SiLift - Benutzerhandbuch für Endanwender

Universität Siegen - Praktische Informatik

14. April 2014

Inhaltsverzeichnis

1 Einleitung

SiLift ist ein *Eclipse*-basiertes Framework mit dessen Hilfe sich Differenzen von *EMF-Modellen semantisch liften* lassen. Des Weiteren kann SiLift dazu verwendet werden auf Basis einer solchen Differenz einen *Patch* zu generieren, um diesen auf ein anderes Modell anzuwenden. Ebenfalls besteht die Möglichkeit des *3-Wege-Mischens* mit entsprechender Konflikterkennung.
Dieses Benutzerhandbuch umfasst eine Installationsanleitung sowie einführende Tutorials zur Anwendung von SiLift als Endanwender.

2 Voraussetzung und Installation

SiLift ist als *Eclipse-Feature* unter folgender *Update-Site* erhältlich:
`http://pi.informatik.uni-siegen.de/Projekte/SiLift/updatesite`.

Hinweis: Vergewissern Sie sich, ob ihr Eclipse die notwendigen Voraussetzungen erfüllt. Eine Liste der benötigten Plugins ist unter `http://pi.informatik.uni-siegen.de/Projekte/SiLift/download.php` zu finden. Bitte beachten Sie dabei die entsprechenden Hinweise zu den jeweiligen Versionen.

Sofern alle Voraussetzungen erfüllt sind, kann SiLift wie gewohnt über den Menüpunkt `Help` ▷ `Install New Software...` installiert werden (Abb. 1).

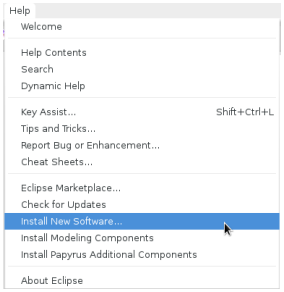

Abbildung 1: Eclipse: Install New Software...

Es sollten Ihnen vier Kategorien angezeigt werden (vgl. Abb. 2).

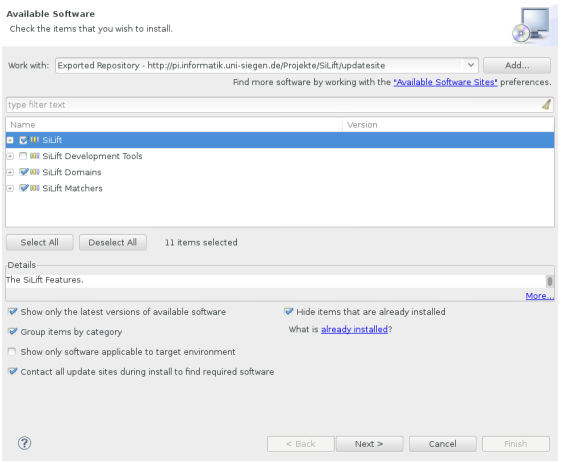

Abbildung 2: SiLift Update Site

Für die folgenden Tutorials benötigen wir alle Features aus der Kategorie `SiLift`, das Feature `SiLift Ecore Domain` aus der Kategorie `SiLift Domains`, sowie den `SiLift Named Element` und `SiLift UUID Matcher` aus `SiLift Matchers`. Sie könnenn aber auch wie in Abbildung 2 die kompletten Kategorien auswählen. Danach klicken Sie auf `Next` und folgen dem Installationsassistenten.

Hinweis: Generell unterstützt SiLift alle *EMF-basierten* Modellierungsprachen, sofern die entsprechenden *Editieroperationen* implementiert wurden. In der aktuellen Version stehen diese bereits für *Ecore-* und *Feature-Modelle* zur Verfügung.[1]

3 SiLift benutzen

3.1 Vergleich von Modellen

Die Bedienung von SiLift als Vergleichswerkzeug soll am folgenden Beispiel demonstriert werden. Ausgangsbasis sind die *Ecore-Modelle* in Abbildung 3.

[1] Informationen zur Integration weiterer Modelltypen finden Sie im **SiLift - Benutzerhandbuch für Entwickler**.

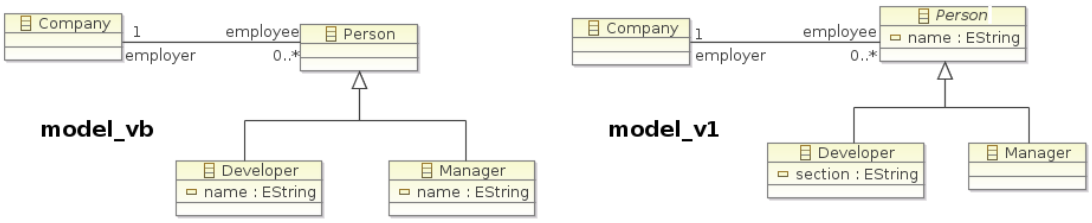

Abbildung 3: Ecore-Modelle

Dabei stellt `model_vb` das Ausgangsmodell und `model_v1` das im Laufe eines Entwicklungsprozesses veränderte Modell dar. Die entscheidenden Änderungen sind zum einen das Attribut `name`, welches durch den Entwicklungsprozess in die nun *abstrakte* Klasse `Person` verschoben wurde und das neue Attribut `section` in der Klasse `Developer`.
Als nächstes selektieren Sie die beiden *ecore*-Files im *Package Explorer* und öffnen mit der rechten Maustaste das Kontextmenü. Wählen Sie `SiLift` ▷ `Compare with each other` aus (vgl. Abb. 4).

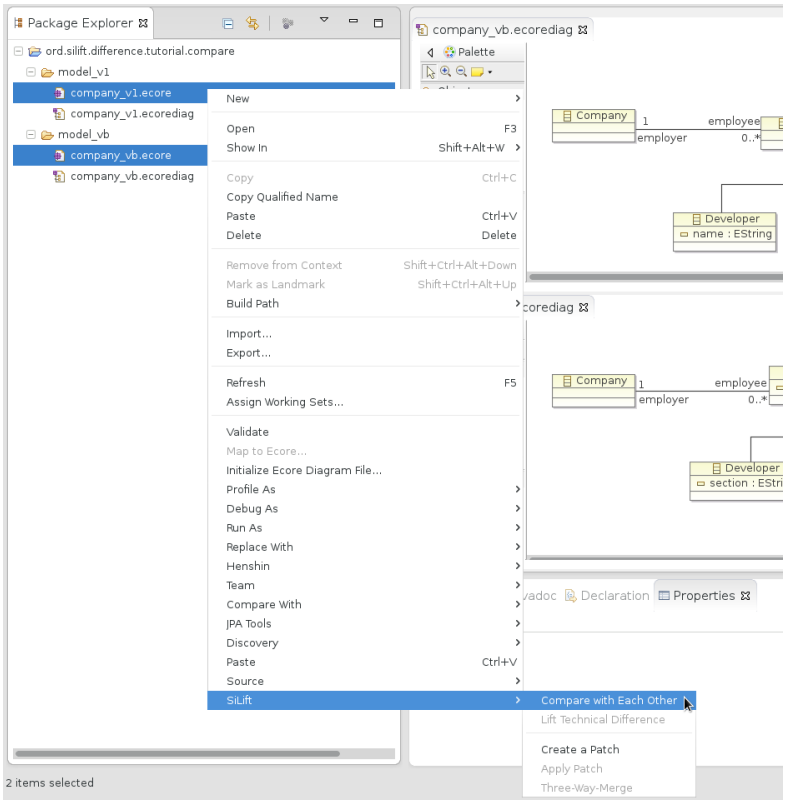

Abbildung 4: SiLift über das Kontextmenü starten

Es öffnet sich ein Wizard-Dialog, der sich über zwei Seiten erstreckt und mehrere Konfigurationsmöglichkeiten bietet (Abb. 5 und 6). Um diese besser zu verstehen, folgt ein kleiner Exkurs über die Architektur von *SiLift*.

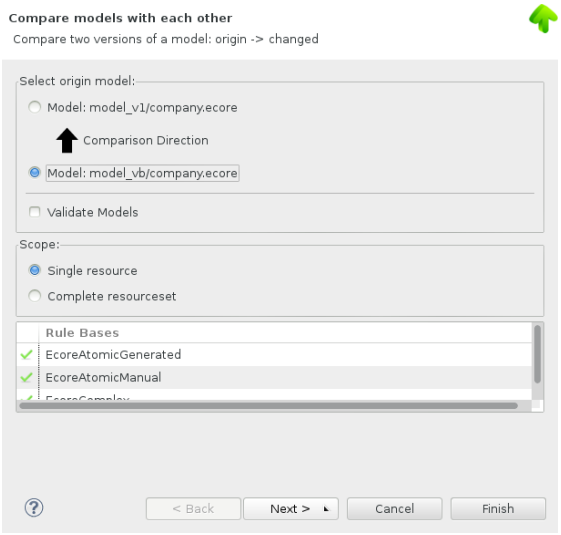

Abbildung 5: Einstellungen für das Erstellen gelifteter Differenzen: Seite 1

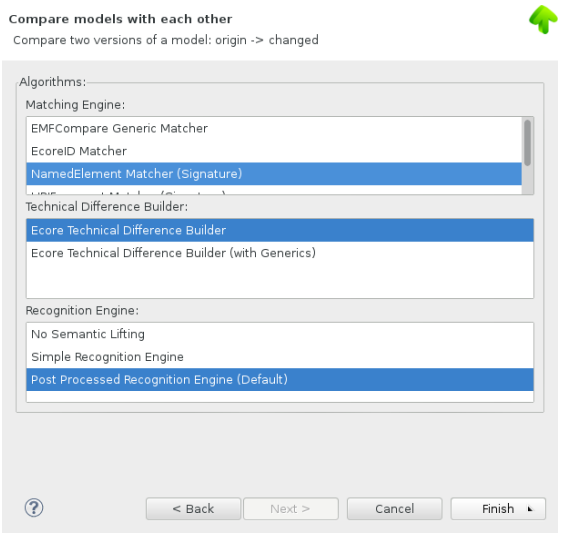

Abbildung 6: Einstellungen für das Erstellen gelifteter Differenzen: Seite 2

Exkurs: Differenz-Pipeline

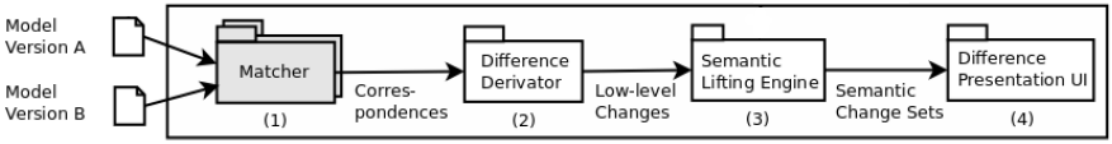

Abbildung 7: SiLift Processing Pipeline

Die Vorgehensweise von SiLift lässt sich am besten mit einer vierstufigen *Pipeline*, wie in Abbildung 7 dargestellt, vergleichen. Als Eingabe dienen immer zwei Versionen eines Modells, z.B. `model_A.ecore` und `model_B.ecore` (Abb. 3):

1. **Matching**: Aufgabe eines *Matcher* 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 unterschieden zwischen *ID-*, *signatur-* und *ähnlichkeitsbasierten* Verfahren.
   In SiLift stehen unter anderem folgende Matcher-Engines zur Verfügung:

   - `EcoreID Matcher`: Ein *ID-basierter* Matcher (nutzt Werte von Attributen, die im Metamodell als ID-Attribute deklariert sind).
   - `EMF Compare`: Unterstützt alle drei Verfahren. `EMF Compare` kann unter `Window` ▷ `Preferences`: `EMF Compare` konfiguriert werden. [2]
   - `NamedElement Matcher`: Ein signaturbasierter Matcher, welcher die entsprechenden Korrespondenzen anhand der Werte der jeweiligen Namensattribute bestimmt.
   - `URIFragment Matcher`: Ein signaturbasierter Matcher, welcher die entsprechenden Korrespondenzen anhand der Werte der *Uri* der Elemente bestimmt (z.B. `eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore-#//EString"`).
   - `UUID Matcher`: Ein ID-basierter Matcher. (basiert auf XMI-IDs der XMI-Repräsentationen der Modelle, falls vorhanden).

   (Abb. 6). Diese Liste ist keineswegs abgeschlossen und kann durch zusätzliche

[2] Informationen zum `EMF Compare Project` finden Sie unter `http://www.eclipse.org/emf/compare`.

Matching-Engines, wie z.B. *SiDiff* oder auch eigener Matcher ergänzt werden.[3]

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 (vgl. Abb. 8).

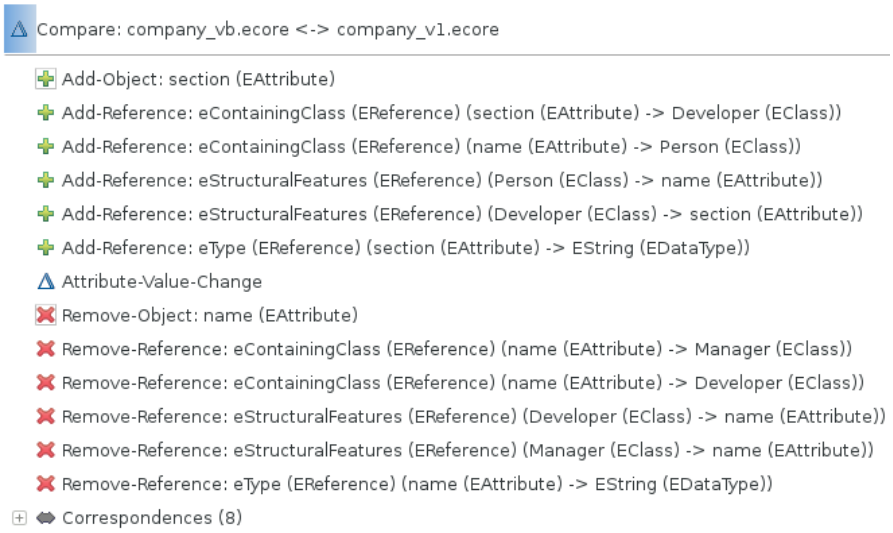

Abbildung 8: technische Differenz von `model_vb.ecore` und `model_v1.ecore`

Die Berechnung der technischen Differen kann durch die Wahl des *Technical Difference Builder* beeinflusst werden (vgl. Abb. 6).

3. **Semantic Lifting**: Die zuvor berechnete technische Differenz enthält alle Änderungen auf Basis des Metamodells. Diese sollen nun *semantisch geliftet* werden. Dazu werden die einzelnen Änderungen mit Hilfe von *Erkennungsregeln* (egnl. *recognition rules*) in sogenannte *Semantic Change Sets* gruppiert, die jeweils eine vom Benutzer ausgeführte *Editieroperation* repräsentieren. Mit der Wahl einer *Rule Base* wird festgelegt, welche Erkennungsregeln zum liften benutzt werden sollen. Dabei wird zwischen folgenden *Rule Bases* unterschieden: `AtomicGenerated`, `AtomicManual` und `Complex` (vgl. Abb. 5). Während atomare Regeln das Erzeugen,

[3] Weitere Informationen zu *SiDiff* finden Sie unter `http://pi.informatik.uni-siegen.de/Projekte/sidiff`

Löschen, Verschieben von Elementen und das Ändern der Attributwerte von Elementen umfassen, setzen sich die komplexen Editierregeln i.d.R. aus den atomaren und anderen komplexen Regeln zusammen.[4] Welchen Einfluss die Wahl einer oder mehrerer *Rule Bases* für das *semantische Liften* hat wird später noch am Beispiel demonstriert.

4. **Difference Presentation UI**: *SiLift* stellt einen *Compare View* bereit, um Differenzen übersichtlich darzustellen.

Damit endet die Pipeline. Über den Wizard kann man an jeder Position der Pipeline eingreifen und somit das Verhalten von SiLift beeinflussen. Zusätzlich lassen sich noch folgende Einstellungen vornehmen:

- **Select source model**: Die Differenzberechnung zwischen zwei Modellen ist nicht kommutativ. I.d.R. handelt es sich bei den zu vergleichenden Modellen um unterschiedliche Revisionen ein und desselben Modells. In den meisten Fällen wird man die ältere Revision (Modell VB) mit der neueren (Modell V1) vergleichen wollen. Dennoch kann auch der andere Fall eintreten. In *Select source model* können Sie die Richtung der Differenzberechnung festlegen.
  Zusätzlich kann man die Modelle vor der Differenzbildung noch validieren (vgl. Abb. 5).

- **Scope**: Ein Modell muss nicht in sich geschlossen sein, sondern kann auf andere Modelle bzw. deren Elemente verweisen. Mit Hilfe des Scopes kann man festlegen, ob diese Modelle bei der Erzeugung der Differenz ignoriert (`Single resource`), oder mit in diese aufgenommen werden sollen (`Complete resourceset`, vgl. Abb. 5).

- **Recognition-Engine**: Mit der Wahl einer *Recognition-Engine* wird festgelegt, ob und wie die technischen Differenzen geliftet werden. Wie der Name bereits andeutet, wird bei der Auswahl von `No Semantic Lifting` keine semantische Differnz erzeugt. Für das Erzeugen einer semantischen Differenz stehen zum einen die `Simple Recognition Engine`, zum anderen die `Post Processed Recognition Engine` zur Verfügung. Der Unterschied liegt im Auftreten von Überlappungen der *Semantic Change Sets*, die vor allem bei der zusätzlichen Verwendung von komplexen *Rule Bases* auftreten. Wenn Sie komplexe Erkennungsregeln nutzen

[4] Weitere Informationen zu den *Recognition Rules* finden Sie im **SiLift - Benutzerhandbuch für Entwickler**.

(und auch sonst) ist es daher ratsam die `Post Processed Recognition Engine` zu nutzen, um eben diese Überlappungen zu vermeiden (vgl. Abb. 6).

Nachdem Sie nun die Konfigurationsmöglichkeiten kennengelernt haben wird es Zeit *SiLift* auf die zuvor erstellten Modelle anzuwenden. In unserem Beispiel ist `company_vb` unsser Basismodell. Des Weiteren wählen wir den `NamedElement Matcher` und deaktivieren zunächst die komplexen Erkennungsregeln (vgl. Abb. 9).

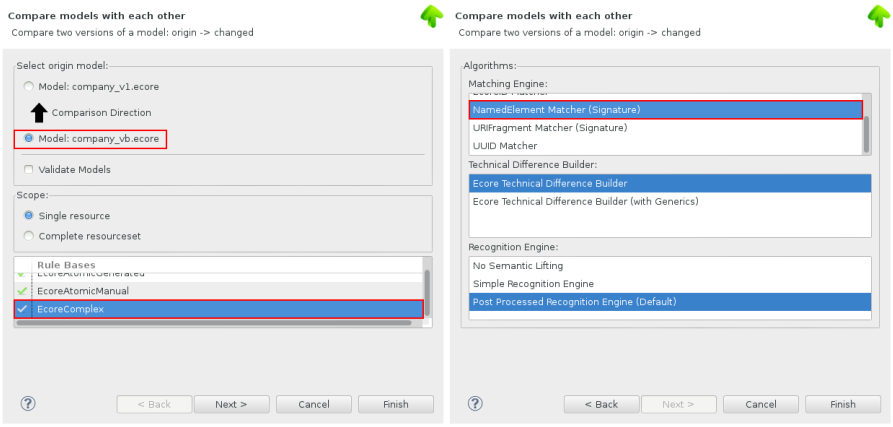

Abbildung 9: Einstellungen für das Erstellen einer gelifteten Differenz ohne komplexe Erkennungsregeln

Das Ergebnis wird in der `company_vb_x_company_v1_NamedElement_lifted_postprocessed.symmetric` gespeichert und lässt sich mit dem *Difference Model Editor* öffnen (vgl. Abb. 10).

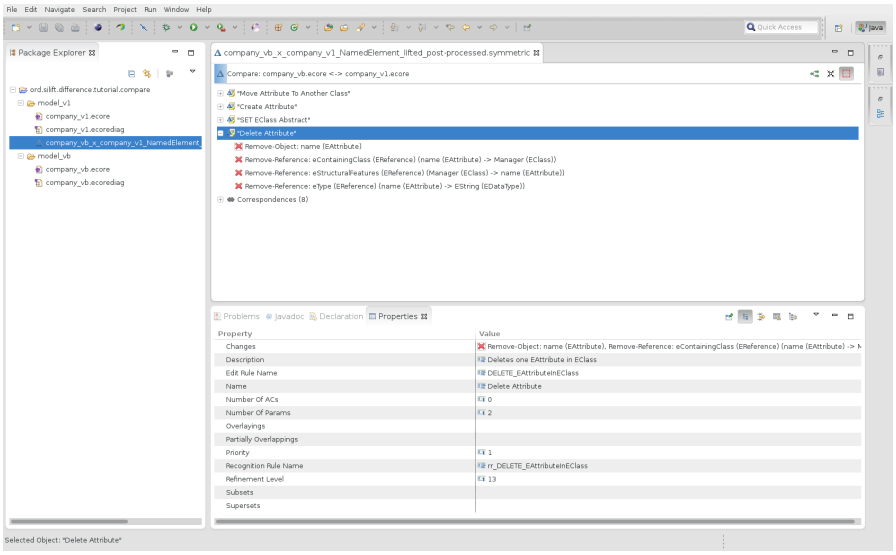

Abbildung 10: `company_vb_x_company_v1_NamedElement_lifted_postprocessed.symmetric`

Jedes *Sematic Change Set* steht für eine *Editieroperation*, welche im Laufe des Entwicklungszyklus auf Modell VB angewandt wurde. Somit werden Ihnen die Differenzen nun auf eine intuitive Weise präsentiert, ohne dass Sie das Metamodell in alle Einzelheiten kennen müssen. Durch das Aufklappen eines Change Sets können jedoch die jeweiligen technischen Differenzen weiterhin angezeigt werden (vgl. Abb. 10). Zusätzlich werden die gefundenen Korrespondenzen aufgelistet.
Neben dem baumbasierten Editor lassen sich die Differenzen mittels eines graphischen Editors anzeigen. Dieser lässt sich, wie in Abbildung 11 dargestellt, aufrufen.

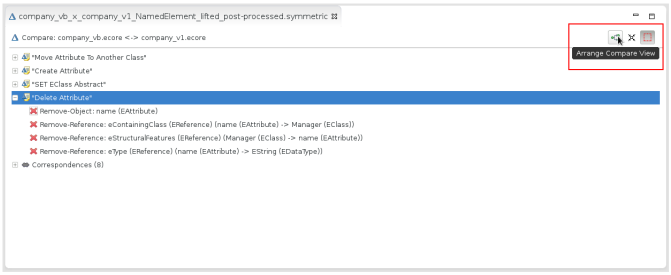

Abbildung 11: Aufruf des graphischen Editors (Compare View)

Durch Auswahl eines Change Sets werden die betroffenen Elemente in den Diagrammen *gehighlightet* (vgl. Abb. 12).

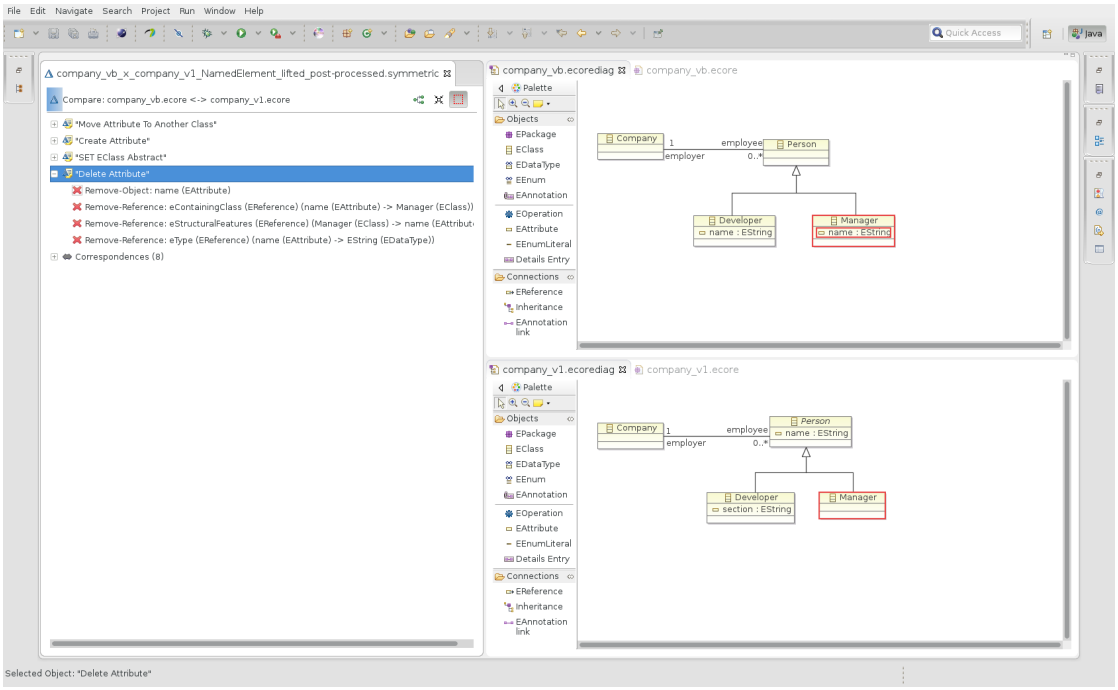

Abbildung 12: SiLift Compare View: vb nach v1 (nur atomare Editieroperationen)

Wenn wir uns nochmals die oben beschriebenen Änderungen ins Gedächtnis rufen, so lassen sich diesen die Editieroperationen wie folgt zuordnen:

- `Move Attribute To Another Class`: Verschiebe Attribut `name` von `Developer` nach `Person`.
- `Create Attribute`: Erstelle neues Attribut `section` in `Developer`.
- `SET EClass Abstract`: Die Klasse `Person` ist nun *abstrakt*.
- `Delete Attribute`: Entferne Attribut `name` aus `Manager`.

Nochmal zur Erinnerung: Die geliftete Differenz aus Abbildung 12 wurde mit Hilfe der atomaren Erkennungsregeln erstellt. Diese werden wiederum aus den atomaren Editierregeln abgeleitet. Eine atomare Editierregel kann nicht in noch kleinere Regeln aufgeteilt werden, ohne dass deren Anwendung zu einem inkonsistenten Modell führen würde. Diese Regeln umfassen i.d.R. das Erstellen (`create`), Entfernen (`remove`) und Verschieben (`move`) von Modellelementen sowie das Ändern von Attributwerten (`set`).
Betrachten wir die beiden Change Sets `Delete Attribute` und `Move Attribute To Another Class`. In diesem Szenario wurde das Attribut `name` in der Klasse `Manager`

gelöscht und aus der Klasse `Developer` nach `Person` verschoben. Gleichzeitig ließe sich diese Differenz der Modelle jedoch auch als ein *Refactoring* der Vererbungsbeziehung verstehen, indem übereinstimmende Attribute der Unterklassen in die Oberklasse verschoben wurden. Die für ein solches Refactoring erforderliche Erkennungsregel umfasst also mehrere atomare Regeln. Um solche Refactorings zu erkennen starten wir SiLift nun zusätzlich mit der komplexen *Rule Base* (vgl. Abb. 13). Die restlichen Einstellungen können Sie aus Abbildung 9 übernehmen.

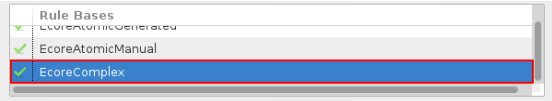

Abbildung 13: Einstellungen für das Erstellen einer gelifteten Differenz mit komplexen Erkennungsregeln

Das Ergebnis ist eine geliftete Differenz die anstatt vier nur noch drei Change Sets beinhaltet (Abb. 14). Hier wurden die atomaren Regeln `Delete Attribute` und `Move Attribute To Another Class` durch die komplexe Regel `Pull Up Attribute` ersetzt.

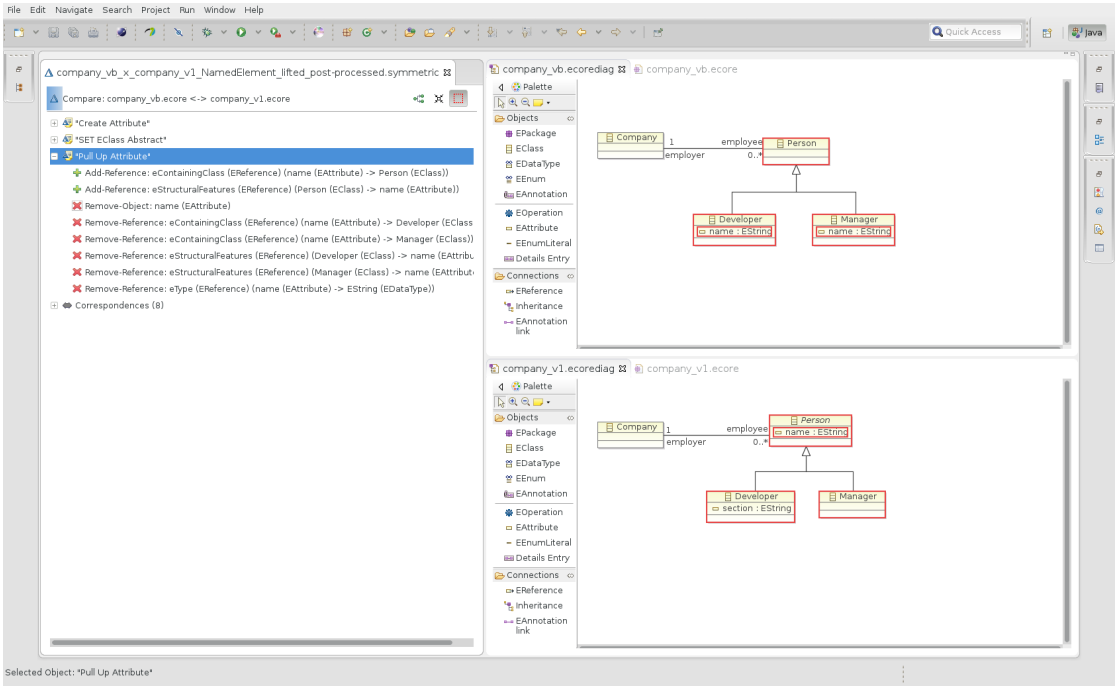

Abbildung 14: SiLift Compare View: vb nach v1 (incl. komplexen Editieroperationen)

Wie bereits erwähnt setzen sich komplexe Regeln aus atomaren und anderen komplexen Regeln zusammen. Betrachten wir Abbildung 15, so deckt die komplexe Regel `Pull Up Attribute` alle technischen Differenzen der beiden atomaren Regeln `Delete Attribute` und `Move Attribute To Another Class` ab.

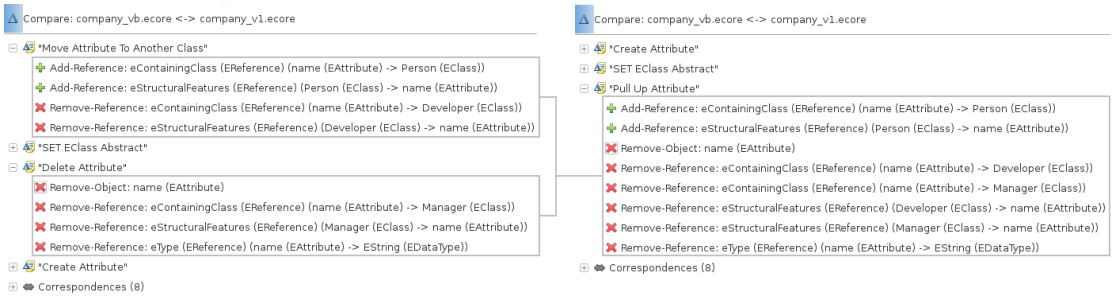

Abbildung 15: Vergleich atomarer und komplexer Erkennungsregeln

Durch komplexe Erkennungsregeln lassen sich somit größere Refactorings auf eine intuitive Weise darstellen.

3.2 Patchen von Modellen

Neben dem semantischen Liften von Differenzen besteht die Möglichkeit einen Patch zu erstellen.
Abbildung 16 zeigt ein typisches Szenario der Patch-Anwendung.

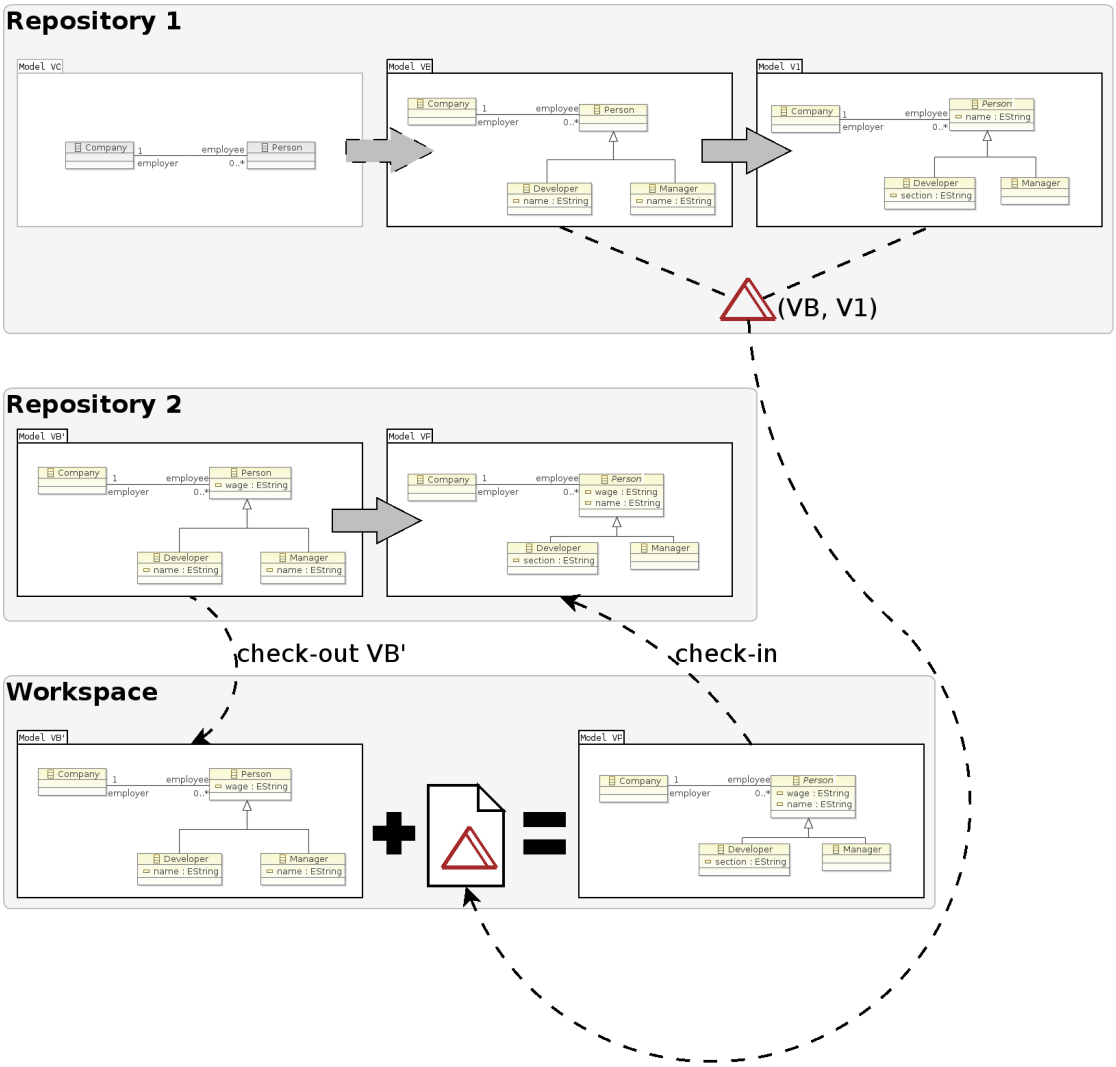

Abbildung 16: Ablauf einer Patch-Anwendung

`Repository 1` stellt den Entwichklungsprozess eines Modells dar, welches zu einem bestimmten Zeitpunkt (hier `Model VB`) in ein zweites Repository übertragen und dort ggf. weiterentwickelt wird (vgl. 16, `Repository 2`, `Model VB'`). Des Weiteren wird das Modell auch in Repository 1 weiterentwickelt bzw. überarbeitet. Dieses Refactoring soll nun

auf Model VB' in Repository 2 angewandt werden, ohne dass die bereist vorgenommenen Änderungen an diesem verloren gehen. Zu diesem Zweck wird eine asymmetrische Differenz zwischen Model VB und Model V1 gebildet und in Form eines Patches auf Model VB' angewandt (vgl. 16, `Workspace`).
Sofern in beiden Modellen Änderungen an dem gleichen Element vorgenommen wurden können Konflikte auftreten. An dieser Stelle muss man zwichen der Anwendung eines Patches und dem Mischen von Modellen unterscheiden. Ein Patch wird i.d.R. auf eine Menge von Kopien eines Modells angewandt. Sofern diese Kopien parallel weiterentwickelt wurden und dies zu Konflikten führt, werden die betroffenen Änderungen verwofen und durch die des Patches ersetzt. Bei größeren Änderungen kann es jedoch sein, dass der Patch nicht mehr als Ganzes anwendbar ist. Beim Mischen von Modellen wird bei einem Konflikt eine Entscheidung getroffen, welche Änderung beibehalten bzw. verworfen wird (vgl. Abschnitt 3.3).

3.2.1 Erstellen eines Patches

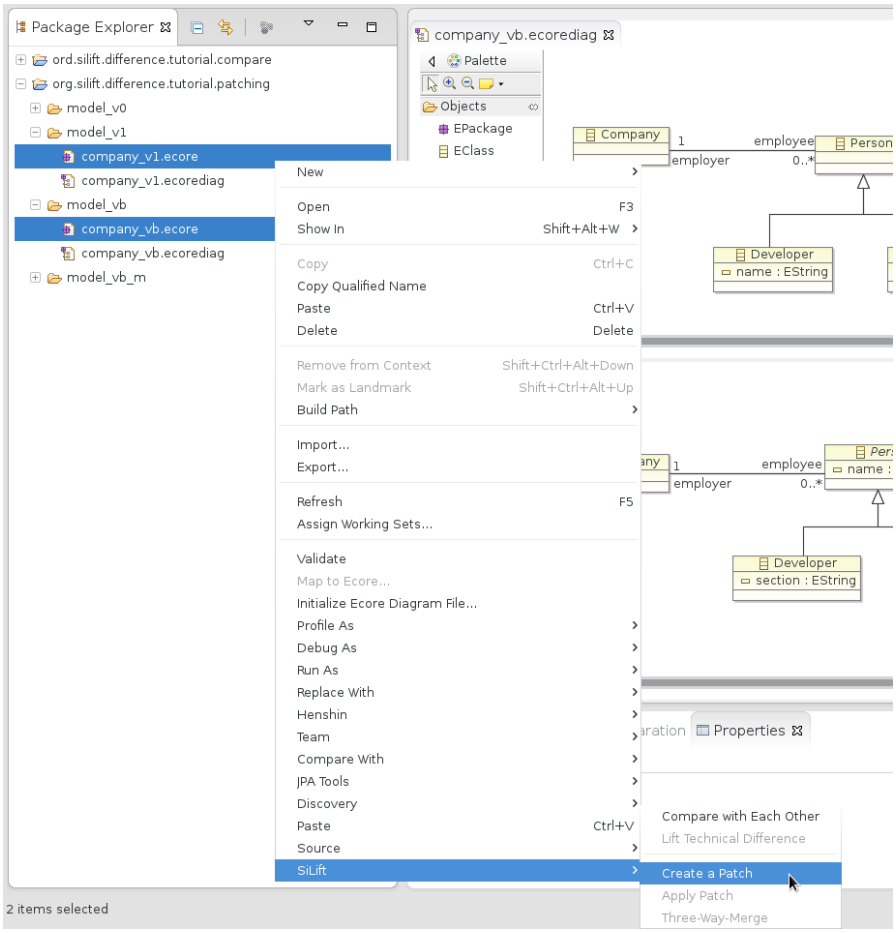

Abbildung 17: SiLift: Patch erstellen

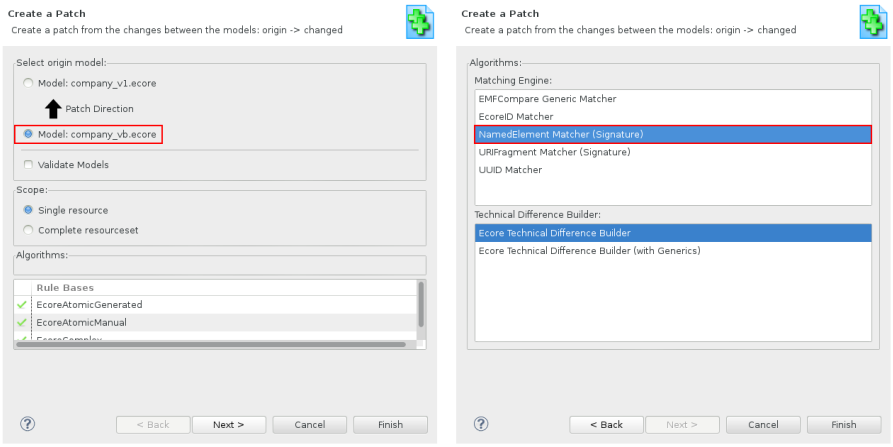

Abbildung 18: Patch-Config

3.2.2 Anwenden eines Patches

– TBD –

3.3 Mischen von Modellen

– TBD –

ENDE

4 Links und weitere Informationen

- **EMF-Compare**: `http://www.eclipse.org/emf/compare`
- **SiDiff**: `http://pi.informatik.uni-siegen.de/Projekte/sidiff/`
- **SiLift**: `http://pi.informatik.uni-siegen.de/Projekte/SiLift/`