Taming EMF Compare 2

Swathi Raghavan

Syed Aoun Raza

Source: http://www.millionstock.com - 2017
Large scale incremental models

Textual XML comparison

CUSTOMIZED UI

EMF

DIFF MODEL DELTA

Special Merge
EMF Compare is the *de facto* comparison tool used in most model-based eclipse projects.
Building Blocks of EMF Compare 2

Core Components

Scope Preparation

Matching

Differencing

Merging

UI Component

Visualization
Comparison Scope Strategies

→ Default
- Retrieves all content under a ECore Resource
- Filters out some specific contents e.g., EGenericTypes
- Follows containment structure
Comparison Scope Strategies

- Default
  - Retrieves all content under a ECore Resource
  - Filters out some specific contents e.g., EGenericTypes
  - Follows containment structure

```java
IComparisonScope comparisonScope = EMFCompare.createDefaultScope(resourceSet1, resourceSet2);
```
Comparison Scope Strategies

- **Default**
  - Retrieves all content under a ECore Resource
  - Filters out some specific contents e.g., EGenericTypes
  - Follows containment structure

```java
IComparisonScope comparisonScope = EMFCompare.createDefaultScope(resourceSet1, resourceSet2);
```

- **Tailored**
  - Every use case can define its own scope (Extensibility)
  - Our Model
    - Optimized EMF-Model
    - Follows reference structure (everything is not covered containment)
  - Switch the default scope with tailored implementation
Comparison Scope Strategies

- **Default**
  - Retrieves all content under a ECore Resource
  - Filters out some specific contents e.g., EGenericTypes
  - Follows containment structure

```java
IComparisonScope comparisonScope = EMFCompare.createDefaultScope(resourceSet1, resourceSet2);
```

- **Tailored**

```java
import org.eclipse.emf.compare.scope.DefaultComparisonScope;

public class ExtendedComparisonScope extends DefaultComparisonScope {

    /**
     * {@inheritDoc}
     */
    @Override
    public Iterator<? extends EObject> getChildren(final EObject eObject) {
        ...
        return scopedEObjects;
    }
}

ExtendedComparisonScope comparisonScope = new ExtendedComparisonScope(leftEObject, rightEObject, null);
// Instantiates and returns a comparison object
EMFCompare build = EMFCompare.builder().build();
build.compare(comparisonScope);
```
Comparison Scope Strategies

- **Default**
  - Retrieves all content under a ECore Resource
  - Filters out some specific contents e.g., EGenericTypes
  - Follows containment structure

```java
IComparisonScope comparisonScope = EMFCompare.createDefaultScope(resourceSet1, resourceSet2);
```

- **Tailored**

```java
import org.eclipse.emf.compare.scope.DefaultComparisonScope;

public class ExtendedComparisonScope extends DefaultComparisonScope {

    /**
     * {@link InheritDoc}
     */
    @Override
    public Iterator<? extends EObject> getChildren(final EObject eObject) {
        ...
        return scopedEObjects;
    }
}

ExtendedComparisonScope comparisonScope = new ExtendedComparisonScope(leftEObject, rightEObject, null);
// Instantiates and returns a comparison object
EMFCompare build = EMFCompare.builder().build();
build.compare(comparisonScope);
```
Comparison Scope Strategies

→ Default
  • Retrieves all content under an ECore Resource
  • Filters out some specific contents e.g., EGenericTypes
  • Follows containment structure

```java
IComparisonScope comparisonScope = EMFCompare.createDefaultScope(resourceSet1, resourceSet2);
```

→ Tailored

```java
import org.eclipse.emf.compare.scope.DefaultComparisonScope;

public class ExtendedComparisonScope extends DefaultComparisonScope {

    /**
     * @inheritDoc
     */
    @Override
    public Iterator<? extends EObject> getChildren(final EObject eObject) {
        ... return scopedEObjects;
    }
}

ExtendedComparisonScope comparisonScope = new ExtendedComparisonScope(leftEObject, rightEObject, null);
// Instantiates and returns a comparison object
EMFCompare build = EMFCompare.builder().build();
build.compare(comparisonScope);
```
Comparison Scope Strategies

→ Default
  - Retrieves all content under a ECore Resource
  - Filters out some specific contents e.g., EGenericTypes
  - Follows containment structure

```java
IComparisonScope comparisonScope = EMFCompare.createDefaultScope(resourceSet1, resourceSet2);
```

→ Tailored

```java
import org.eclipse.emf.compare.scope.DefaultComparisonScope;

public class ExtendedComparisonScope extends DefaultComparisonScope {
    /* {@inheritdoc} */
    @Override
    public Iterator<? extends EObject> getChildren(final EObject eObject) {
        ... your scoping logic ...
        return scopedEObject;
    }
}

ExtendedComparisonScope comparisonScope = new ExtendedComparisonScope(leftEObject, rightEObject, null);
// Instantiates and returns a comparison object
EMFCompare build = EMFCompare.builder().build();
build.compare(comparisonScope);
```
Comparison Scope Strategies

- **Default**
  - Retrieves all content under a ECore Resource
  - Filters out some specific contents e.g., EGenericTypes
  - Follows containment structure

  ```java
  IComparisonScope comparisonScope = EMFCompare.createDefaultScope(resourceSet1, resourceSet2);
  ```

- **Tailored**

  ```java
  import org.eclipse.emf.compare.scope.DefaultComparisonScope;
  public class ExtendedComparisonScope extends DefaultComparisonScope {
      /**
       * @inheritDoc
       * @Override
       
       public Iterator<? extends EObject> getChildren(final EObject eObject) {
       ... return scopedEObjects;
     }
  
  ExtendedComparisonScope comparisonScope = new ExtendedComparisonScope(leftEObject, rightEObject, null);
  EMFCompare build = EMFCompare.builder().build();
  build.compare(comparisonScope);
  ```
Building Blocks of EMF Compare 2

Core Components
- Scope Preparation
- Matching
- Differencing
- Merging

UI Component
- Visualization
Matching

Matching Strategies

Default

- Mapping of scoped input elements
  - EObjects identifiers either XMI:ID or attribute ID
  - Some distance function (proximity)
Matching Strategies

Default
- Mapping of scoped input elements
  - EObjects identifiers either XMI:ID or attribute ID
  - Some distance function (proximity)

```java
IEObjectMatcher matcher = DefaultMatchEngine.createDefaultEObjectMatcher(UseIdentifiers.WHEN_AVAILABLE);
```
Matching Strategies

➡️ Default
• Mapping of scoped input elements
  • EObjects identifiers either XMI:ID or attribute ID
  • Some distance function (proximity)

```java
IEObjectMatcher matcher = DefaultMatchEngine.createDefaultEObjectMatcher(UseIdentifiers.WHEN_AVAILABLE);
```

➡️ Tailored
• Specific model-elements can be compared against each other
• Meta-model consideration
Matching Strategies

- **Default**
  - Mapping of scoped input elements
  - EObjects identifiers either XMI:ID or attribute ID
  - Some distance function (proximity)

```java
IEObjectMatcher matcher = DefaultMatchEngine.createDefaultEObjectMatcher(UseIdentifiers.WHEN_AVAILABLE);
```

- **Tailored**

```java
import org.eclipse.emf.compare.match.eobject.IEObjectMatcher;

public class MyEObjectMatcher implements IEObjectMatcher {
    /**
     * @inheritDoc
     */
    @Override
    public void createMatches(final Comparison comparison, final Iterator<? extends EObject> leftEObjects,
                                final Iterator<? extends EObject> rightEObjects, final Iterator<? extends EObject> originEObjects,
                                final Monitor monitor) {
        variability.addAll(comparison.getMatches(), this.matches);
    }

    // default match engine
    final IMatchEngine customMatchEngine = new DefaultMatchEngine(new MyEObjectMatcher(), comparisonFactory);
    matchEngineFactoryRegistry.add(new MatchEngineFactoryImpl() { ... return customMatchEngine; });
    //Register match engine
    EMFCompare.builder().setMatchEngineFactoryRegistry(matchEngineFactoryRegistry);
```
Matching Strategies

- **Default**
  - Mapping of scoped input elements
  - EObjects identifiers either XMI:ID or attribute ID
  - Some distance function (proximity)

```java
IEObjectMatcher matcher = DefaultMatchEngine.createDefaultEObjectMatcher(UseIdentifiers.WHEN_AVAILABLE);
```

- **Tailored**

```java
import org.eclipse.emf.compare.match.eobject.IEObjectMatcher;

public class MyEObjectMatcher implements IEObjectMatcher {

    /* [@inheritdoc]
     */
    @Override
    public void createMatches(final Comparison comparison, final Iterator<?> extendsEObject, final Iterator<?> extendsEObject, final Iterator<?> extendsEObject, final Monitor monitor) {
        ...
        this.matches;
    }
}
```

// default match engine
final IMatchEngine customMatchEngine = new DefaultMatchEngine(new MyEObjectMatcher(), comparisonFactory);
matchEngineFactoryRegistry.add(new MatchEngineFactoryImpl() {
    ... return customMatchEngine;
});

//Register match engine
EMFCompare.builder().setMatchEngineFactoryRegistry(matchEngineFactoryRegistry);

Matching Strategies

➤ Default
  • Mapping of scoped input elements
  • EObjects identifiers either XMI:ID or attribute ID
  • Some distance function (proximity)

```java
EOObjectMatcher matcher = DefaultMatchEngine.createDefaultEOObjectMatcher(UseIdentifiers.WHEN_AVAILABLE);
```

➤ Tailored

```java
import org.eclipse.emf.compare.match.eobject.EOObjectMatcher;

public class MyEOObjectMatcher implements EOObjectMatcher {
  /**
   * @inheritDoc
   */
  @Override
  public void createMatches(final Comparison comparison, final Iterator<? extends EObject> leftEObjects,
                             final Iterator<? extends EObject> rightEObjects, final Iterator<? extends EObject> originEObjects,
                             final Monitor monitor) {
    matchables.addAll(comparison.getMatches(), this.matches);
  }
  // default match engine
  final IMatchEngine customMatchEngine = new DefaultMatchEngine(new MyEOObjectMatcher(), comparisonFactory);
  matchEngineFactoryRegistry.add(new MatchEngineFactoryImpl()) { ... return customMatchEngine; };
  //Register match engine
  EMFCompare.builder().setMatchEngineFactoryRegistry(matchEngineFactoryRegistry);
```
Matching Strategies

- Default
  - Mapping of scoped input elements
  - EObjects identifiers either XMI:ID or attribute ID
  - Some distance function (proximity)

```java
IEObjectMatcher matcher = DefaultMatchEngine.createDefaultEObjectMatcher(UseIdentifiers.WHEN_AVAILABLE);
```

- Tailored

```java
import org.eclipse.emf.compare.match.eobject.IEObjectMatcher;
public class MyEObjectMatcher implements IEOBJjectMatcher {
    /*
     * @override
     */
    public void createMatches(final Comparison comparison, final Iterator<? extends EObject> leftEObject, final Iterator<? extends EObject> rightEObject, final Iterator<? extends EObject> originEObject, final Monitor monitor)
    {
        yours mapping logic
    }
    final IMatchEngine customMatchEngine = new DefaultMatchEngine(new MyEObjectMatcher(), comparisonFactory);
    EMFCompare.builder().setMatchEngineFactoryRegistry(matchEngineFactoryRegistry);
} // default match engine
```

Swathi Raghavan (RBEI/EMT1) | 30/10/2014 | © Robert Bosch Engineering and Business Solutions Limited 2014. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
Matching

Matching Strategies

⇒ Default
  • Mapping of scoped input elements
  • EObjects identifiers either XMI:ID or attribute ID
  • Some distance function (proximity)

```java
IEObjectMatcher matcher = DefaultMatchEngine.createDefaultEObjectMatcher(UseIdentifiers.WHEN_AVAILABLE);
```

⇒ Tailored

```java
import org.eclipse.emf.compare.match.eobject.IEObjectMatcher;

public class MyEObjectMatcher implements IEObjectMatcher {

/*
 * @inheritDoc
 */
@override
public void createMatches(final Comparison comparison, final Iterator<? extends EObject> leftEOBjects,
    final Iterator<? extends EObject> rightEOBjects, final Iterator<? extends EObject> originEOBjects,
    final Monitor monitor) {

    matchables.addAll(comparison.getMatches(), this.matches);
}

// default match engine
final IMatchEngine customMatchEngine = new DefaultMatchEngine(new MyEObjectMatcher(), comparisonFactory);
matchEngineFactoryRegistry.add(new MatchEngineFactoryImpl() { ... return customMatchEngine; });
//Register match engine
EMFCompare.builder().setMatchEngineFactoryRegistry(matchEngineFactoryRegistry);
```
Building Blocks of EMF Compare 2

Core Components

Scope Preparation

Matching

Differencing

Merging

Post Processing

Post Processing

UI Component

Visualization
Building Blocks of EMF Compare 2

Core Components

Scope Preparation → Matching → Differencing → Merging

UI Component

Visualization
Differencing

Diffing Strategies

- Default
  - Detects modified attributes and references of matched elements
  - Very generic and provides sufficient functionality for any model
Differencing

Diffing Strategies

⇒ Default
  • Detects modified attributes and references of matched elements
  • Very generic and provides sufficient functionality for any model

⇒ Tailored
  • Diff based on specific attributes
  • Intelligent Reference Diff
    • Reference following contained in element values
Diffing Strategies

➢ Default
  • Detects modified attributes and references of matched elements
  • Very generic and provides sufficient functionality for any model

➢ Tailored

```java
import org.eclipse.emf.compare.diff.DefaultDiffEngine;
public class MyDiffEngine extends DefaultDiffEngine {

/**
 * {@inheritDoc}
 */
@Override
protected void checkForDifferences(final Match match, final Monitor monitor) {
  ...
  computeDifferences(match, specialEAttribute, false);
}

// diff engine registration
IDiffEngine customDiffEngine = new MyDiffEngine(new DiffBuilder());
EMFCompare.builder().setDiffEngine(customDiffEngine);
```
Differencing

Diffing Strategies

➡ Default
  • Detects modified attributes and references of matched elements
  • Very generic and provides sufficient functionality for any model

➡ Tailored

```java
import org.eclipse.emf.compare.diff.DefaultDiffEngine;

public class MyDiffEngine extends DefaultDiffEngine {

/**
 * {@inheritDoc}
 */
@override
protected void checkForDifferences(final Match match, final Monitor monitor) {
  ...
  computeDifferences(match, specialEAttribute, false);
}

// diff engine registration
IDiffEngine customDiffEngine = new MyDiffEngine(new DiffBuilder());
EMFCompare.builder().setDiffEngine(customDiffEngine);
```
Diffing Strategies

- Default
  -Detects modified attributes and references of matched elements
  -Very generic and provides sufficient functionality for any model

- Tailored

```java
import org.eclipse.emf.compare.diff.DefaultDiffEngine;

public class MyDiffEngine extends DefaultDiffEngine {

    /**
     * @inheritDoc
     */
    @Override
    protected void checkForDifferences(final Match match, final Monitor monitor) {
        computeDifferences(match, specialEAttribute, false);
    }

    // diff engine registration
    IDiffEngine customDiffEngine = new MyDiffEngine(new DiffBuilder());
    EMFCompare.builder().setDiffEngine(customDiffEngine);
```
Differencing

Diffing Strategies

- Default
  - Detects modified attributes and references of matched elements
  - Very generic and provides sufficient functionality for any model

- Tailored

```java
import org.eclipse.emf.compare.diff.DefaultDiffEngine;

public class MyDiffEngine extends DefaultDiffEngine {

/**
 * @inheritDoc
 */

@override
protected void checkForDifferences(final Match match, final Monitor monitor) {
    // your diff logic
    computeDifferences(match, specialEAttribute, false);
}

// diff engine registration
IDiffEngine customDiffEngine = new MyDiffEngine(new DiffBuilder());
EMFCompare.builder().setDiffEngine(customDiffEngine);
```
Differencing

Diffing Strategies

- Default
  - Detects modified attributes and references of matched elements
  - Very generic and provides sufficient functionality for any model

- Tailored

```java
import org.eclipse.emf.compare.diff.DefaultDiffEngine;

public class MyDiffEngine extends DefaultDiffEngine {

    /**
     * {@inheritDoc}
     */
    @Override
    protected void checkForDifferences(final Match match, final Monitor monitor) {
        ... 
        computeDifferences(match, specialEAttribute, false);
    }

    // diff engine registration
    IDiffEngine customDiffEngine = new MyDiffEngine(new DiffBuilder());
    EMFCompare.builder().setDiffEngine(customDiffEngine);
```
Recalling “Post-Processing”

- Default
  - Doesn’t perform any post-processing
  - Configurable through code/extension point
Recalling “Post-Processing”

→ Default
  • Doesn’t perform any post-processing
  • Configurable through code/extension point

→ Yes, we needed it
  • Our use case is to refine the Diff model based on some specific rules.
  • Few example for rules,
    • E.g., False/false/FALSE/0 and True/true/TRUE/1
    • ON/1 and OFF/0
Recalling “Post-Processing”

➔ Default
  • Doesn’t perform any post-processing
  • Configurable through code/extension point

➔ Yes, we needed it

```java
import org.eclipse.emf.compare.postprocessor.IPostProcessor;

public class CustomPostProcessor implements IPostProcessor {
    /**
     * @inheritdoc
     */
    @Override
    public void postDiff(final Comparison comparison, final Monitor monitor) {
        EList<Diff> differences = comparison.getDifferences();
        ...
    }
}

//Registration
IPostProcessor postProcessor = new CustomPostProcessor();
IPostProcessor.Descriptor descriptor = new BasicPostProcessorDescriptorImpl(postProcessor, null, null);

PostProcessor.Registry registry = new PostProcessorDescriptorRegistryImpl();
registry.put(comparisonScope, descriptor);
EMFCompare.builder().setPostProcessorRegistry(registry);
```
Recalling “Post-Processing”

→ Default
  • Doesn’t perform any post-processing
  • Configurable through code/extension point

→ Yes, we needed it

```java
import org.eclipse.emf.compare.postprocessor.IPostProcessor;

public class CustomPostProcessor implements IPostProcessor {

    /**
     * {@inheritDoc}
     */
    @Override
    public void postDiff(final Comparison comparison, final Monitor monitor) {
        EList<Diff> differences = comparison.getDifferences();
        ...
    }

    //Registration
    IPostProcessor postProcessor = new CustomPostProcessor();
    IPostProcessor.Descriptor descriptor = new BasicPostProcessorDescriptorImpl(postProcessor, null, null);
    PostProcessor.Registry registry = new PostProcessorDescriptorRegistryImpl();
    registry.put(comparisonScope, descriptor);
    EMFCompare.builder().setPostProcessorRegistry(registry);
}
```
Recalling “Post-Processing”

- Default
  - Doesn’t perform any post-processing
  - Configurable through code/extension point

- Yes, we needed it

```java
import org.eclipse.emf.compare.postprocessor.IPostProcessor;

public class CustomPostProcessor implements IPostProcessor {
    /**
     * {@inheritDoc}
     */
    @Override
    public void postDiff(final Comparison comparison, final Monitor monitor) {
        EList<Diff> differences = comparison.getDifferences();
        ...}
}

//Registration
IPostProcessor postProcessor = new CustomPostProcessor();
IPostProcessor.Descriptor descriptor = new BasicPostProcessorDescriptorImpl(postProcessor, null, null);
PostProcessor.Registry registry = new PostProcessorDescriptorRegistryImpl();
registry.put(comparisonScope, descriptor);
EMFCompare.builder().setPostProcessorRegistry(registry);
```
Recalling “Post-Processing”

- Default
  - Doesn’t perform any post-processing
  - Configurable through code/extension point

- Yes, we needed it

```java
import org.eclipse.emf.compare.postprocessor.IPostProcessor;

public class CustomPostProcessor implements IPostProcessor {

    /**
     * @inheridDoc
     */
    @Override
    public void postDiff(final Comparison comparison, final Monitor monitor) {
        EList<Diff> differences = comparison.getDifferences();
        ...
    }
}

//Registration
IPostProcessor postProcessor = new CustomPostProcessor();
IPostProcessor.Descriptor descriptor = new BasicPostProcessorDescriptorImpl(postProcessor, null, null);
PostProcessor.Registry registry = new PostProcessorDescriptorRegistryImpl();
registry.put(comparisonScope, descriptor);
EMFCompare.builder().setPostProcessorRegistry(registry);
```

your post-processing logic
Recalling “Post-Processing”

- Default
  - Doesn’t perform any post-processing
  - Configurable through code/extension point

- Yes, we needed it

```java
import org.eclipse.emf.compare.postprocessor.IPostProcessor;

public class CustomPostProcessor implements IPostProcessor {
    /**
     * {@inheritDoc}
     */
    @Override
    public void postDiff(final Comparison comparison, final Monitor monitor) {
        EList<Diff> differences = comparison.getDifferences();
        ...
    }
}

// Registration
IPostProcessor postProcessor = new CustomPostProcessor();
getPostProcessor().setPostProcessorDescriptor(new BasicPostProcessorDescriptorImpl(postProcessor, null, null));
PostProcessor.Registry registry = new PostProcessorDescriptorRegistryImpl();
registry.put(comparisonScope, descriptor);
EMFCompare.builder().setPostProcessorRegistry(registry);
```
Building Blocks of EMF Compare 2

Core Components

- Scope Preparation
- Matching
- Differencing
- Merging

UI Component

- Visualization
Merging

Default
- Merger modifies the differences based on the intended user action
- Performs prompt modification to the model
- Different types of merger implementations
Merging

- Default
  - Merger modifies the differences based on the intended user action
  - Performs prompt modification to the model
  - Different types of merger implementations
Merging

Tailored

- Single model instance with possible multi-processing
- No interim changes on the model (model consistency)
- Need of more merger options (copyAsNew..)
- Multi-Selectable merge
Tailored
- Single model instance with possible multi-processing
- No interim changes on the model (model consistency)
- Need of more merger options (copyAsNew..)
- Multi-Selectable merge
Building Blocks of EMF Compare 2

Core Components
- Scope Preparation
- Matching
- Differencing
- Merging

UI Component
- Visualization
Visualization

Default

- Out of the box support for UI input model (Compare Editor)
- Minimal integration effort
- Difficult to follow the differences if ordering is important
Visualization

Default

Source: http://eclipsemde.blogspot.de
Visualization by Customization

Tailored

- Adjacent visualization
- Easily understandable for tool users
- Irrelevant element ordering
- Loaded model Vs External resources
- Embedded Details view
Visualization by Customization

Tailored

<table>
<thead>
<tr>
<th>Name</th>
<th>Library_1</th>
<th>Library_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book</td>
<td>Book_2</td>
<td>Book_2</td>
</tr>
<tr>
<td></td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>Category</td>
<td>TRAVEL</td>
<td>SPORTS</td>
</tr>
<tr>
<td>Title</td>
<td>On the road</td>
<td>Heaven is a playground</td>
</tr>
<tr>
<td>Page</td>
<td>Page_0</td>
<td>Page_1</td>
</tr>
<tr>
<td>Number</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PageLinks</td>
<td>/RB/UBK/Project/EcucModuleConfigurationValues/Library_1/Catalogue_1</td>
<td>/RB/UBK/Project/EcucModuleConfigurationValues/Library_2/Catalogue_2</td>
</tr>
<tr>
<td>Title</td>
<td>My dream</td>
<td>Healthy foods</td>
</tr>
<tr>
<td></td>
<td>The Story of Art</td>
<td>Food as a medicine</td>
</tr>
<tr>
<td>Page</td>
<td>Page_2</td>
<td>Page_1</td>
</tr>
<tr>
<td>Page</td>
<td>Page_3</td>
<td>Page_4</td>
</tr>
<tr>
<td>Catalogue</td>
<td>Catalogue_1</td>
<td>Catalogue_2</td>
</tr>
<tr>
<td>Magazine</td>
<td>magazine_1</td>
<td>magazine_1</td>
</tr>
</tbody>
</table>

- Merger options
- Difference in the value
- Elements present in one side only
- Special differencing on boolean element
- Intelligent differencing on reference element
- Complete value of the selected element
Our Success Story!!

Large Scale EMF Model Optimized

>20,000 Model Elements

Automotive Domain

Domain Specific Model Comparison

Adaptation of EMFCompare framework

Performance

User satisfaction

Source: http://www.clker.com - license free
Takeaways !!

- Default implementation for simple models
- In reality specialization required
- Separation of Concerns
  - Benefit from post processing
- Customization throughout EMF Compare workflow
- Plugin third party components
Thank you for your attention