Diff & Merge Strategy

Diffing and Merging Models is the main purpose of *LemonTree*. Since models can be highly connected and concurrent editing can lead to conflicts, *LemonTree* employs some strategies for handling such cases.

Diffing

Diffing is a two-stage process.

In the first stage, *LemonTree* examines the models and quickly determines which elements have changed. The result of this step is a list of elements which *may* have undergone significant changes, but it's not a conclusive one. Put differently, the result of this operation is a strict superset of the changes in the models.

In the second stage, the elements which may have changed will be examined further. *LemonTree* determines which structural changes happened (Moves, Deletes, Additions), and also examines the internals of elements, basically determining *what* has changed for each element.
This is also the stage where conflicts will be detected.

Conflicts

Conflicts are detected when changes in both branches are not reconcilable.

Conflict detection is currently done on single elements, meaning their surrounding is not examined - only the elements and their changes themselves are examined.

In the matrix below you'll find the instances where *LemonTree* currently detects a conflict, and which version *LemonTree* will prefer.

Change Type	Unmodified in A	New in A	Modified in A	Moved in A	Removed in A
Unmodified in B	No Conflict	Not Possible	No Conflict	No Conflict	No Conflict
New in B	Not Possible	No Conflict (1)	Not Possible	Not Possible	Not Possible
Modified in B	No Conflict	Not Possible	Conflict:A (2)	No Conflict	Conflict:B
Moved in B	No Conflict	Not Possible	No Conflict	Conflict:A (3)	Conflict:B
Removed in B	No Conflict	Not Possible	Conflict:A	Conflict:A	No Conflict

1) In case the new elements are identical in both models, *LemonTree* doesn't detect a conflict (since Release 2.1.4). "New-New" scenarios with same GUIDs can happen if parts of the model are imported via .xmi.
2) Only a conflict if there is at least one property which is modified in both models.
3) In case the element is moved to the same location in A and B, *LemonTree* doesn't detect a conflict.

Change Type	Description
Modified	The element properties or tagged values have been changed
Moved	The element has a new owner
New	The element is new in the model (the element in the other branch is marked as **Doesn't exist**)
Removed	The element has been deleted from the model
Unmodified	There have been **no direct modifications** at the properties of the element, for example: 1. There are just graphical changes (which can be seen at the *Diagram Representations*) 2. The element is needed for displaying (structuring in *Impacted Elements / Impacted Diagrams List*)
Child Modified	At least one child element of the element has been changed and the element itself was not directly changed

While the diff is symmetrical, the merge is not. When merging, the conflict resolution step favors the 'A' model.

Merging

Although *LemonTree* can be used as a diff tool, there is the option to merge different versions of models. For this, a diff is performed followed by a conflict resolution step. The result of these steps is the *Merge Preview* which is a model where changes from both branches have been applied.

Pre-Merge

In case of a conflict in the diffing step, *LemonTree* still generates a *Merge Preview* where it employs some merge strategies to make sure that the resulting model is sound.

- At first, the changes are examined and the non conflicting changes are applied to the *Merge Preview*.
- In case of a conflict, the change which did not remove an element is applied.
  - If there is no such change, the change in 'A' is applied.
- If the element has been modified in both branches, but the modified properties are not the same, *LemonTree* combines the changes done in both branches.

Conflict Resolution

After this pre-merge step, the applied changes are analyzed further to find out if...

- an element has all its dependencies met (for example, the owner of a new element must also be part of the *Merge Preview*).
- there are higher-level conflicts (for example a new connector to an unmodified element in 'B' while the element has been removed in 'A').

Problems found here will be corrected, and the model will be analyzed again until no such problems are found.

After this phase, the *Merge Preview* is ready to be written.

User Modifications

When using *LemonTree* interactively, you have the option to override the changes *LemonTree* did to the model by using the GUI. Because *LemonTree* tries to ensure model consistency after every (interactive) step, some actions may yield surprising results. As an example, if you choose to take a (deleted in the other branch) connector by using the UI functionality to do so, this will also take the elements connected by the connector to ensure that the connector is valid. If such an element has been marked as deleted by the merge algorithm, it will change its state and becomes part of the *Merge Preview*. Depending on the structure of your project, a tiny change such as this can have a huge impact to the model because the owning elements and other dependencies also have to be taken into account.

We will add some means to visualize the impact of such changes. In the meantime, be careful when you are doing manual changes on potentially very well connected models.

Existential Dependencies

Most elements in models can't exist on their own - they have dependencies to other elements which are existential to this element. We call these dependencies *Existential Dependencies*.
Examples of existential dependencies are

- Owner dependencies. A child can't exist without its parent.
- Connector ends. A connector end (each connector has at least two of those) can't exist without its connector. At the same time, the connector can't exist without its ends.
- Connected objects. A connector end ties the part of a connector to an element. It can't exist without this element, it has to be connected to something.
- Representations on diagrams. Elements are not directly part of diagrams. Instead, they are represented by so called diagram objects. These diagram objects can't exist without the elements they represent.

LemonTree adds these elements if they are required to preserve the integrity of the model.

Integral Features

Some elements in Models are pretty complete on their own, but still have some dependencies which are defining characteristics of that element. We call these dependencies *Integral Features*.
Examples of integral features are

- Representations on diagrams

    - Representations of elements on diagrams are not necessary for the element to exist, but still the representations should be part of the result if you select the represented element to be part of the merged model.
- Images
    - Elements may contain references to images. Although the element can exist without the referenced image, the image is still an important part of the element, and should be preserved as well.

LemonTree respects such integral features. If you select an element to be part of the merge, for example because it was deleted in one branch and you don't want it to be deleted in the merged model, *LemonTree* also selects the integral features of this element, so that the element is complete. However, if an integral feature has been deleted in one branch, *LemonTree* respects this and will not resurrect this element, in contrast to existential dependencies where this feature would have to be resurrected in order to get a valid model.