Klassengenerierung und Regelkompilierung

Allgemein

Für die Regelprogrammierung müssen Informationen zu den Businessobjekten, den Statusmodellen und anderen Nuclos-Einheiten zur Verfügung stehen. Damit der Zugriff und die Verwendung möglichst einfach und fehlerfrei bleibt, werden diese Daten objektiviert als Java-Klassen bereit gestellt. Diese Java-Klassen können dann aus den Regeln heraus aufgerufen und benutzt werden.

Folgende Typen von unterstützenden Klassen gibt es:

Typ	Beschreibung	Suffix

BusinessObjekt	Ein BusinessObjekt ist z.B. *Auftrag* oder *Kunde*. Die dazugehörenden Felder werden aus den Meta-Informationen gelesen und als Attribute in die Klasse geschrieben. Ja nach Art des Attributs und der Zugriffsrechte werden setter und getter-Methoden zum Setzen und Auslesen von Werten zur Verfügung gestellt.
	
	Wird mittels Businessobjekt eine neus Businessobjekt angelegt oder eine bestehende verändert oder gelöscht, werden mit dem erfolgreichen Abschluss des Speichervorgangs alle Änderungen in dem dazugehörenden BusinessObjekt vorgenommen und im Classloader abgelegt. Damit stehen die Änderungen sofort zur Verfügung.
	
	Allgemeine Konventionen:
	
	• Der Name des BusinessObjekts entspricht dem java-tauglich überarbeiteten Namen des Businessobjekts. So sind weder Sonderzeichen noch Leerzeichen erlaubt.
	• Jedes BusinessObjekt besitzt weiterhin eine Package-Angabe. Wurde ein Businessobjekt keinem Nuclet zugewiesen, wird als Default das Package "*org.nuclet.businessentity*" verwendet, ansonsten der Pfad des Nuclets, z.B: *org.meineFirma.meinNuclet*
	• Die Felder des Businessobjekts können mit gettern und setter befüllt, bzw. ausgelesen werden. Beispiel: *setBestelldatum (Date dat)*. Sonderzeichen, Leerzeichen oder auch Zahlen zu Beginn des Namens sind nicht erlaubt und werden bei der Generierung der Klassen automatisch entfernt/ersetzt, weshalb es zu Abweichungen zwischen den Namen der Methoden /Klassen und den Meta-Informationen aus dem Businessobjekt kommen kann.
	• Für die Abfrage der Dependents stehen ebenfalls Methoden zur Verfügung, die - wenn gewünscht - mit Flags erweitert werden können, um eine Einschränkung deren Status vornehmen zu können. Beispiel: *public List<Bestellposition> getBestellposition(Flag... flags).* Mögliche Flags sind: NONE, UPDATE, INSERT oder DELETE. Somit kann festgelegt werden, ob nur als gelöscht markierte Dependents zurückgegeben werden sollen oder z.B. solche, die auf der Oberfläche neu hinzugefügt wurden.
	
	• Das BusinessObjekt umfasst ebenfalls Konstanten, welche für die Erstellung von Abfragen im QueryProvider verwendet werden können.
	
	Beispiel eines BusinessObjekts "Adresse":
	
	`public class Adresse extends AbstractBusinessObject implements Modifiable {`
	
	`// Konstanten für den QueryProvider`
	`public static final Attribute<Boolean> Standard =`
	`new Attribute<Boolean>("Standard", "org.nuclet.basis", "Adresse", new Long (40005905), "standard", new Long(40006275), Boolean.class );`
	
	`public static final StringAttribute<String> Postfach =`
	`new StringAttribute<String>("Postfach", "org.nuclet.basis", "Adresse", new Long(40005905), "postfach", new Long(40006271), String.class );`
	
	`public java.lang.Boolean getStandard() {`
	`return getField("standard", java.lang.Boolean.class);`
	`}`
	
	`public void setStandard(java.lang.Boolean pStandard) {`
	`setField("standard", pStandard);`
	`}`
	
	`public java.lang.String getPostfach() {`
	`return getField("postfach", java.lang.String.class);`
	`}`
	
	`public void setPostfach(java.lang.String pPostfach) {`
	`setField("postfach", pPostfach);`
	`}`
	`}`
	
	Im Constructor des BusinessObjects werden alle Boolean-Pflichtfelder mit "FALSE" vorbelegt. Der Constructor sieht dann beispielsweise so aus:
	
	`public Auftrag() {`
	`super("lXyJlGbh830vO6CejAQF");`
	`setWichtig(Boolean.FALSE);`
	`setNuclosLogicalDeleted(Boolean.FALSE);`
	`}`
		--

<table>
<tr>
<td>Statusmodell-Klasse</td>
<td>Eine Statusmodell-Klasse ist eine Java-Klasse, die fachlich einem in Nuclos erstellen Statusmodell entspricht. Statusmodell-Klassen werden in der Regelprogrammierung verwendet, um eine einfache und sichere Gestaltung von beispielsweise Statuswechseln zu gewährleisten.

Allgemeine Konventionen:

• Der Name der Statusmodell-Klasse entspricht dem java-tauglich überarbeiteten Namen des Statusmodells. So sind weder Sonderzeichen noch Leerzeichen erlaubt.
• Jede Statusmodell-Klasse besitzt weiterhin eine Package-Angabe. Wurde ein Statusmodell keinem Nuclet zugewiesen, wird als Default das Package "org.nuclet.statemodel" verwendet, ansonsten der Pfad des Nuclets, z.B: org.meineFirma.meinNuclet
• Die Statusmodell-Klasse umfasst alle ihr zugewiesenen Status als statische Konstanten mit folgender Namenskonvention: "State_[Statusnumeral]. Beipsiel: AuftragSM.State_10

Beispiel der Statusmodell-Klasse "AuftragSM":

<pre>public class AuftragSM{
        public static State State_10 =
                    new StateImpl(IdUtils.toLongId(40006496), "Entwurf", "Geplant", 10,
IdUtils.toLongId(40006477));
        public static State State_90 =
                     new StateImpl(IdUtils.toLongId(40006497), "Abgeschlossen",
"Abgeschlossen", 90, IdUtils.toLongId(40006477));
        public static State State_50 =
            new StateImpl(IdUtils.toLongId(40006498), "Offen", "Offen", 50, IdUtils.toLongId
(40006477));
        public static State State_99 =
            new StateImpl(IdUtils.toLongId(40006499), "Storniert", "Storniert", 99, IdUtils.
toLongId(40006477));
}</pre>
Die hinterlegten Status können für manuelle Statuswechsel im StatemodelProvider verwendet werden.</td>
<td>SM</td>
</tr>
<tr>
<td>Arbeitsschritt-Klasse</td>
<td>In Nuclos können Arbeitsschritte erstellt werden. Um auch innerhalb von Regeln Arbeitsschritte manuell ausführen zu können, werden nach dem erfolgreichen Speichern eines Arbeitsschrittes sogenannte Arbeitsschritt-Klassen oder auch Generation-Klassen erstellt. Diese können über den GenerationProvider gestartet werden. Eine Arbeitsschritt-Klasse ist mittels Generics typsicher aufgebaut und legt Quell- und Zielobjekte als BusinessObjekte fest.

Allgemeine Konventionen:

• Der Name der Arbeitsschritt-Klasse entspricht dem java-tauglich überarbeiteten Namen des Arbeitsschritt. So sind weder Sonderzeichen noch Leerzeichen erlaubt.
• Jede Arbeitsschritt-Klasse besitzt weiterhin eine Package-Angabe. Wurde ein Arbeitsschritt keinem Nuclet zugewiesen, wird als Default das Package "org.nuclet.generation" verwendet, ansonsten der Pfad des Nuclets, z.B: org.meineFirma.meinNuclet
• Als Quell- und Zielklassen werden die BusinessObjekte verwendet, die im Arbeitsschritt angegeben wurden.

Beispiel der Arbeitsschritt-Klasse "ErstelleRechnungAusAuftragGEN":

<pre>public class ErstelleRechnungAusAuftragGEN implements Generation<Auftrag, Rechnung> {
        public Class<Auftrag> getSourceModule() {
                 return Auftrag.class;
        }
        public Class<Rechnung> getTargetModule() {
                return Rechnung.class;
        }
}</pre></td>
<td>GEN</td>
</tr>
</table>

ReportDatasource-Klasse	ReportDatasource-Klassen stellen objektivierte Datenquellen aus "Report und Formular" dar. Werden in Nuclos Quellen für Report und Formular angelegt, wird eine entsprechende Report-Klasse erstellt. Diese kann in der Regelprogrammierung als Quelle verwendet und über den DatasourceService aufgerufen werden.
	
	Allgemeine Konventionen:
	
	• Der Name der ReportDatasource-Klasse entspricht dem java-tauglich überarbeiteten Namen des Reports. So sind weder Sonderzeichen noch Leerzeichen erlaubt.
	• Jede ReportDatasource-Klasse besitzt weiterhin eine Package-Angabe. Wurde ein Source keinem Nuclet zugewiesen, wird als Default das Package "*org.nuclet.datasource*" verwendet, ansonsten der Pfad des Nuclets, z.B: *org.meineFirma.meinNuclet*
	• Zur weiteren Information beinhaltet eine Report-Klassen Id, Namen und Beschreibung der Datasource. Da keine Eingangsinformation, bzw. zu verarbeiten Daten angegeben werden müssen, ist eine ReportDatasource-Klasse schlicht gehalten.
	
	Beispiel der ReportDatasource-Klasse "ExportRechnungsdatenDS":
	
	`public class ExportRechnungsdatenDS implements Datasource {`
	
	`public static final Long id = new Long(40016085);`
	`public static final String name = "Export: Rechnungsdaten";`
	`public static final String description = "Export: Rechnungsdaten - für alle Kunden";`
	`public Long getId() { return new Long(40016085);}`
	`}`
		DS

Report-Klasse	Wird in Nuclos ein Report angelegt, wird eine entsprechende Report-Klasse erstellt. Diese kann in der Regelprogrammierung verwendet und über den ReportService ausgeführt werden. Innerhalb der generierten Java-Klasse werden alle Ausgabeformate als Konstanten abgelegt, die der Benutzer in Nuclos für diesen Report hinterlegt hat. Soll aus einer Regel heraus ein Report ausgeführt werden, muss das Ausgabeformat mit angegeben werden.
	
	ⓘ Aktuell wird nur das Ausgabeformat "PDF" akzeptiert. Weitere Formate werden bei der Generierung der Klassen nicht berücksichtigt und fehlen somit.
	
	Allgemeine Konventionen:
	
	• Der Name der Report-Klasse entspricht dem java-tauglich überarbeiteten Namen des Reports. So sind weder Sonderzeichen noch Leerzeichen erlaubt.
	• Jede Report-Klasse besitzt weiterhin eine Package-Angabe. Wurde ein Report keinem Nuclet zugewiesen, wird als Default das Package "*org.nuclet.report*" verwendet, ansonsten der Pfad des Nuclets, z.B: *org.meineFirma.meinNuclet*
	• Zur weiteren Information beinhaltet eine Report-Klassen die dazugehörige Id.
	
	Beispiel der Report-Klasse "MeinAuftragReportRE":
	
	`public class MeinAuftragReportRE implements Report {`
	`public static final OutputFormat Auftrag = new OutputFormat(new Long(40314675L));`
	
	`public Long getId() {`
	`return new Long(40314673L);`
	`}`
	`}`
		RE

Printout-Klassen (Formulare)	Wird in Nuclos ein Formular angelegt, wird eine entsprechende Formular-Klasse, bzw. Printout-Klasse erstellt. Diese kann in der Regelprogrammierung verwendet und über den PrintoutService ausgeführt werden. Innerhalb der generierten Java-Klasse werden alle Ausgabeformate als Konstanten abgelegt, die der Benutzer in Nuclos für dieses Formular hinterlegt hat. Soll aus einer Regel heraus ein Formular ausgeführt werden, muss das Ausgabeformat und die Id des Datensatzes, der zur Befüllung des Formulars verwendet werden soll, mit angegeben werden.
	
	Aktuell wird nur das Ausgabeformat "PDF" akzeptiert. Weitere Formate werden bei der Generierung der Klassen nicht berücksichtigt und fehlen somit.
	
	Allgemeine Konventionen:
	• Der Name der Printout-Klasse entspricht dem java-tauglich überarbeiteten Namen des Formulars. So sind weder Sonderzeichen noch Leerzeichen erlaubt.
	• Jede Printout-Klasse besitzt weiterhin eine Package-Angabe. Wurde ein Formular keinem Nuclet zugewiesen, wird als Default das Package "*org.nuclet.printout*" verwendet, ansonsten der Pfad des Nuclets, z.B: *org.meineFirma.meinNuclet*
	• Zur weiteren Information beinhaltet eine Printout-Klassen die dazugehörige Id.
	
	Beispiel der Printout-Klasse "AngebotPO":
	`public class AngebotPO implements Printout {`
	`public static final OutputFormat Angebot_Druck = new OutputFormat(new Long (40019472L));`
	`public static final OutputFormat Angebot_Email = new OutputFormat(new Long (40298009L));`
	
	`public Long getId() {`
	`return new Long(40019470L);`
	`}`
		PO

ImportStrukturDefintion-Klassen	Wird in Nuclos eine ImportStrukturDefinition angelegt, wird eine entsprechende Definition-Klasse erstellt. Diese kann in der Regelprogrammierung verwendet und über den ImportProvider ausgeführt werden. Innerhalb der generierten Java-Klasse wird die interne Nuclos-Id der Strukturdefinition abgelegt.
	
	Allgemeine Konventionen:
	• Der Name der ImportStrukturDefintion-Klasse entspricht dem java-tauglich überarbeiteten Namen des Sturkturdefinition. So sind weder Sonderzeichen noch Leerzeichen erlaubt.
	• Jede ImportStrukturDefintion-Klasse besitzt weiterhin eine Package-Angabe. Wurde eine Sturkturdefinition keinem Nuclet zugewiesen, wird als Default das Package "*org.nuclet.importstructure*" verwendet, ansonsten der Pfad des Nuclets, z.B: *org.meineFirma.meinNuclet*
	
	`public class MaterialImportIDS implements ImportStructureDefinition {`
	`public Long getId() {`
	`return new Long(41763890L);`
	`}`
		ISD

System- und Nucletparameter	**NucletParameter**: Wird in Nuclos ein Nuclet Parameter angelegt, wird für das entsprechende Nuclet eine Parameter-Klasse generiert, die in der Regelprogrammierung mittels ParameterProvider verwendet werden kann. Innerhalb der generierten Java-Klasse wird der neue Parameter als Konstante angelegt. Allgemeine Konventionen: • Der Name der Parameter-Klasse entspricht dem java-tauglich überarbeiteten Namen des Nuclets und der Erweiterung "NucletParameter", z.B. ZahlungsverkehrNucletParameter. Weder Sonderzeichen noch Leerzeichen sind erlaubt. • Jede Parameter-Klasse besitzt weiterhin eine Package-Angabe, die dem Package des Nuclets entspricht. Besitzt das Nuclet keine Package-Angabe, wird der Default *org.nuclet.parameter* eingesetzt. • Jede Parameter-Klasse besitzt die Parameter, die für das Nuclet im "Nuclet Management" angelegt wurden. Der Name des NucletParameters entspricht dem java-tauglich überarbeiteten Namen des Parameters.	

```
package org.nuclet.zahlungsverkehr;

import org.nuclos.api.parameter.NucletParameter;

public class ZahlungsverkehrNucletParameter {

    public static final NucletParameter MT940_DIRECTORY = new NucletParameter
("gV8rfNzwf59Dx6fsiVoc");
    public static final NucletParameter MT940_FILE_ENCODING = new NucletParameter
("pz3QpFCqSUumGgqNxZD9");
    public static final NucletParameter MT940_FILE_EXTENSION = new NucletParameter
("tn9x29Y7Rn5kDilAVyaY");
    public static final NucletParameter MT940_REFERENCE_TYPE = new NucletParameter
("8dTfwTM26hCR2GqQRNdt");

}
```

SystemParameter:

Werden in Nuclos Systemparameter über "Parameter" angelegt, wird die Klasse "NuclosSystemParameter" generiert. Diese beinhaltet alle SystemParameter, die nicht einem Nuclet zugewiesen sind, und kann mittels ParameterProvider verwendet werden.

Allgemeine Konventionen:

- Der Name der Parameter-Klasse ist immer "NuclosSystemParameter"
- Die Parameter-Klasse besitzt das Package *org.nuclos.parameter*
- Die Parameter-Klasse besitzt alle SystemParameter als Konstanten, deren Name dem java-tauglich überarbeiteten Namen des Parameters entspricht.

```
package org.nuclos.parameter;

import org.nuclos.api.parameter.SystemParameter;

public class NuclosSystemParameter {

    public static final SystemParameter POP3Username = new SystemParameter
("MSBTcg7xpw1rGrQdQfUN");
    public static final SystemParameter POP3Server = new SystemParameter
("oY9S8h6GfxXTN7nZgZYx");
    public static final SystemParameter POP3Port = new SystemParameter
("EPFVPnfPTKPZPN4o9QsH");
    public static final SystemParameter POP3Password = new SystemParameter
("7FcWwzSt9gxl4lEKne8F");
    public static final SystemParameter IMAPServer = new SystemParameter
("qGDEZAUGXpVpMi4A83SV");
    public static final SystemParameter IMAPPort = new SystemParameter
("ZjDGZCA6EN8ckspRgYXy");
    public static final SystemParameter IMAPProtocol = new SystemParameter
("cbv5h8m9zDBJlY6swmWD");
    public static final SystemParameter IMAPUsername = new SystemParameter
("xn2JskFs6nr2sChu5UR7");
    public static final SystemParameter IMAPPassword = new SystemParameter
("OJMsKlFfH0ujcSPH5j15");
}
```

Klassengenerierung und Regelkompilierung

Allgemein

Für die oben beschriebenen Klassen gelten bestimmte Regeln und Reihenfolgen, in der sie erstellt und bereitgestellt werden. Die Klassen werden je nach Typ kompiliert und gruppiert in einer eigenen Jar im CodeGenerator-Verzeichnis abgelegt. Von dort aus werden die Archive in den Classloader aufgenommen und können verwendet werden.

Typ	Klasse	Reihenfolge der Generierung
BusinessObjekte	BOEntities.jar	Die BusinessObjekte können als erstes erstellt werden, da sie lediglich auf die Meta-Informationen der Businessobjekt zurückgreifen müssen.
Report-Datasource-Klassen	ReportDSEntities.jar	Die Report-Klassen können unabhängig von anderen Klassen erstellt werden, da sie lediglich auf die Meta-Informationen der Reports zugreifen.
Arbeitsschritt-Klasse	Generation.jar	Die Arbeitsschritte verwenden als Quell- und Zielobjekte BusinessObjekte. Die Arbeitsschritt-Klassen können somit erst erstellt und kompiliert werden, wenn die BusinessObjekte in der BOEntities.jar fehlerfrei gebaut wurden und zur Verfügung stehen.
Statusmodell-Klasse	statemodels.jar	Die Statusmodell-Klassen können unabhängig von anderen Klassen erstellt werden, da sie lediglich auf die Meta-Informationen der Statusmodelle zugreifen.
Regeln und Bibliotheken	Nuclet.jar	Die Regeln werden immer als letztes gebaut, das sie auf alle BusinessObjekte, Report-, Statusmodel- und Arbeitsschritt-Klassen zugreifen können. Die Generierung der genannten Klassen ist somit eine notwendige Vorbedingung zur Erstellung der Nuclet.jar. Sollte es im Vorfeld Fehler geben, wird die Nuclet.jar nicht gebaut. Ein weiterer Grund dafür, dass die Nuclet.jar nicht gebaut werden kann, sind ungültige Referenzen, die Regeln untereinander besitzen. Wird z.B. die Signatur einer Utility-Klasse, die von einer Regel aufgerufen wird, verändert, dann die Regel nicht mehr kompiliert werden. Da beide zum Typ "Regeln und Bibliotheken" gehören, lässt sich die Nuclet.jar trotz korrekter Arbeitsschritt-Klassen, BusinessObjekte etc. nicht kompilieren. Sind diese jedoch strukturell in Ordnung, werden alle Klassen des Typs "Regeln und Bibliotheken" in einem Schritt kompiliert und im Archiv abgelegt.
Report-Klassen	ReportEntities.jar	Die Report-Klassen können unabhängig von anderen Klassen erstellt werden, da sie lediglich auf die Meta-Informationen der Reports zugreifen.
Printout-Klassen	PrintoutEntities.jar	Die Printout-Klassen können unabhängig von anderen Klassen erstellt werden, da sie lediglich auf die Meta-Informationen der Formulare zugreifen.
ImportStrukturDefintion-Klassen	ImportStructDefsEntities.jar	Die ImportStrukturDefintion-Klassen können unabhängig von den anderen Klassen erstellt werden, da sie lediglich auf die Meta-Informationen der Definitionen zugreifen.
System- und Nucletparameter	Parameter.jar	Die System- und Nucletparameter können unabhängig von den anderen Klassen erstellt werden, da sie lediglich auf die Meta-Informationen der Nuclets / Systemparameter zugreifen.

Zeitpunkt der Generierung

Systemstart

Bei Systemstart werden alle der genannten Klassen neu gebaut und im Classloader bereitgestellt. Dazu werden alle vorhanden Klassen und Archive aus dem Codegenerator-Verzeichnis gelöscht. Sollte es an dieser Stelle zu einem Fehler kommen, bleiben die Verzeichnisse (teilweise) leer. Der Server startet weiterhin.

Import eines Nuclets

Mit dem Import eines Nuclets ändert sich in Nuclos alles. Es werden neue Strukturen eingespielt, neue Businessobjekte, Statusmodelle, etc. Logischerweise müssen alle bereits erstellen Klassen gelöscht und der neuen Umgebung entsprechend generiert werden. Auch hier kann es bei ungültigen Daten zu Fehlern und somit zu Fehlermeldungen bei der Kompilierung von Klassen kommen.

Laufzeit

Auch zur Laufzeit können sich Strukturen von Businessobjekten, Statusmodell, etc. ändern, was eine Anpassung und Neuerstellung der Java-Klassen notwendig macht. Wird eine Businessobjekt verändert, muss das entsprechende BusinessObjekt neu erstellt werden. Das Nuclet.jar und das Generation.jar mit den Arbeitsschritten im Anschluss ebenfalls, da sie mit dem betroffenen BusinessObjekt evtl. arbeiten. Für die Sicherstellung der Datenstrukturen ist das notwendig.

Report-Klassen, Arbeitsschritt-Klassen und Statusmodell-Klassen fordern ebenfalls eine Neugenerierung des Nuclet-jars, sollte im entsprechenden Editor das Modell verändert worden sein (löschen, anlegen oder aktualisieren).

Beispiel: In Nuclos wird ein Statusmodell im Editor geladen und dadurch verändert, dass ein Status hinzugefügt wird: Status 40 "Auftrag in Bearbeitung". Mit dem Speichern des Modells wird nun eine neue Statusmodell-Klasse "AuftragSM" erstellt, die diesen Status beinhaltet. Daraufhin erstellen wir eine Regel, die über den StatusmodelProvider eine gegebene Instanz eines Auftrags auf den neuen Status 40 anheben soll. Also einen Statuswechsel vornimmt. Mit dem Speichern der Regel soll die Nuclet.jar neu gebaut werden. Wurde "AuftragSM" korrekt erstellt, kann das Nuclet.jar mit unserer neuen Regel problemlos erstellt werden.
Nun soll aus nicht näher bekannten Gründen der neue Status ein anderes Numeral erhalten. Der Wert wird von 40 auf 50 gesetzt. Mit dem Speichern wird wie zuvor eine neue Statusmodell-Klasse erzeugt, die mit dem Status 50 aktuell ist. Dennoch erscheint jetzt eine Fehlermeldung, die besagt, dass eine Regel nicht kompiliert werden kann! Was ist passiert? Die Statusmodell-Klasse wurde zwar erfolgreich verändert und neu kompiliert. Unsere Regel möchte aber immer noch einen Auftrag auf den Status 40 anheben und findet diesen Status nun nicht mehr. Damit ist die Regel nicht mehr kompilierbar und die Nuclet.jar nicht baubar. Als Folge müssen alle Referenzen auf den Status 40 manuell aktualisiert werden. In unseren Fall eine Regel.

Strukturell müssen alle Informationen korrekt sein und dürfen gegen keine Java-Konvention verstoßen, da sonst die Klassen nicht erstellt werden können. Weitere Informationen finden Sie unter Probleme und Lösungen.