Vespucci-Tutorial

Eine kurze Einführung in Vespucci
Stand: 31. März 2012

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik

Dr.-Ing Michael Eichberg
Dipl.-Inform. Ralf Mitschke

Inhaltsverzeichnis

1 Installation

1.1 Benötigte Eclipse-Plugins

Um die korrekte Darstellung von Vespucci zu garantieren, müssen einige Plugins zu Eclipse hinzugefügt werden. Dies geht am einfachsten, wenn man sie über „Help → Install New Software...“ in die Entwicklungsumgebung einbindet. Werden zu viele Plugins installiert, kann es zu fehlerhaften Ausgaben kommen. Es ist nicht empfehlenswert, das komplette „Modeling“-Packet einzubinden, da dies dann zu besagtem fehlerhaften Verhalten des Plugins führt. Folgende Plugins werden benötigt:

- Indigo - http://download.eclipse.org/release/indigo
  - Modeling
    - Ecore Tools SDK
    - EMF - Eclipse Modeling Framework SDK
    - Extended Editing Framework - SDK for Indigo
    - Graphical Editing Framework GEF SDK
    - Operational QVT SDK
- „http://download.eclipse.org/modeling/gmp/gmf-tooling//updates/releases“
  - GMF Tooling - SDK
    - Graphical Modeling Framework SDK
- „http://download.eclipse.org/modeling/mdt/ocl/updates“
  - Eclipse OCL-3.1.1RC4
    - OCL All-In-One SDK
- „http://www.opal-project.de/vespucci“
  - Opal project

Unter dem Punkt „What is already installed?“ sollten jetzt unter „Installed Software“ folgende Plugins aufgelistet sein:

- de.tud.cs.st.vespucci.feature
- Eclipse IDE for Java Developers
- Ecore Tools SDK
- EMF - Eclipse Modeling Framework SDK
- Extended Editing Framework - SDK for Indigo
- Graphical Editing Framework GEF SDK
- Graphical Modeling Framework SDK
- OCL All-In-One SDK
- Operational QVT SDK

1.2 Einrichten von Git und Einbinden von Vespucci

1. Anmelden bzw. Registrieren bei www.github.com
2. Die Anmeldenamen an den Veranstalter (Ralf Mitschke) schicken, damit er das Projekt für diese freigeben kann
3. „Set Up Git“ anklicken und dem Tutorial folgen (nur dem ersten Teil!)
4. Klonen des Projektes mit folgendem Code:
   $ git clone git@github.com:mitschke/vespucci.git
5. Masterbranch auslesen:
   $ git checkout master
   $ git fetch origin master
   ACHTUNG: Der Masterbranch sollte niemals von euch überschrieben werden!
6. In Eclipse: Rechtsklick → „Import...“ → „General → Existing Projects into Workspace“ → „Select root directory“ → neu angelegten Ordner „vespucci“auswählen, um das Projekt und alle Veränderungen, die gemacht werden, in Eclipse einzubinden.
7. Danach bei „Run“ → „Run Configurations“ → „Eclipse Application“ → „New configuration“ → „Arguments“ → „VM arguments“ den Befehl „-XX:MaxPermSize=256m“ (bei Bedarf auch mehr - also z.B. 512) hinzufügen. Dieser verhindert einen „out of perm-space“ Error, indem er den virtuellen Speicher von Eclipse automatisch erhöht.

2 „Erste Schritte“

2.1 Starten des Plugins

Zum Öffnen einer neuen Instanz von Eclipse, führe folgende Schritte aus:

- Java-Projekt „de.tud.cs.st.vespucci“ öffnen
- Ordner „META-INF“ öffnen
- Rechtsklick auf „MANIFEST.MF“ → „Run As“ → „Eclipse Application“

2.2 Erstellen einer *.sad-Datei

- Erstelle ein neues Java-Projekt
- Jetzt gibt es drei Möglichkeiten, ein neues Vespucci-Diagram zu erstellen:
  1. File → New → Software Architecture Constraint Diagram.
  2. File → New → Other... → Software Architecture Constraint Diagram.
  3. File → New → Other... → Software Architecture Visualization → Software Architecture Constraint Diagram.
- Beim Erstellen des ersten Diagrammes wird direkt eine Datei <NameDesProjektes>.sam erstellt. Diese Datei ist das *GlobalRepository*.
- Die Module „Ensemble“ und „Empty Ensemble“ sowie alle Constraints können per Drag&Drop in die Datei gezogen werden

2.3 Testen

- Jede BP-Gruppe hat zu diesem Projekt spezifische Testfälle generiert, die unter „de.tud.cs.st.vespucci.diagram“ → „Doc“ → „GUI-Checkliste“ zu finden sind
- Tipp: Führt diese Testfälle zu Anfang des Praktikums mit dem Masterbranch durch, um euch in das Projekt einzufinden und um zu Testen, ob alles richtig funktioniert.
- Diese Tests, sowie dieses Dokument und alle weiteren Dokumente müssen von euch regelmäßig auf den neusten Stand gebracht werden.

3 Das Projekt

Die „ecore-model“-Datei und die dazugehörigen „mapping“-Dateien sind im Package „de.tud.cs.st.vespucci“ im Ordner „model“ zu finden.

3.1 Die wichtigsten Komponenten

de.tud.cs.st.vespucci: Metamodell für Vespucci

- Ordner „model“:
  - vespuci.ecore
  - vespuci.ecore_diagram
  - vespucci.genmodel
  - vespucci.gmfgen
  - vespucci.gmfgraph
  - vespucci.gmfmap
  - vespucci.gmftool
  - vespucci.trace

de.tud.cs.st.vespucci.2011-06-01: Eine Version des alten Vespuccis,die für das Versioning benötigt wird. Näheres kann in dem Tutorial "Versioning HowTo.pdf"nachgelesen werden.

de.tud.cs.st.vespucci.diagram: Das VespucciDiagram

de.tu.cs.st.vespucci.diagram.dnd & de.tu.cs.st.vespucci.diagram.dnd.JavaType: Hier werden die Drag&Drop Funktionen implementiert.

de.tud.cs.st.vespucci.diagram.menuItems: Hier werden die Einträge für das „Edit Constraint“/„SetDependency“-Menu erstellt.

Doc In diesem Ordner befinden sich sämtliche Dokumentationen zu dem Projekt.

Ordner „model“: Eine graphische Erklärung in Form von .sad Dateien, die eine Vorstellung liefern, wie die Verbindungen zwischen alle Klassen und Packages aussehen.

plugin.xml: Sehr wichtige Initialisierungen und Implementierungen von allen Sachen, die sich in dem Diagramm befinden.(z.B. Menüs, Properties, etc.)

de.tud.cs.st.vespucci.diagram.resources: Hier befindet sich u.a. der „icons-Ordner“. Dieses Java-Projekt ist für alle Ressourcen zuständig, die für das Diagramm benötigt werden.

de.tud.cs.st.vespucci.versioning: Versioning Transformationen

de.tud.cs.st.vespucci.versioning: Das Package de.tud.cs.st.vespucci.versioning

Activator.java: kontrolliert den Plugin-Lebenzyklus

VespucciTransformationHelper.java: Das ist eine Template-Klasse, die die Methoden zur Tranformation in Vespucci unterstützt

VespucciVersionChain.java: Bildet die Versionskette für Vespucci

de.tud.cs.st.vespucci.versioning.versions: Alle Versionen in Form von Java Klassen, in denen Informationen über das Datum der Version und die QVT-Transformationen beschrieben werden.

Ordner „model„: Alle ecore-Modelle von dem verschiedenen Versionen.

Ordner Transformationen: QVT Transformationen die für das Update von einer alten Version zu der aktuellsten Version benötigt werden.

3.2 Einige wichtige Funktionen

- Um Änderungen in model zu übernehmen, muss der neue Code generiert werden:
  1. Generiere den „model“-Code: Doppelklick auf die genmodel-Datei → Rechtsklick auf den Knoten „vespucci model“ → wähle „Generate Model Code“.
  2. Generiere den „edit“-Code: Doppelklick auf die genmodel-Datei → Rechtsklick auf den Knoten „vespucci model“ → wähle „Generate Edit Code“.
  3. Generiere den ’„editor“-Code: Doppelklick auf die genmodel-Datei → Rechtsklick auf den Knoten „vespucci model“ → wähle „Generate Editor Code“.
  4. Generiere „diagram“-Code: Rechtsklick auf die gmfgen-Datei → wähle „Generate diagram code“
  5. Lösche den Ordner „icons/“ im Package „de.tud.cs.st.vespucci.diagram“ (dieser wurde in das neue Package „de.tud.cs.st.vespucci.diagram.resources“ übertragen).

4 Git und seine Befehle

Wer sich schon mit Git bzw. eGit auskennt, kann dieses Kapitel gerne überspringen. Für alle anderen soll es als ein kurze Referenz der Befehle dienen, die zu Anfang benötigt werden bzw. das Tool eGit vorstellen.

4.1 Eine kurze Befehlsreferenz für die Konsole

einen Branch erstellen: git branch branchname

eine Kopie eines Repository herunterladen: git fetch branchname
Beachte: Hierbei wird zwar eine Kopie heruntergeladen, aber nicht in den lokalen Branch gemerged!

einen Branch updaten: git pull (origin) branchname

einen Branch wechseln: git checkout branchname

alle lokalen Branches anzeigen lassen: git branch

alle Branches anzeigen lassen: git branch -a oder git branch –all

einen Branch reseten: git reset –hard

einen lokalen Branch löschen: git branch -d branchname oder git branch -D branchname

einen Branch aus dem Repository löschen: git push (origin) :branchname

in einen Branch „committen”: git commit -a -m -v „Jetzt eine Nachricht einfügen“
-a: all; -m: message; -v: view (your changes)
-a -v -m sind optional, allerdings sollte IMMER eine „Message“ geschrieben werden für die Teammitglieder

einen Branch hochladen: git push (origin) branchname

Änderungen in Git einsehen: gitk –all

Quellen:

- *help.github.com/git-cheat-sheets/*
- *help.github.com/remotes/*
- *gitref.org*

4.2 Das Tool eGit

Folgendes PlugIn ermöglicht ebenfalls den Zugriff auf Git und ist bequem über Eclipse zu bedienen. Der Download ist wieder mittels „Help → Install New Software...“ möglich.

- Indigo - http://download.eclipse.org/release/indigo
  - Collaboration
    - Eclipse EGit Mylyn GitHub Feature

Dann über „Window“ → „open Perspective“ → „Git Repositories“ hinzufügen. Nun einfach mittels „Add an existing local Git Repository to this view“ den Ordner mit Vespucci hinzufügen. Von hier aus lassen sich jetzt ebenfalls die in 4.1 genannten Aktionen durchführen.

5 Tipps & Tricks

- Sollte es zu unerwarteten Fehlern kommen (z.B. in der Instanz von Eclipse Rechtsklick auf eine freie Fläche → unter „Add“ sind plötzlich die gelöschten(!) Geoshapes (z.B. Diamond, Oval,...) zu finden) kann es helfen, im workspace-Ordner den Ordner „.metadata“ zu löschen. Dies sollte nur im Notfall gemacht werden, da danach der workspace in Eclipse wieder komplett neu eingerichtet werden muss.
- Nachdem man die Instanz einmal nach dem Muster von Punkt 2.1 gestartet hat, kann man danach wie gewohnt auch einfach auf „Run“ drücken.