TECHNISCHE UNIVERSITÄT DRESDEN

ResUbic

QualiFune

Fakultät Informatik Institut für Software- und Multimediatechnik - Lehrstuhl für Softwaretechnologie

Elucidative Development for Model-Based Documentation

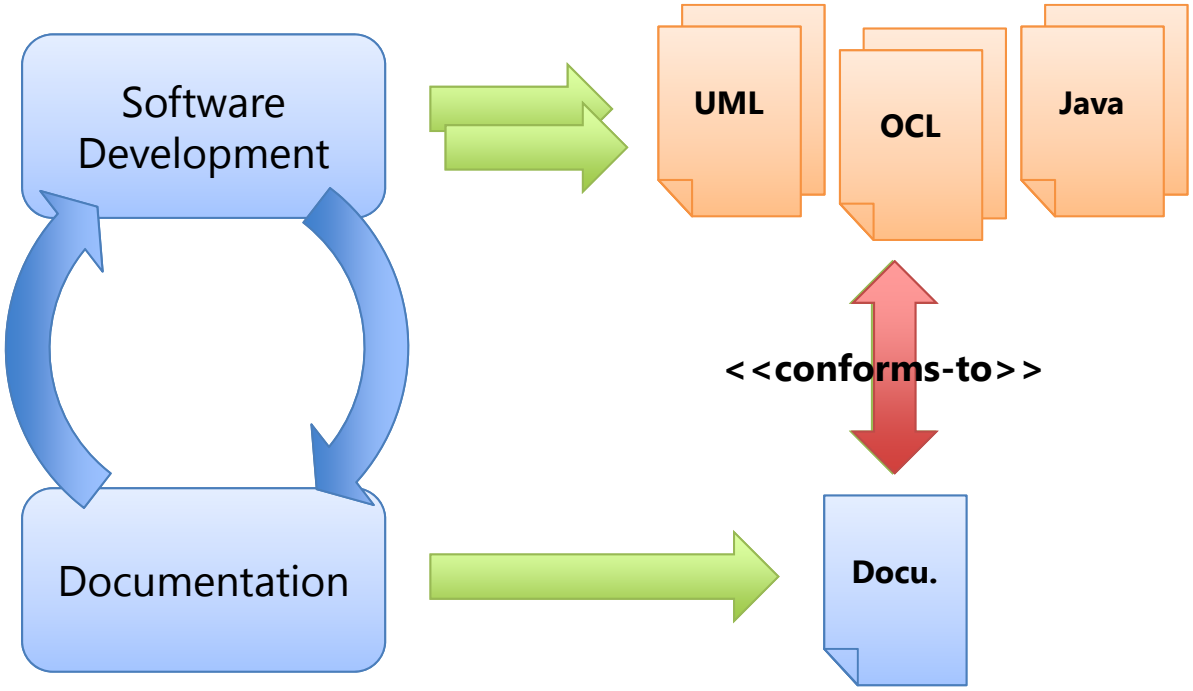
Claas Wilke, Andreas Bartho, Julia Schroeter, Sven Karol, and Uwe Aßmann

29.05.2012

Europa fördert Sachsen. ESF Europäischer Sozialfonds SACHSEN

DRESDEN concept Exzellenz aus Wissenschaft und Kultur

PROBLEM


Software Development
Documentation
UML
OCL
Java
<<conforms-to>>
Docu.

PROBLEM

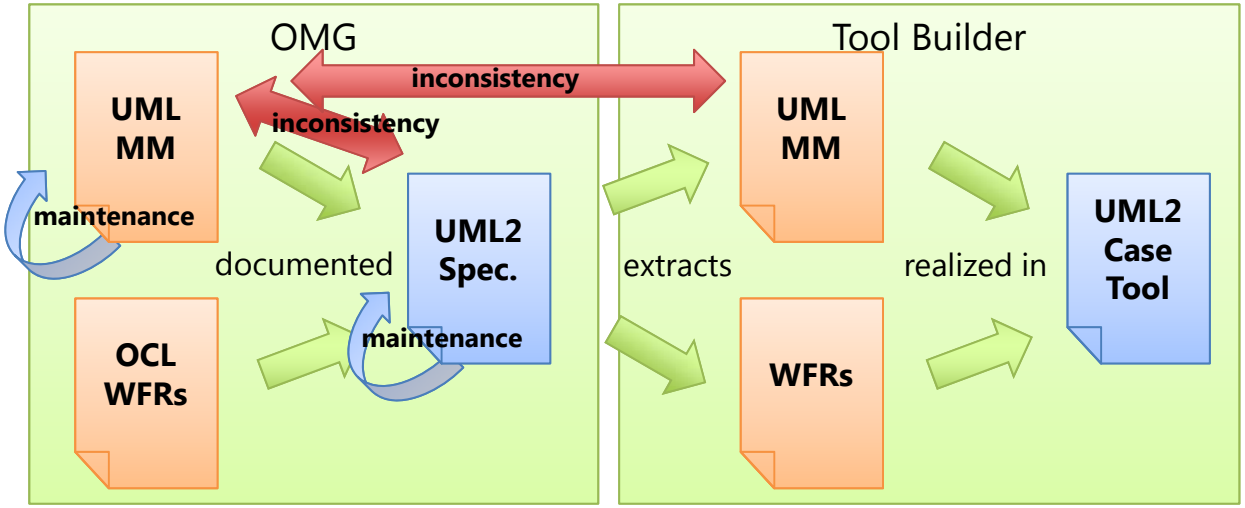
- **Development** and **documentation** activities are **separated**
  - Often, **documentation** is a **postprocess**

- **Documentation** is **maintained manually**

  → **Inconsistencies** are typical

  - Howtos, **code examples**
  - Tutorials, **screenshots**
  - Model examples, **diagrams**
  - Manuals, **specifications**
  - **…**

AN EXAMPLE: UML SPECIFICATION



Target:

- **Consistent** metamodel and **specification**
- **Release** of both **metamodel and specification**
- **Shorter** revision/maintenance **cycles**

AN EXAMPLE: UML SPECIFICATION

- Each **metaclass** has ist own **section**
  - Class diagram
  - Generalizations
  - Description
  - Attributes
  - Associations
  - Constraints
  - Additional Operations

…

Constraints

…

Additional Operations

[1] The query mustBeOwned() indicates whether elements of this type must have an owner.

Package::mustBeOwned() : Boolean

mustBeOwned = false

…

AN EXAMPLE: UML SPECIFICATION

Identified maintenance **problems:**

1. **Graphical representations**
2. **Continous text**
3. **Textual representations**
4. **External references**

Additional Operations

…

[3] The query makesVisible() …

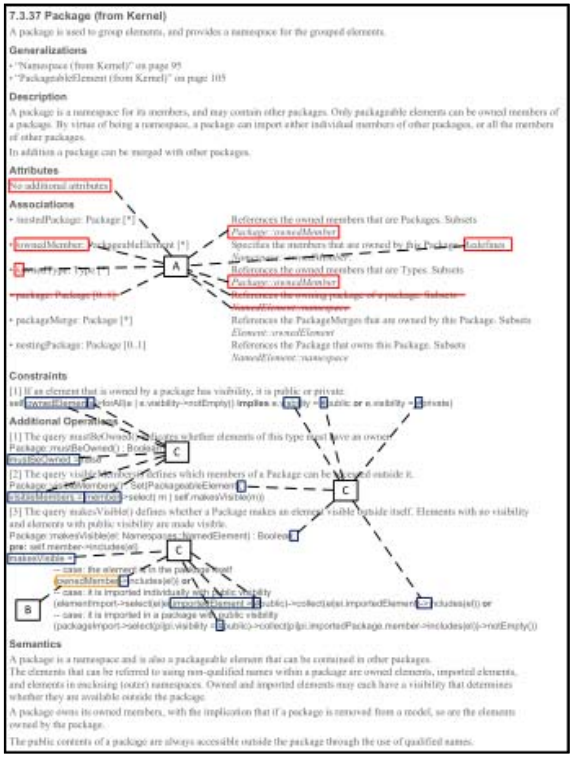
```
pre: self.member->includes(el)
makesVisible =
    (ownedMember->includes(el)) or
    (elementImport>select(ei|
            ei.importedElement = #public)
            ->collect(ei|ei.importedElement)
            ->includes(el)) or
```

…

TECHNISCHE UNIVERSITÄT DRESDEN

AN EXAMPLE: UML SPECIFICATION

- First version (**2.0**) released **in 2005**
- **Since then**
  - **Continuous evolution**
  - **Manual maintenance**
  - Several **inconsistencies**


7.3.37 Package (from Kernel)

A package is used to group elements, and provides a namespace for the grouped elements.

Generalizations

- "Namespace (from Kernel)" on page 95
- "PackageableElement (from Kernel)" on page 105

Description

A package is a namespace for its members, and may contain other packages. Only packageable elements can be owned members of a package. By virtue of being a namespace, a package can import either individual members of other packages, or all the members of other packages.

In addition a package can be merged with other packages.

Attributes

No additional attributes

Associations

- importedPackage: Package [*]
- ownedMember: PackageableElement [*]
- ownedType: Type [*]
- package: Package [0..1]
- packageMerge: PackageMerge [*]
- nestingPackage: Package [0..1]

References the owned members that are Packages. Subsets Package::ownedMember

Specifies the members that are owned by this Package. Subsets Namespace::ownedMember

References the owned members that are Types. Subsets Package::ownedMember

References the owning package of a package. Subsets NamedElement::namespace

References the PackageMerges that are owned by this Package. Subsets Element::ownedElement

References the Package that owns this Package. Subsets NamedElement::namespace

Constraints

[1] If an element that is owned by a package has visibility, it is public or private.

Additional Operations

[1] The query mustBeOwned() indicates whether elements of this type must have an owner.

Package::mustBeOwned() : Boolean

[2] The query visibleMembers() defines which members of a Package can be visible outside it.

Package::visibleMembers() : Set{PackageableElement}

[3] The query makesVisible() defines whether a Package makes an element visible outside itself. Elements with no visibility and elements with public visibility are made visible.

Package::makesVisible(e: NamespaceOwnedElement) : Boolean

Semantics

A package is a namespace and is also a packageable element that can be contained in other packages.

The elements that can be referred to using non-qualified names within a package are owned elements, imported elements, and elements in enclosing (outer) namespaces. Owned and imported elements may each have a visibility that determines whether they are available outside the package.

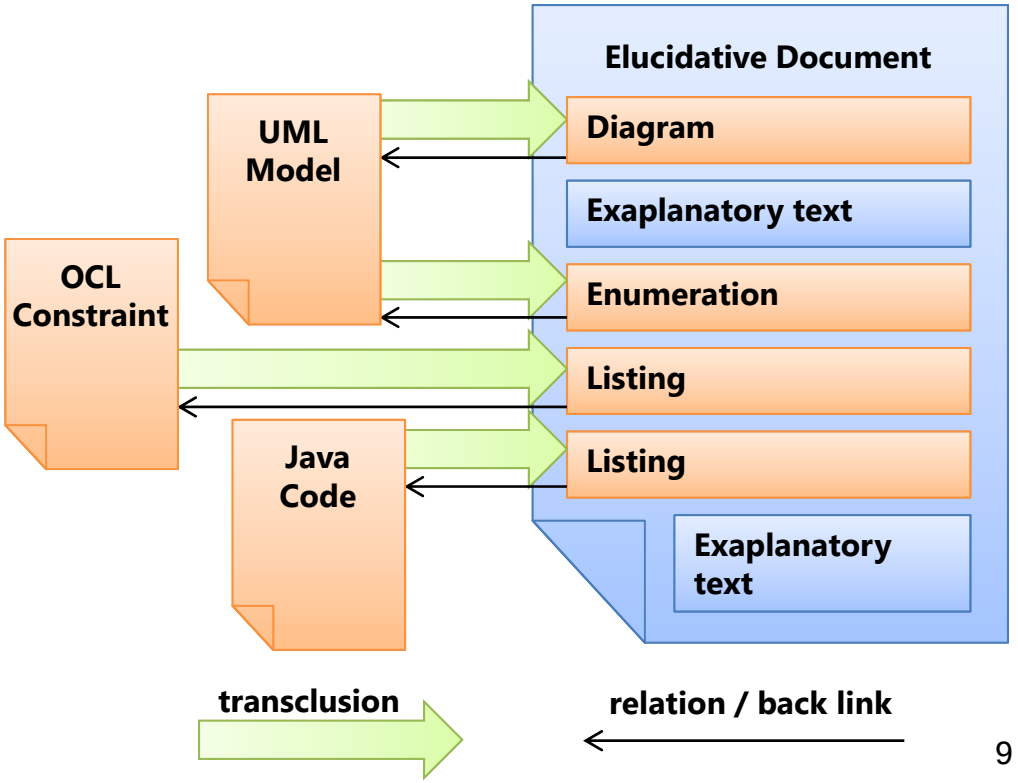
A package owns its owned members, with the implication that if a package is removed from a model, so are the elements owned by the package.

The public contents of a package are always accessible outside the package through the use of qualified names.

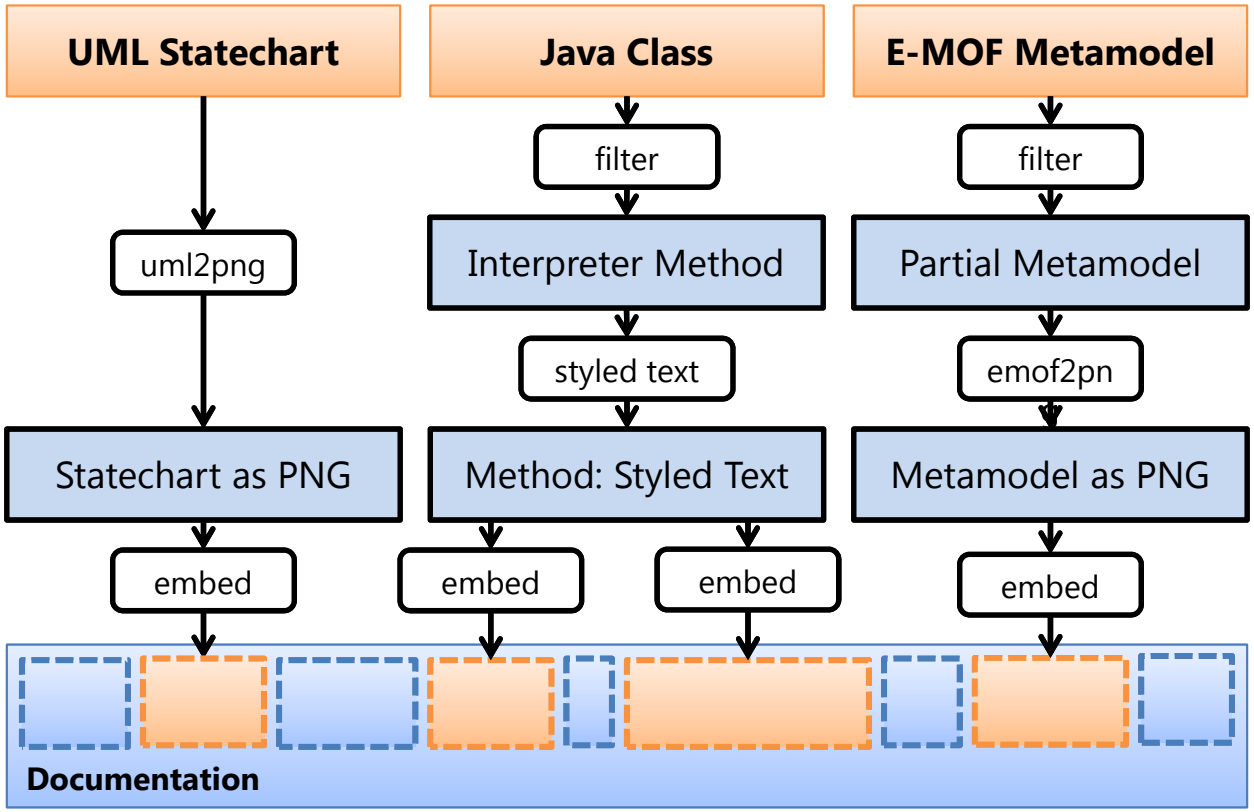
ELUCIDATIVE DEVELOPMENT

ELUCIDATIVE DEVELOPMENT

- **Transclusion:** content propagation from one document to another
- **Transconsistency:** updates are triggered when artifacts change and changes are transcluded to the document (**hot update**)

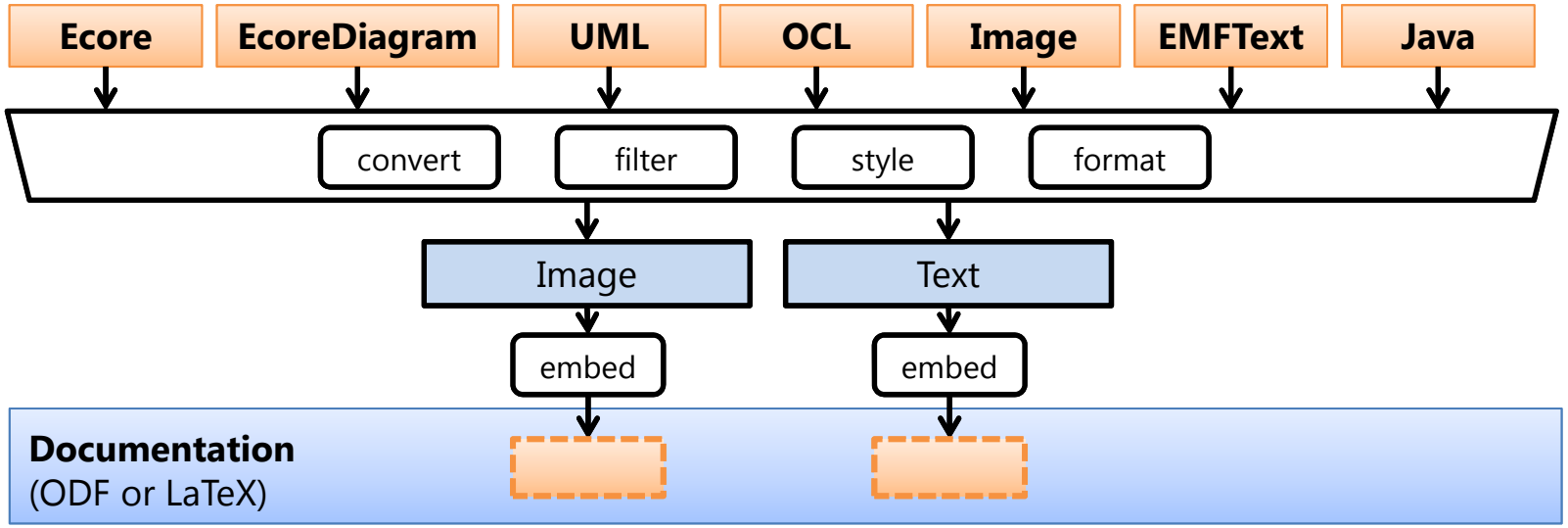


EXAMPLE ELUCIDATIVE DOCUMENT



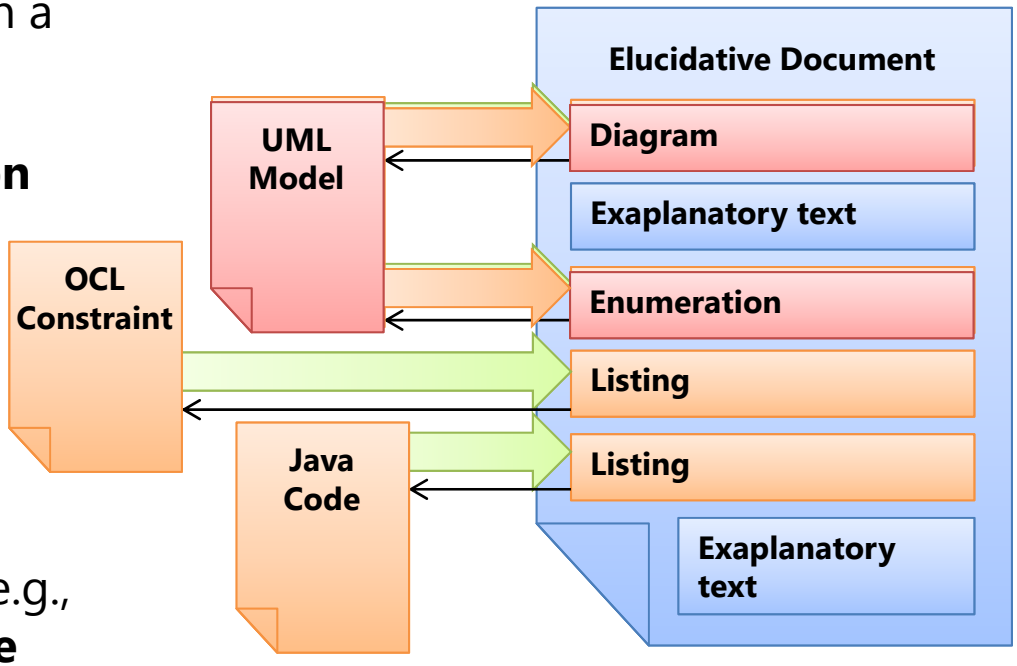
DEFT

- Prototypical **creation and maintenance** of **elucidative documents**
- **Hot update** and **notifications**
- **Several** typical **MDSD artifacts**
- http://deftproject.org/



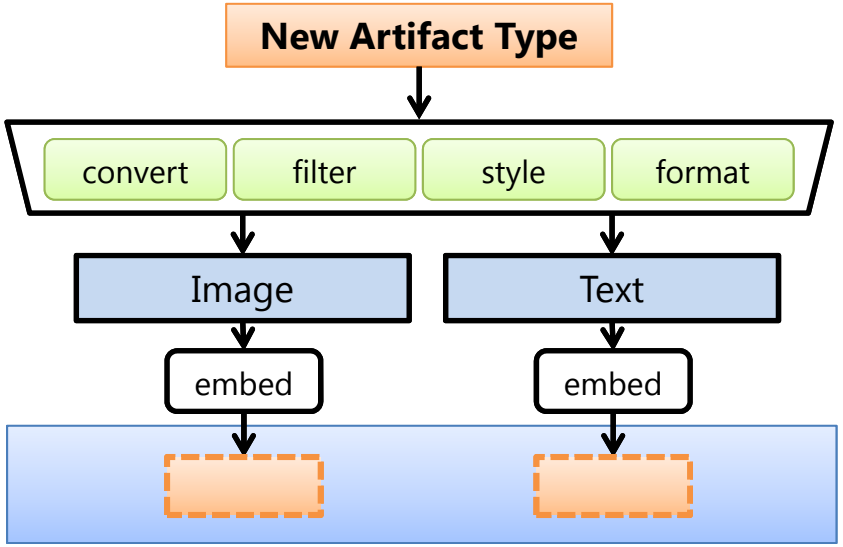
CHANGE DETECTION IN DEFT

- **Artifacts** maintained in a **repository**
- Basic **change detection** based **on timestamps**
  - → **Transclusion** of new content
  - → Author **notification**
- **Extension** with **diffs** (e.g., EMFCompare) **possible**

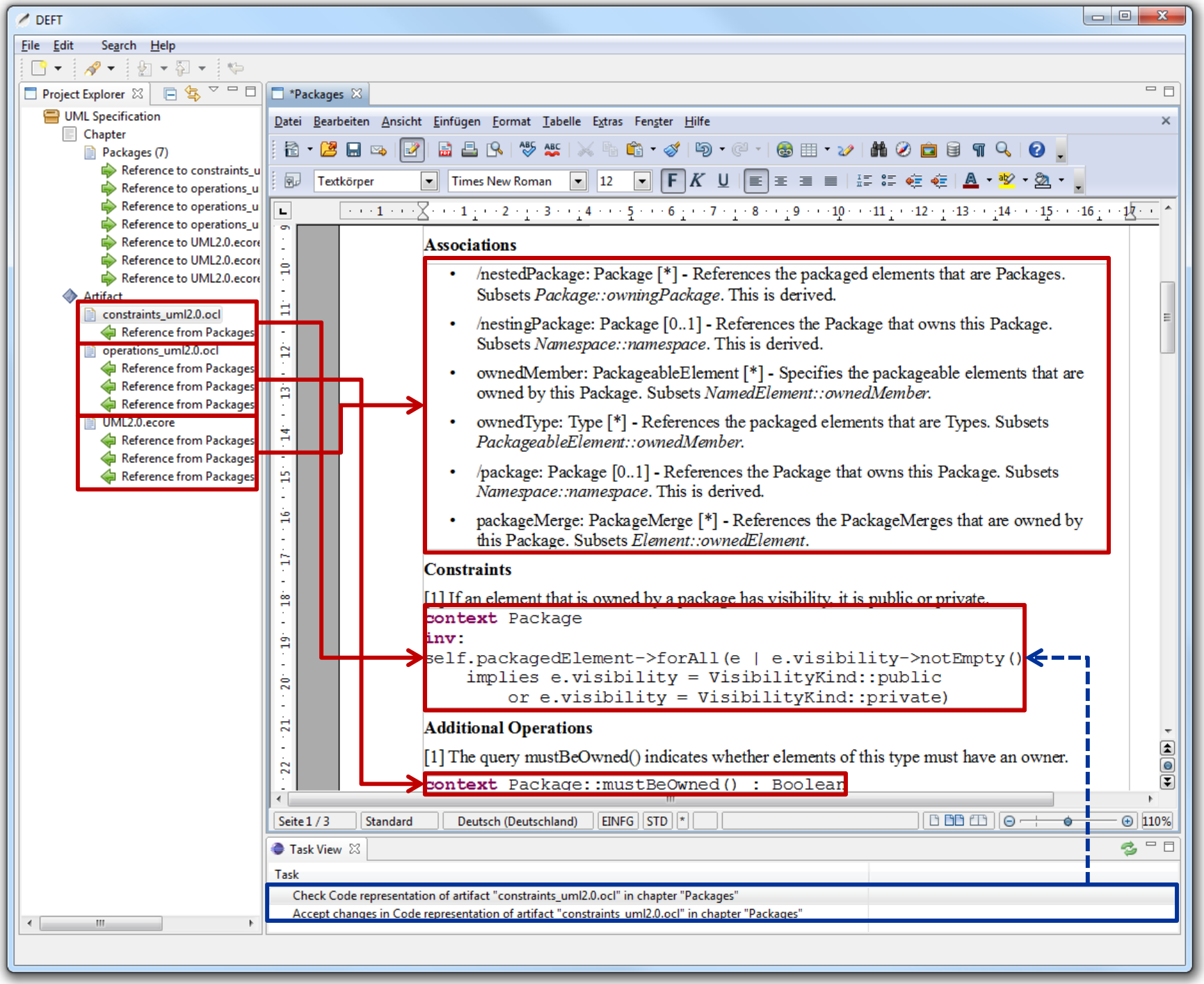


INTEGRATION OF NEW ARTIFACT TYPE

- Simple **implementation of operators** required for transclusion
- **Either**
  - Artifact-**to-Text** or
  - Artifact-**to-Image**
- **Optional**
  - Filter
  - Style
  - Format



ELUCIDATIVE DEVELOPMENT FOR UML


DEFT
File Edit Search Help
Project Explorer
UML Specification
Chapter
Packages (7)
Reference to constraints_u
Reference to operations_u
Reference to operations_u
Reference to operations_u
Reference to UML2.0.ecore
Reference to UML2.0.ecore
Reference to UML2.0.ecore
Artifact
constraints_uml2.0.ocl
Reference from Packages
operations_uml2.0.ocl
Reference from Packages
Reference from Packages
Reference from Packages
UML2.0.ecore
Reference from Packages
Reference from Packages
Reference from Packages
*Packages
Datei Bearbeiten Ansicht Einfügen Format Tabelle Extras Fenster Hilfe
Textkörper
Times New Roman
12
Associations
/nestedPackage: Package [*] - References the packaged elements that are Packages. Subsets Package::owningPackage. This is derived.
/nestingPackage: Package [0..1] - References the Package that owns this Package. Subsets Namespace::namespace. This is derived.
ownedMember: PackageableElement [*] - Specifies the packageable elements that are owned by this Package. Subsets NamedElement::ownedMember.
ownedType: Type [*] - References the packaged elements that are Types. Subsets PackageableElement::ownedMember.
/package: Package [0..1] - References the Package that owns this Package. Subsets Namespace::namespace. This is derived.
packageMerge: PackageMerge [*] - References the PackageMerges that are owned by this Package. Subsets Element::ownedElement.
Constraints
[1] If an element that is owned by a package has visibility, it is public or private.
context Package
inv:
self.packagedElement->forAll(e | e.visibility->notEmpty()
implies e.visibility = VisibilityKind::public
or e.visibility = VisibilityKind::private)
Additional Operations
[1] The query mustBeOwned() indicates whether elements of this type must have an owner.
context Package::mustBeOwned() : Boolean
Seite 1 / 3
Standard
Deutsch (Deutschland)
EINFG
STD
110%
Task View
Task
Check Code representation of artifact "constraints_uml2.0.ocl" in chapter "Packages"
Accept changes in Code representation of artifact "constraints_uml2.0.ocl" in chapter "Packages"

UML SPECIFICATION WITH DEFT

1. **Graphical representations** transcluded ✓
2. **Continuous text** transclusion possible in some cases ✓
   (manual proofreading might be necessary)
3. **Textual representations** transcluded ✓
4. **External references** can be checked for consistency ✓
   (integration of existing tooling, e.g., OCL parsers)

UML SPECIFICATION WITH DEFT

- **Continuous maintenance** of **metamodel and specification**

→ **Identification** of **inconsistencies**

→ **Fast release**, of **up-to-date specifications** possible

→ **Artifacts** (e.g., metamodel) **can be shipped** with the specification

CONCLUSION

CONCLUSION

- Elucidative Development for **continuous**, **transconsistent documentation**
  - Several kinds of artifacts
  - Multiple development phases
- Application to **UML specification** as one usage example
  - Specification process improved
  - Identification of inconsistencies
- **Future work**
  - Further **case studies**
  - **Product line** engineering
  - **Language processing** for descreptive text