# New concepts for EASy-Producer (Languages)

This living document contains a summary of discussions about new EASy-Producer concepts, partly just as a memory protocol and not fully elaborated. Second part contains already realized new concepts.

# In discussion

## Extension of enums

1. Klaus...
2. Type refinement

Example:

project A {

enum Color {red, green};

}

project B {

enum MyColor **refines Color** {yellow};

}

Potential problem:

* Constraints: shall work, however the refined enum can be used wherever the original enum was used (akin to compounds) so that equality and unequality checks can then be applied to a broader range, which might lead to a different semantic as intended.
* Workaround: Make explicit which enums can be extended, which not (“final”)
* Ordered enums: no overriding of existing ordinal numbers

## Evals

Current state (and decision): The evaluation sequence determined by an enum is applied only once for the first evaluation. Actually, evals shall only improve reasoning speed rather than implicitly introduce an evaluation order leading to different results.

To check: The “initialization” of “anchor” types in QualiMaster did not work independently of the sequence (reasoner).

Type consistency

Current state: Although not proven, this shall be ok for now due to the assignment compatibility along compound refinements, the base-type compatibility of typedefs and even in the case of typedefs on refined compounds.

## Mandatory

Problem: isConfigured(.) applies immediately and breaks staged configuration.

A mandatory modifier for variable forcing a check at the end of reasoning would potentially fall too short. A specification of the “variable lifecycle phase” (or however we call that) would be more generic. Further discussion is needed.

## Specifying details of a reference (new)

Current solution – might be the most generic:

compound Flow {

refTo(Node) next;

// additional information

}

compound Node {

setOf(Flow) next;

}

Problem: If node hierarchy becomes complex and flows shall be limited to specific parts of a topology, we need a flow and a node hierarchy.

Discussion:

* Allow co-variance in compounds – redefine slot with more specific type.

TBD: Detail example – does this help?

* Some form of “association class”

TBD: Detail example – does this help? Is it only helpful for undirected references.

* Do we need undirected / bi-directed references for general topology modeling?

## VIL: Creating a configuration (new)

Testing (as well as profiling of pipelines in QM) requires the creation of a permanent or temporary configuration. So VIL shall be able to create new instances of project, imports, variables (not necessarily types), the related configuration and to pass the new instances on for further instantiation as well as to store them if needed (what about the existing model).

Background: Currently, in QM we need the creation of projects and variables as well as value assignment. Thereby, it would be beneficial to copy values of existing variables on demand (also for differentiating instances of configured elements at runtime).

# Done / Realized

## VIL/VTL: explicit for and while loops

Clearify semantics of map vs loops and introduce explicit for and while loops.

## Constants (new)

Defining a constant value that can be assigned instead of repeating over and over the same value would be helpful. BTW, TVL has constants.

Agreed solution: constant modifier for variables.