

Introduction to SiDiff 2.0

Compare Functions

Timo Kehrer, Pit Pietsch

17. September 2009

Inhaltsverzeichnis

1	About this Document	3
2	Introduction	3
2.1	Comparison Rules	3
2.2	Compare Functions	3
2.3	Configuration File	4
2.4	Compare Function Interface	4
3	Trivial Compare Functions	4
4	Komplexe Vergleichsfunktionen	5
4.1	Einführendes Beispiel	5
4.2	Modularisierung durch Trennung der Aspekte WAS und WIE	5
4.2.1	Konzeptuelle Unterscheidung	5
4.2.2	Realisierung in SiDiff	6
4.3	Komparatoren	6
4.3.1	Prinzipielle Funktionsweise	6
4.3.2	Konfiguration und Anpassung	7
4.3.3	Die Komparator-Schnittstelle	7
5	Bedingte Vergleichsfunktionen	8
6	Existierende Vergleichsfunktionen und Komparatoren	8
6.1	Allgemeines	8
6.2	Vergleichsfunktionen	9
6.3	Komparatoren	9

Dokumenthistorie

Dieses Dokument wird fortlaufend gepflegt. Die nachfolgende Tabelle gibt eine Übersicht über die Änderungen in einzelnen Versionen.

Datum	Änderungen
07.09.09	erste Version (Grobstruktur)
11.09.09	erste vollständige Version

1 About this Document

Vergleichsfunktionen sind ein elementarer Bestandteil von SiDiff. In diesem Dokument wird zunächst der Einsatzkontext sowie die prinzipielle Funktionsweise von Vergleichsfunktionen beschrieben. Nach einer kurzen Betrachtung einfacher Vergleichsfunktionen wird ein mächtiger Mechanismus zur Komposition und Konfiguration komplexer Vergleichsfunktionen mittels Komparatoren beschrieben. Anschließend wird das Konzept der bedingten Vergleichsfunktionen betrachtet. Abschließend folgt eine Übersicht über die in SiDiff vorhandenen Vergleichsfunktionen und Komparatoren.

2 Introduction

The similarity computation is one of the key features of SiDiff. SiDiff supports the definition of arbitrary similarity heuristics to determine whether two model elements correspond or not. The heuristics are given by configuration files that define comparison rules for each type of element.

2.1 Comparison Rules

A comparison rule defines the element properties which are relevant for the similarity of two elements of the same type (i.e. elements that are instances of the same class in the metamodel). The properties are either local attributes (e.g. names) or other elements in the neighborhood.

Each compare function returns a value between 0.0 and 1.0; a value of 0.0 stands for no similarity between the properties, a value of 1.0 expresses equality. The comparison rules assign each property with a weight indicating the relevance of the property for the similarity of two elements. Further, each comparison rule specifies a threshold, i.e. a minimum similarity for two elements of this type to be eligible as corresponding elements.

More formally, the similarity between two elements is defined as the weighted arithmetic mean of the similarities of the similarity-relevant properties.

$$sim_{e_1, e_2} = \sum_{p \in P} w_p \cdot compare_p(e_1, e_2),$$

where e_1 and e_2 are the elements to be compared, P is the set of similarity-relevant properties, w_p gives the weight of property p and $compare_p$ is the compare function (see Section 2.2) for property p .

2.2 Compare Functions

According to the formula introduced in Section 2.1, the comparison rules assign a compare function to each property considered for the similarity. SiDiff provides several compare functions. A very simple example is a compare function that simply checks whether the model elements that should be compared are equal (cf. Section 3).

Generally, a compare function takes two model elements, one from model A and one from model B, as input and returns a value between 0.0 and 1.0. A value of 1.0 means that both model elements are identical with respect to the property that is considered by a specific compare function. A value of 0.0 means that both model elements have no similarity with respect to a specific property.

2.3 Configuration File

Comparison rules and compare functions are defined within a configuration file. The following listing shows a very simple configuration rule containing exactly one trivial compare function (cf. Section 3):

```
<Class name="PrimitiveType" threshold="1.0">
  <CompareFunction class="Equality" weight="1.0"/>
</Class>
```

The example is taken from a configuration file for UML models. The comparison rule states, that model elements of type `PrimitiveType` should be compared with respect to a trivial property: equality. The threshold for this comparison rule is 1.0; two elements of type `PrimitiveType` can only correspond if they are equal.

The actual comparison with respect to equality is done by a compare function called `Equality`. The weight of the compare function is of course 1.0 because it is the only compare function used by this comparison rule. In case of complex compare functions, these can be further configured by means of an additional parameter string. For further information on the available parameters and the syntactical structure of the parameter, please refer to the JavaDoc of the respective compare function.

2.4 Compare Function Interface

Die Schnittstelle von Vergleichsfunktionen ist sehr einfach aufgebaut. Sie besteht aus einem Anteil zur Konfiguration und der eigentlichen Methode zur Berechnung der Ähnlichkeit zwischen zwei Modellelementen. Da zur Repräsentation von Modellen in Sidiff das Eclipse Modeling Framework (EMF, s. WhitePaper "Introduction to Modelmanagement") eingesetzt wird, werden Modellelemente als Instanz der EMF-Klasse *EObject* repräsentiert. Die Initialisierung wird während der Instanziierung von Vergleichsfunktionen durchgeführt. Jede Vergleichsfunktion ist von der abstrakten Klasse `CompareFunction` abzuleiten.

3 Trivial Compare Functions

In Abschnitt 4 wird ein sehr mächtiger Mechanismus zur Realisierung von Vergleichsfunktionen eingeführt. Dabei werden konfigurierbare Komparatoren benutzt, um komplexe Vergleichsfunktionen wie bspw. den Vergleich einer bestimmten Eigenschaft aller Kindelemente zweier zu vergleichender Modellelemente zu realisieren. Als triviale Vergleichsfunktionen hingegen bezeichnen wir solche Vergleichsfunktionen, die nicht nach diesem Mechanismus aufgebaut sind.

Dies ist dann der Fall, wenn es sich um sehr spezielle, nicht weiter konfigurierbare Vergleichsfunktionen für einen ganz bestimmten Einsatzkontext handelt, oder die zu vergleichende Eigenschaft sehr einfach ist. In Abschnitt 2 wurde bereits die Eigenschaft der Gleichheit als ein Vertreter trivialer Vergleichsfunktionen eingeführt. Weitere Beispiele sind `MaximumSimilarity` und `NoSimilarity` welche stets die Werte 1.0 bzw. 0.0 als Ähnlichkeit für zwei Modellelemente des gleichen Typs zurückliefern.

4 Komplexe Vergleichsfunktionen

In den Abschnitten 4.2 und 4.3 wird der prinzipielle Aufbau und die Konfiguration komplexer Vergleichsfunktionen beschrieben. Zur anschaulichen Erläuterung dient das in Abschnitt 4.1 eingeführte Beispiel.

4.1 Einführendes Beispiel

Als Beispiel soll eine Vergleichsfunktion dienen, welche zwei Modellelemente anhand ihrer Kindelemente vergleicht. Wir gehen davon aus, dass die Kindelemente geordnet sind. Der Vergleich der beiden Listen soll mittels LCS durchgeführt werden. Basis des Vergleichs einzelner Kindelemente im Rahmen des LCS soll eine lokale Eigenschaft der Kindelemente, hier als Attribut bezeichnet, darstellen. Der Datentyp des Attributwerts entscheidet schließlich darüber, wie die Attributwerte miteinander verglichen werden sollen. In unserem Fall soll ein einfacher String-Vergleich ohne Beachtung von Groß- und Kleinschreibung durchgeführt werden.

4.2 Modularisierung durch Trennung der Aspekte WAS und WIE

Wir beschreiben im Folgenden sowohl die konzeptuelle Unterscheidung der beiden Aspekte WAS und WIE, als auch die Realisierung der Unterscheidung im Rahmen von SiDiff.

4.2.1 Konzeptuelle Unterscheidung

Zunächst stellt sich bei der Realisierung komplexer Vergleichsfunktionen die Frage, WAS miteinander verglichen werden soll. So z.B.

- lokale Eigenschaften,
- die Kindelemente,
- die Elternelemente,
- über einen Pfadausdruck erreichbare Modellelemente,
- etc.

der zu vergleichenden Modellelemente. In dem in Abschnitt 4.1 eingeführten Beispiel besteht der WAS-Aspekt also aus den beiden geordneten Listen der Kindelemente.

Zusätzlich zur Frage, WAS im Kontext einer Vergleichsfunktion für den Vergleich zweier Modellelemente herangezogen werden soll muss spezifiziert werden, WIE die ausgewählten lokalen Eigenschaften bzw. Modellelemente zu vergleichen sind. In unserem Referenzbeispiel muss spezifiziert werden, WIE die Kindelemente miteinander verglichen werden.

4.2.2 Realisierung in SiDiff

Zur Selektion der zu vergleichenden Modellelemente oder Attributwerte existiert in SiDiff bereits eine Menge von *Vergleichsfunktionen* (s. Abschnitt 6.2). Es ist zu beachten, dass der Begriff *Vergleichsfunktion* an dieser Stelle überladen wird. Bisher wurde in diesem Dokument unter einer Vergleichsfunktion eine Funktion zum Vergleich zweier Modellelemente auf Basis bestimmter Eigenschaften verstanden, welche die Aspekte WAS und WIE gleichermaßen umfasst. Hier führen wir die Vergleichsfunktion als funktionalen Bestandteil einer gesamten Vergleichsfunktion ein, welcher lediglich die Selektion der zu vergleichenden Elemente bzw. Attributwerte, also den WAS-Aspekt, umfasst. Die jeweilige Bedeutung des Begriffs Vergleichsfunktion sollte sich anhand des Kontexts erschließen lassen.

WIE die selektierten Modellelemente bzw. Attributwerte miteinander zu vergleichen sind wird an weitere funktionale Einheiten delegiert; die Komparatoren. Deren Aufbau und Konfiguration werden im Folgenden näher betrachtet.

4.3 Komparatoren

In diesem Abschnitt werden die prinzipielle Funktionsweise sowie die Konfiguration von Komparatoren beschrieben.

4.3.1 Prinzipielle Funktionsweise

Die Dekomposition in kleine, funktionale Einheiten, sowie die hierarchische Schachtelung von Komparatoren ist das Kernprinzip der SiDiff-Komparatoren. Ähnlich wie Vergleichsfunktionen den eigentlichen Vergleich zweier selektierter Eigenschaften an Komparatoren delegieren, können auch Komparatoren Teile des Vergleichs wieder an weitere Komparatoren delegieren.

Jeder Komparator erfüllt dabei eine klar definierte, abgeschlossene Aufgabe. Komplexe Vergleiche lassen sich durch die Komposition von Komparatoren zu einer Art Pipeline realisieren. Zur Realisierung des eigentlichen Vergleichs der Kindelemente in unserem Referenzbeispiel lassen sich bspw. folgende Komparatoren einsetzen: Zunächst führt ein Komparator zum abstrakten Vergleich von Listen nach dem LCS-Prinzip den Vergleich der Beiden Listen durch. Um die Ähnlichkeit der in den Listen enthaltenen Modellelemente zu bestimmen, wird ein weiterer Komparator benutzt, hier ein Komparator zum Vergleich von lokalen Attributen. Dieser wiederum kann den eigentlichen Wertevergleich der lokalen Attribute an einen weiteren Komparator abhängig vom

Typ (numerisch, String, etc.) der zu vergleichenden Werte delegieren. Hier wird ein Komparator zum case-insensitiven Vergleich von String-Werten eingesetzt.

Ähnlich wie Vergleichsfunktionen den eigentlichen Vergleich zweier selektierter Eigenschaften an Komparatoren delegieren, können auch Komparatoren Teile des Vergleichs wieder an weitere Komparatoren delegieren. So z.B. den Vergleich zwei Listen von Modellelementen. Eine typische Konfiguration spezifiziert einen Komparator, der ein Verfahren zum abstrakten Vergleich von Listen realisiert, so z.B. nach dem LCS-Prinzip. Um die Ähnlichkeit der in den Listen enthaltenen Modellelemente zu bestimmen, wird ein weiterer Komparator benutzt, z.B. ein Komparator zum Vergleich von lokalen Eigenschaften. Dieser wiederum kann den eigentlichen Wertevergleich der lokalen Attribute an einen weiteren Komparator abhängig vom Typ (numerisch, String, etc.) der zu vergleichenden Werte delegieren.

4.3.2 Konfiguration und Anpassung

Die Mächtigkeit des Komparator-Konzepts besteht in der Konfiguration sowie der hierarchischen Schachtelung von Komparatoren im Rahmen der Konfigurationsdatei. Das folgende Listing bezieht sich auf das im Verlauf dieses Kapitels behandelte Referenzbeispiel.

```
<CompareFunction class="ChildrenCF" weight="0.2"
  parameter="LCLongestCommonSubsequence [
    ECAttributeStatic [VCString [ci]; SomeType; someAttributeName]; 0.6
  ]"/>
```

Komparatoren werden, wie auch Vergleichsfunktionen, über einen Parameter-String konfiguriert. Die einzigen syntaktischen Elemente eines Parameter-String sind das Semikolon (“;”) sowie die eckigen Klammern (“[” und “]”). Eckige Klammern umfassen den Parameter-String eines Komparators, das Semikolon trennt einzelne Segmente eines solchen Parameter-Strings voneinander ab. Im dargestellten Beispiel wird als erster Komparator in der Pipeline der LCLongestCommonSubsequence benutzt. Dieser erwartet offenbar vier Parameter: Einen weiteren Komparator (hier: ECAttributeStatic) und einen numerischen Wert (hier: 0.6). Bei dem numerischen Wert handelt es sich um einen Threshold, dessen genaue Bedeutung der Code-Dokumentation der Klasse LCLongestCommonSubsequence zu entnehmen ist. Der Komparator ECAttributeStatic erwartet drei Parameter: den konkreten Komparator zum Vergleich der Attributwerte (hier: Ein String-Komparator (VCString) unter Nichtbeachtung von Groß- und Kleinschreibung¹), eine Typinformation der zu vergleichenden Elemente (hier: SomeType) sowie den Namen des zu vergleichenden Attributs (hier:someAttributeName).

4.3.3 Die Komparator-Schnittstelle

Die Schnittstelle von Komparatoren ist ähnlich aufgebaut wie die der Vergleichsfunktionen. Sie besteht aus einem Anteil zur Initialisierung des Komparators sowie einer

¹der Parameter ci steht für case-insensitive, s. auch JavaDoc

Methode zur Durchführung des eigentlichen Vergleichs. Im Gegensatz zu Vergleichsfunktionen werden hier jedoch nicht zwangsläufig zwei Modellelemente (also Instanzen der EMF-Klasse EObject), sondern beliebige Java-Objekte übergeben. So z.B. Attributwerte, Listen von Modellelementen, Modellelemente etc.

5 Bedingte Vergleichsfunktionen

TODO: Future Work

6 Existierende Vergleichsfunktionen und Komparatoren

6.1 Allgemeines

Eine Unterscheidung von Vergleichsfunktionen konzeptueller Natur ist die bereits beschriebene Unterscheidung von trivialen (s. Abschnitt 3 und komplexen Vergleichsfunktionen (s. Abschnitt 4).

Weiterhin lassen sich die verschiedenen Vergleichsfunktionen anhand eines technischen Merkmals unterscheiden: der Abhängigkeiten zu anderen SiDiff-Komponenten (OSGI-Bundles). Um die Abhängigkeit zu anderen Bundles bei der Konfiguration von SiDiff-Applikationen möglichst auf ein Minimum zu reduzieren haben wir uns dazu entschlossen, die Vergleichsfunktionen anhand ihrer Abhängigkeit zu anderen SiDiff-Komponenten auf verschiedene Bundles zu verteilen. Momentan existierende folgende Bundles:

- `org.sidiff.core.compare.comparefunctions:`
Vergleichsfunktionen ohne jegliche Abhängigkeiten zu anderen Bundles. Insbesondere fallen unter diese Kategorie Vergleichsfunktionen ohne Modellzugriff, so z.B. die in Abschnitt 3 bereits angesprochene Vergleichsfunktion `MaximumSimilarity`. Alle folgenden Vergleichsfunktionen greifen auf die Modelle zu.
- `org.sidiff.core.compare.comparefunctions.emf:`
Ausschließlich EMF-spezifische Vergleichsfunktionen
- `org.sidiff.core.compare.comparefunctions.emf.annotations:`
Vergleichsfunktionen welche Annotationswerte auslesen
- `org.sidiff.core.compare.comparefunctions.emf.correspondence:`
Vergleichsfunktionen auf Basis von Korrespondenzen
- `org.sidiff.core.compare.comparefunctions.emf.similarity:`
Vergleichsfunktionen auf Basis von Ähnlichkeiten
- `org.sidiff.core.compare.comparefunctions.emf.correspondenceandsimilarity:`
Vergleichsfunktionen welche die Korrespondenz- und die Ähnlichkeitstabelle benutzen

6.2 Vergleichsfunktionen

Triviale Vergleichsfunktionen Die in SiDiff vorhandenen, trivialen Vergleichsfunktionen wurden bereits in Abschnitt 3 vollständig aufgeführt und sollen hier nicht weiter betrachtet werden.

Vergleichsfunktionen zur Verwendung in Verbindung mit Komparatoren Abb. 1 liefert einen Überblick über die in SiDiff verfügbaren Vergleichsfunktionen, welche in Verbindung mit Komparatoren eingesetzt werden können. Diese Vergleichsfunktionen erfüllen den in Abschnitt 4.2 beschriebenen WAS-Aspekt bei der Realisierung komplexer Vergleichsfunktionen. Ihre primäre Aufgabe besteht also darin, lokale Eigenschaften oder Elementmengen bzw. -listen zu berechnen und die den weiteren Vergleich an Komparatoren zu delegieren. Die Semantik der jeweiligen Vergleichsfunktionen ist der JavaDoc der entsprechenden Bundles (s. Abschnitt 6.1) zu entnehmen.

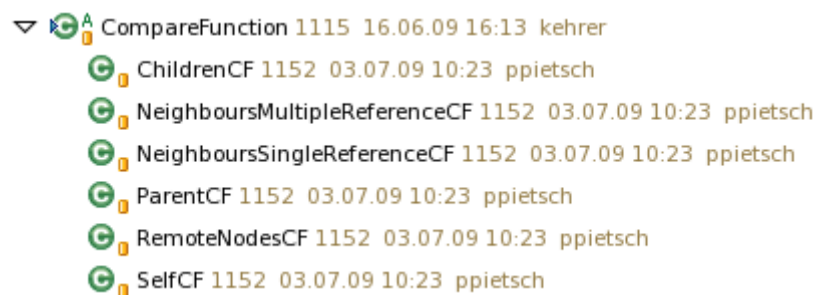


Abbildung 1: Verfügbare Vergleichsfunktionen zur Verwendung in Verbindung mit Komparatoren

6.3 Komparatoren

SiDiff stellt bereits einige Komparatoren für verschiedenste Einsatzzwecke zur Verfügung. Eine Übersicht über alle Komparatoren ist Abb. 2 zu entnehmen. Es ist dabei zu beachten, dass es sich lediglich bei den Blättern der in Abb. 2 dargestellten Vererbungshierarchie um konkrete Komparatoren handelt. Alle anderen Komparatoren sind abstrakt und dienen lediglich einer Klassifizierung der Funktionsweise der Komparatoren.

TODO: Hier sollten wohl eher die einzelnen Kategorien und deren Intention näher beschrieben werden, als einfach nur die vorhandenen Komparatoren abzubilden!?






























- ▼   AbstractComparator 1152 03.07.09 10:23 ppietsch
 - ▼  AbstractCollectionComparator 1152 03.07.09 10:23 ppietsch
 - ▼  AbstractListComparator 1152 03.07.09 10:23 ppietsch
 -  LCAAlignedList 1152 03.07.09 10:23 ppietsch
 -  LCLongestCommonSubsequence 1319 26.08.09 10:51 theod
 - ▼  AbstractSetComparator 1152 03.07.09 10:23 ppietsch
 -  SCGreedyMatchedOrSimilar 1355 04.09.09 14:00 wenzel
 -  SCGreedySimilarity 1151 03.07.09 10:22 ppietsch
 -  SCMatched 1301 21.08.09 16:06 wenzel
 - ▼  AbstractSingleComparator 1152 03.07.09 10:23 ppietsch
 - ▼  AbstractElementComparator 1152 03.07.09 10:23 ppietsch
 -  ECAAnnotation 1391 09.09.09 17:20 kehrer
 -  ECAAttributeDynamic 1152 03.07.09 10:23 ppietsch
 -  ECAAttributeStatic 1152 03.07.09 10:23 ppietsch
 -  ECMatched 1152 03.07.09 10:23 ppietsch
 -  ECMatchedOrSimilar 1152 03.07.09 10:23 ppietsch
 -  ECSimilarity 1151 03.07.09 10:22 ppietsch
 - ▼  AbstractValueComparator 1152 03.07.09 10:23 ppietsch
 -  CEquals 1152 03.07.09 10:23 ppietsch
 -  CEqualType 1259 06.08.09 09:29 ppietsch
 -  VCCharacterEqualsCI 1324 26.08.09 15:07 andres
 -  VCGauss 1322 26.08.09 14:48 theod
 -  VCStringEMFCompare 1229 24.07.09 16:05 ppietsch
 -  VCStringEqualsCI 1259 06.08.09 09:29 ppietsch
 -  VCStringIndexOf 1304 24.08.09 13:22 andres
 -  VCStringLCS 1396 09.09.09 17:45 kehrer
-  CHeaviside 1320 26.08.09 14:37 andres
-  CHeavisideReverse 1334 28.08.09 14:27 theod

Abbildung 2: Verfügbare Komparatoren in SiDiff