1 Architecture

Die Architektur des Frameworks lässt sich mit einer mehrstufigen Pipeline, wie im unteren Teil von Abbildung 1 zu sehen, vergleichen. Dabei können einzelne Komponenten ausgetauscht bzw. über die *Meta tool chain* (siehe Abbildung 1, oberer Teil) konfiguriert werden.

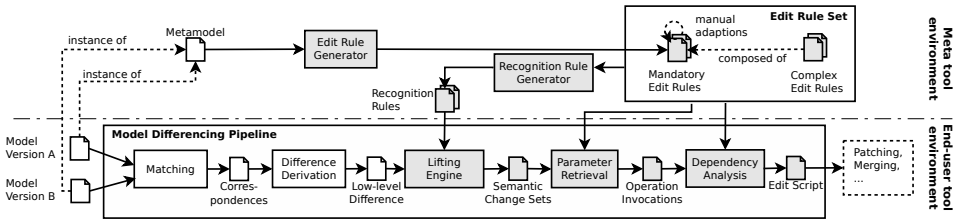

Abbildung 1: Differencing tool chain and meta-tools

Im folgenden werden die einzelnen Komponenten genauer erläutert.

1.1 Matching

Die Aufgabe des *Matchers* ist es, die korrespondierenden Elemente aus Modell A und Modell B, also die Elemente, die in beiden Modellen übereinstimmen, zu identifizieren. Dabei ist das Ergebnis vor allem davon abhängig anhand welcher Kriterien der Matcher eine Übereinstimmung festlegt. Hier wird unter anderem zwischen *ID-*, *signatur-* und *ähnlichkeitsbasierten* Verfahren unterschieden.
Das Framework definiert einen Extension Point (siehe 3.1), welcher es ermöglicht weiter Matcher in das Framework zu integrieren. Die konzeptuelle Struktur des Ergebnisses, also eines *Matchings*, ist dabei durch das in Abbildung 2 dargestellte, EMF-basierte Metamodell definiert.
Ein `Matching` zwischen den beiden Modellen `eResourceA` und `eResourceB` besteht aus einer Menge von `Correspondences`. Eine `Correspondence` verweist auf die beiden korrespondierenden Elemente `matchedA` und `matchedB` vom Typ `EObject`. Zusätzlich kann diese Referenzen auf die `Correspondence` der Kind- (`containmentCorrespondences`) bzw. der Elternelemente (`contnainerCorrespondence`) halten. Des Weiteren erbt eine `Correspondence` von `ExtendableObject`, wodurch es möglich ist dieser weitere Eigenschaften, wie beispielsweise der Verlässlichkeit, zuzuordnen.

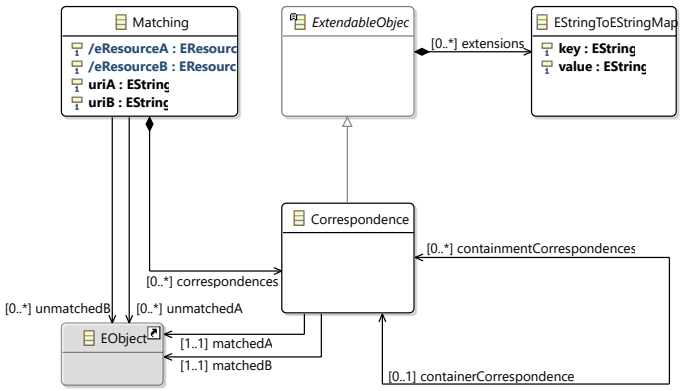

Abbildung 2: Abstract Syntax of a model matching

1.2 Difference Derivation

Ausgehend von den gefunden Korrespondenzen berechnet der *Difference Derivator* eine technische Differenz (*low-level difference*) der Modelle. Alle Objekte und Referenzen, für die keine Korrespondenz existiert, müssen demnach entweder in Modell B hinzugefügt, oder aus Modell A entfernt worden sein. Durch die Verwendung eines *Technical Difference Builders* können Modellelemente von der Differenzberechnung ausgeschlossen werden. Dieser kann ebenfalls über einen Extension Point (siehe 3.2) erweitert und für die jeweilige Domäne angepasst werden.
Die konzeptuelle Struktur einer Differenz ist durch das in Abbildung 3 dargestellte, EMF-basierte Metamodell definiert.
Ein `SymmetricDifference` zwischen den beiden Modellen `modelA` und `modelB` besteht aus einer Menge von `Changes`. Dabei wird zwischen `Add-` und `RemoveObjects`, `Add-` und `RemoveReferences` sowie `AttributValueChanges` unterschieden. Ein Änderung vom Typ `AddObject/RemoveObject` hält eine Referenz `obj` auf das Objekt aus `modelB/modelA`. Da es sich bei den Referenzen auf Ebene des Metamodells um Eigenschaften der Objekte handelt, können diese nicht durch die Änderungen `Add/RemoveReferences` referenziert werden. Daher halten diese jeweils eine Referenz auf das `src-` und `tgt-`Objekt aus `modelA` (bei `RemoveReference`) oder `modelB` (bei `AddReference`). Zusätzlich wird noch eine Referenz `type` auf den Typen der Referenz gehalten, also einer Instanz des Metametamodells. Ähnlich verhält es sich bei einer Änderung vom Typ `AttributValueChange`. Auch hier handelt es sich um eine Eigenschaft des Laufzeitobjektes aus einem der Modelle. Daher wird eine Referenz `obj` auf den Container des Attributes gehalten sowie eine Referenz `type` auf den Typen des Attributes.

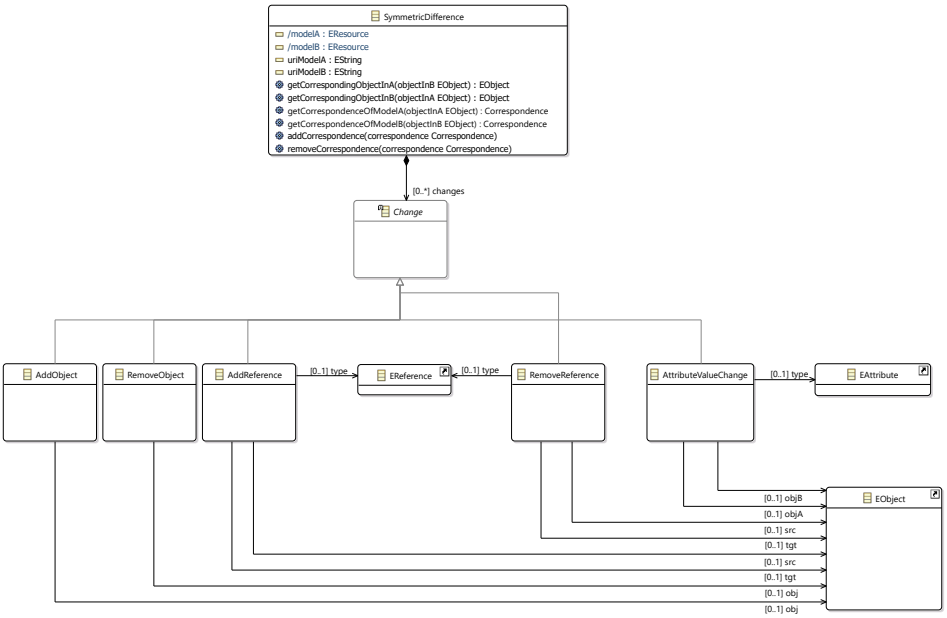

Abbildung 3: Abstract Syntax of a technical model difference

1.3 Lifting Engine

Aufgabe der *Lifting Engine* ist es, die zuvor berechnete technische Differenz semantisch zu liften. Eine technische Differenz enthält alle Änderungen auf Basis des Metamodells, also der abstrakten Syntax. Beim *semantischen liften* werden die einzelnen Änderungen zu sogenannten *Semantic Change Sets* gruppiert, welche Editieroperation auf Ebene der konkreten Syntax repräsentieren.
Die konzeptuelle Struktur einer gelifteten Differenz ist durch das in Abbildung 4 dargestellte, EMF-basierte Metamodell definiert.

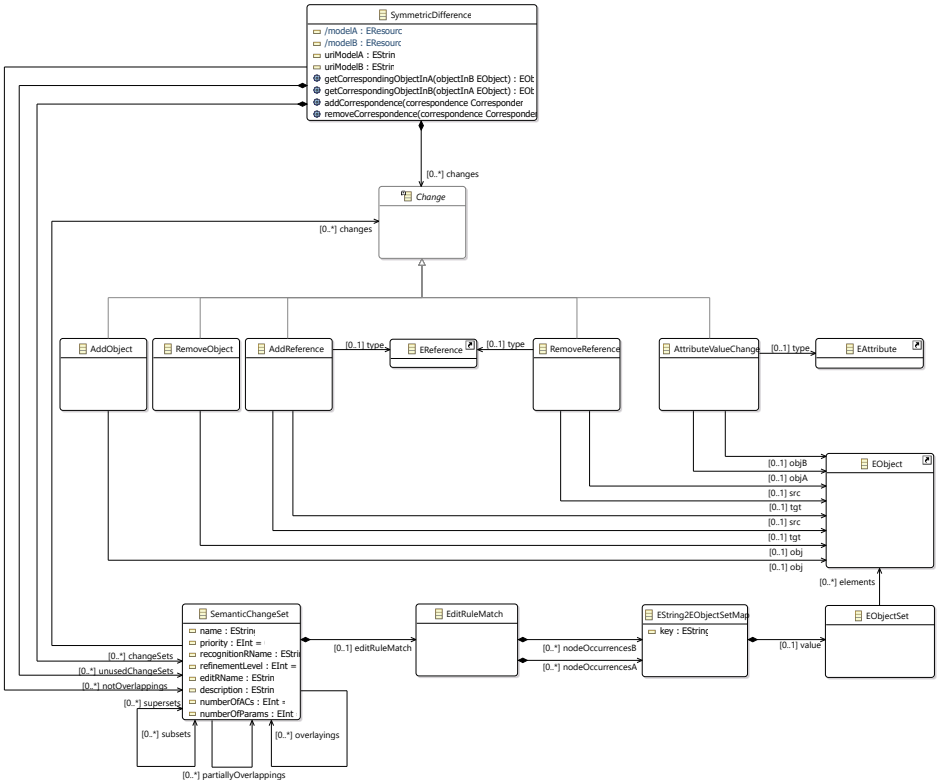

Abbildung 4: Abstract Syntax of a descriptive model difference

1.4 Rule Base

2 Application Programming Interface

Für jede Stufe der Pipeline existiert ein Plug-in, bestehend aus einer *Facade-Klasse* und einer *Settings-Klasse*. Erstere stellt diverse statische Methoden zur Berechnung und zum Serialisieren der jeweiligen Ergebnisse bereit. Letztere erbt von der Klasse `BaseSettings` (vgl. Abbildung 7) oder einer der Subklassen (siehe Abschnitt 2.1 - 2.4) und dient der Konfiguration der jeweiligen Komponente.

Package: org.sidiff.common.settings
Name: BaseSettings
Generalization: AbstractSettings
Description:

Die Klasse `AbstractSettings` hält eine Liste von `ISettingsChangedListener`,

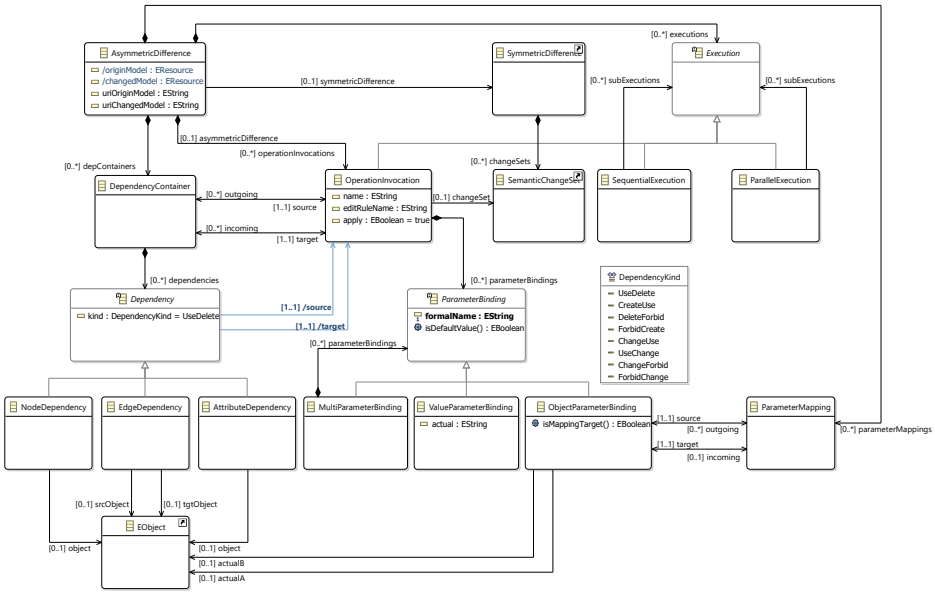

Abbildung 5: Abstract Syntax of a prescriptive model difference

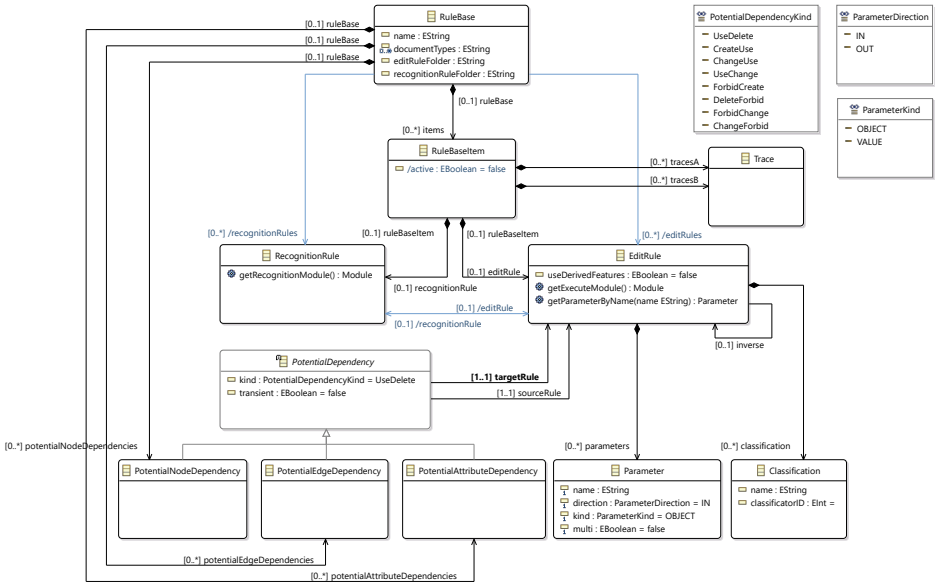

Abbildung 6: Conceptual structure of a rule base

mit deren Hilfe auf Änderungen einzelner Konfigurationsparameter reagiert werden kann. So kann die Belegung einzelner Parameter beispielsweise weitere Konfigurationsparameter aktivieren. Zusätzlich definiert die Klasse die Methode `validateSettings()`, um die Konfiguration zu validieren.

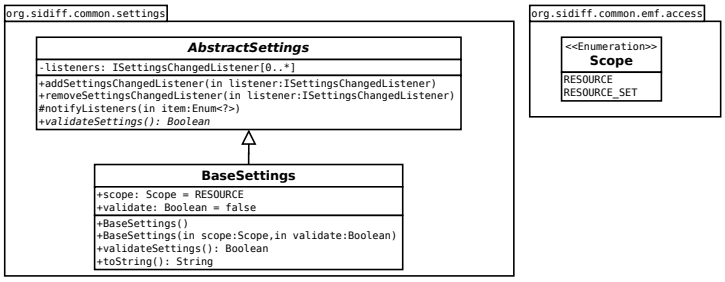

Abbildung 7: Base Settings

Diese Methode muss von der jeweiligen Unterklasse implementiert bzw. überschrieben werden.
Die Klasse `BaseSettings` erbt von `AbstractSettings` und besitzt die beiden Konfigurationsparameter `scope` und `validate`. Ersterer legt fest, ob die folgenden Berechnungen auf Basis einer einzelnen Ressource (`Scope.RESOURCE`) oder einer Menge verbundener Ressourcen (`Scope.RESOURCE_SET`) erfolgen. Über den Konfigurationsparameter `validate` kann festgelegt werden, ob die Eingabemodelle vor den folgenden Berechnungen validiert werden sollen.

2.1 Matching Engine (org.sidiff.matching.api)

Package: org.sidiff.matching.api
Name: MatchingFacade
Generalization: none
Description:

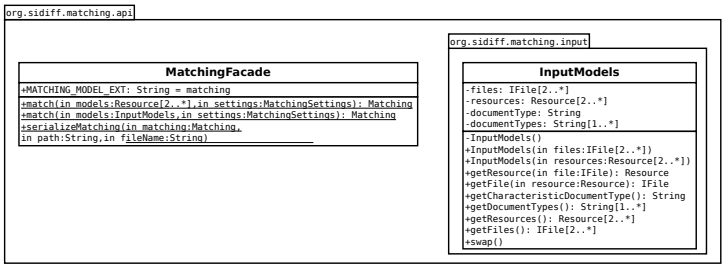

Abbildung 8: Matching Facade

Die Klasse `MatchingFacade` stellt Methoden zur Verfügung, um Korrespondenzen zwischen mehreren Modellen zu berechnen (siehe Abbildung 8:

`match(...)`) und zu serialisieren (siehe Abbildung 8: `serializeMatching(Matching matching, String path, String fileName)`). Anstatt der Methode `match()` eine Menge von Ressourcen vom Typ `org.eclipse.emf.ecore.resource.Resource` zu übergeben, können diese auch in Form eines Objekts vom Typ `InputModels` übergeben werden. Diese Klasse ermöglicht den Zugriff auf die Ressourcen unter Verwendung von Ressourcen vom Typ `org.eclipse.core.resources.IFile`. Des Weiteren bietet die Klasse die Möglichkeit, die Reihenfolge der Eingabemodelle umzukehren.

Package: org.sidiff.matching.api.settings
Name: MatchingSettings
Generalization: BaseSettings
Description:

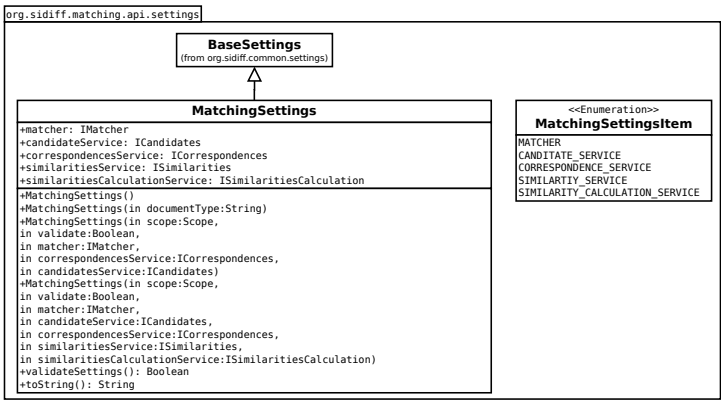

Abbildung 9: Matching Settings

2.2 Difference Derivator (org.sidiff.difference.technical.api)

Package: org.sidiff.difference.technical.api
Name: TechnicalDifferenceFacade
Generalization: MatchingFacade
Description:
Die Klasse *TechnicalDifferenceFacade* stellt Methoden zur Verfügung, um eine technische Differenz zwischen zwei Modellen zu berechnen und zu serialisieren.

Package: org.sidiff.difference.technical.api.settings
Name: DifferenceSettings
Generalization: MatchingSettings
Description:

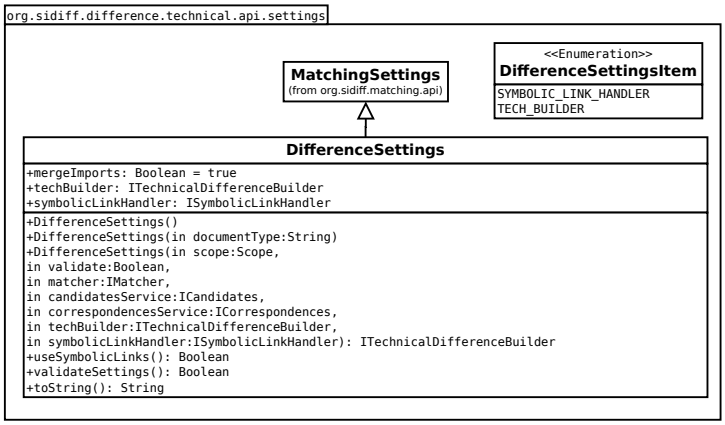

Abbildung 10: Difference Settings

2.3 LiftingEngine (org.sidiff.difference.lifting.api)

Package: org.sidiff.difference.lifting.api
Name: LiftingFacade
Generalization: TechnicalDifferenceFacade
Description:
Die Klasse *LiftingFacade* stellt Methoden zur Verfügung, um eine technische Differenz semantisch zu lfiten und zu serialisieren.

Package: org.sidiff.difference.asymmetric.api
Name: AsymmetricDiffFacade
Generalization: LiftingFacade
Description:
Die Klasse *AsymmetricDiffFacade* stellt Methoden zur Verfügung, um eine geliftete, ausführbare, asymmetrische Differenz zu berechnen und zu serialisieren.

Package: org.sidiff.difference.lifting.api.settings

Name: LiftingSettings
Generalization: DifferenceSettings
Description:

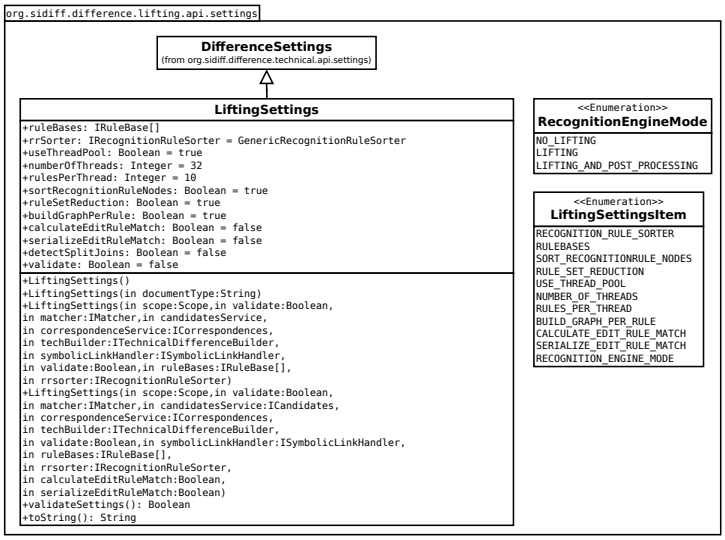

Abbildung 11: Lifting Settings

2.4 Patch Derivation and Application (org.sidiff.patching.api)

3 Extension Points

3.1 Matching Engine

Plug-in ID: org.sidiff.correspondences
Extension Points: org.sidiff.correspondences.extensionpoint
Extensions: none
Description:
TODO

Plug-in ID: org.sidiff.candidates
Extension Points: org.sidiff.candidates.extensionpoint
Extensions: none
Description:
TODO

Plug-in ID: org.sidiff.matcher
Extension Points: org.sidiff.matcher.extensionpoint
Extensions: none
Description:
This extension point is used to add new matchers. A plugin that adds this extension point has to implement the IMatcher interface.

Package: org.sidiff.matcher
Name: MatcherUtil
Generalization: none
Description:
Die Klasse *MatcherUtil* stellt diverse Hilfsmethoden zur Verfügung, um auf die Erweiterungen des Extension Points *org.sidiff.matcher.extensionpoint* zuzugreifen.

Plug-in ID: org.sidiff.similarities
Extension Points: org.sidiff.similarities.extensionpoint
Extensions: none
Description:
TODO

3.2 Difference Derivator

Plug-in ID: org.sidiff.difference.technical
Extension Points: org.sidiff.difference.technical.technical_difference_builder_extension
Extensions: none
Description:
This extension point is used to add a technical difference builder to the lifting engine. A plugin that adds this extension point has to implement the ITechnicalDifferenceBuilder interface. This interface offers the following information:

- The document type the technical difference builder was implemented for.

Package: org.sidiff.difference.technical.util
Name: TechnicalDifferenceBuilderUtil

Generalization: none
Description:
Die Klasse *TechnicalDifferenceBuilderUtil* stellt diverse Hilfsmethoden zur Verfügung, um auf die Erweiterungen des Extension Points *org.sidiff.difference.technical.technical_difference_builder_extension* zuzugreifen.

3.3 Lifting Engine

Plug-in ID: org.sidiff.difference.rulebase
Extension Points: org.sidiff.difference.rulebase.rulebase_extension
Extensions: org.eclipse.emf.ecore.generated_package
Description:
This extension point is used to add new rulebases to the recognition engine. A plugin that adds this extension point had to implement the IRuleBase interface. This interface offers the following information to the recognition engine:

- A description name of the rulebase.
- The document type the rulebase was generated for. In other words, the meta model the edit rules were implemented for.
- A list containing all rulebase recognition rules which will be applied by the recognition engine.

Package: org.sidiff.difference.rulebase.util
Name: RuleBaseUtil
Generalization: none
Description:
Die Klasse *RuleBaseUtil* stellt diverse Hilfsmethoden zur Verfügung, um auf die Erweiterungen des Extension Points *org.sidiff.difference.rulebase .rulebase_extension* zuzugreifen.

Plug-in ID: org.sidiff.difference.lifting.recognitionrulesorter
Extension Points: org.sidiff.difference.lifting.recognitionrulesorter.recognition_rule_sorter_extension
Extensions: none
Description:
This extension point is used to add new recognition rule sorter. A recognition rule sorter sorts a Henshin recognition rule to be optimized for matching

on a technical difference. A plugin that adds this extension point had to implement the IRecognitionRuleSorter interface. This interface offers the following information to the recognition engine:

- A description name of the sorter.
- The document type the sorter was implemented for.

3.4 Merging Engine

Plug-in ID: org.sidiff.conflicts.modifieddetector
Extension Points: org.sidiff.conflicts.modifieddetector
Extensions: None
Description:
TODO