An Open Modeling Infrastructure

Olaf Kath
Marc Born
Agenda

- Architecture Guidelines
- Users View
- Medini Toolset
- Model Transformations
- Conclusions
Our Vision on MDA: A Modeling Infrastructure open to a variety of modeling and model transformation techniques and open to different domains
Architecture Guidelines

- All modeling instruments and principles are defined by models themselves, model databases are derived
  - We use the Meta Object Facility (MOF) for this purpose
- Based on these metamodels, domain specific visual modeling languages are defined
  - We use different techniques here, e.g. UML based languages, textual languages, new graphical languages
- Also based on the metamodels, further processes like code generation or model transformations are defined
  - We keep notation and these processes separate
Architecture Guidelines: Concepts, Notations and Mappings

- Definition of modeling concepts
- Notation guide for concepts
- Automatic derivation of model databases
- Model database
- Model transformation rules
- Model transformators or code generators
- Apply notation guides to express concepts in models
- Apply transformators for automatic system construction
If the class is not abstract, an IDL component definition is being generated in the same module as the abstract interface with the name concatenate ( format_1 ( <class identifier> ), "Component" ). The component is declared to support the generated abstract interface for the class.

If the class is not abstract, a home interface declaration for the component with the name concatenate ( format_1 ( <class identifier> ), "Home" ) is being generated, managing the component generated following Rule (9). This home interface contains a factory operation to create a component without parameters with the name concatenate ( "create_", format_2 ( <class identifier> ) ).

**Constraints in OCL**

```
if not self.isAbstract then
    self.c_target->size() = 1 and self.home_target->size() = 1 and
    self.c_target.identifier = concat( format_1( self.identifier ), "Component") and
    self.home_target.identifier = concat( format_1( self.identifier ), "Home") and
    self.container.target = self.c_target.definedIn and
    self.container.target = self.home_target.definedIn and
    self.home_target.manages = self.c_target and
    self.c_target.supports = self.target
else
    self.c_target->size() = 0 and self.home_target->size() = 1 and
    Class.allInstances()->forAll( c | not c.isAbstract implies
        c = self.home_target.factory and
        c.identifier = concat( "create_", format_2(self.identifier) ) )
endif
```
Users View:

PIM Modeling using EDOC

Rational Rose based Tool for EDOC PIM Modeling

Rational Rose based Tool for OSP/CCM PSM Modeling

eclipse Programming Environment

EDOC Model Database

CCM/OSP Model Database

modell transformators
Users View:
PSM Refinement using Modeling Tool

Rational Rose based Tool for EDOC PIM Modeling

Rational Rose based Tool for OSP/CCM PSM Modeling

eclipse Programming Environment

EDOC Model Database

OSP/CCM Model Database

modell transformers
Users View:
Generation of Java Code out of the PSM

Rational Rose based Tool for EDOC PIM Modeling

Rational Rose based Tool for OSP/CCM PSM Modeling

eclipse Programming Environment

EDOC Model Database

OSP/CCM Model Database

modell transformers
Tools supporting the Modeling Infrastructure

Automatically generate model databases out of the metamodels, using IKVs medini tool chain.

These model databases support mechanisms for:
- transactions
- model change notifications
- XML import/export
- ...

EDOC

<<metamodel>>

PIM Model Database

enago OSP

<<metamodel>>

PSM Model Database
Applying the Medini tool set

**Rules in English**

9] If the class is not abstract, an IDL component definition is being generated in the same module as the abstract interface with the name concatenate ( format_1 ( <class identifier> ), "Component" ). The component is declared to support the generated abstract interface for the class.

11] If the class is not abstract, a home interface declaration for the component with the name concatenate ( format_1 ( <class identifier> ), "Home" ) is being generated, managing the component generated following Rule (9). This home interface contains a factory operation to create a component without parameters with the name concatenate ( "create_", format_2 ( <class identifier> ) ).

**Constraints in OCL**

9,11] if not self.isAbstract then
    self.c_target->size() = 1 and self.home_target->size() = 1 and
    self.c_target.identifier = concat( format_1( self.identifier ), "Component") and
    self.home_target.identifier = concat( format_1( self.identifier ), "Home") and
    self.container.target = self.c_target.definedIn and
    self.container.target = self.home_target.definedIn and
    self.home_target.manages = self.c_target and
    self.c_target.supports = self.target
  else
    self.c_target->size() = 0 and self.home_target->size() = 1 and
    Class.allInstances()->forAll( c | not c.isAbstract implies
      c = self.home_target.factory and
      c.identifier = concat( "create_", format_2(self.identifier) ) )
  endif
Infrastructure: Generation of Repositories

develop the meta model with UML

generate

MOF
Meta-Meta-Model

Any Metamodel

Any Model

Model Database

store and access
// IDL

interface ProcessComponentDef
: ProcessComponentDefClass,
 PortOwnerDef,
 CompositionDef,
 UsageContextDef,
 ECA::ModelManagement::PackageDef
{
    ECA::CCA::GranularityKindDef granularity ()
        raises ( Reflective::MofError );
    void set_granularity ( in
        ECA::CCA::GranularityKindDef new_value)
        raises ( Reflective::MofError );
    boolean is_persistent ( )
        raises ( Reflective::MofError );
    // ...
};
Generation of the EDOC Repository
(cont.)

class ProcessComponentDefServant
: virtual public
ProcessComponentDefClassServant,
virtual public PortOwnerDefServant,
virtual public CompositionDefServant,
virtual public UsageContextDefServant,
virtual public ModelManagement::PackageDefServant
{
public:
    virtual ECA::CCA::GranularityKindDef_Ptr
granularity ( )
    {
        virtual bool is_persistent ( );
        virtual void set_is_persistent (bool new_value);
        void set_granularity ( in ECA::CCA::GranularityKindDef new_value);
    };
// ...
}
EDOC Model Repository

- Repositories are generated with:
  - MOF2IDL Generator
  - MOF2C++ Generator

- EDOC repository implements the generated CORBA IDL interfaces

- Repository Control Center provides
  - Debugging and Monitoring capabilities
  - Serialization to XML
  - In the future: Control over Medini extensions
Medini Extensions to repositories

- Repository implements the generated CORBA IDL interfaces plus:
  - Transaction capabilities
  - Persistency
  - Medini Extensions under development

- **Consistency Management** through active communication of model changes as Notifications
  - Medini ActiveRepository Service (MARS)

- **Validity Maintenance**: repository checks meta-model constraints at run-time
Model Transformations

MOF Meta Meta Model

Source Meta Model e.g. Business Meta Model
Define Formal Transformation Rules
Target Meta Model e.g. Platform Meta Model

Source Model e.g. Business Model
Apply Transformation Rules
Target Model e.g. Platform Specific Software Components
Transformations from EDOC to OSP/CCM

- Transformations are defined between meta-models
  - EDOC meta-model <> enago OSP/CCM meta-model
  - They specify, how any EDOC compliant model (i.e. instances of the EDOC meta-model) is transformed to the OSP/CCM platform

- Transformations are by itself instances of a (transformation-)meta-model
  - this meta-model determines, how to define the model-transformations
  - currently no OMG standard exists (initial submission deadline has just passed)
Approaches on transformations

- **Rule based transformations**
  - Specify how to construct a target from a source model
  - Example language: UML ActionSemantics

- **Declarative transformations**
  - Specify only the result of a transformation (i.e. as invariants and postconditions)
  - Example language: Object Constraint Language (OCL)

- Other approaches use templates or scripting/semiformal languages
Approaches on transformations (cont.)

• Rule based transformations
  - Specify how to construct a target from a source model
  - Example language: UML ActionSemantics

• Advantages:
  - Significant and intuitive formalism
  - Simple implementation derivation

• Disadvantages:
  - Lack of (automatic) testability
  - Rules are only “one-way”
    • Changes late in the software process can not be propagated back
Approaches on transformations (cont.)

• **Declarative transformations**
  - Specify only the result of a transformation (i.e. as invariants and postconditions)
  - Example language: Object Constraint Language

• **Advantages:**
  - Testability & Traceability
  - “Two-way” transformations

• **Disadvantages:**
  - Complex formalism (error-prone)
  - Implementation more difficult
Our Approach for Transformations

- Three steps towards a **MOF Transformator Generator** (MTG):
  1. Specification and development of a Transformator Core Generator ("Skeleton Generator")
     - independent of any transformation concepts
     - produces meta-model specific transformers (for EDOC and OSP)
  2. Development of an EDOC to OSP/CCM transformator based on generated skeleton
     - Parallel implementation of code generators
     - Realization of traceability, testability, and parameterization (or interaction)
The MOF Transformer Generator (MTG)

- Specification and development of a Transformer Core Generator ("Skeleton Generator")
  - First part of the whole Generator
  - Input:
    - Source (EDOC) or target (OSP/CCM) meta-models
  - Output:
    - meta-model specific Skeletons ("Walker") for model-transformations (EDOC) or code-generation (OSP/CCM)
  - Status: The Skeleton Generator produces already utilizable transformers for half-automated development
Operation of the generated Transformer Skeleton

Skeleton „walks“ over source tree

transformation_procedure_1 ()
transformation_procedure_2 ()
transformation_procedure_3 ()

EDOC Source Model
M1-level Instance
(in repository)

OSP/CCM Target Model
M1-level instance
(in repository)
Transformations: final step (vision)

- Replacement of hand-written code with evaluated transformation rules
  - Rule based or declarative approach for the transformations (possibly a combination)
  - Gained experience from FhG FOKUS / IKV++ joint submission to the MOF2.0-IDL Request For Proposal and OMG process will influence the decision
Advantages of our approach

- **Platform independence** through a formal specification of the transformations

- **Testability**
  - rule and/or constraint based specification of transformations allow for validation and/or verification

- **Traceability**
  - the trace of elements is easily extractable from transformation models

- Highest **flexibility**, late determination of transformation model
Conclusion

- Medini tool set supports infrastructure
  - Integration of modeling tools, transformations
  - Extensions during the project needed

- Transformations will be one essential
  - Already realized part of the
  - Possibly combined approach of rule and constraint based specification in future