Abgabetermin: Fr. 16. Dez. 2016, 23:55 Uhr

Abgabe: Über Moodle, eine entsprechende Aufgabe ist dort eingerichtet.

Benennung der ZIP-Datei: Nachname_Vorname.zip. Darin sollen die jeweiligen Projekte als Ordner enthalten sein.

Erstellen Sie ein PDF Dokument welches den relevanten Sourcecode (z.B. Code für Validierung, Code für Generierung, XText Syntax) enthält (übersichtlich formatiert und nach einzelnen Aufgaben gegliedert) und entsprechende Testfälle (Screenshots).

Entwicklung einer DSL (24 Punkte + 6 Zusatzpunkte)

Überlegen Sie sich selbstständig eine Domänen-Spezifische Sprache (DSL). Die DSL sollte einen möglichst sinnvollen Einsatz ermöglichen. Überlegen Sie, ob sie ev. aus ihrem beruflichen Umfeld (falls vorhanden) eine Idee generieren können. Versuchen sie so weit als möglich die Konzepte von Ecore/XText einzusetzen.

Konkrete Aufgaben:

- Beschreiben Sie kurz den Zweck und das Einsatzgebiet ihrer DSL sowie die Kernkonzepte hinter Ihrer DSL in schriftlicher Form (ca ½ - 1 Seite).
- Entwickeln Sie für Ihre DSL eine passende textuelle Syntax mit Hilfe von XText. Dokumentieren Sie ihre Designentscheidungen (warum sieht Ihre textuelle Syntax so aus wie sie aussieht).
- Beschreiben Sie schriftlich Einschränkungen (Constraints) für Ihre DSL und begründen Sie, warum die jeweilige Einschränkung nicht (oder nicht einfach) im Metamodell bzw. mit XText direkt verwirklicht werden kann.
- Implementieren Sie mindestens jeweils 5 eigene, unterschiedliche Validierungen und dazu passende Quickfixes
- Implementieren sie geeignete Formatierungshilfen für Ihre DSL.
- Generieren sie für Ihre DSL passende Artefakte. Dies kann z.B. wie in der Übung Java Code sein. Lässt sich für Ihre DSL kein Code generieren, so kann z.B. auch eine Dokumentation als PDF (verwenden Sie eine Java PDF Bibliothek) oder ähnliches generiert werden. Dokumentieren Sie auch hier kurz, wie welches Element abgebildet wird und die Intention hinter den jeweiligen Regeln.
- Erstellen Sie auch ein Beispielmodell, das die Anwendung Ihrer textuellen Syntax zeigt. Zeigen Sie dabei auch mittels passender Screenshots die Validierungs- und Formatierungshilfen, die Sie eingebaut haben. Verwenden Sie das Beispielmodell auch zum Generieren des Codes.

Abzugeben ist die Dokumentation sowie der gesamte Source Code und das Beispielmodell als zip-File über Moodle.

Zusatzteil (6 Extrapunkte)

Entwickeln Sie für Ihre DSL auch eine grafische Syntax mit SIRIUS. Überlegen Sie passende Layer und Conditional Styles um die Anzeige entsprechend anpassen zu können. Überlegen Sie sich auch einen passenden zweiten Viewport (z.B. Tabellen Ansicht wie in der Übung). Dokumentieren Sie sämtliche Designentscheidungen wieder kurz. Erstellen Sie ein Beispielmodell, dass die Anwendung zeigt

1 Beschreibung

In dieser Übung wird eine *DSL* entwickelt, mit der ein Modul für eine bestehende Anwendung beschrieben werden kann. Mit dieser Beschreibung wird eine Maven-Projektstruktur wie folgt aufgelistet erstellt:

- ***[MODULE_KEY]****/model/pom.xml*
  Parent für alle Modell Projekte.
- ***[MODULE_KEY]****/model/jpa/pom.xml*
  Das Projekt der *JPA*-Modells.
- ***[MODULE_KEY]****/service/pom.xml*
  Parent für alle Service Projekte.
- ***[MODULE_KEY]****/service/api/pom.xml*
  Das Projekt mit der Service Spezifikation
- ***[MODULE_KEY]****/service/impl/pom.xml*
  Das Projekt mit der Service Implementierung.

Es können folgende *Java*-Ressourcen definiert werden:

- ***MessageBundles*** sind Beschreibungen von Klassen, die sprachspezifische Texte für einen Schlüssel und eine *Locale* verwalten.
- ***Observers*** sind Beschreibungen von Beobachtermethoden, die mittels einen *Delegate* auf einem definierten *CDI*-Event reagieren können.
- ***JpaConfig*** ist die Beschreibung des *JPA*-Projekts, dem *Observer* und *MessageBundles* hinzugefügt werden können.
- ***ServiceConfig*** ist die Beschreibung der *Service*-Projekte, dem *Observer* und *MessageBundles* hinzugefügt werden können.

Das Ziel dieser *DSL* ist es den initialen Aufwand beim erstellen eines Moduls für eine bestehende Anwendung zu erleichtern. Ein Module besteht aus mehreren *Maven*-Projekten, die Mühsam erstellt werden müssen. Mit dieser *DSL* kann ein Modul einfach beschrieben werden und daraus einen *Maven*-Projektstruktur zu erstellen.

2 XTEXT Grammatik

Listing 1: ProjectGeneratorDsl.xtext

```
grammar at.ooe.fh.mdm.herzog.dsl.proj.ProjectGenerator with org.eclipse.xtext.common.Terminals

generate projectGenerator "http://www.ooe.at/fh/mdm/herzog/dsl/proj/ProjectGenerator"

Module:
    'module' name=ID '{'
    'key' key=STRING';'
    'cdiEnabled' cdiEnabled=Boolean';'
    ('messageBundles' '{' (messageBundles+=Localized+) '}' ';')?
    ('observers' '{' (observers+=Observer+) '}' ';')?
    'jpaConfig' jpaConfig=JpaConfig';'
    'serviceConfig' serviceConfig=ServiceConfig';'
    '}';

// Service configuration
ServiceConfig: '{'
    ('observers' observers+=Observer+)?
    ('messageBundles' '{' (messageBundles+=[Localized]+) '}' ';')
    '}';

Observer:
    name=ID '{'
    'type' type=CLASSNAME';'
    'during' during=During';'
    'notify' notify=Notify';'
    'delegate' className=CLASSNAME';'
    ('qualifier' qualifier=CLASSNAME';')?
    '}';

// Jpa configuration
JpaConfig: '{'
    'localizedEnums' '{' (localizedEnums+=[Localized]+) '}'';'
    ('observers' observers+=Observer+)?
    '}';

// Common
Localized:
    name=ID '{'
    ('values' '{' (values+=LocalizedEntry+) '}' ';')?
    '}';

LocalizedEntry: '{'
    'key' localizedKey=LOCALIZEDKEY';'
    'values' '{' (values+=LocalizedValue+) '}' ';'
    ('args' '{' (args+=STRING+) '}' ';')?
    '}';

LocalizedValue: '{'
    'locale' locale=Locale';'
    'value' value=STRING';'
    '}';

// Constants
enum Locale: DE_DE='de_DE' | EN_US='en_US';
enum Boolean: TRUE='true' | FALSE='false';
enum During: IN_PROG='InProgress' | AFTER_COMPLETION='AfterCompleition';
enum Notify: ALWAYS='Always' | EXISTS='Exists';
terminal fragment UAZ: 'A'..'Z';
terminal fragment LAZ: 'a'..'z';
```

```
60 terminal fragment UAZN: 'A'..'Z'|'0'..'9';
61 terminal CLASSNAME: (LAZ+ '.')+ (UAZ LAZ*)+;
62 terminal LOCALIZEDKEY: UAZN+ ('_' UAZN)*;
```

2.1 Regeln

Die gesamte Konfiguration befindet sich im Wurzelobjekt des Typ Modul, das ein Modul beschreibt. Objekte, die in mehreren Objekten referenziert werden können, werden auf Ebene des Moduls einmalig beschrieben und dann von anderen Objekten referenziert. Sofern ein Name benötigt wird, wurde das Attribut Name den Regeln, die Objekte beschreiben, hinzugefügt, wobei der Name ebenfalls als Schlüssel für diese Objekte fungiert, damit diese Objekte referenziert werden können. Im Fall der *Observer* und *Localized* wird aus dem Namen der Klassenname für Quelltextgenerierung abgeleitet.

2.2 Konstanten

Die Konstanten, wie die unterstützten *Locale*, boolsche Werte und *Observer* spezifische Konstanten wurden als Aufzählungen abgebildet. Es wurde darauf geachtet, das über die Zeichenkettenrepräsentation der Aufzählungen sich leicht die *Java*-Datentypen erstellen lassen.

2.3 Terminal Regeln

Die Terminal Regeln wurden eingeführt, damit die Klassennamen für die *Observer*-Beschreibungen einen gültigen voll qualifizierten *Java*-Klassennamen darstellen und damit die Schlüssel der sprachspezifischen Einträge einer *Localized*-Beschreibung, der Konvention von Schlüsseln einer *Properties*-Datei folgen.

3 Constraint

Folgende Auflistung beschreibt die implementierten Validierungen:

- ***Leere Beschreibung***: Es wurde eine Validierung eingeführt, die auf eine leere Beschreibung eines Moduls prüft, damit ein *Quickfix* eine initiale Beschreibung erstellen kann.
- ***Duplikate Namen***: Da die Namen vom Datentyp *ID* sind, wurde zusätzlich eine Prüfung eingeführt, ob die Namen bereits vergeben wurden.
- ***Camel case Namen***: Da die Namen vom Datentyp *ID* sind, wurde zusätzlich eine Prüfung eingeführt, ob die Namen in *camel case* sind.
- ***Duplikate Locale Einträge***: Es wurde eine Validierung eingeführt, die prüft ob es Duplikate der *Locale* für ein *Localized* gibt.
- ***Duplikate Referenzen***: Es wurde eine Validierung eingeführt, die prüft ob es Duplikate bei den gesetzten Referenzen gibt.

Mit einer Grammatik kann nicht beschrieben werden, dass es keine Duplikate bei den verwendeten Namen von Objekten in einer Auflistung gibt. Das ist so, da es sich hierum Semantik handelt und nicht Grammatik, ob ein Name nur einmalig oder mehrmalig vergeben werden darf.

Wenn das Attribut Name bereits als *ID* festgesetzt wurde, dann kann nicht zusätzlich eine Terminalregel angewendet werden, die sicherstellt, das der Name auch z.B.: in *camel case* ist.

Wenn eine Auflistung definiert wurde wie *(localizedEnums+=[Localized]+)*, dann kann mit der Grammatik nicht verhindert werden, das Duplikate bei den Referenzen angeben werden, da es sich hier auch um Semantik handelt. Das Attribut *localizedEnums* wird als Liste abgebildet.

Das ein *Localized* für eine *Locale* nur einmal einen Eintrag definieren kann, ist eine Semantik, die auch nicht über die Grammatik abgebildet werden kann.

4 Validation und Quickfix

Listing 2: ProjectGeneratorValidator.xtend

```
1  /*
2   * generated by Xtext 2.10.0
3   */
4  package at.ooe.fh.mdm.herzog.dsl.proj.validation
5
6  import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.JpaConfig
7  import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.Localized
8  import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.Module
9  import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.Observer
10 import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.ProjectGeneratorPackage
11 import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.ServiceConfig
12 import java.util.Objects
13 import java.util.regex.Pattern
14 import org.eclipse.xtext.validation.Check
15
16 import static java.util.stream.Collectors.*
17
18 /**
19  * This class contains custom validation rules.
20  *
21  * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#validation
22  */
23 class ProjectGeneratorValidator extends AbstractProjectGeneratorValidator {
24
25   /**
26    * The Class holding the validator ids.
27    */
28   public static class ValidatorId {
29     // Module
30     public static final String MODULE_EMPTY = "MODULE_EMPTY";
31     public static final String MODULE_NAME_CAMEL_CASE = "MODULE_NAME_CAMEL_CASE";
32     public static final String MODULE_KEY_UPPER_CASE = "MODULE_KEY_UPPER_CASE";
33     // Observer
34     public static final String OBSERVER_NAME_CAMEL_CASE = "OBSERVER_NAME_CAMEL_CASE";
35     public static final String OBSERVER_NAME_DUPLICATE = "OBSERVER_NAME_UNIQUE";
36     // Localized
37     public static final String LOCALIZED_NAME_CAMEL_CASE = "LOCALIZED_NAME_CAMEL_CASE";
38     public static final String LOCALIZED_NAME_DUPLICATE = "LOCALIZED_NAME_UNIQUE";
39     public static final String LOCALIZED_ENTRY_DUPLICATE = "LOCALIZED_ENTRY_DUPLICATE";
40     public static final String LOCALIZED_ENTRY_UNDEFINED = "LOCALIZED_ENTRY_UNDEFINED";
41     // ServiceConfig
42     public static final String SERVICE_CONFIG_MESSAGE_BUNDLE_DUPLICATE =
   ↪  "SERVICE_CONFIG_MESSAGE_BUNDLE_DUPLICATE";
43     // JpaConfig
44     public static final String JPA_LOCALIZED_ENUMS_DUPLICATE = "JPA_LOCALIZED_ENUMS_DUPLICATE";
45   }
46
47   static Pattern CAMEL_CASE_PATTERN = Pattern.compile("([A-Z]{1}[a-z0-9]+)+");
48
49   @Check
50   def checkForEmptyModule(Module module) {
51     if (module.key == null && module.messageBundles.empty && module.observers.empty &&
   ↪   module.jpaConfig == null &&
52       module.serviceConfig == null) {
53       val errorMsg = "Module is empty";
54       error(errorMsg, ProjectGeneratorPackage.Literals.MODULE__NAME, ValidatorId.MODULE_EMPTY);
55     }
56   }
57
```

```
58    // Module: Name must be camel case
59    @Check
60    def checkForCamelCaseModuleName(Module _module) {
61      if (!CAMEL_CASE_PATTERN.matcher(_module.name).matches) {
62        val errorMsg = "Module name must be a camel case string (e.g.: MyModuleName)";
63        error(errorMsg, ProjectGeneratorPackage.Literals.MODULE__NAME,
      ↪  ValidatorId.MODULE_NAME_CAMEL_CASE);
64      }
65    }
66
67    // Module: Key must be upper case
68    @Check
69    def checkForUpperCaseModuleKey(Module _module) {
70      for (Character c : _module.key.toCharArray) {
71        if (Character.isLowerCase(c)) {
72          val errorMsg = "Module key must be upper case";
73          error(errorMsg, ProjectGeneratorPackage.Literals.MODULE__KEY,
      ↪  ValidatorId.MODULE_KEY_UPPER_CASE);
74          return;
75        }
76      }
77    }
78
79    // Localized: Name must be camel case
80    @Check
81    def checkForCamelCaseLocalizedName(Localized _localized) {
82      if (!CAMEL_CASE_PATTERN.matcher(_localized.name).matches) {
83        val errorMsg = "Localized name must be a camel case string (e.g.: MyLocalizedName)";
84        error(errorMsg, ProjectGeneratorPackage.Literals.LOCALIZED__NAME,
      ↪  ValidatorId.LOCALIZED_NAME_DUPLICATE);
85      }
86    }
87
88    // Localized: Name must be unique
89    @Check
90    def checkForUniqueLocalizedName(Localized _localized) {
91      val module = _localized.eContainer as Module;
92      Objects.requireNonNull(module, "eContainer should be an instance of Module");
93      val count = module.messageBundles.stream.filter[name.equals(_localized.name)].distinct.count;
94      if (count > 1) {
95        val errorMsg = "Localized name is used by '" + (count - 1) + "' other Localized instances";
96        error(errorMsg, ProjectGeneratorPackage.Literals.LOCALIZED__NAME,
      ↪  ValidatorId.LOCALIZED_NAME_CAMEL_CASE);
97      }
98    }
99
100   // Localized: must contain at least one localized entry
101   @Check
102   def checkForDefinedLocaleEntries(Localized localized) {
103     if (localized.values.empty) {
104       error("If attribute 'values' is defined, then at least one localized values must be given",
105         ProjectGeneratorPackage.Literals.LOCALIZED__VALUES,
      ↪  ValidatorId.LOCALIZED_ENTRY_UNDEFINED);
106     }
107   }
108
109   // LocalizedEntry: Duplicate locale entries for a localized key not allowed
110   @Check
111   def checkForDuplicateLocaleEntries(Localized _localized) {
112     val duplicateEntries =
      ↪  _localized.values.stream.collect(groupingBy[localizedKey]).entrySet.stream.filter [
113       value.size > 1
114     ].map[key].distinct.collect(toList);
```

```
115
116     if (!duplicateEntries.empty) {
117       val errorMsg = "Duplicate locale entries found. " +
    ↪   duplicateEntries.stream.collect(joining(",", "[", "]"));
118       error(errorMsg, ProjectGeneratorPackage.Literals.LOCALIZED__VALUES,
    ↪   ValidatorId.LOCALIZED_ENTRY_DUPLICATE,
119         duplicateEntries.toArray(newArrayOfSize(duplicateEntries.size)));
120     }
121   }
122
123   // Observer: Name must be camel case
124   @Check
125   def checkForCamelCaseObserverName(Observer _observer) {
126     if (!CAMEL_CASE_PATTERN.matcher(_observer.name).matches) {
127       val errorMsg = "Observer name must be a camel case string (e.g.: MyObserverName)";
128       error(errorMsg, ProjectGeneratorPackage.Literals.OBSERVER__NAME,
    ↪   ValidatorId.OBSERVER_NAME_CAMEL_CASE);
129     }
130   }
131
132   // Observer: Name must be unique
133   @Check
134   def checkForUniqueObserverName(Observer _observer) {
135     val module = _observer.eContainer as Module;
136     Objects.requireNonNull(module, "eContainer should be an instance of Module");
137     val count = module.observers.stream.filter[name.equals(_observer.name)].distinct.count;
138     if (count > 1) {
139       val errorMsg = "Observer name is used by '" + (count - 1) + "' other Observer instances";
140       error(errorMsg, ProjectGeneratorPackage.Literals.OBSERVER__NAME,
    ↪   ValidatorId.OBSERVER_NAME_DUPLICATE);
141     }
142   }
143
144   // ServiceConfig: Message Bundles must be unique
145   @Check
146   def checkForUniqueServiceConfigMessagebundles(ServiceConfig _serviceConfig) {
147     val duplciateBundles =
    ↪   _serviceConfig.messageBundles.stream.collect(groupingBy[name]).entrySet.stream.filter [
148       value.size > 1
149     ].map[key].distinct.collect(toList);
150     if (!duplciateBundles.empty) {
151       val errorMsg = "Duplicate message bundles found. " +
152         duplciateBundles.stream.collect(joining(",", "[", "]"));
153       error(errorMsg, ProjectGeneratorPackage.Literals.SERVICE_CONFIG__MESSAGE_BUNDLES,
154         ValidatorId.SERVICE_CONFIG_MESSAGE_BUNDLE_DUPLICATE,
155         duplciateBundles.toArray(newArrayOfSize(duplciateBundles.size)));
156     }
157   }
158
159   // JpaConfig: Message Bundles must be unique
160   @Check
161   def checkForUniqueJpaConfigMessagebundles(JpaConfig _jpaConfig) {
162     val duplciateBundles =
    ↪   _jpaConfig.localizedEnums.stream.collect(groupingBy[name]).entrySet.stream.filter [
163       value.size > 1
164     ].map[key].distinct.collect(toList);
165     if (!duplciateBundles.empty) {
166       val errorMsg = "Duplicate localized enums found. " +
167         duplciateBundles.stream.collect(joining(",", "[", "]"));
168       error(errorMsg, ProjectGeneratorPackage.Literals.JPA_CONFIG__LOCALIZED_ENUMS,
169         ValidatorId.JPA_LOCALIZED_ENUMS_DUPLICATE,
170         duplciateBundles.toArray(newArrayOfSize(duplciateBundles.size)));
171     }
```

```
172   }
173 }
```

Listing 3: ProjectGeneratorQuickfixProvider.xtend

```
1  /*
2   * generated by Xtext 2.10.0
3   */
4  package at.ooe.fh.mdm.herzog.dsl.proj.ui.quickfix
5
6  import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.Boolean
7  import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.During
8  import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.JpaConfig
9  import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.Locale
10 import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.Localized
11 import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.Module
12 import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.Notify
13 import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.Observer
14 import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.ProjectGeneratorFactory
15 import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.ServiceConfig
16 import at.ooe.fh.mdm.herzog.dsl.proj.validation.ProjectGeneratorValidator.ValidatorId
17 import java.util.Objects
18 import java.util.stream.Collectors
19 import org.eclipse.xtext.ui.editor.quickfix.DefaultQuickfixProvider
20 import org.eclipse.xtext.ui.editor.quickfix.Fix
21 import org.eclipse.xtext.ui.editor.quickfix.IssueResolutionAcceptor
22 import org.eclipse.xtext.validation.Issue
23
24 /**
25  * Custom quickfixes.
26  *
27  * See https://www.eclipse.org/Xtext/documentation/310_eclipse_support.html#quick-fixes
28  */
29 class ProjectGeneratorQuickfixProvider extends DefaultQuickfixProvider {
30
31   /**
32    * Generates a default structure.
33    */
34   @Fix(ValidatorId.MODULE_EMPTY)
35   def fixModuleInitGeneration(Issue issue, IssueResolutionAcceptor acceptor) {
36     acceptor.accept(issue, 'Initialize default', 'Initialize default', '') [ element, context |
37       val module = (element as Module);
38       module.key = module.name.toUpperCase;
39       module.cdiEnabled = Boolean.TRUE;
40
41       // Default jpa Localized
42       val localizedJpa = ProjectGeneratorFactory.eINSTANCE.createLocalized;
43       localizedJpa.name = "JpaConstantBundle";
44       val localizedJpaEntry = ProjectGeneratorFactory.eINSTANCE.createLocalizedEntry;
45       localizedJpaEntry.localizedKey = "KEY_1";
46       val localizedJpaValue = ProjectGeneratorFactory.eINSTANCE.createLocalizedValue;
47       localizedJpaValue.locale = Locale.EN_US;
48       localizedJpaValue.value = "EN_KEY_1";
49
50       localizedJpaEntry.values.add(localizedJpaValue);
51       localizedJpa.values.add(localizedJpaEntry);
52
53       // Default service Localized
54       val localizedError = ProjectGeneratorFactory.eINSTANCE.createLocalized;
```

```
55        localizedError.name = "ErrorMessagebundle";
56        val localizedErrorEntry = ProjectGeneratorFactory.eINSTANCE.createLocalizedEntry;
57        localizedErrorEntry.localizedKey = "ERROR_1";
58        val localizedErrorValue = ProjectGeneratorFactory.eINSTANCE.createLocalizedValue;
59        localizedErrorValue.locale = Locale.EN_US;
60        localizedErrorValue.value = "EN_ERROR_1";
61
62        localizedErrorEntry.values.add(localizedErrorValue);
63        localizedError.values.add(localizedErrorEntry);
64
65        // Default Observer
66        val observer = ProjectGeneratorFactory.eINSTANCE.createObserver;
67        observer.name = "YourObserverName";
68        observer.type = "com.clevercure.common.event.StartedEvent";
69        observer.during = During.IN_PROG;
70        observer.notify = Notify.ALWAYS;
71        observer.className = "com.clevercure." + module.key.toLowerCase +
    ↪   ".event.observer.YourObserverClass";
72
73        // Default jpaConfig
74        val jpaConfig = ProjectGeneratorFactory.eINSTANCE.createJpaConfig;
75        jpaConfig.localizedEnums.add(localizedJpa);
76
77        // Default serviceConfig
78        val serviceConfig = ProjectGeneratorFactory.eINSTANCE.createServiceConfig;
79        serviceConfig.messageBundles.add(localizedError);
80
81        module.messageBundles.add(localizedJpa);
82        module.messageBundles.add(localizedError);
83        module.observers.add(observer);
84        module.jpaConfig = jpaConfig;
85        module.serviceConfig = serviceConfig;
86      ]
87    }
88
89    /**
90     * Module: Make Module.key upper case
91     */
92    @Fix(ValidatorId.MODULE_KEY_UPPER_CASE)
93    def fixModuleKey(Issue issue, IssueResolutionAcceptor acceptor) {
94      acceptor.accept(issue, 'Convert to upper case', 'Convert to upper case', '') [ element,
    ↪   context |
95        val module = (element as Module);
96        module.key = module.key.toUpperCase;
97      ]
98    }
99
100   /**
101    * Localized: Remove or rename duplicate Localized matched by their name.
102    */
103   @Fix(ValidatorId.LOCALIZED_NAME_DUPLICATE)
104   def fixDuplicateLocalizedName(Issue issue, IssueResolutionAcceptor acceptor) {
105     // Remove entry
106     acceptor.accept(issue, 'Remove', 'Remove', '') [ element, context |
107       val localized = (element as Localized);
108       Objects.requireNonNull(localized, "Element should be Localized instance");
109       val module = (localized.eContainer as Module);
110       Objects.requireNonNull(module, "eContainer of Localized should be Module instance");
111
112       module.messageBundles.remove(localized);
113     ]
114     // Rename entry
115     acceptor.accept(issue, 'Rename', 'Rename', '') [ element, context |
```

```
116       val localized = (element as Localized);
117       Objects.requireNonNull(localized, "Element should be Localized instance");
118       val module = (localized.eContainer as Module);
119       Objects.requireNonNull(module, "eContainer of Localized should be Module instance");
120
121       val duplicates = module.messageBundles.stream.filter[name.equals(localized.name)].collect(
122         Collectors.toList);
123       for (var i = 1; i <= duplicates.size; i++) {
124         val loc = duplicates.get(i);
125         loc.name = loc.name + i;
126       }
127     ]
128   }
129
130   /**
131    * Localized: Remove or rename duplicate LocalizedEntry instances of the given Localized
    ↪  instance.
132    */
133   @Fix(ValidatorId.LOCALIZED_ENTRY_DUPLICATE)
134   def fixLocalizedEntryDuplicates(Issue issue, IssueResolutionAcceptor acceptor) {
135     // Remove entry
136     acceptor.accept(issue, 'Remove', 'Remove', '') [ element, context |
137       val localized = (element as Localized);
138       Objects.requireNonNull(localized, "Element should be Localized instance");
139
140       if ((issue.data != null) && (issue.data.length > 0)) {
141         for (_key : issue.data) {
142           val optionalDuplicate =
    ↪  localized.values.stream.filter[localizedKey.equals(_key)].findFirst;
143           if (optionalDuplicate.isPresent) {
144             localized.values.remove(optionalDuplicate.get);
145           }
146         }
147       }
148     ]
149     // Rename entry
150     acceptor.accept(issue, 'Rename', 'Rename', '') [ element, context |
151       val localized = (element as Localized);
152       Objects.requireNonNull(localized, "Element should be Observer instance");
153
154       if ((issue.data != null) && (issue.data.length > 0)) {
155         for (_key : issue.data) {
156           val optionalDuplicate =
    ↪  localized.values.stream.filter[localizedKey.equals(_key)].findFirst;
157           if (optionalDuplicate.isPresent) {
158             optionalDuplicate.get.localizedKey = optionalDuplicate.get.localizedKey + "_1";
159           }
160         }
161       }
162     ]
163   }
164
165   /**
166    * Observer: Remove or rename duplicate Observer matched by their name.
167    */
168   @Fix(ValidatorId.OBSERVER_NAME_DUPLICATE)
169   def fixDuplicateObserverName(Issue issue, IssueResolutionAcceptor acceptor) {
170     // Remove entry
171     acceptor.accept(issue, 'Remove', 'Remove', '') [ element, context |
172       val bserver = (element as Observer);
173       Objects.requireNonNull(bserver, "Element should be Observer instance");
174       val module = (bserver.eContainer as Module);
175       Objects.requireNonNull(module, "eContainer of Observer should be Module instance");
```

```
176
177       module.observers.remove(bserver);
178     ]
179     // Rename entry
180     acceptor.accept(issue, 'Rename', 'Rename', '') [ element, context |
181       val observer = (element as Observer);
182       Objects.requireNonNull(observer, "Element should be Observer instance");
183       val module = (observer.eContainer as Module);
184       Objects.requireNonNull(module, "eContainer of Observer should be Module instance");
185
186       val duplicates =
    ↪  module.observers.stream.filter[name.equals(observer.name)].collect(Collectors.toList);
187       for (var i = 1; i <= duplicates.size; i++) {
188         val obs = duplicates.get((i - 1));
189         obs.setName(obs.getName() + i);
190       }
191     ]
192   }
193
194   /**
195    * ServiceConfig: Remove duplicate mapped message bundle matched by their name
196    */
197   @Fix(ValidatorId.SERVICE_CONFIG_MESSAGE_BUNDLE_DUPLICATE)
198   def fixServiceConfigDuplicateMessageBundles(Issue issue, IssueResolutionAcceptor acceptor) {
199     // Remove entry
200     acceptor.accept(issue, 'Remove', 'Remove', '') [ element, context |
201       val config = (element as ServiceConfig);
202       Objects.requireNonNull(config, "Element should be ServiceConfig instance");
203
204       if ((issue.data != null) && (issue.data.length > 0)) {
205         for (_name : issue.data) {
206           val optionalDuplicate =
    ↪  config.messageBundles.stream.filter[name.equals(_name)].findFirst;
207           if (optionalDuplicate.isPresent) {
208             config.messageBundles.remove(optionalDuplicate.get);
209           }
210         }
211       }
212     ]
213   }
214
215   /**
216    * JpaConfig: Remove duplicate mapped message bundles mapped by their name.
217    */
218   @Fix(ValidatorId.JPA_LOCALIZED_ENUMS_DUPLICATE)
219   def fixJpaConfigDuplicateMessageBundles(Issue issue, IssueResolutionAcceptor acceptor) {
220     // Remove entry
221     acceptor.accept(issue, 'Remove', 'Remove', '') [ element, context |
222       val config = (element as JpaConfig);
223       Objects.requireNonNull(config, "Element should be JpaConfig instance");
224
225       if ((issue.data != null) && (issue.data.length > 0)) {
226         for (_name : issue.data) {
227           val optionalDuplicate =
    ↪  config.localizedEnums.stream.filter[name.equals(_name)].findFirst;
228           if (optionalDuplicate.isPresent) {
229             config.localizedEnums.remove(optionalDuplicate.get);
230           }
231         }
232       }
233     ]
234   }
235 }
```

5 Formatter

Listing 4: ProjectGeneratorFormatter.xtend

```
/*
 * generated by Xtext 2.10.0
 */
package at.ooe.fh.mdm.herzog.dsl.proj.formatting2

import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.Localized
import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.LocalizedEntry
import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.LocalizedValue
import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.Module
import at.ooe.fh.mdm.herzog.dsl.proj.projectGenerator.ServiceConfig
import at.ooe.fh.mdm.herzog.dsl.proj.services.ProjectGeneratorGrammarAccess
import com.google.inject.Inject
import org.eclipse.xtext.formatting2.AbstractFormatter2
import org.eclipse.xtext.formatting2.IFormattableDocument

class ProjectGeneratorFormatter extends AbstractFormatter2 {

  @Inject extension ProjectGeneratorGrammarAccess

  def dispatch void format(Module module, extension IFormattableDocument document) {
    module.interior[indent];
    module.regionFor.keyword("{").prepend[noSpace];
    module.regionFor.keyword("{").append[newLine];
    module.regionFor.keyword("}").prepend[newLine];

    module.regionFor.keyword("key").prepend[newLine];
    module.regionFor.keyword("cdiEnabled").prepend[newLine];
    module.regionFor.keyword("messageBundles").prepend[newLine];
    module.regionFor.keyword("observers").prepend[newLine];
    module.regionFor.keyword(";").prepend[noSpace].append[newLine];

    // TODO: format HiddenRegions around keywords, attributes, cross references, etc.
    module.messageBundles.forEach[format];
    module.observers.forEach[format];
    module.getJpaConfig.format;
    module.getServiceConfig.format;
  }

  def dispatch void format(Localized localized, extension IFormattableDocument document) {
    localized.interior[indent];
    localized.regionFor.keyword("{").prepend[noSpace].append[newLine];
    localized.interior[indent];
    localized.regionFor.keyword("values").prepend[newLine];
    localized.regionFor.keyword("}").prepend[newLine];
    localized.regionFor.keyword(";").prepend[noSpace].append[newLine];

    localized.values.forEach[format];
  }

  def dispatch void format(LocalizedEntry localizedEntry, extension IFormattableDocument document)
  ↪  {
    localizedEntry.interior[indent]
    localizedEntry.regionFor.keyword("{").prepend[newLine].append[newLine];
    localizedEntry.regionFor.keyword("key").prepend[newLine];
    localizedEntry.regionFor.keyword("values").prepend[newLine];
    localizedEntry.regionFor.keyword("}").prepend[newLine];
    localizedEntry.regionFor.keyword(";").prepend[noSpace].append[newLine];

    localizedEntry.values.forEach[format];
```

```
59
60    }
61
62    def dispatch void format(LocalizedValue localizedValue, extension IFormattableDocument document)
      ↪  {
63      localizedValue.interior[indent]
64      localizedValue.regionFor.keyword("{").prepend[newLine].append[newLine];
65      localizedValue.regionFor.keyword("locale").prepend[newLine];
66      localizedValue.regionFor.keyword("value").prepend[newLine];
67      localizedValue.regionFor.keyword("}").prepend[newLine];
68      localizedValue.regionFor.keyword(";").prepend[noSpace].append[newLine];
69    }
70
71    def dispatch void format(ServiceConfig serviceConfig, extension IFormattableDocument document) {
72      serviceConfig.observers.forEach[format];
73    }
74
75  // TODO: implement for JpaConfig, Localized, LocalizedEntry
76  }
```