> Develop the human mental configuration from the XML file.
> Derive and express the maximum information contained within the XML file.
> Check the requirements of the parsed transformer.
> Execute the transformation.

In the end, transformations are sheets that share a common interface:
- Transformation (source, target)
- Transformation (target, source)
- Transformation (source, source)
- Transformation (target, target)

The generator allows these sheets and annotates the corresponding methods.

The transformation becomes standard in the right model can be achieved using the proxy interface.

**Challenges**
- Share the same add-on interface and extend XML and transformation (source, target) at the same time.
- Identify components that are transformation (source, target) and do not have a common interface.
- Identify composition (source, source) and transformation (source, target).
- Identify composition (source, target) and transformation (target, target).
- Identify composition (source, target) and transformation (source, target).

**Summary**
- The use of new concepts of software architects.
- The use of new concepts in the field of software architecture.
- New concepts are well integrated into the field of software architecture.
- The source of transformation normally refers to the common configuration transformation (source, target).
- Challenges: comprehensibility and communicability, new transformations, composition or source, practical adaptability, and changes in transformations.
Feature Model Driven Generation of Software Artifacts

Roman Táborský and Valentino Vranič

Institute of Informatics and Software Engineering

crudecrude@gmail.com vranic@stuba.sk fiit.sk/~vranic

WAPL 2015 @ FedCSIS 2015
Łódź, September 13, 2015
> A feature model defines variants in a structured way

> Domain engineering / software product lines

> Configurability first; appropriate implementation mechanisms selected accordingly

> Need not necessarily involve any academic feature modeling notation


> A simple FODA-like style used here

> Abstracting from advanced concepts like binding time, additional constraints among features, feature cardinality...
Feature Modeling

Variants

But how exactly to implement features?
What are they in code?
> Features sometimes can be implemented as well localized components

> But often the realization of one feature is spread over several components

> Sometimes this involves crosscutting: a case for aspect-oriented programming
Features sometimes can be implemented as well localized components

But often the realization of one feature is spread over several components

Sometimes this involves crosscutting: a case for aspect-oriented programming

A simple conditional compilation can cope with this, but the code of crosscutting features would be scattered

Generative approaches rely on the set of selected features to generate highly specialized code

Generators can be external, but also metaprogramming based

In either case, features are understood as the structure and behavior to be added, adapted, or removed
Our idea: make each feature's realization a transformation

Selecting a feature activates the corresponding transformation

A transformation can affect all code, including the other transformations

Virtually, this is a distributed generator embedded into a feature model

The actual generator is then simple and generic

This raises a number of issues...
> Metatransformations

> Complexity of transformations
Where to keep the transformations?

One way is to attach them directly to the feature model that may conveniently be kept in the XML format

```xml
<feature>
  <feature>
    <transformation>
      <!-- Transformation information including the event chain, metatransformation information, requirements, etc. -->
    </transformation>
  </feature>
</feature>
```
Another way is to keep only references to transformations in the feature model (this is what we did)

The actual code of the transformations is in the corresponding files
<feature name="DynamicContent-AccessLog" ID="4"

<feature name="DynamicContentProvider" ID="5"

<feature name="ASP.Net" ID="7"
</feature>

<feature name="DatabaseBackend" ID="6"

<feature name="MSSQL" ID="8"

<feature name="XML" ID="9"
</feature>

<feature name="MicroformatSupport" ID="10"

<feature name="CustomHTMLHeader" ID="11"

<feature name="GeneratorDoc" ID="12"
</feature>
</featuremodel>
> Parse the feature model configuration from the XML file

> Parse and execute the metatransformations contained within the transformations from the XML file

> Check the requirements of the parsed transformations

> Execute the transformations
In the end, transformations are objects that share a common interface

```java
public interface ITransformation {
    CheckPrerequisites();
    ExecuteTransformation();
    GetParameterNames();
    GetMetaTransformations();
    ...
}
```

The generator takes these objects and executes the corresponding methods

The transformations have to be executed in the right order: can be adjusted using the priority attribute
Challenges

> Assess the comprehensibility and maintainability of a software system expressed by a number of transformations: does this preserve the intent?

> Explore a non-exclusive application of the approach in which only some features would be realized as transformations

> How to deal with complex feature models consisting of several separate feature diagrams with references between them

> Foster the practical adoption of the approach by providing directly reusable transformations, transformation templates, and transformation schemes or examples
Digression:

Transformations as changes

Aspect-oriented change realization

> Feature model driven generation of software artifacts

> Besides an enhanced feature model, no other models are necessary

> Transformations are not limited in affecting the software system

> The concept of metatransformations: modify the common transformations prior to their execution

> Challenges: comprehensibility and maintainability, non-exclusive application, complex feature models, practical adoption, and changes as transformations

vranic@stuba.sk
fiit.sk/~vranic