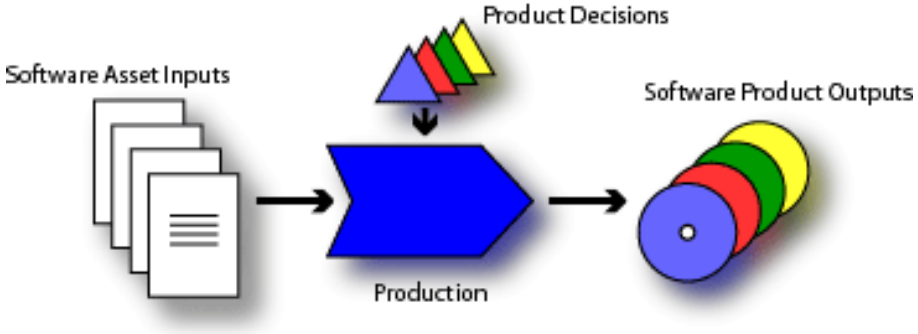
Software Product Lines

Dennis Wagelaar
Viviane Jonckers
Software Languages Lab
Vrije Universiteit Brussel

Software Product Lines (SPL)

- SPL origins, goals
- SPL concepts
  - Core assets, features
  - Product decisions, output
- SPL process
  - SCV analysis, feature modelling
  - Configuration, product derivation
- MDA & SPL integration
- Further reading



From http://www.softwareproductlines.com

SPL origins:
Mass customisation & commonality

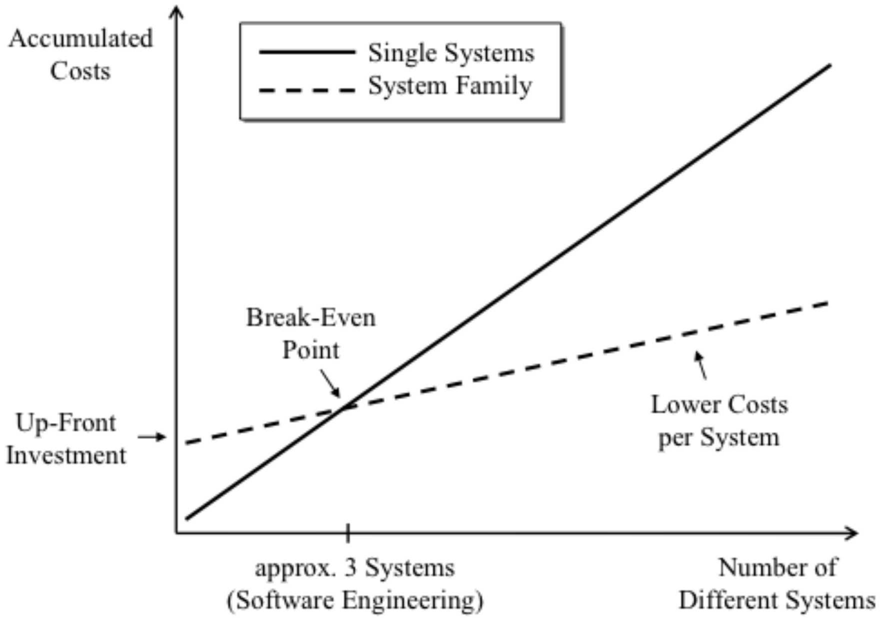
- Software Product Lines (SPLs) follow the idea of regular product lines, such as:
  - Ford automobile product line
  - Kodak camera product line
  - HP printer product line
- Product lines aim to combine two principles:
  - **Mass customisation**: realise many versions of one car model (configured and assembled in one factory)
  - **Mass production**: from a pool of carefully architected car parts (produced in dedicated factories)

SPL origins:
Definition

- “A software product line is a **set of software-intensive systems** sharing a **common, managed set of features** that satisfy the specific needs of a particular market segment or mission and that are developed from a **common set of core assets** in a prescribed way.” [Clements & Northrop 2001]
- Also known as Software Families or Family-Oriented Software Development
- Classic reuse is opportunistic: a general software component is put in a library in hope that opportunities for reuse will arise
- In SPL reuse is predictive: software artefacts are created because reuse is predicted in one or more products in a well defined product line

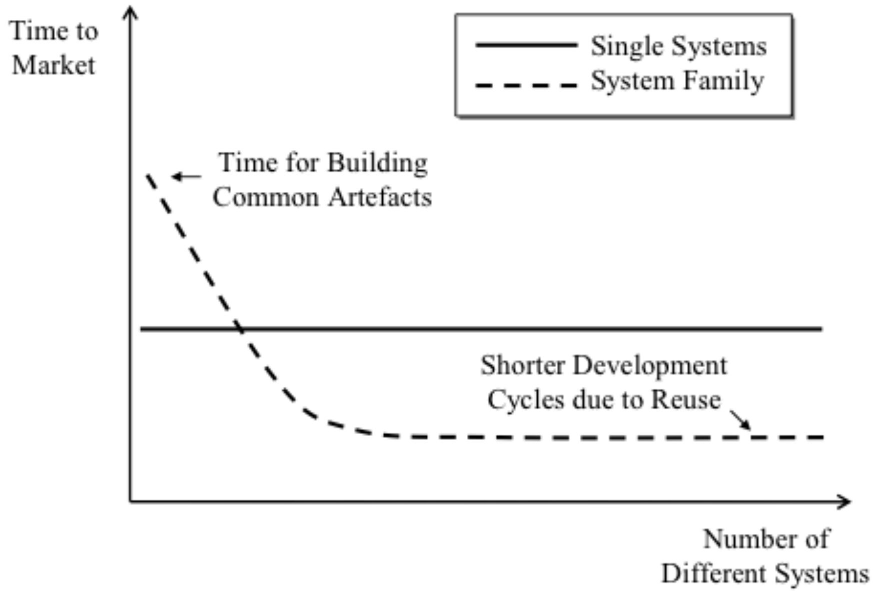
SPL goals:
Envisioned benefits (1/2)

- Reduction of development costs
  - Fixed up-front investment in product line infrastructure pays back as system family grows



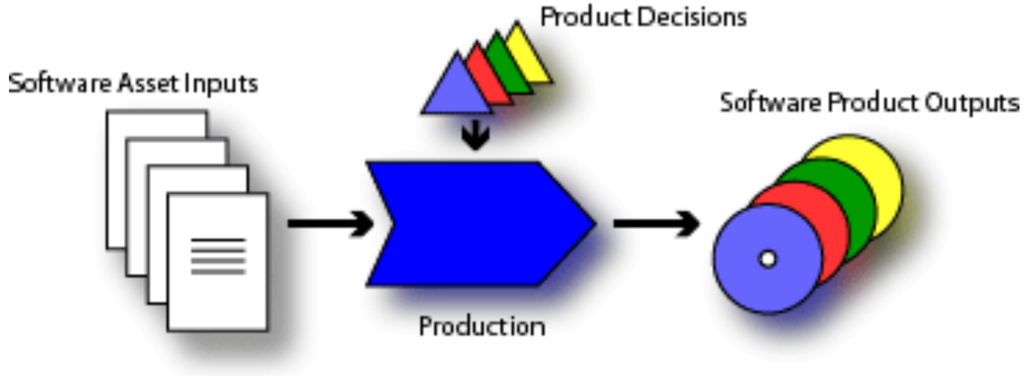
SPL goals:
Envisioned benefits (2/2)

- Reduction of time to market
  - Fixed up-front investment in product line infrastructure pays back once it is in place



SPL goals:
Commonality and variability

- **Capitalise on commonality** within a set of software products, thereby avoiding duplication and divergence.
- **Manage variation** by clearly defining the variation points for a given set of software products.



Source: http://www.softwareproductlines.com/introduction/concepts.html

SPL concepts (1)

- Software Asset Inputs (core assets)
  - a collection of software artefacts – such as requirements, architecture, source code, components, test cases, domain models, documentation, ... – that can composed in different ways to create all products in a product line
  - each asset has a well defined role within a common architecture for the product line, i.e. it contributes to realise a **feature** of a product
  - some assets are fixed, they occur in every product (e.g. a platform artefact, a core architecture), some assets are configurable, they occur in some products (e.g. a plug-in, a component)
  - assets may have internal variation points

SPL concepts (2)

- "A **feature** is a **system property** that is **relevant to some stakeholder** and is used to **capture commonalities** or **discriminate between systems**." [Czarnecki, Helsen & Eisenecker 2004]
- Feature model (decision model)
  - A description of optional and variable features for the products in the product line
- Product decisions (configuration)
  - Choices that are made for each of the optional and variable features in the decision model

SPL concepts (3)

- Production mechanism
  - A (technological) means for composing and configuring products from the software asset inputs (e.g. a plugin architecture, a middleware platform in which components are deployed, MDA style code generation, …)
- Software product outputs
  - Deployable software systems (products) that can be generated from the core assets (e.g. integrated source code of a product, a make-file with deployment descriptors, a packaged product or product installer, …)

SPL process: Overview

1) Scope, commonality & variability (SCV) analysis
   - Determine scope of software product outputs
   - Determine common & variable features for that scope
2) Feature modelling
   - Model feature relationships/dependencies
3) Configuration
   - Select features for a specific software product
4) Derive software products
   - Implement the configurations

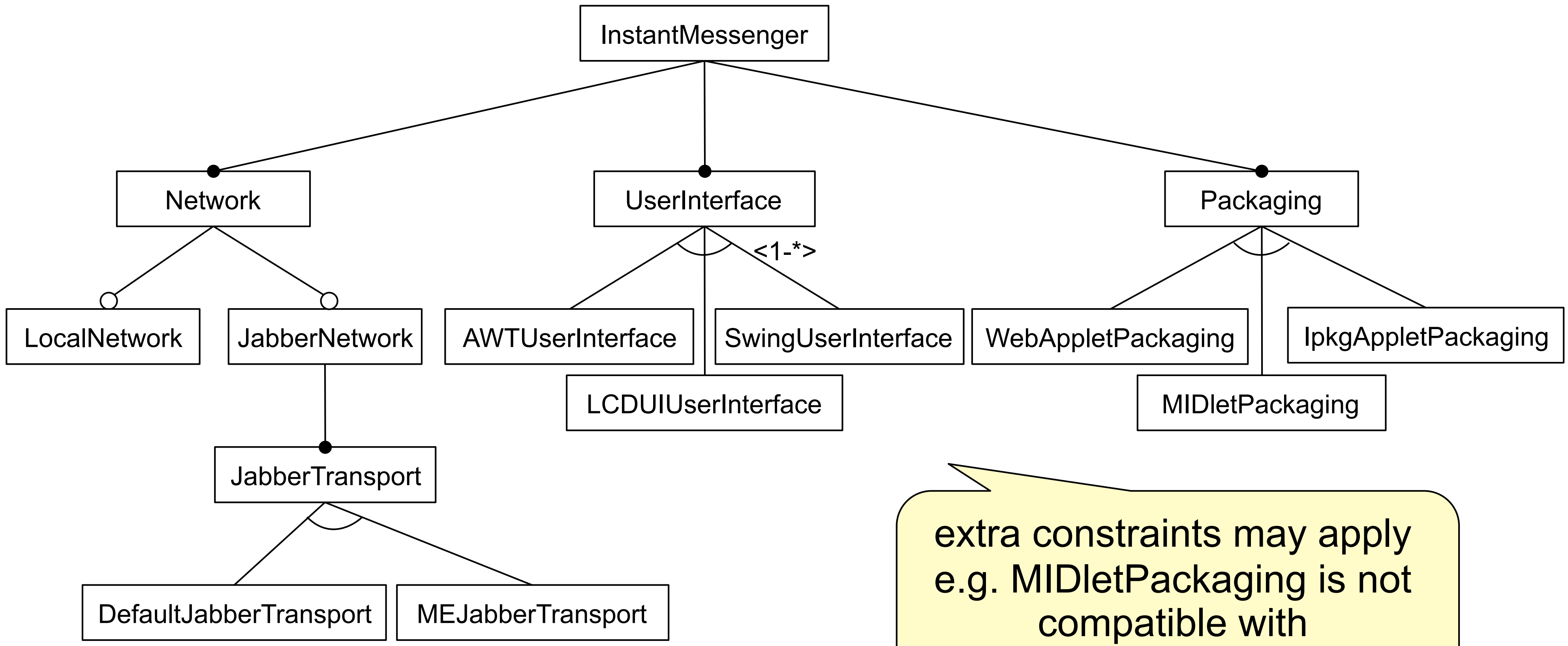
SPL process: SCV analysis

- Scope
  - Range of software products that we want to derive from the software asset inputs
- Scope management ranges between:
  - **Proactive:** anticipate all products needed on the foreseeable horizon
  - **Reactive:** support only products needed in the immediate term and add new products as the need arises

SPL process: SCV analysis (2)

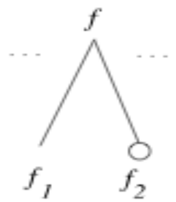
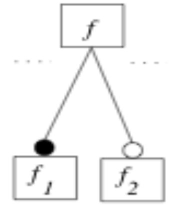
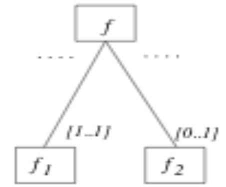
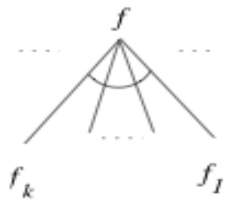
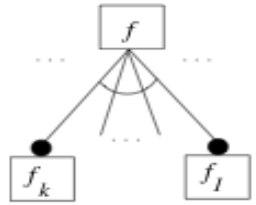
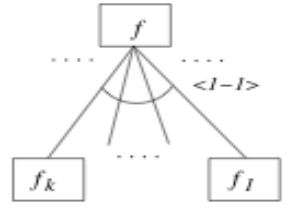
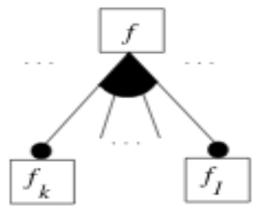
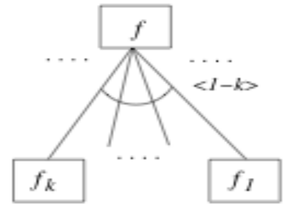
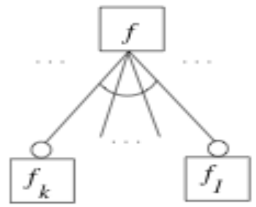
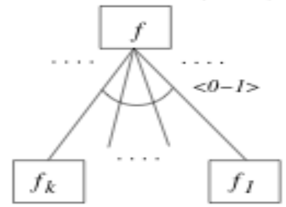
- Commonalities
  - Core assets are built for each commonality
  - A typical core asset is a common architecture for the entire SPL
- Variabilities
  - Are bounded by placing specific limits
  - Are often organised as a hierarchy of sub-variabilities
  - Example: an instant messaging client can support multiple communication protocols (ICQ, MSN, Jabber).
    - A Jabber sub-variability is encryption/no encryption

SPL process:
Feature modelling: example



extra constraints may apply
e.g. MIDletPackaging is not compatible with SwingUserInterface

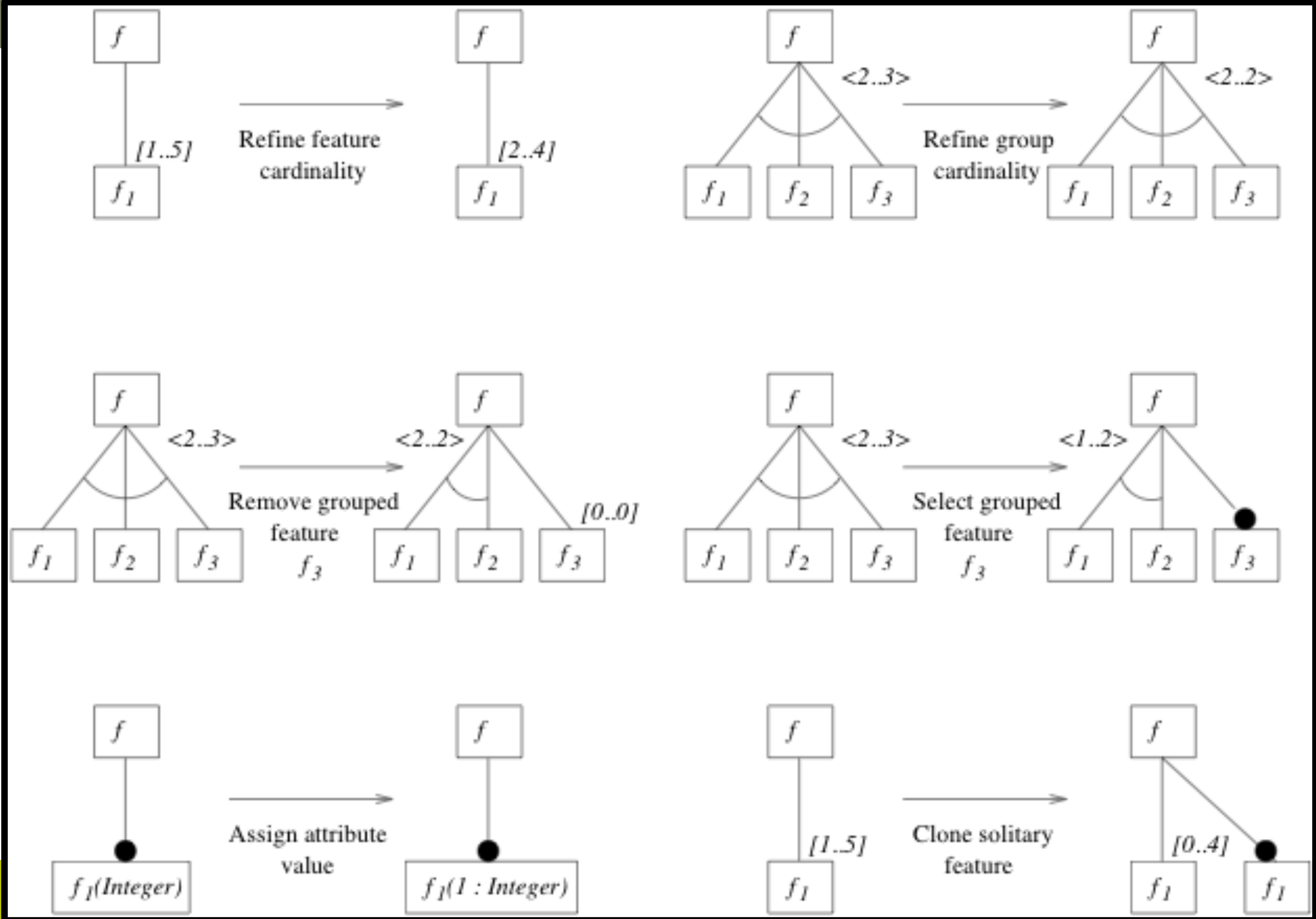
SPL process: Feature modelling

FODA notation (Kang et al., 1990)	**Extended notation (Czarnecki, 1998; Czarnecki and Eisenecker, 2000)**	**Cardinality-based notation**
mandatory and optional subfeatures 	*mandatory and optional subfeatures* 	*mandatory and optional subfeatures* 
alternative subfeatures 	*exclusive-or group* 	*group with cardinality* $\langle 1-1 \rangle$ 
n/a	*inclusive-or group* 	*group with cardinality* $\langle 1-k \rangle$ 
n/a	*exclusive-or group with optional subfeatures* 	*group with cardinality* $\langle 0-1 \rangle$ 

SPL process:
Configuration

- Decision model can be in the form of:
  - Feature model
  - Domain-specific language (DSL) definition
  - Logic rules
- Product decisions conform to the decision model:
  - Constrained feature model (staged configuration)
  - Expression in DSL
  - Logic assumptions

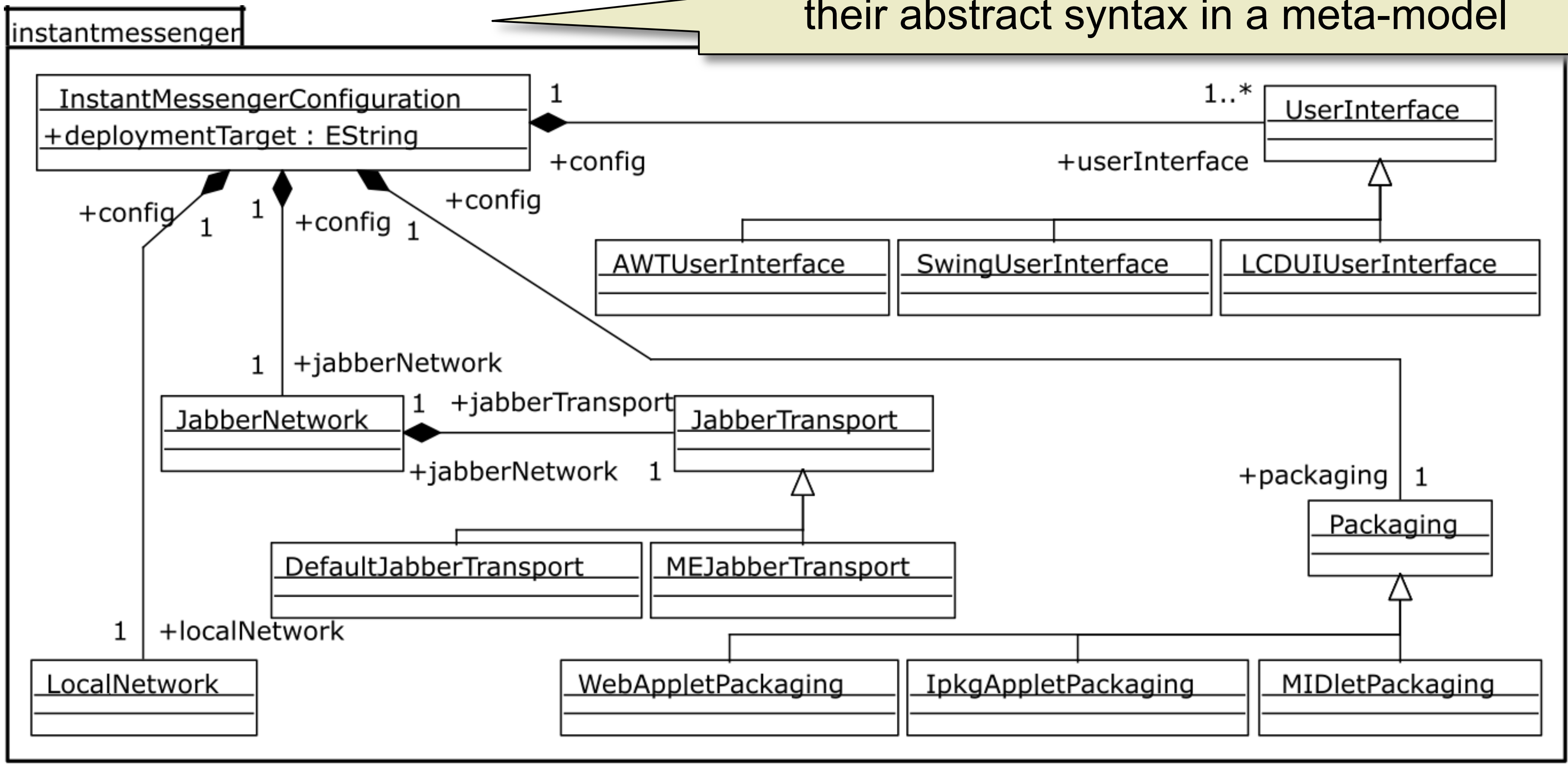
SPL process:
Configuration: Staged configuration



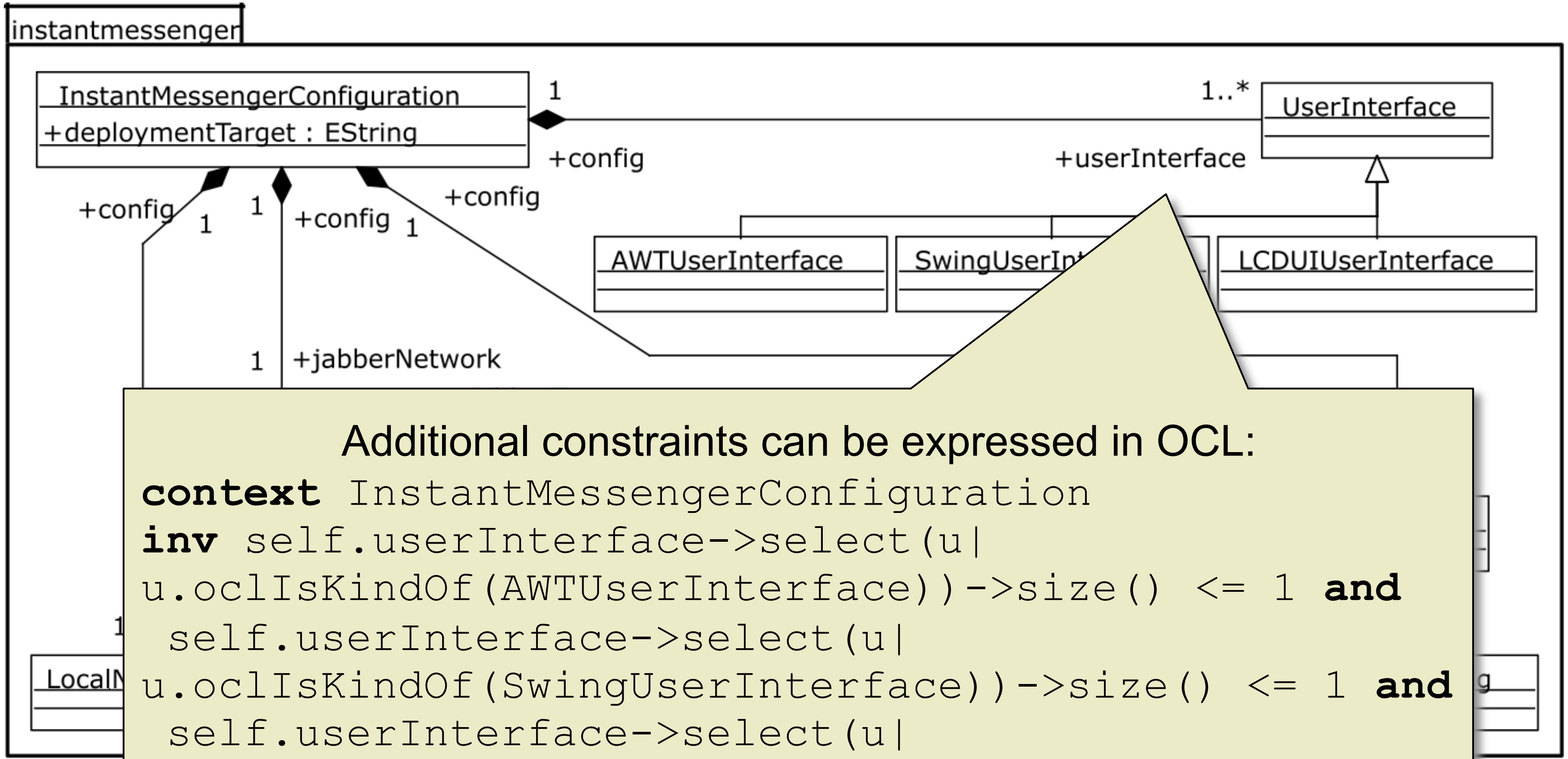
SPL process:

Configuration: DSML

Domain-specific modelling languages define their abstract syntax in a meta-model



SPL process:
Configuration: DSML

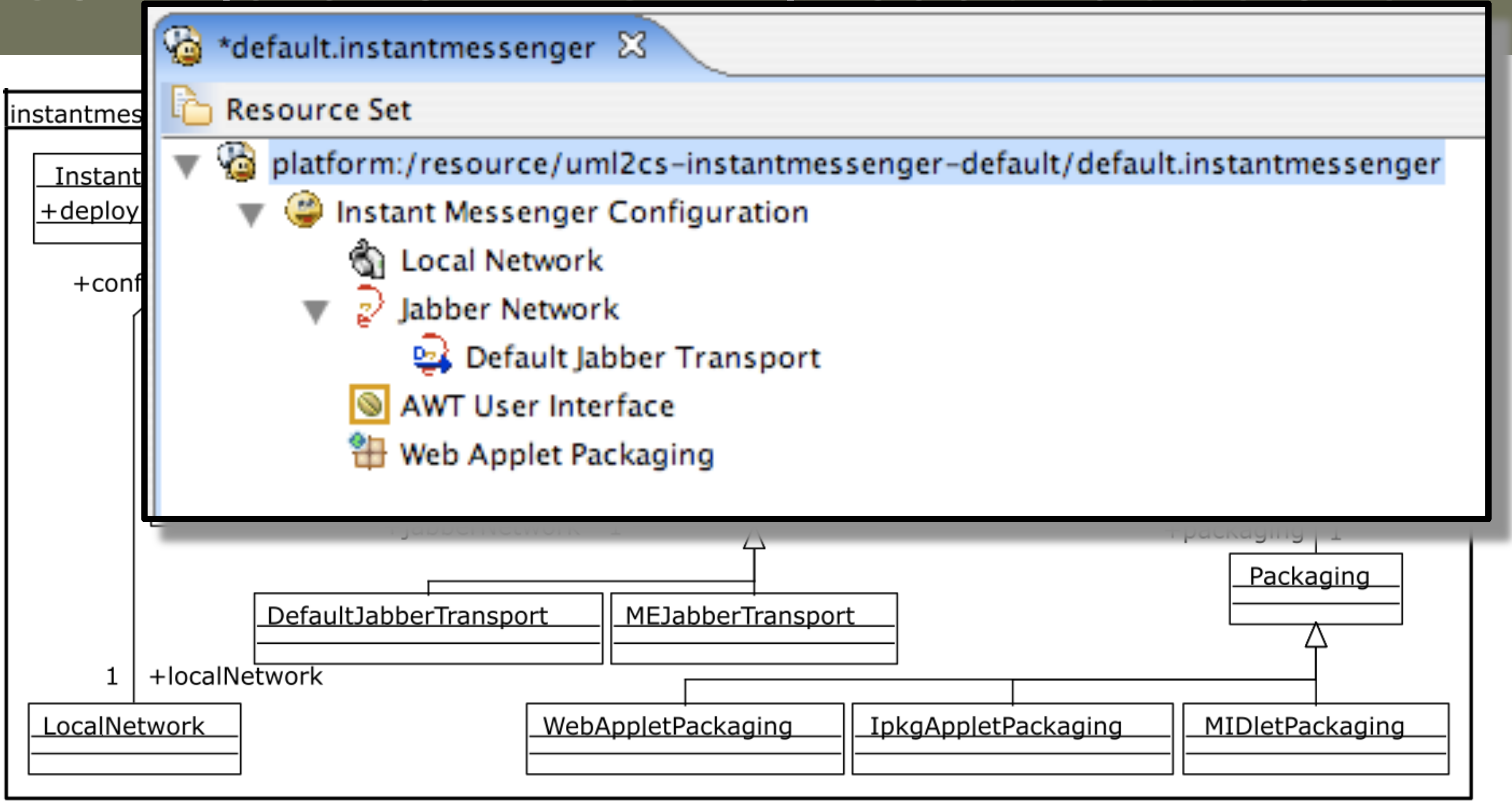


Additional constraints can be expressed in OCL:

```
context InstantMessengerConfiguration
inv self.userInterface->select(u|
u.oclIsKindOf(AWTUserInterface))->size() <= 1 and
 self.userInterface->select(u|
u.oclIsKindOf(SwingUserInterface))->size() <= 1 and
 self.userInterface->select(u|
u.oclIsKindOf(LCDUIUserInterface))->size() <= 1
```

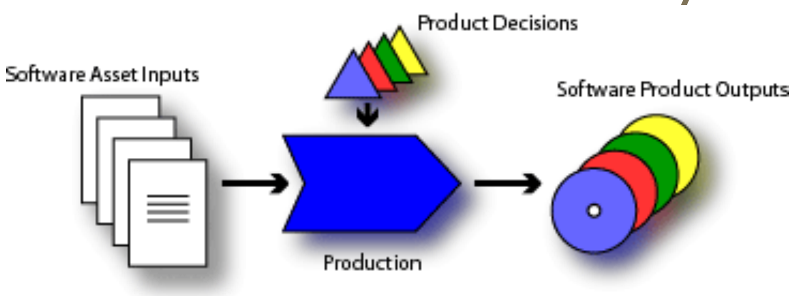
SPL process:

Configuration: DSML product decisions



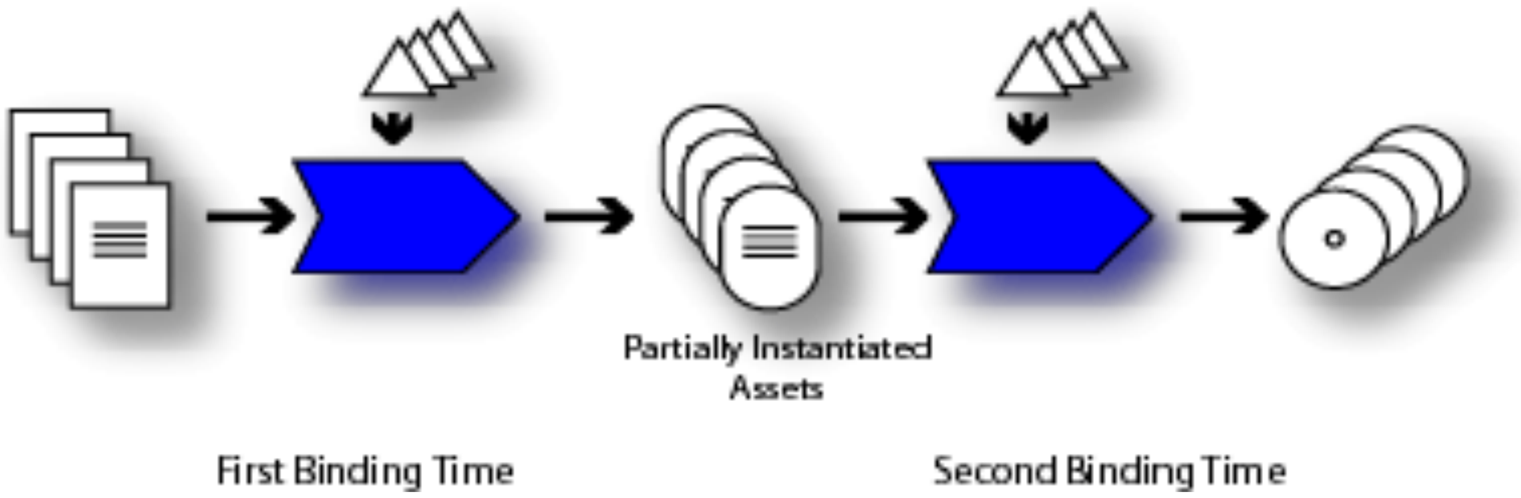
SPL process:
Derive software product

- Manual:
  - When new configurations are rarely made
  - When only few configurations exist
  - Consistency with configuration must be checked by hand
- Automatic:
  - When configuring often
  - When configuring many products
  - Enforces consistent implementation of configurations



SPL process:
Binding times

- SPL core assets can be “bound” to (partial) software products at various times

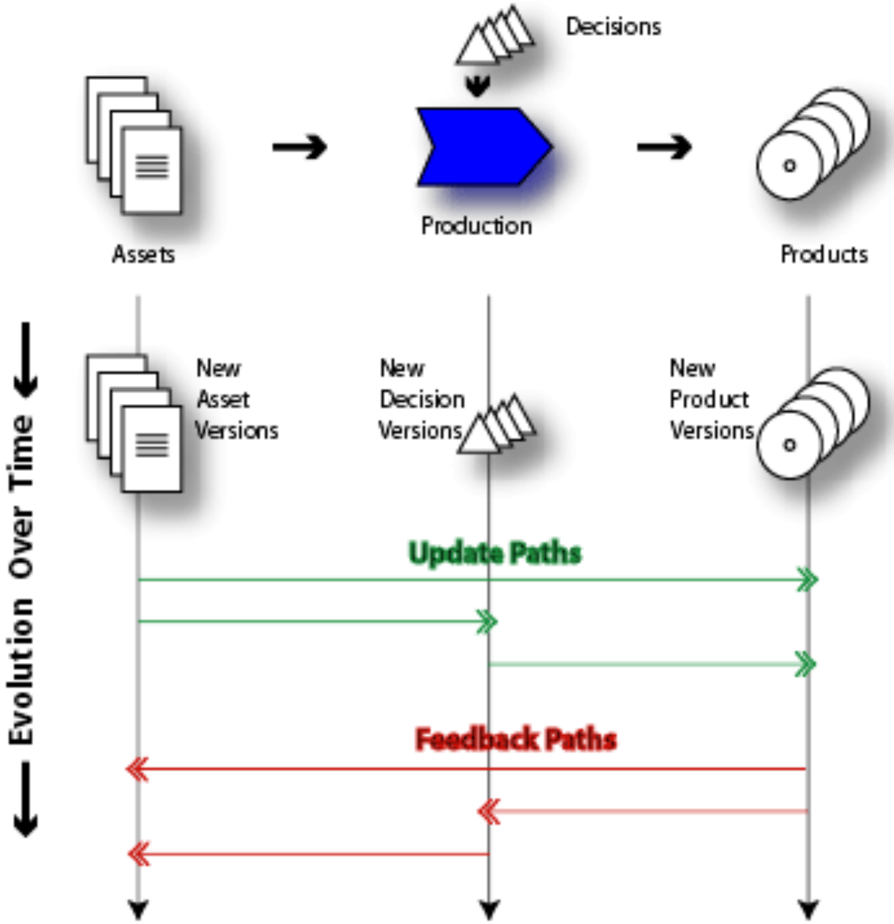


SPL process:
Binding times (2)

- Possible binding times:
  - **Source reuse time:** reuse configurable source artefact
  - **Development time:** architecture, design, coding
  - **Static code instantiation time:** code assembly
  - **Build time:** during compilation
  - **Package time:** deployable packages
  - **Customer customisation:** on-site adaptations
  - **Install time:** during software installation
  - **Startup time:** during software startup
  - **Run-time:** during software execution

SPL process:
Evolution

- Update paths
  - Changes in core assets must be reflected in products
  - Introduction of new or changed assets gives opportunity to evolve all products
- Feedback paths
  - Changes in a product must be generalised in core assets
  - Fixes to core assets can be propagated to all products



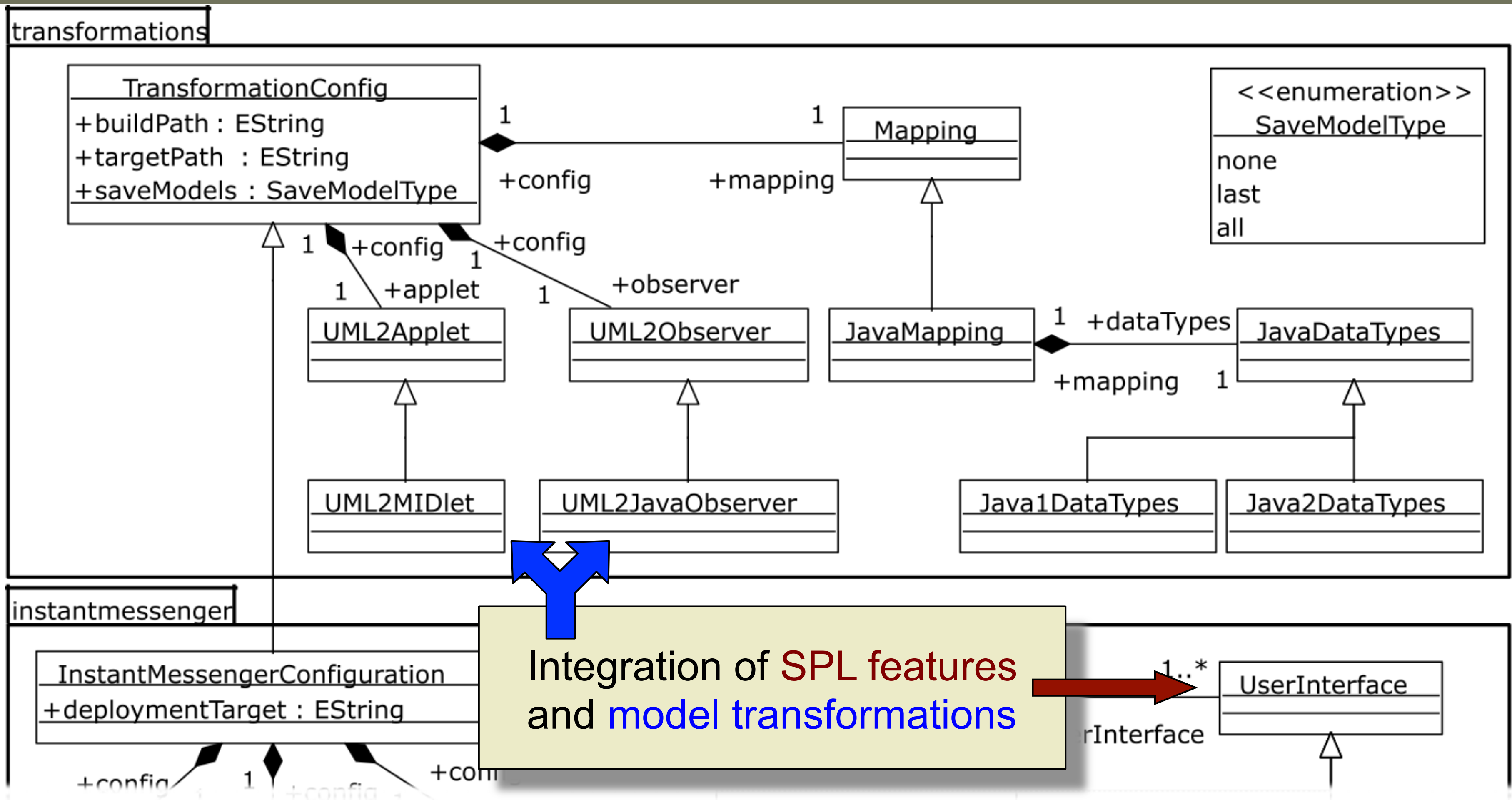
Software Product Lines:
Summary

- SPLs leverage commonalities between related software products while facilitating variabilities
  - Increased (and enforced) software reuse
  - Controlled variation
- SPLs have a specific development process in addition to a traditional software engineering process
  - Introduces (shared) overhead in development effort
  - Difficult to apply on smaller scale

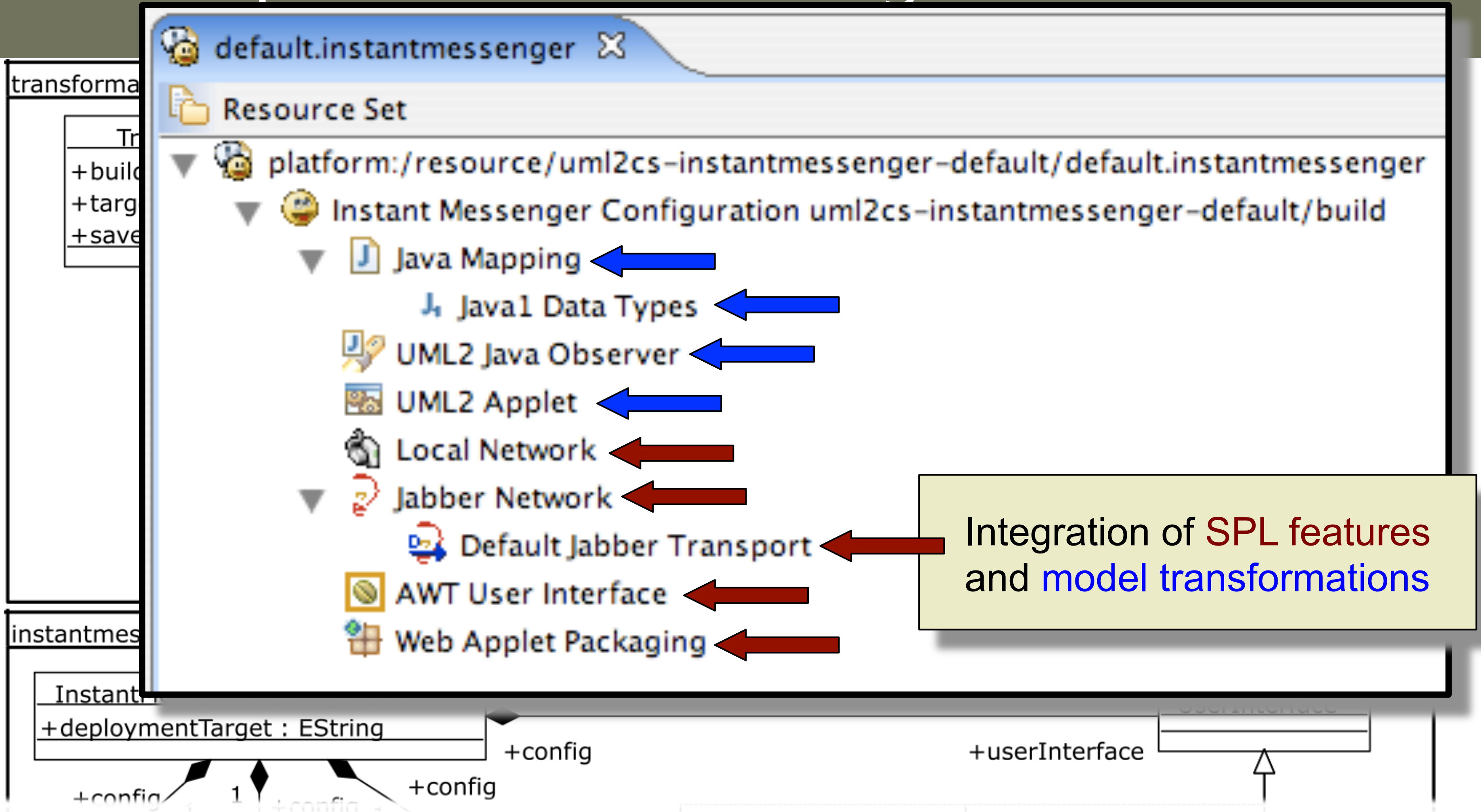
MDA & SPL integration:
Overview

- SPLs use models for configuration
  - Model transformation can be used to automatically generate products
- The MDA targets multiple PSMs
  - PSMs can be considered as products in a SPL
- The MDA has no configuration approach for multiple model transformations
  - SPL configuration is applicable to the MDA

MDA & SPL integration:
Example: MD-SPL configuration



MDA & SPL integration:
Example: MD-SPL configuration



MDA & SPL integration:
Example: MD-SPL configuration

- Integrated configuration DSL
  - Combines model transformation configuration rules with feature configuration rules
- Integrated product generation
  - Generator applies model transformations to all selected features

Further reading:
Books

- K. Pohl, G. Böckle, F. van der Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques* (2005)
  http://www.software-productline.com/
- P. Clements, L. Northrop, *Software Product Lines: Practices and Patterns* (2001)
  http://www.informit.com/store/product.aspx?isbn=0201703327
- D. M. Weiss, C. T. R. Lai, *Software Product-Line Engineering: A Family-Based Software Development Process* (1999)
  http://tinyurl.com/cwjllo
- K. Czarnecki, U. W. Eisenecker, *Generative Programming - Methods, Tools, and Applications* (2000)
  http://www.generative-programming.org/

Further reading:
Papers

- K. Czarnecki, S. Helsen, U. W. Eisenecker, *Staged configuration through specialization and multilevel configuration of feature models*, Software Process: Improvement and Practice **10**(2)
  http://swen.uwaterloo.ca/~kczarnec/spip05b.pdf
- J. Coplien, D. Hoffman, D. Weiss, *Commonality and variability in software engineering*, IEEE Software **15**(6)
  http://doi.ieeecomputersociety.org/10.1109/52.730836
- D. Benavides, A. Ruiz-Cortéz, P. Trinidad, S. Segura, *A Survey on the Automated Analyses of Feature Models*, Proceedings of JISBD'06
  http://www.lsi.us.es/~trinidad/docs/benavides06-jisbd.pdf

Further reading:
Websites

- Software Product Lines website at CMU:
  http://www.sei.cmu.edu/productlines/
- Software Product Lines website by BigLever:
  http://www.softwareproductlines.com
- Software Product Lines Conferences:
  http://splc.net/
- Generative Programming and Component Engineering Conferences:
  http://www.gpce.org/
- VariBru - Variability in Software-Intensive Product Development:
  http://www.varibru.be/