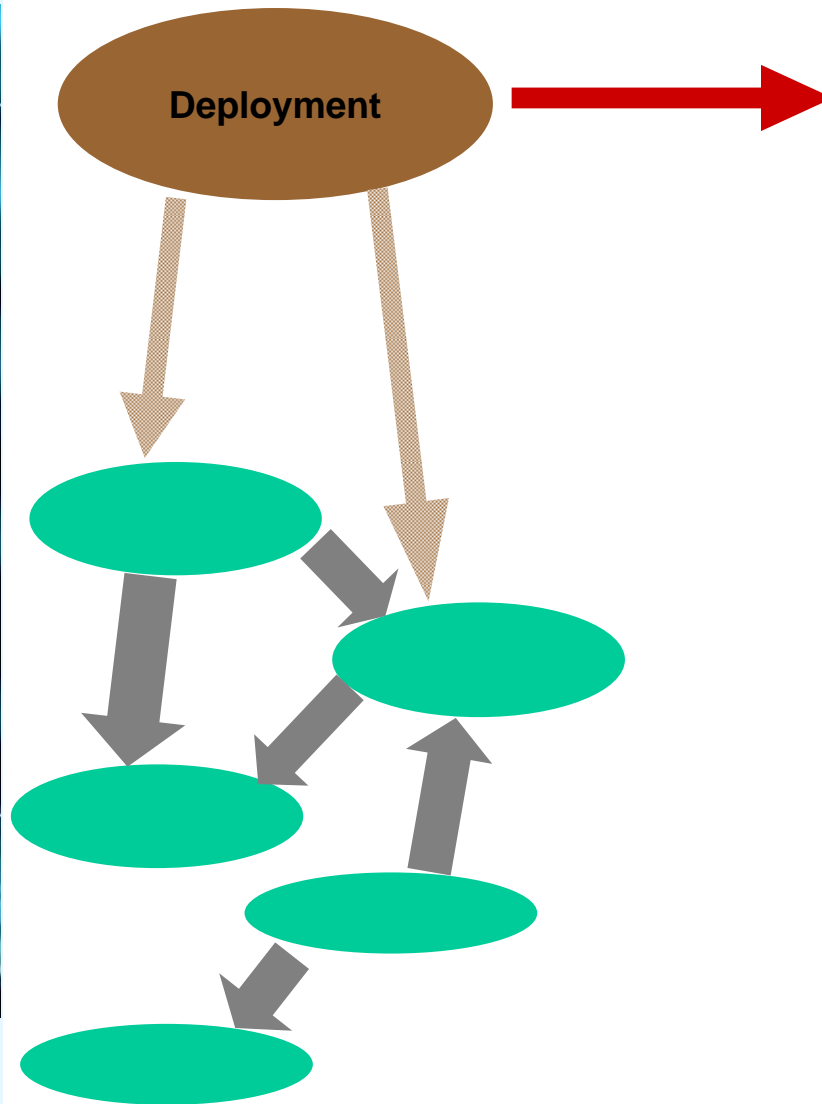


```
<?xml version="1.0" encoding="UTF-8"?>

<project name="uk.co.scisys.demo" default="local-publish">
  <property name="module.dir" value="." />
  <property name="ivy.settings.dir" value="${basedir}"/>
  <property file=".,/ivysettings.properties" />
  <import file="${base.module.dir}/build.xml"/>
</project>
```

```
<ivy-module version="2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://ant.apache.org/ivy/schemas/ivy.xsd">
  <info organisation="uk.co.scisys.demo"
    module="demoService-impl">
    </info>
  <configurations>
    <include file="${master.module.configurations.file}"/>
  </configurations>
  <publications>
    <artifact type="jar" conf="master"/>
    <artifact type="source" ext="jar" conf="master"/>
    <artifact type="javadoc" ext="zip" conf="master"/>
  </publications>
  <!-- additional -->
  </publications>
  <dependencies>
    <!-- Default Conf -->
    <dependency name="demoService-businessdelegate" rev="latest.integration" conf="compile"/>
    <dependency name="demo-domaintypes" rev="latest.integration" conf="compile"/>
  <!-- additional -->
    <dependency org="junit" name="junit" rev="4.1" conf="test"/>
  <!-- additional -->
  </dependencies>
  <!-- additional -->
</ivy-module>
```

Deployment Model & Generation



EAR and WAR artefacts:

- application.xml
- app-server specific deployment descriptors
- web.xml
- many others...

Build artefacts:

EAR & WAR assembly scripts

Screen Model and Generation

- Level of abstraction too close to the code
 - ↳ Model is as detailed as the generated code
 - ↳ No productivity gain over writing the code
- Meta-model is too restrictive:
 - ↳ Unable to model all aspects of screen
 - ↳ E.g. reusable/shareable screen components such as panels
- Quick to initially develop rough screens
 - ↳ but hard to maintain sophisticated screens
- Better GUI development tools available





Part 4 - MDD Issues and considerations

MDD Issues and Considerations

- Selling MDD to your Clients and Management
 - ↪ Resistance to code generation
 - ↪ Hard to explain:
 - Models, meta-models, meta-meta-models, meta-meta-meta-...
 - Model-to-model, model-to-code transformations
 - PIM, PSM, code
 - ↪ Demonstrate MDD
 - ↪ Only tell clients if it is to your advantage

MDD Issues and Considerations

- Which style of MDD/Code Generation Tool?
 - ↳ Out-of-the-box models and transformations
 - ↳ Develop your own meta-models and transformations
 - ↳ Hybrid approach
- What to model and generate and what not to
- Model and code re-generation
 - ↳ Overwrites all existing models and code
 - ↳ Preserves user's changes to model and extensions to generated code
- Costs of adopting MDD
 - ↳ Cost of tools and training
 - ↳ Cost of MDD prototype
 - ↳ Effectively second project to develop meta-models and transformations

MDD Issues and Considerations

- Robustness of tool
 - ↳ Model and code generators must be robust
 - ↳ Developers must like working with tool
 - ↳ Big Models - must cope with models of real-world applications
- Support for concurrent modelling/partitioning of model
- Documentation
 - ↳ For generated application
- MDD tool vision
 - ↳ Build a MDD/code generator for different architectures and technologies from off-the-shelf meta-models and transformations



Part 5 - MDD Advantages

MDD Advantages

- Agility in application development
 - ↳ Ability to quickly develop the application
 - ↳ Ability to re-factor the application following changes in user requirements
 - ↳ Ability to quickly change technology
- Application Maintenance – Simpler and Faster
- Design always kept in line with code
- Increased productivity (Service/EJB tier)
 - ↳ Team of 5 replaced by team of 3 (40% saving)
 - ↳ Development time reduced from 1100 man days to 800 days (25% saving)

Lines of Code Generated

- Total lines of code in application:
 - ↳ 5,519,055 (43,531 files)
- Number of lines of code generated:
 - ↳ 4,101,057 (31,147 files)
- Percentage of application's lines of code generated:
 - ↳ 74% (72% files)



Part 6 - Conclusions

Conclusions

- MDD offers potential for:
 - ↳ Increased productivity and reduced development times and costs
 - ↳ Agility
 - Faster development cycle
 - Able to better respond to changes in requirements
 - Able to respond to changes in technology
 - MDD allowed us to make changes that otherwise would have never been considered

Conclusions

- MDD requires careful thought and planning
 - ↳ Costs – tools, training and development effort
 - ↳ How best to use MDD
 - what to model and generate and what to develop by hand
 - ↳ Your deliverables
 - Will MDD deliver all of them?
 - And if not, how will you create and maintain them?
 - ↳ Choice of MDD/Code Generation Tool and Architecture
 - Out-of-the-box models and transformations
 - Develop your own meta-models and transformations
 - Hybrid approach

Questions



Richard Hensman

richard.hensman@scisys.co.uk

Tony Wilkins

tony.wilkins@scisys.co.uk

www.scisys.co.uk