Variability and Component Composition

Tijs van der Storm

storm@cwi.nl

CWI

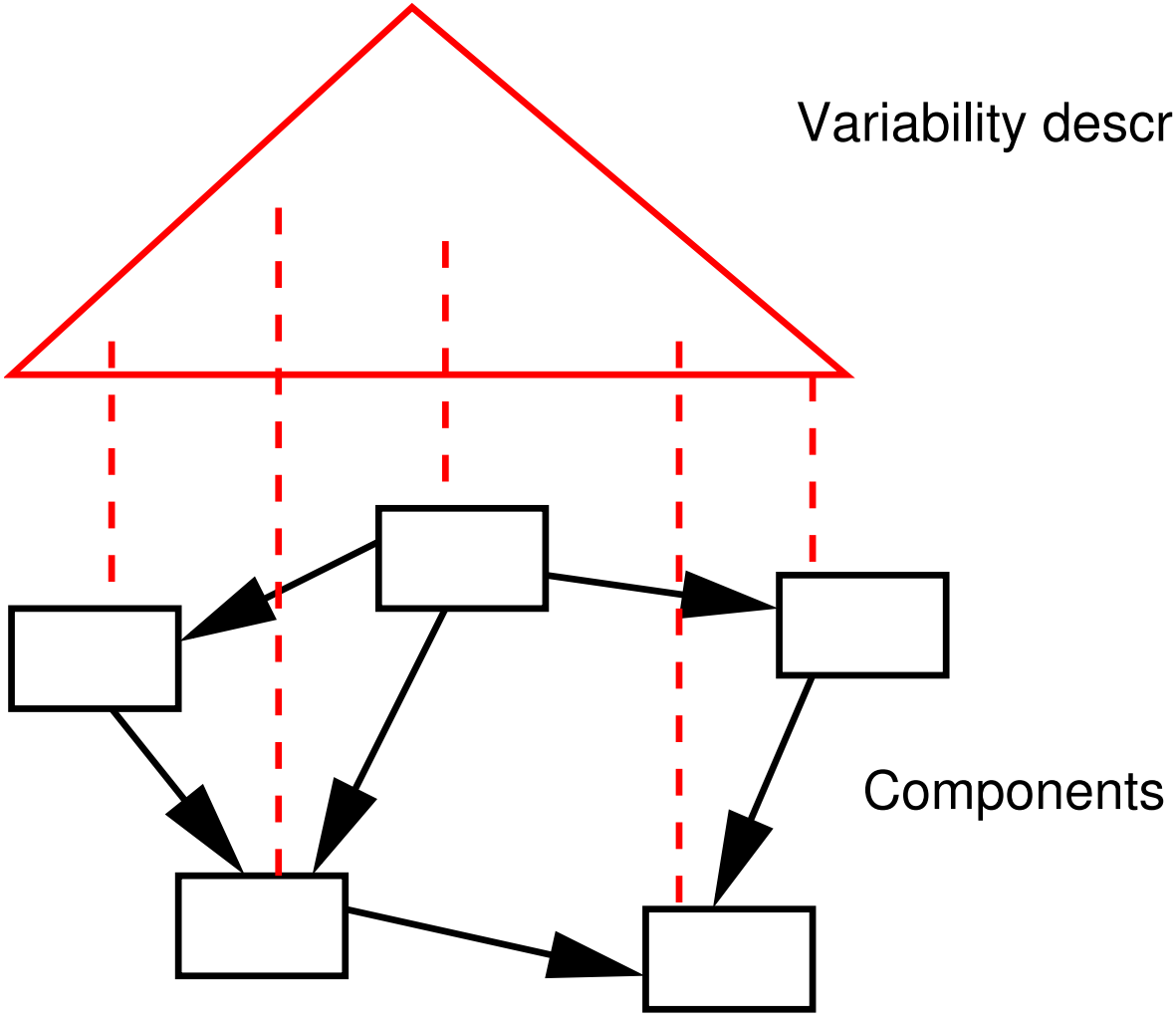
Overview

- Background on variability
- Component description
- Correct composition

Background

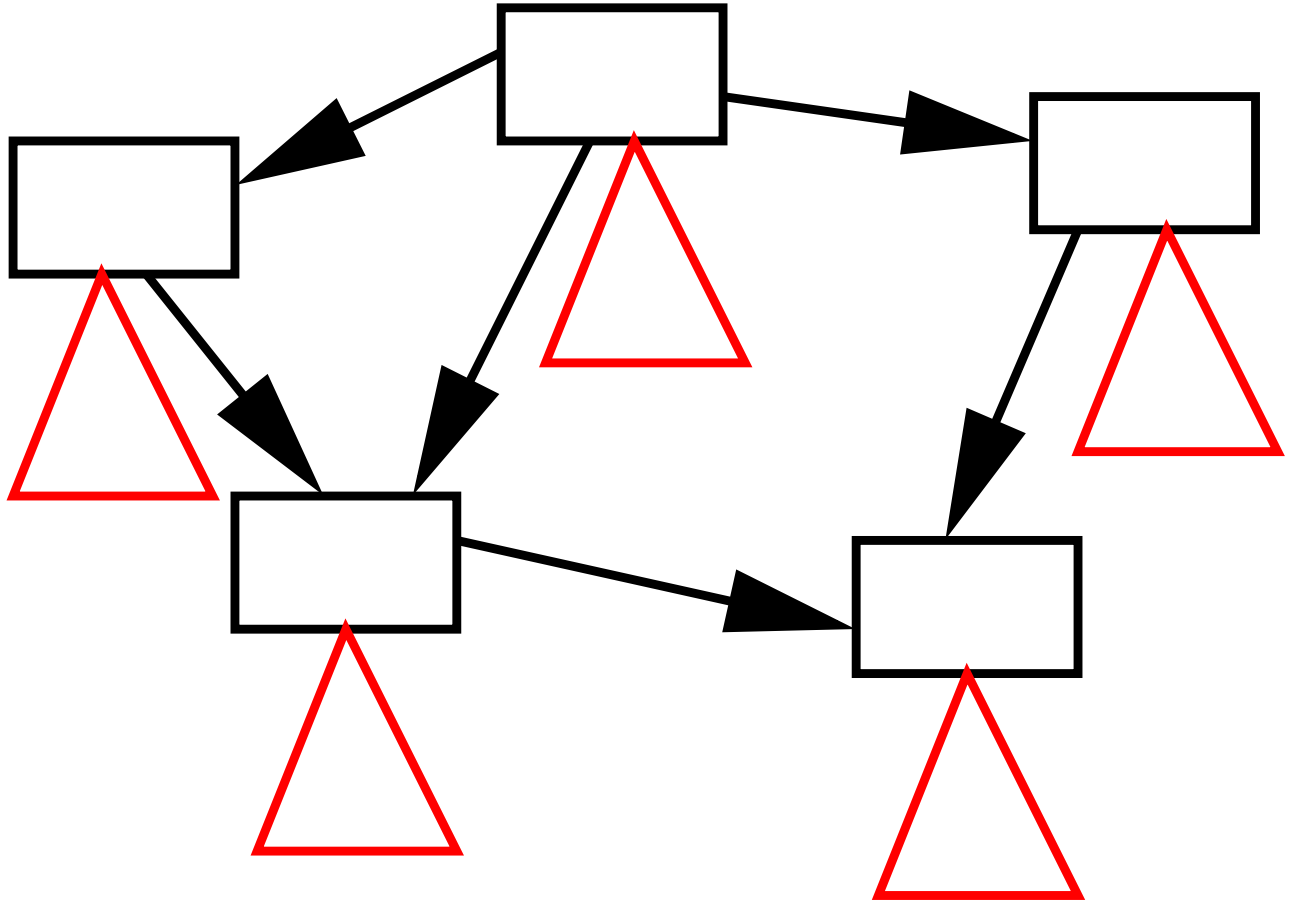
System variability

Variability described at level of system.



Component variability

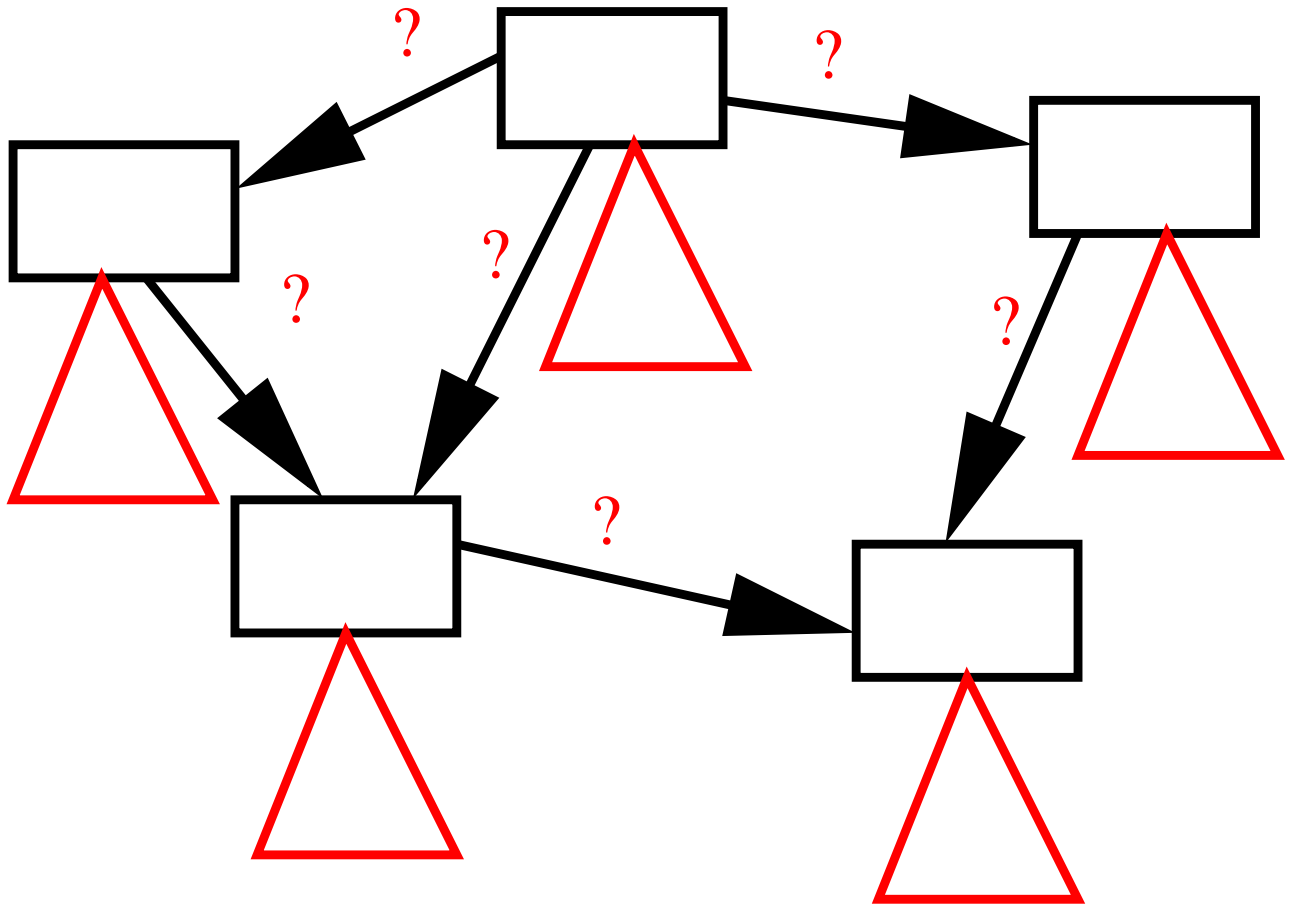
Variability described per component.



Motivation

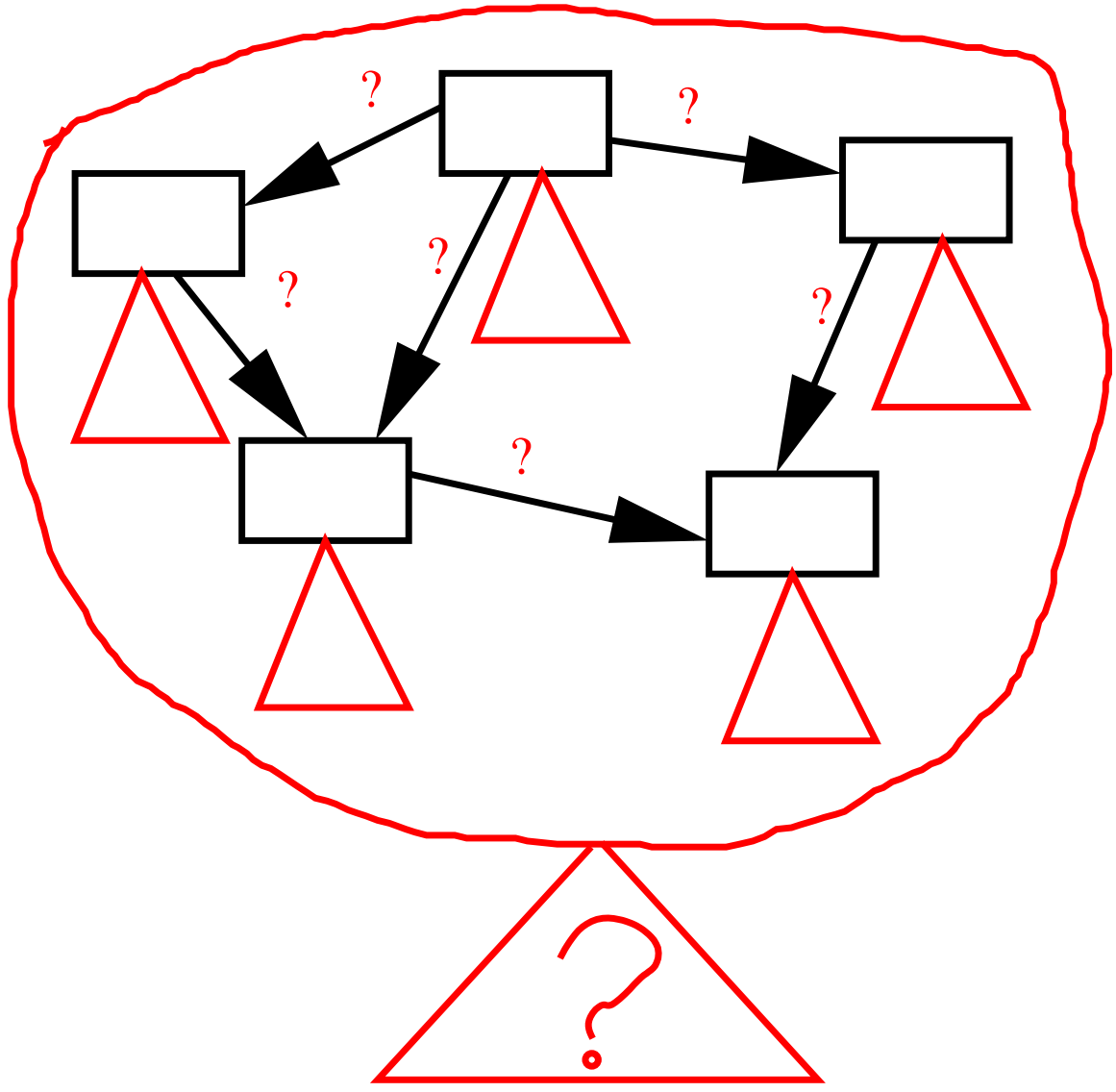
- Components as unit of variation not enough.
  (e.g., fine-grained variation using `#ifdef`s)
- Focus on malleability instead of factorization.
  ("Maximizing reuse minimizes use." [Szyperski])
- Ideology of commercial-off-the-shelf (COTS)
  (e.g. selling ActiveX components)
- Locality of variability description
  (think about a component in isolation)

Component variability



What about composition?

Component variability



Questions

- How is component variability described?
- How is correct composition guaranteed?

Component description

A component description language

A formal language facilitates automatic reasoning, analysis, tranformation.

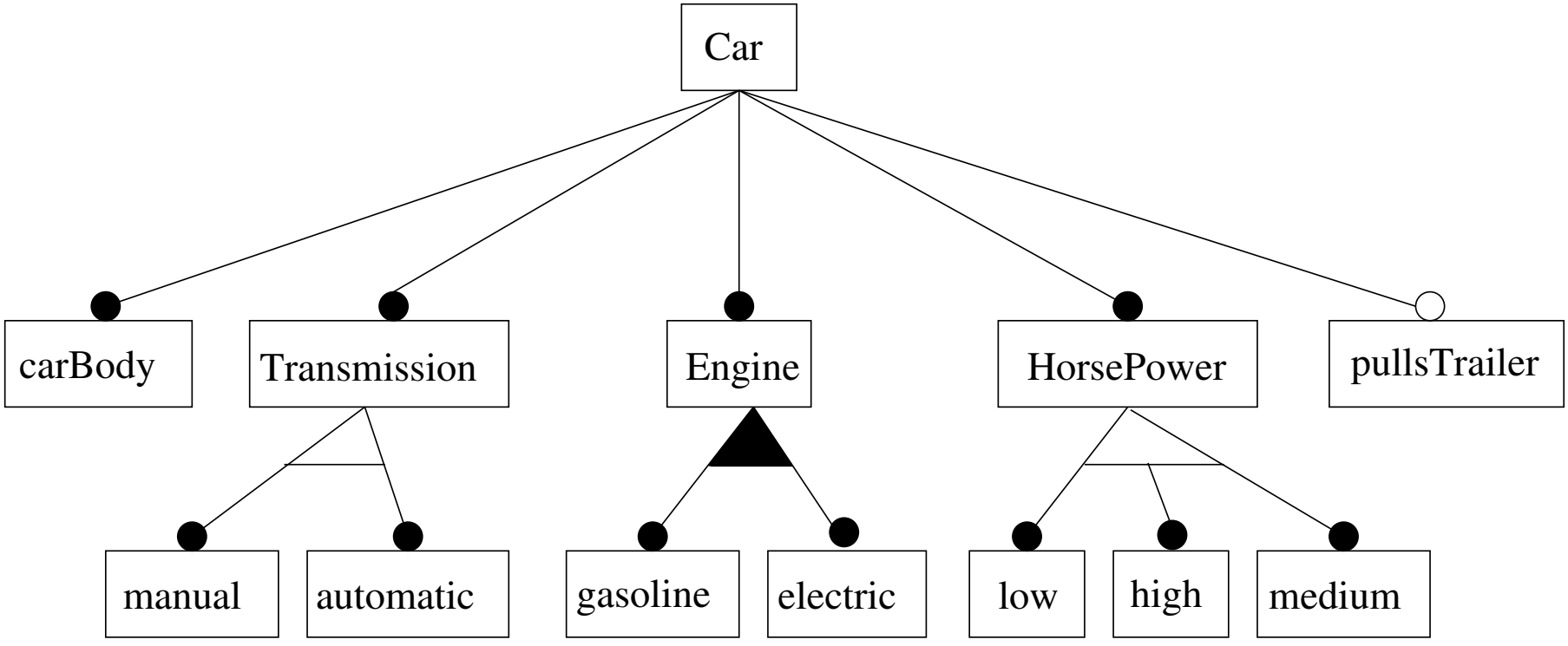
Description of variability:

- Configuration interface (Feature Diagrams)

Description of composition:

- Requires interface (dependencies)

Description of variability



Constraint: `pullsTrailer requires high`

Feature description language

Feature Description Language (FDL):

```
Car: all(carBody, Transmission, Engine,
         HorsePower, pullsTrailer?)
Transmission: one-of(manual, automatic)
Engine: more-of(electric, gasoline)
HorsePower: one-of(high, medium, low)

pullsTrailer requires high
```

Description of composition

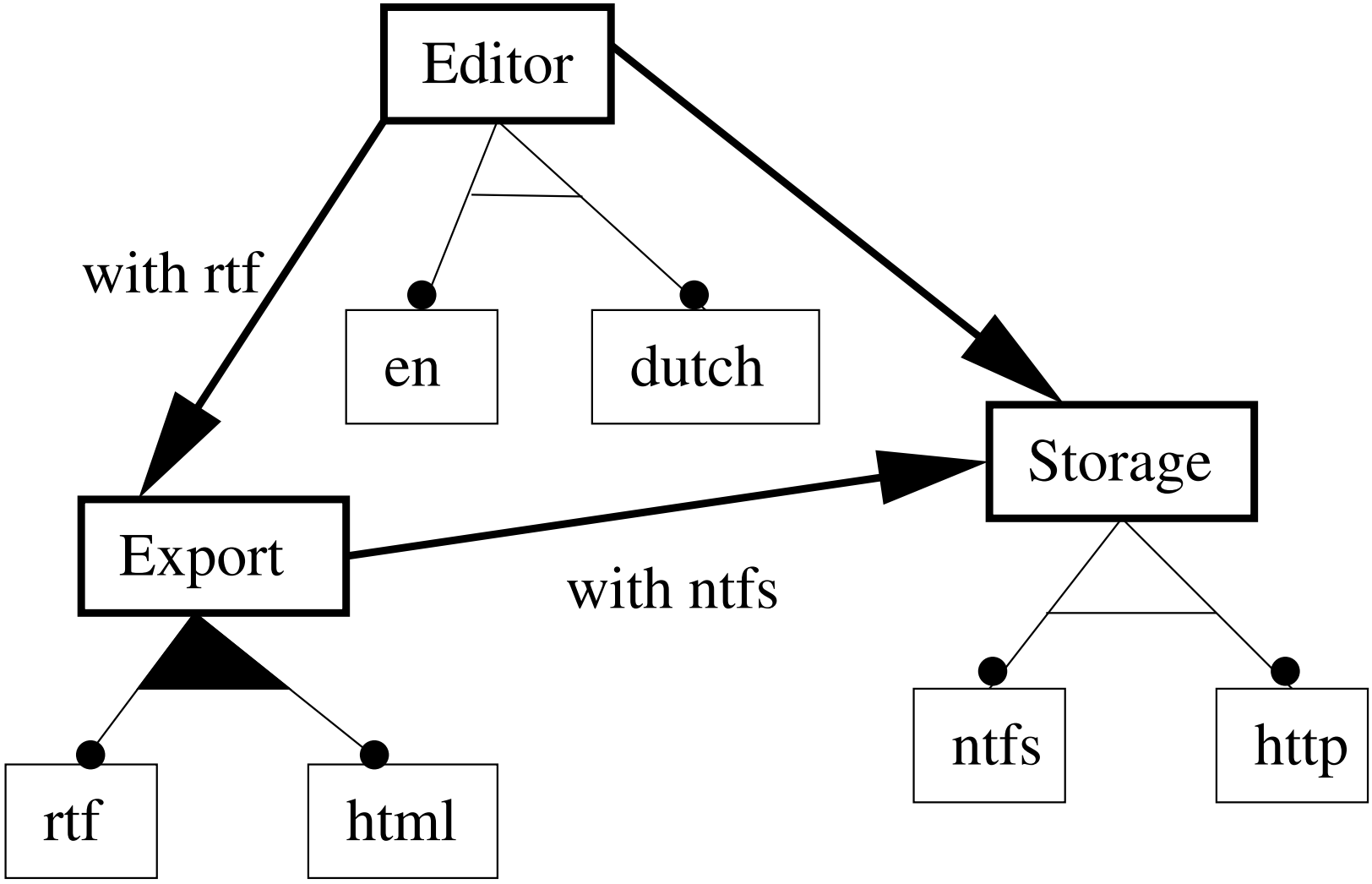
Specification of dependencies.

Examples of dependencies:

- `require(Foo)`
- ```
  require(Car,
     [automatic, gasoline, pullsTrailer]);
  ```

In the last example features are passed to the dependency.
  ```

Example component base



Component description example

```
package Editor {
   Editor: one-of(en,dutch)
   require(Export, [rtf]);
   require(Storage);
}

package Export {
   Export: more-of(rtf, html)
   require(Storage, [ntfs])
}

package Storage {
   Storage: one-of(ntfs, http)
}
```

Correct composition

Consistency requirements

- Description of variability should be internally consistent.

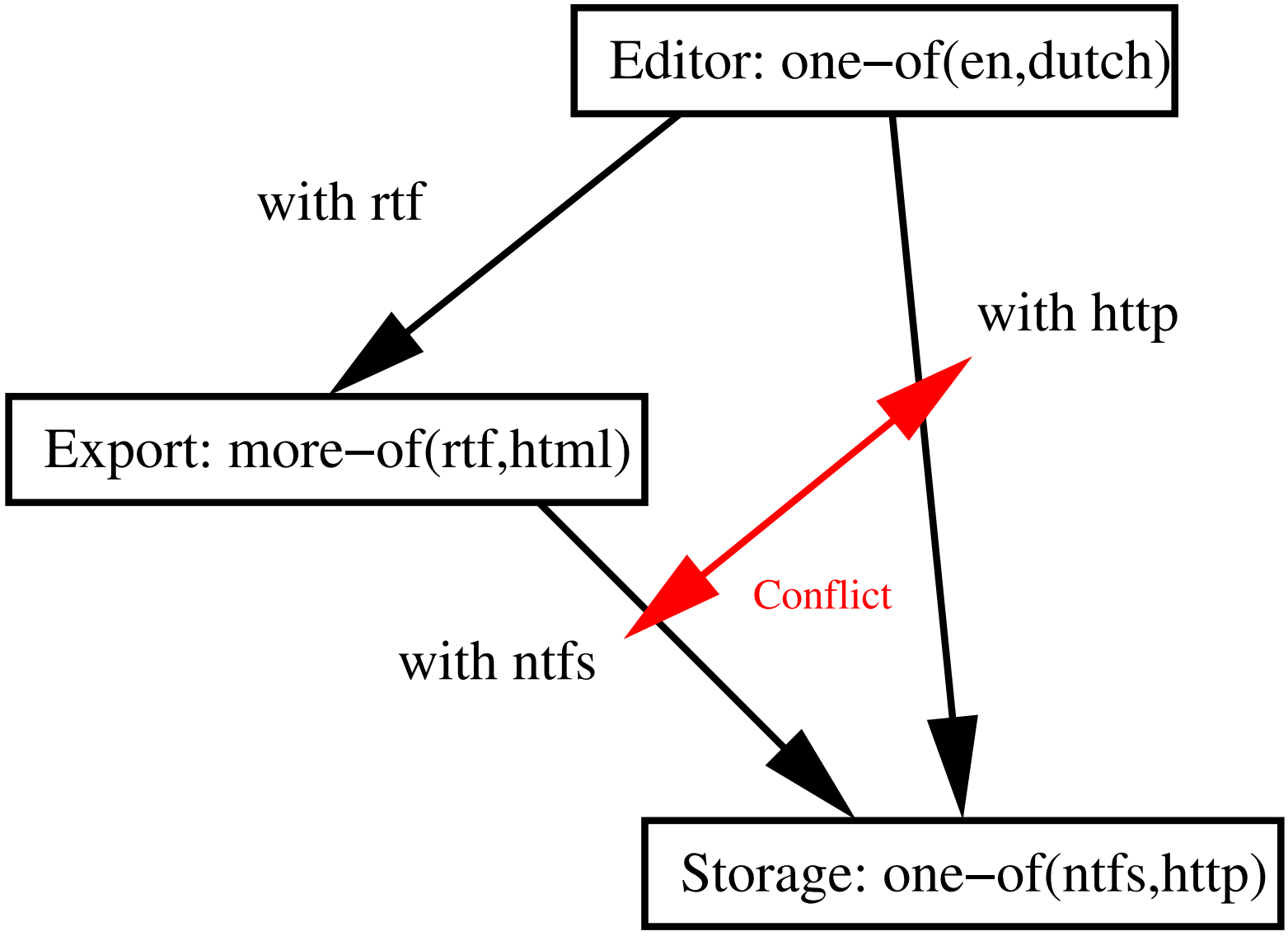
  (e.g. `all(a,b)` and `a excludes b` is inconsistent)

- Composition descriptions should be consistent

  (e.g. enabling `a` and `b` for `one-of(a,b)` is incorrect)

Properties can be automatically checked using our language.

Invalid composition



Correctness of composition

Valid configurations of Storage:

$$\text{configurations}(Storage) = \{\{\text{ntfs}\}, \{\text{http}\}\}$$

The received set $\{\text{ntfs}, \text{http}\}$ is not part of a configuration.

General rule:

$\forall c \in Components$:

$$\exists C \in \text{configurations}(c) : \left(\bigcup_{\text{require}(c,F)} F \right) \subseteq C$$

Configuration of compositions

How to obtain a configuration interface for a composition?

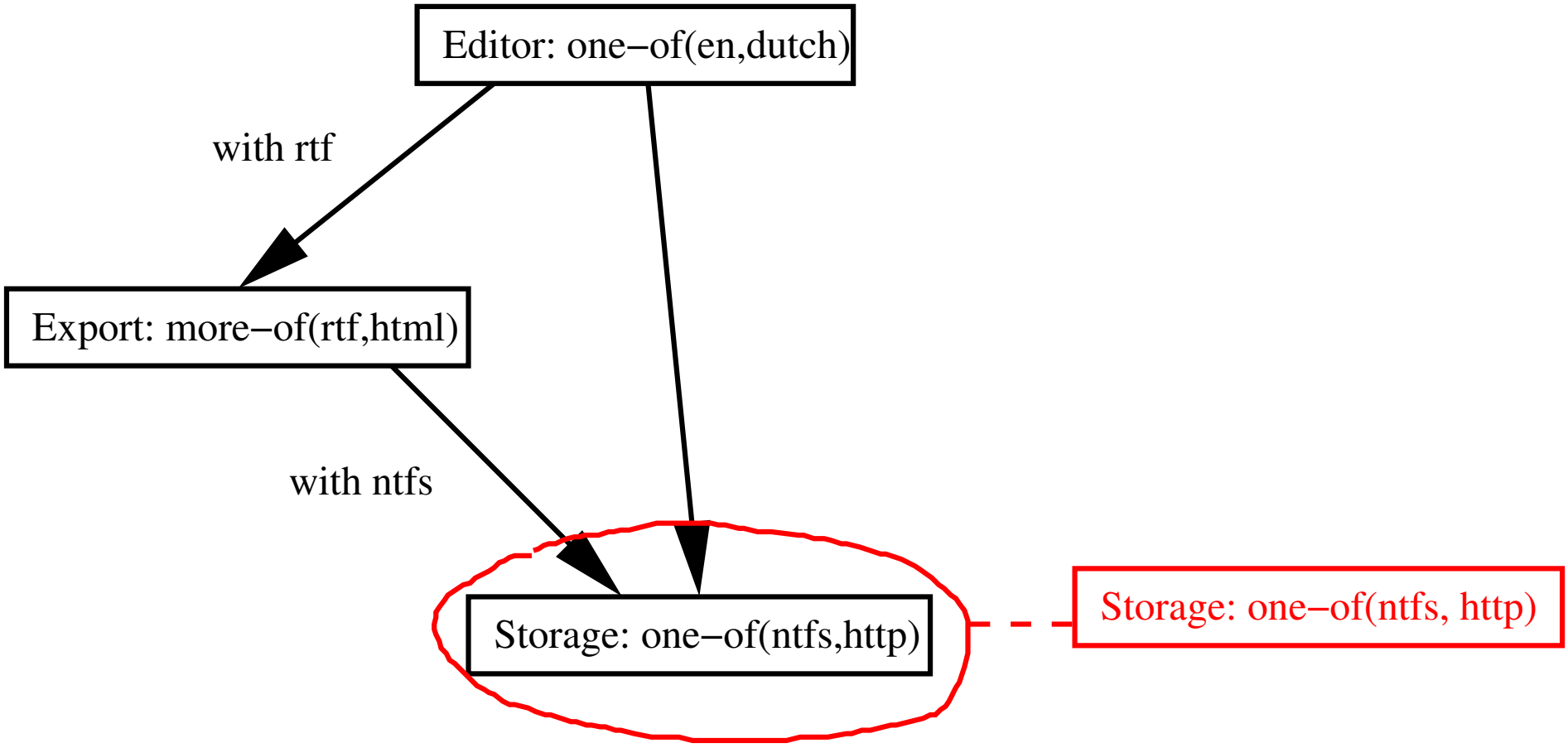
By partial evaluation and variability inheritance:

- components partially instantiate dependencies;
- unbound variability propagates upwards.

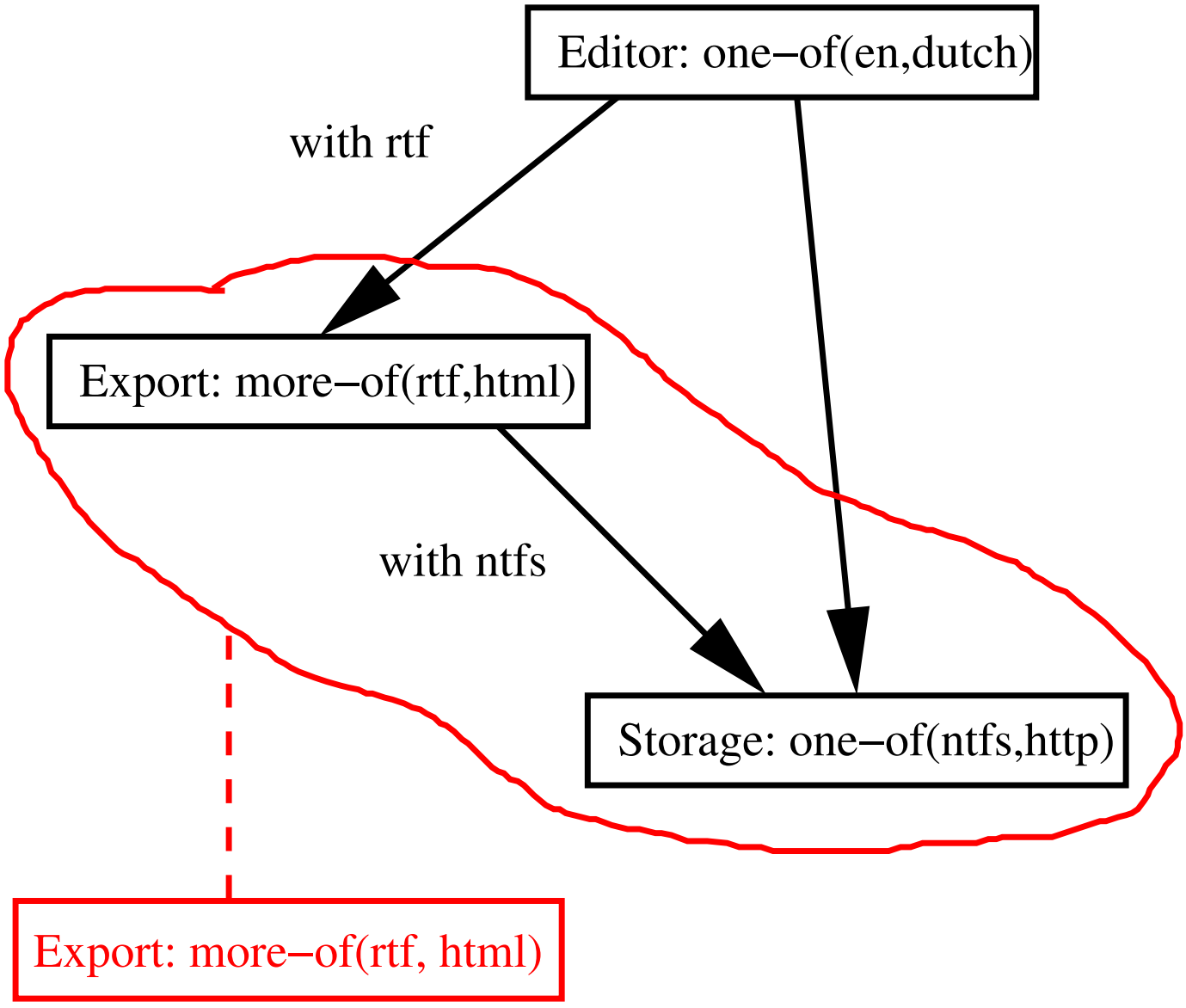
Example of partial evaluation:

`more-of(a,b)` with `a` $\longrightarrow$ `b?`

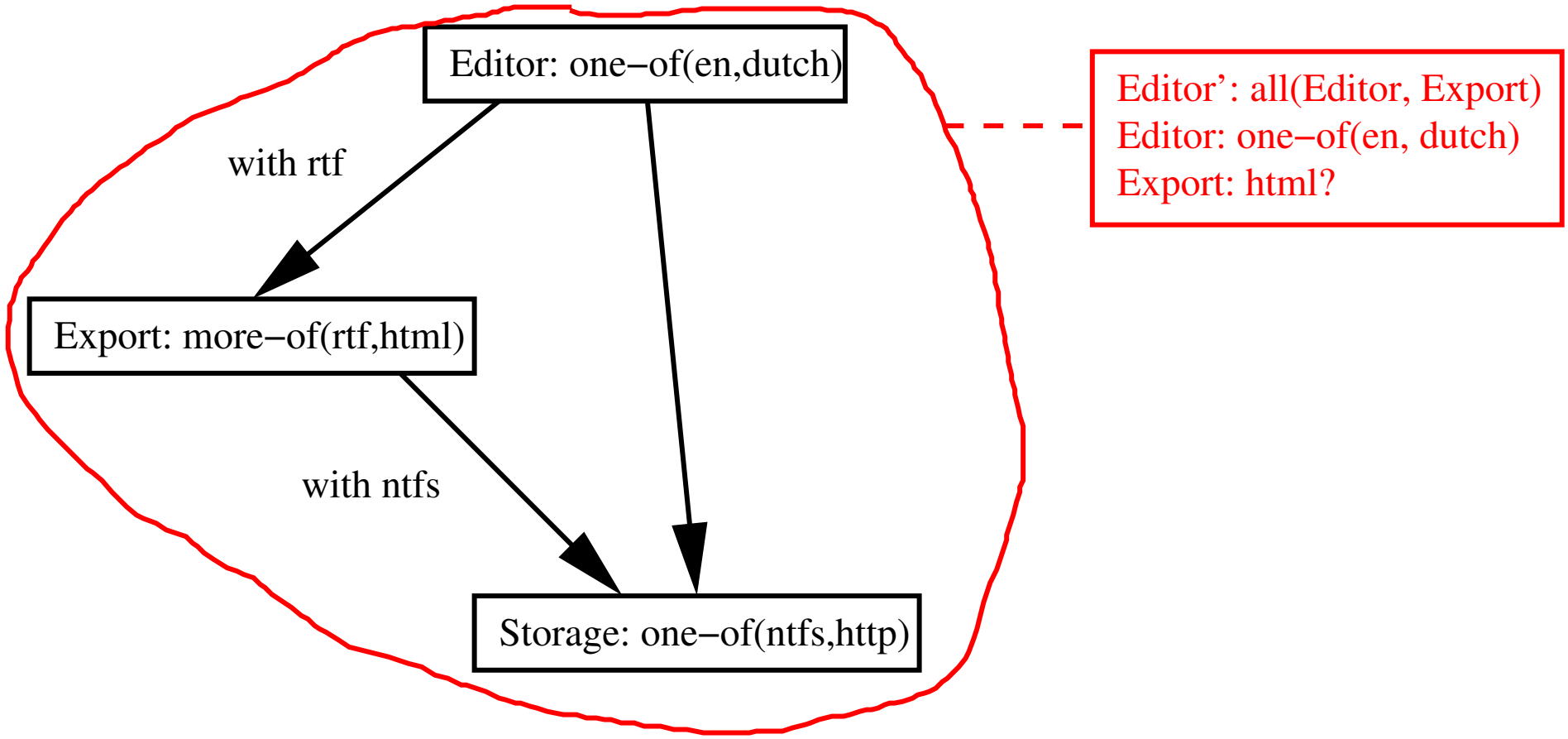
Configuring compositions



Configuring compositions



Configuring compositions



Implementation

Language technology enables automatic checking of correctness properties.

Partial prototype implementation:

- Component description language in ASF+SDF
- Satisfiability of FDL in ASF+SDF (using BDDs)
- Composition computed using relational query evaluator RSCRIPT.

Concluding...

To sum up:

- Feature descriptions are used to describe component variability.
- Compositions can be configured if certain consistency requirements are met.
- The configuration interface of a composition is obtained by partial evaluation and variability inheritance.

Future work

Future work:

- Reconcile system variability and component variability.
- Connecting problem space configuration to solution space variation.

Current focus:

- Instantatiation with Java packages.
- Binding of features using AspectJ.

For ongoing work: `http://www.cwi.nl/~storm`.

Questions?