

Code Generation with openArchitectureWare

Donald Graham



rocketseed

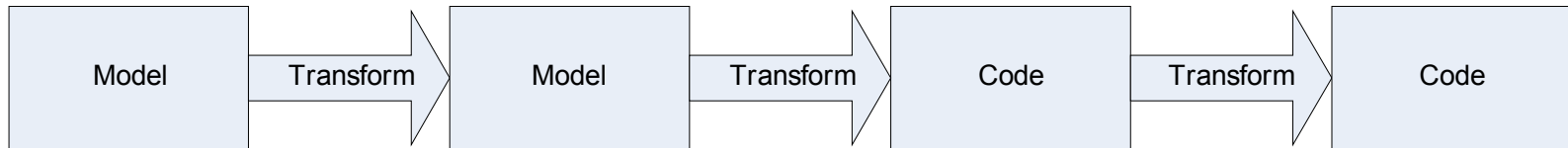
Introduction

- History – war stories
- MDSD – some theory
- openArchitectureWare – an introduction
- Case Study – a data abstraction layer
- Discussion – your ideas

Code Generation

- Wrapper classes – Python script
- Embedded Database – UML
- Second-class citizens
- What kind of code generation experiences have you had?

What is MDSD?



MDSD

Model-Driven Software Development is a software development approach that aims at developing software from domain-specific models.

*Source: Voelter and Bettin,
Patterns for Model-Driven Software-Development*

Pioneers

Markus Voelter

www.voelter.de

openArchitectureWare



Jorn Bettin

www.softmetaware.com



MDA vs MDSD

- MDA is specific to the OMG
- Relies on UML and customizable code generation
- MDA is a kind of MDSD
- MDSD allows for any meta-model

Values and Patterns

- Influenced by Agile Movement
- Many patterns
 - Process and Organisation
 - Domain Modeling
 - Tool Architecture
 - Application Platform Development

Values

We prefer to validate software-under-construction over validating software requirements.

We work with domain-specific assets, which can be anything from models, components, frameworks, generators, to languages and techniques.

Values

We strive to automate software construction from domain models; therefore we consciously distinguish between building software factories and building software applications.

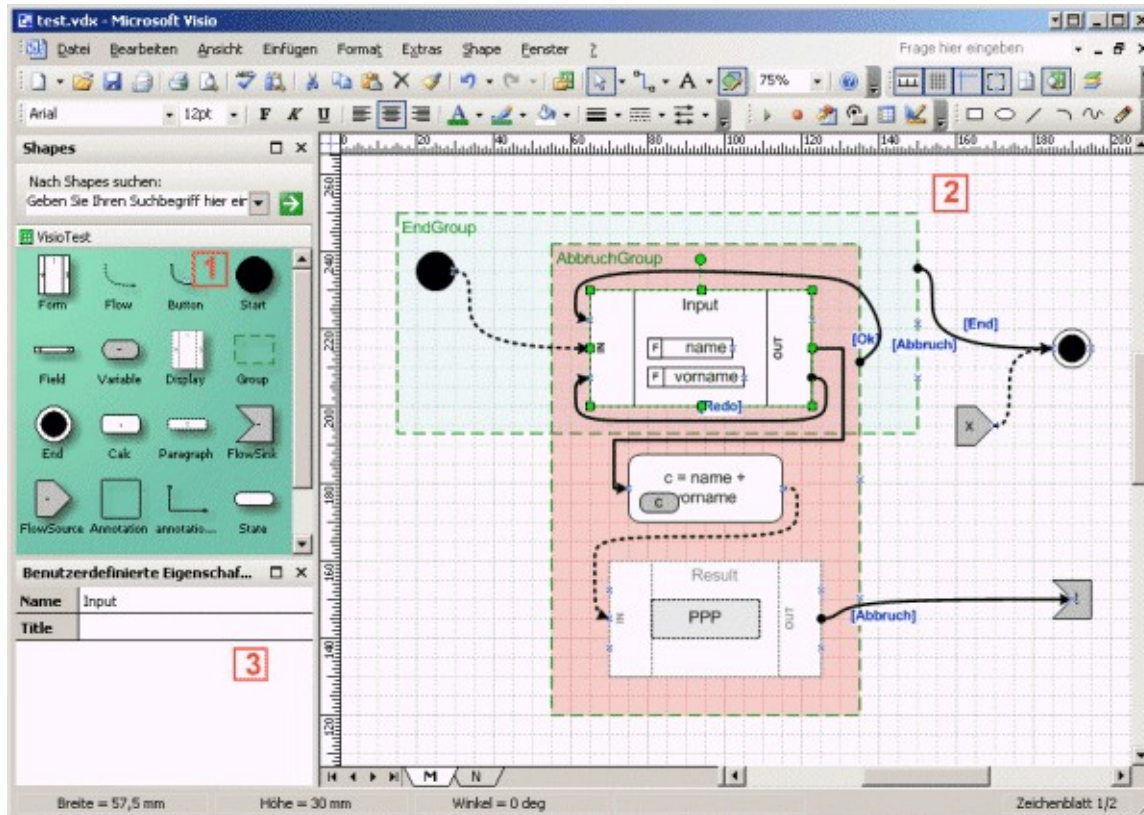
We support the emergence of supply chains for software development, which implies domain-specific specialization and enables mass customization.

Patterns

- Extract the Infrastructure
- Separate generated and non-generated code
- Produce Nice-Looking Code ... Wherever Possible
- Believe in Re-Incarnation
- Transformations as First-Class Citizens

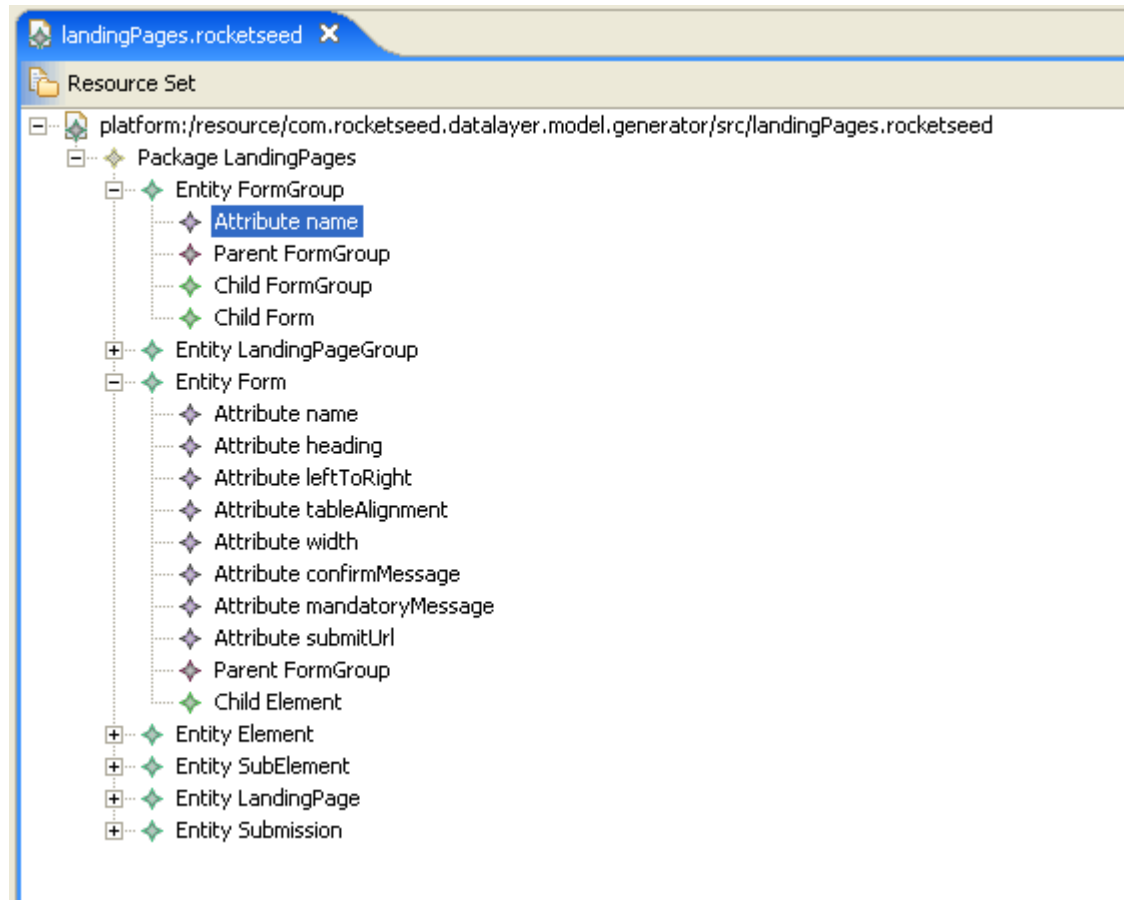
*Source: Voelter and Bettin,
Patterns for Model-Driven Software-Development*

So what is a model?

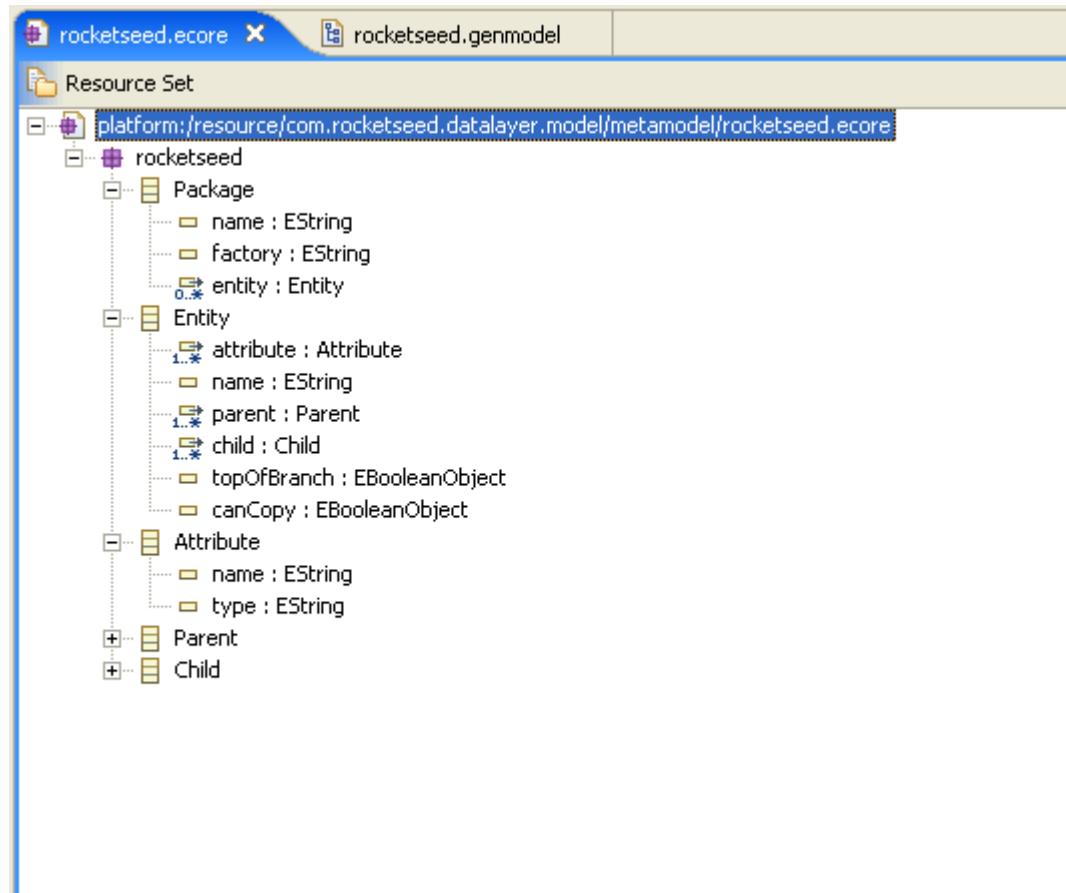


Source: Voelter, openArchitectureWare
Visio integration

A simple model



What's a meta-model?



Why bother with MDSD?

What are the advantages?

More fun!

Less testing required?

Disadvantages

Why don't we write all our code this way?

Is the proliferation of errors a disadvantage?

openArchitectureWare



- Built around Eclipse and EMF
- Wide range of meta-models supported
- Build your own meta-model
- Very flexible tools

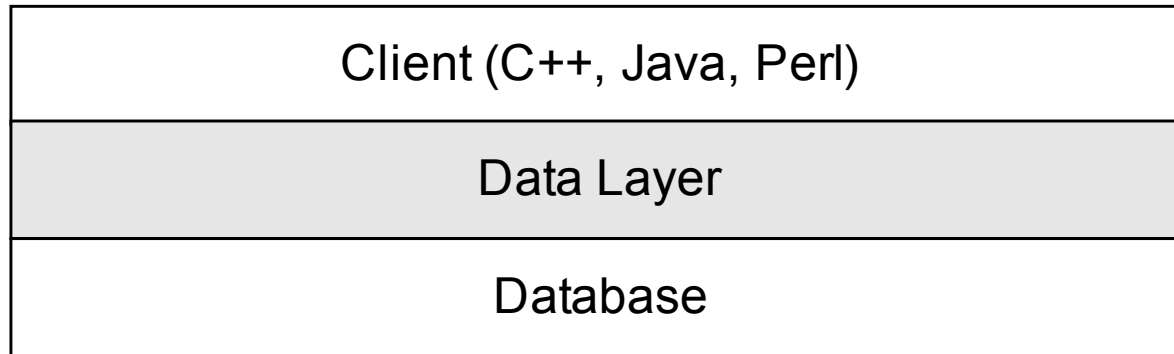
Why oaw?

Why not vanilla EMF?

Why not any old IDE?

Case Study

Data Abstraction Layer

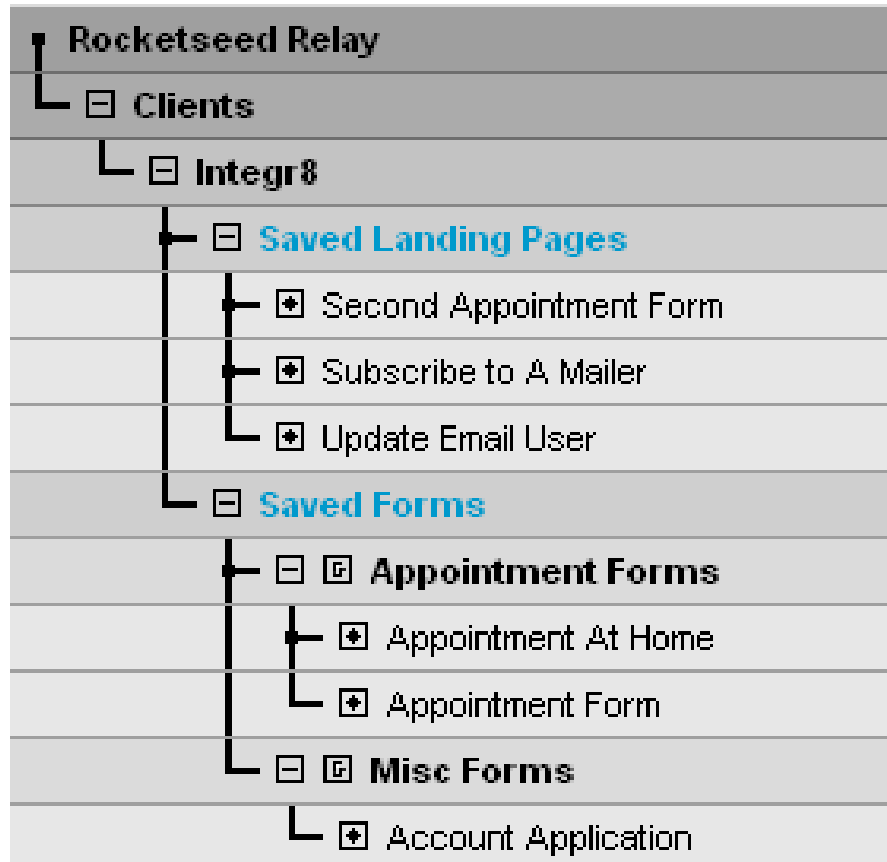


- Object-oriented
- Easy to port application to a new database

Observations

- Thin layer
 - actually, a number of thin layers
- Most of the hard work done by stored procedures
- Very repetitive
- Pilot Project – Landing Pages

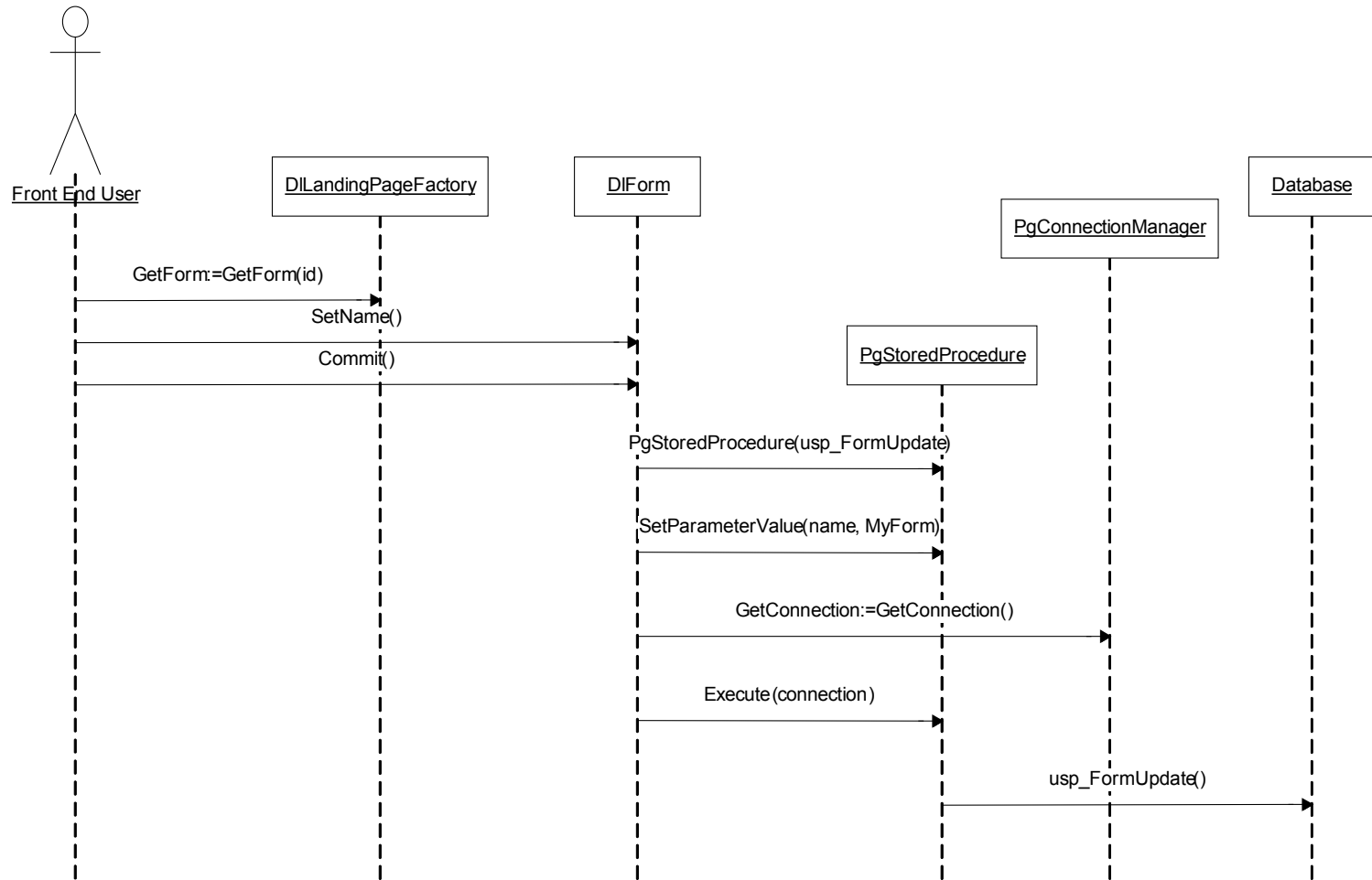
Hierarchical Data



Patterns

- Abstract Factory
 - to create different objects, depending on which database underlies the system
- Proxy
 - to allow for a variety of stored procedure invocation syntaxes

Example: Editing a Form



Hand-write some code

```
bool DlForm::Commit() {
    if (!IsValid()) {
        return false;
    }

    DL_DEBUG(DL_INFO_LEVEL, "Committing changed to DlForm with id: ", GetId());

    DlStoredProcedure sp("mstFormUPD", false, &mErrorHandler);
    sp.SetParameter("id", GetId());
    if (mParentFormGroup != NULL) {
        mParentFormGroupId = mParentFormGroup->GetId();
        DL_DEBUG(DL_INFO_LEVEL, "ParentFormGroupId set to: ", mParentFormGroupId);
    }

    sp.SetParameter("parentFormGroupId", mParentFormGroupId);

    sp.SetParameter("name", mName.c_str());

    // ..... //

    sp.SetParameter("submitUrl", mSubmitUrl.c_str());

    DlResultSet resultSet = sp.Execute();
    return IsValid();
}
```

Then what?

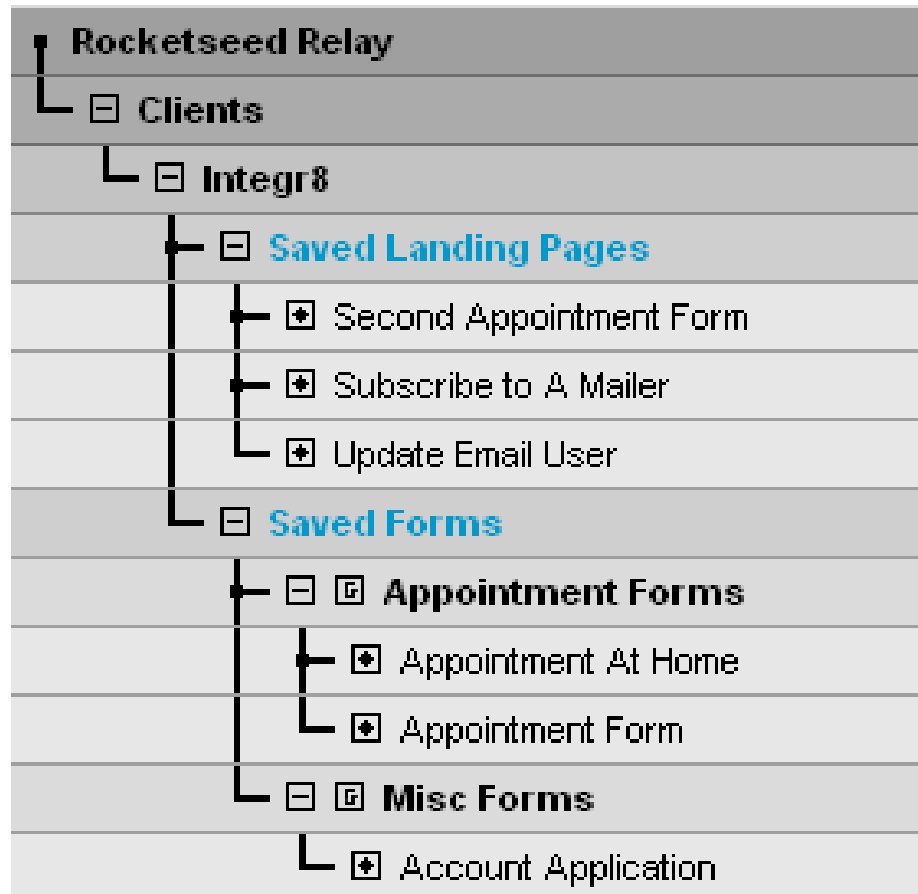
- Review
- Refactor
- Review
- Refactor
- Test
- Review
- Refactor
- Test

Refactor, refactor

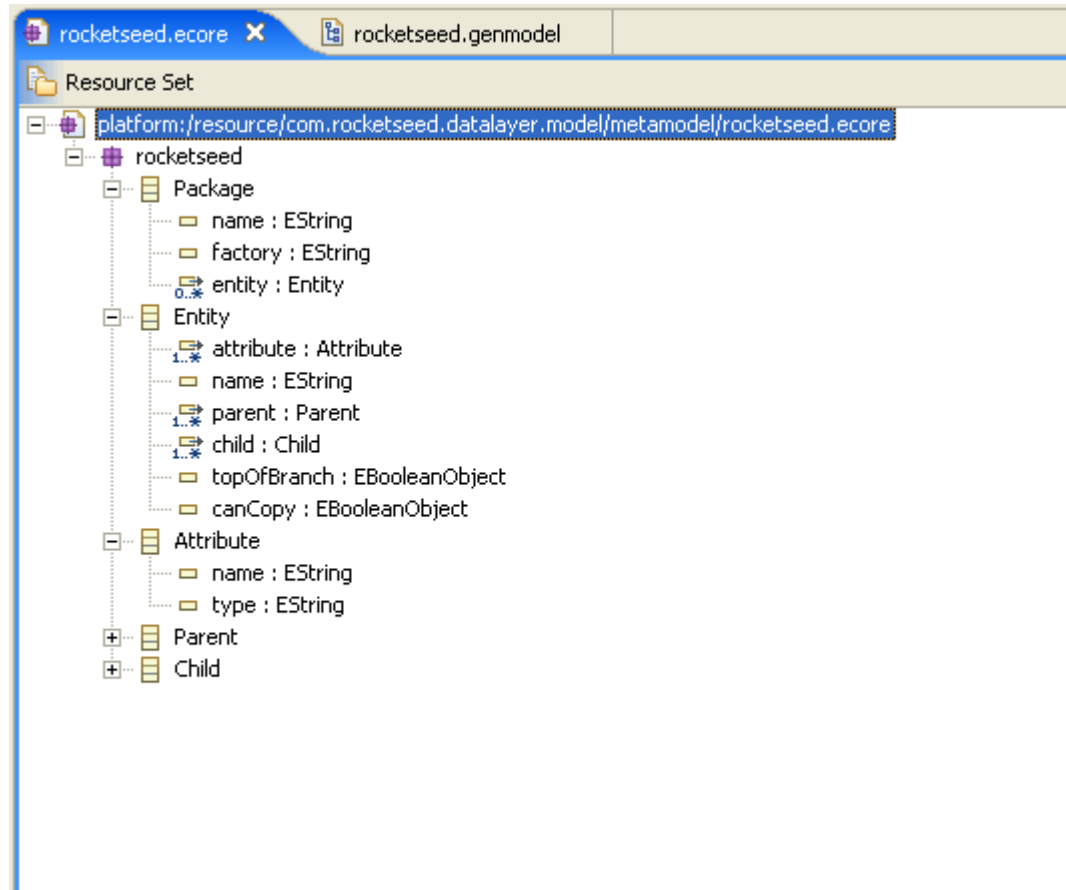
```
DlForm::DlForm(unsigned int id) {  
    DL_DEBUG(DL_INFO_LEVEL, "DlForm created with id: ", id);  
  
    InitialiseMembers();  
    SetId(id);  
    RetrievePropertyValues();  
}
```

Is more generated code a good thing?

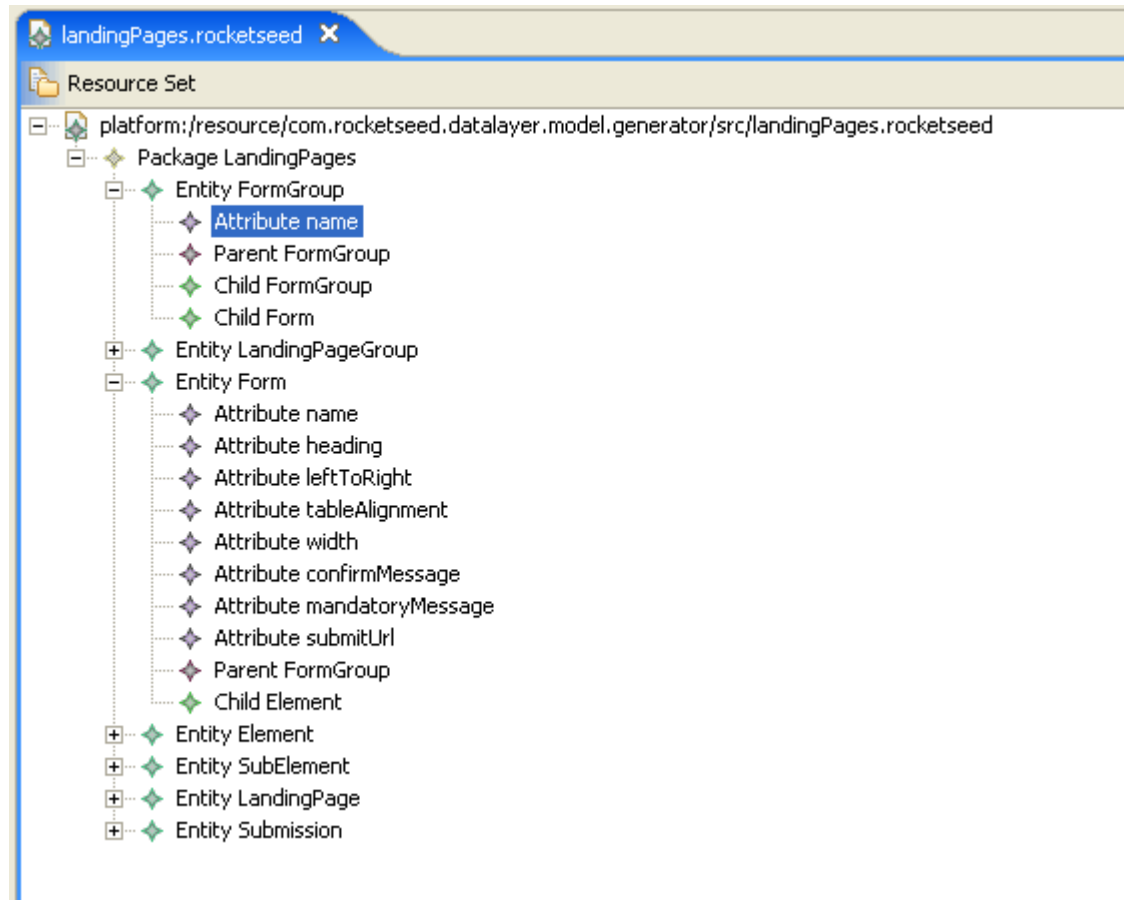
Extract meta-model



Extract meta-model



Construct model



Build templates – Xpand2

```
«DEFINE Entity FOR rocketseed::Entity»
    «FILE headerFileName()»
#ifdef «defineName()»
#define «defineName()»

#include "DlTypes.h"
#include "DlObject.h"

«EXPAND DeclareParent(interfaceName()) FOREACH parent-»
«EXPAND DeclareChild(interfaceName()) FOREACH child-»

class «interfaceName()» : public DlObject {

public:
    «EXPAND GetAndSetAttribute FOREACH attribute-»
    «EXPAND GetAndSetParent FOREACH parent-»
    «EXPAND GetChild FOREACH child-»
«IF canCopy »
    // Make a copy of this object in the database and return the new copy's id
    virtual «interfaceName()»* MakeCopy() = 0;
«ENDIF»
};

#endif // «defineName()»
```

Build templates – Expressions

```
String headerFileName(Entity e) :
    'implementation/' + className(e) + '.h';

String implementationFileName(Entity e) :
    'implementation/' + className(e) + '.cpp';

String defineName(Entity e) :
    className(e).toUpperCase() + '_H_';

String getParent(Parent p) :
    'I' + p.type + '* GetParent' + p.type + '()';

String setParent(Parent p) :
    'void SetParent' + p.type + '(I' + p.type + '* parent' + p.type + ')';
```

... and the code

```
#ifndef IFORM_H_
#define IFORM_H_

#include "DITypes.h"
#include "DIObject.h"

class IFormGroup;

class IElement;

class IForm :
public DIObject
{
public:
    virtual const char* GetName() = 0;

    virtual void SetName(const char* name) = 0;

    virtual const char* GetHeading() = 0;

    virtual void SetHeading(const char* heading) = 0;

// ..... //

    virtual IFormGroup* GetParentFormGroup() = 0;

    virtual void SetParentFormGroup(IFormGroup* parentFormGroup) = 0;

    virtual IElement** GetChildElements() = 0;

    // Make a copy of this object in the database and return the new copy's id
    virtual IForm* MakeCopy() = 0;
};
#endif // IFORM_H_
```

Another template

```
«className()»::«className()»(unsigned int id) {  
    DL_DEBUG(DL_INFO_LEVEL, "«className()» created with id: ", id);  
  
    InitialiseMembers();  
    SetId(id);  
    RetrievePropertyValues();  
}
```

... and the code

```
DlForm::DlForm(unsigned int id) {  
    DL_DEBUG(DL_INFO_LEVEL, "DlForm created with id: ", id);  
  
    InitialiseMembers();  
    SetId(id);  
    RetrievePropertyValues();  
}
```

Do you recognise this?

```
bool <<className()>>::Commit() {
    if (!IsValid()) {
        return false;
    }
    DL_DEBUG(DL_INFO_LEVEL, "Committing changed to <<className()> with id: ", GetId());

    D1StoredProcedure sp("mst<<name>>UPD", false, &mErrorHandler);
    sp.SetParameter("id", GetId());
    <<EXPAND SetParentId FOREACH parent>>
    <<EXPAND SetAttributeCommit FOREACH attribute->>
    D1ResultSet resultSet = sp.Execute();
    return IsValid();
}
```

In case you don't ...

```
bool DlForm::Commit() {
    if (!IsValid()) {
        return false;
    }

    DL_DEBUG(DL_INFO_LEVEL, "Committing changed to DlForm with id: ", GetId());

    DlStoredProcedure sp("mstFormUPD", false, &mErrorHandler);
    sp.SetParameter("id", GetId());
    if (mParentFormGroup != NULL) {
        mParentFormGroupId = mParentFormGroup->GetId();
        DL_DEBUG(DL_INFO_LEVEL, "ParentFormGroupId set to: ", mParentFormGroupId);
    }

    sp.SetParameter("parentFormGroupId", mParentFormGroupId);

    sp.SetParameter("name", mName.c_str());

    // ..... //

    sp.SetParameter("submitUrl", mSubmitUrl.c_str());

    DlResultSet resultSet = sp.Execute();
    return IsValid();
}
```

So what can we generate?

- Factory and Factory Methods
- Header files
- Implementations
- Perl wrapper – SWIG input file
- Java wrapper – SWIG input file
- Makefile
- Unit tests
 - Why bother?

SWIG input file - Perl

```
// SubElement

%newobject LandingPageFactory::GetSubElement(unsigned int id);
%newobject LandingPageFactory::CreateSubElement(IElement* parentElement);

// LandingPage

%newobject LandingPageFactory::GetLandingPage(unsigned int id);
%newobject LandingPageFactory::CreateLandingPage(ILandingPageGroup* parentLandingPageGroup);

// Submission

%newobject LandingPageFactory::GetSubmission(unsigned int id);
%newobject LandingPageFactory::CreateSubmission(ILandingPage* parentLandingPage);

%{
#include "LandingPageFactory.h"
#include "IFormGroup.h"
#include "ILandingPageGroup.h"
#include "IForm.h"
#include "IElement.h"
#include "ISubElement.h"
#include "ILandingPage.h"
#include "ISubmission.h"
%}

// Grab the declarations from hpp file
#include "../../common/include/DlCreator.h"
#include "../../common/include/DLObject.h"
#include "../../src/interface/LandingPageFactory.h"
#include "../../src/interface/IFormGroup.h"
#include "../../src/interface/ILandingPageGroup.h"
```

Enhancements

- A real domain specific language – not just EMF
- Generating stored procedures, etc
- Front end – MVC/HTML
- Any ideas?

Recap – Case Study

- Already worth the investment – many times over
- Only just starting to reap the benefits
- No silver bullet
- Is this “real” MDSD?
 - How could real MDSD help this project?

Conclusion

- A success story
- Some theory
- History
- Would you like to try this?
 - On what project?

Questions and Suggestions