

Software Engineering II

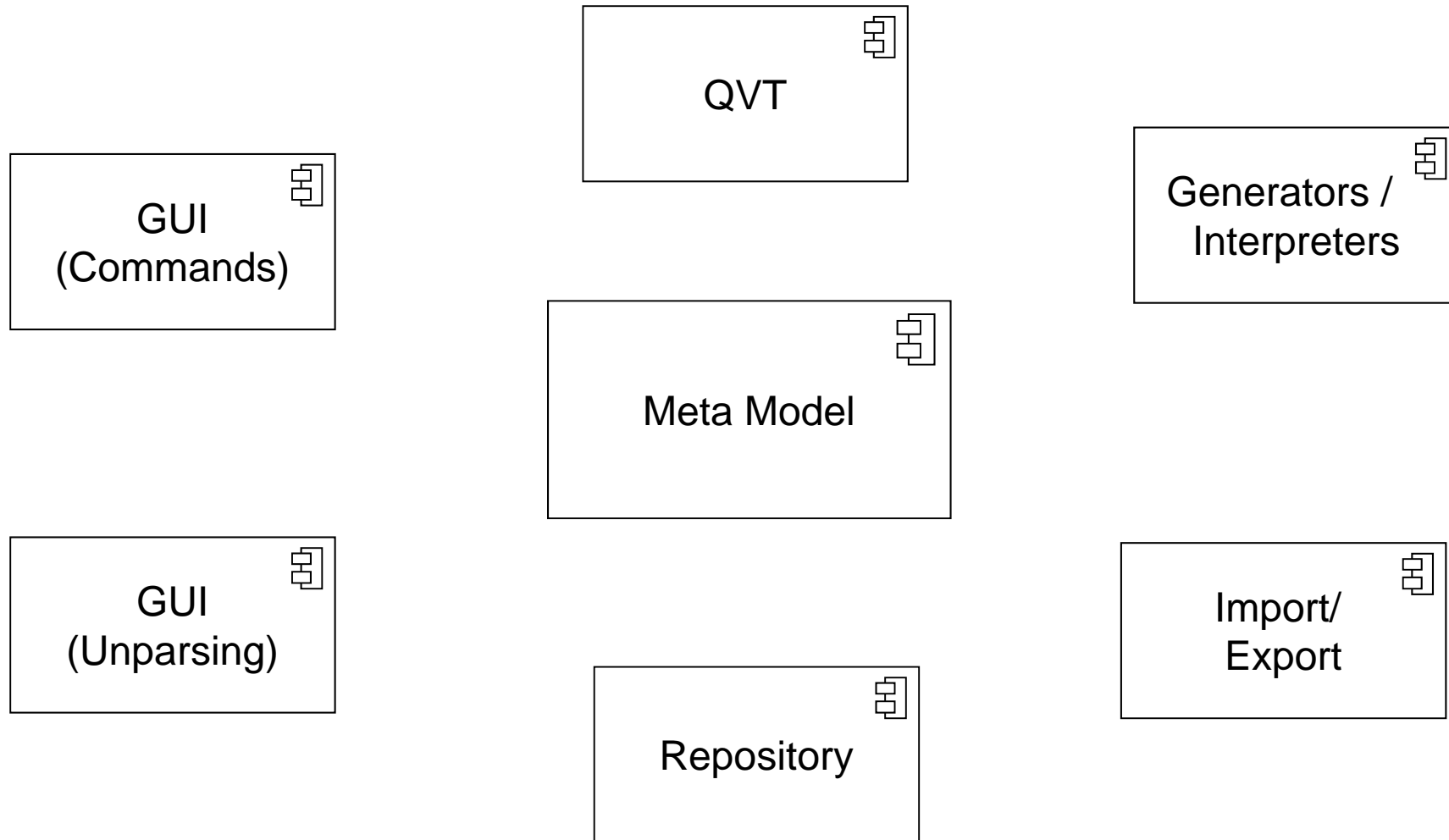
Model Validation

Wintersemester 10/111

Fachgebiet Software Engineering

Albert Zündorf / Nina Geiger

Referenzarchitektur



Wiederholung

- **Bisher im Laufe des Semesters umgesetzt:**
 - Modellierung eines Meta-Modells für die Hausautomatisierung mit Fujaba
 - Generierung von EMF Quelltext aus dem Fujaba Klassendiagramm
 - Generieren eines Editor und eines Edit Eclipse Plugins für das EMF Modell (baumartiger Editor)
 - Anfügen von Annotation an des Klassendiagramm und automatische Generierung des grafischen Editors aus diesen Annotationen mit Hilfe von Eugenia
 - Implementierung von Refactoring Operationen mittels Fujaba Storydiagrammen.
 - Implementierung einer einfachen Interpreterlogik für den grafischen Editor
 - Custom Figures zur weiteren Verfeinerung der grafischen Darstellung des Editors
 - Codegenerierung mittel Modeling Workflow Engine

Nächstes Ziel

- **Validierung des gezeichneten Diagramms**
 - Diagramme können Inkonsistenzen oder unerlaubte Zustände enthalten.
 - Prüfung der Diagramme um Fehler zu finden automatisieren
 - Regeln (Constraints) für den Aufbau der Diagramme definieren, die über die im Meta-Modell festgelegten Eigenschaften hinaus gehen.
 - Switch darf nur ausgehende und keine eingehenden Channels haben
 - Light darf nur eingehende und keine ausgehenden Channels haben
 - etc.

Constraints mit Fujaba bauen

- **Neue Constraint Klasse**
 - Erbt von `AbstractModelConstraint` aus `EclipseClasses.ctr`
- `validate(context: IValidationContext): Istatus` als Storydiagramm implementieren
- **Evtl. neues Validation package für die Validierungsklassen.**
 - Codestyle ist java bzw. inherited, NICHT emf
- **Code generieren**

Validierung in Manifest.MF einbauen

- **Neues package mit exportieren (Runtime Tab)**
- **Extensions Tab:**
 - Neuen „constraintProviders“ Extension Point hinzufügen
 - Neue category hinzufügen (Rechtsklick -> New -> category)
 - Neuen constraintProvider hinzufügen (Rechtsklick -> New -> constraintProvider)
 - Cache : true
 - Auf constraintProvider:
 - Rechtsklick -> New -> package: Namespace eures Modells
 - Rechtsklick -> New -> constraints:
 - categories: category von oben eintragen
 - Rechtsklick -> New -> constraint

Constraint eintragen

- **Klasse die ihr gerade implementiert habt bei class eintragen**
- **name beliebig vergeben**
- **lang: Java**
- **id: am besten gleich der Klasse**
- **statusCode: 0**
- **severity: ERROR**
- **mode: Batch**
- **Message mit sinnvollem Fehlertext füllen.**
- **Neues target anlegen: Klasse**
- **Description anlegen und mit sinnvollem Text füllen**

constraintBindings Extension point

- **Neuen constraintBindings Extension point anlegen**
 - Neuen clientContext anlegen (Rechtsklick -> New -> clientContext)
 - id: sinnvoll vergeben
 - default: true
 - In clientContext : Rechtsklick -> New -> enablement
 - Auf enablement: Rechtsklick -> New -> and
 - Auf and:
 - Rechtsklick -> New -> instanceof
 - Value: Eobject
 - Rechtsklick -> New -> test
 - Property: ns eures Modells + .ePackage
 - Value: Ns URI eures Modells

binding

- **Im constraintBindings ExtensionPoint:**
 - Rechtsklick -> New -> binding
 - Context : id des clientContext
 - Category: id der category aus dem constraintProviders ExtensionPoint

propertyTesters Extension point

- **Neuen propertyTesters Extension point anlegen**
 - Neuen propertyTester anlegen: Rechtsklick -> New -> propertyTester
 - class: EObjectPropertyTester (kommt aus CodeGenClasses.ctr)
 - properties: ePackage
 - namespace: euer Namespace
 - type: EObject
 - id: sinnvoll vergeben

Running

- Evtl. diagramm und edit/editor neu bauen, sollte aber eigentlich nicht notwendig sein
- Laufen lassen
- **Im Runtime Eclipse:**
 - Diagramm mit gewünschtem Fehler malen
 - Edit -> Validate
 - Fehlermeldung sollte im Problems View angezeigt werden und Fehler Icon am entsprechenden Diagrammteil angezeigt werden.