Debugging Structure-based Configuration Models

ECAI 2008 – Configuration Workshop
21.-22.07.2008, Patras, Greece

Thorsten Krebs
HITeC e.V. c/o University of Hamburg

Contents

Contents

Introduction

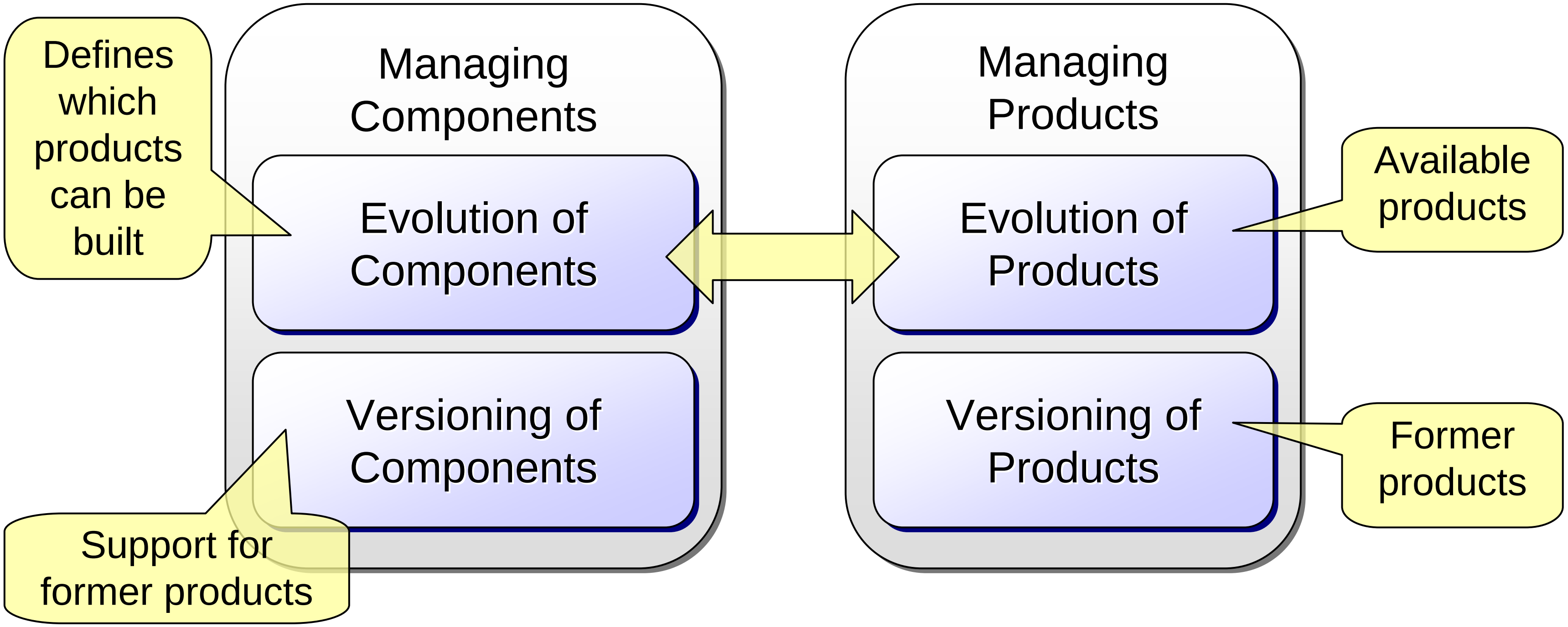
- **Motivation**
  - Typical configuration domains consist of
    - several hundreds or thousands of components, and
    - restrictions on how the components can be combined
    - in one *configuration model*.
  - Environment in which configurable products and components continually evolve
    - Overview can easily get lost
    - Difficulty to manage both conceptual representation and constraints
    - Consistency of configuration model is a prerequisite for deterministic configuration results

Introduction

- **Well-formedness**
  - A configuration model is *well-formed* when it adheres to the language specification
  - But well-formed knowledge may still be inconsistent!
- **Consistency**
  - A concept is *consistent* when it allows at least one instance
    - A component representation is consistent when it allows instances
    - A product representation is consistent when all required parts (components) are consistent

Knowledge Representation

- **Knowledge Management**
  - Semantic differentiation between
    - concepts that represent components, and
    - concepts that represent products



Introduction

- **Solution Approach**
  - Product-centered framework
    - Semantic distinction between conceptual representation of components and the configurable products
    - Improves reasoning about impacts of changes
  - Three typical use cases:
    1. Which Components are relevant for a Configurable Product?
    2. Which Components are not relevant for any Configurable Product?
    3. Which Components are “Reachable”?

Introduction

- **Context**
  - The work belongs to a larger framework: *Knowledge Management Supporting the Evolution of Configurable Products* [Krebs, 2007]
  - Evolution processes directly dealing with impacts that
    - Changing components has on configurable products, and
    - Changing configurable products has on the required components
  - More use cases: “introducing a product”, “retiring a product”, “which products are affected by changing a component”, “identifying common, widely used, rarely used and unused property values”, etc.

Contents

Knowledge Representation

- Based on **Description Logics** (*ALCQI+(D)*)…
  - Concepts
    - Sets of objects
  - Roles
    - Relations between objects
  - Instances
    - Specific objects
- …and the **Semantic Web Rule Language** (SWRL)
  - Antecedent: defines a conceptual pattern
    - Evaluates to true when a matching instance structure exists
  - Consequent: defines action
    - Is executed when the pattern evaluates to true

Knowledge Representation

- **Modeling Facilities**
  - *Concepts*
    - Map to DL concepts
  - *Attributes*
    - Map to DL roles with concrete domains as filler
  - *Composition Relations*
    - Map to DL roles with concepts as filler
    - Allow to specify a cardianlity
  - Attributes and composition relations are *properties*
  - *Instances*
    - Map to DL instances
  - *Constraints*
    - Map to SWRL rules

Knowledge Representation

- **Abstract and Concrete Concepts**
  - *Abstract* concepts
    - Generic concepts used for taxonomically grouping *similar components*
  - *Concrete* concepts
    - Concepts representing *specific components* that can actually be assembled for realizing a product
    - Leaf concepts with fully specified property values
  - Configurable products may specify abstract concepts as their parts
    - Instances of abstract concepts are specialized to be instances of a concrete concept

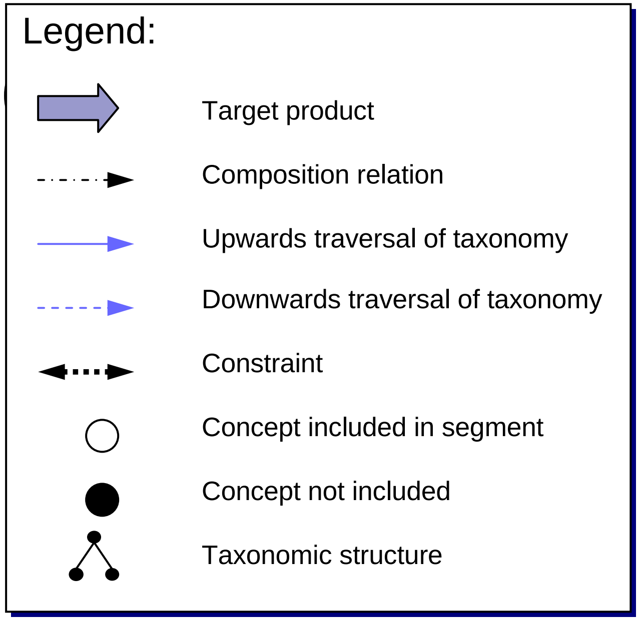
Contents

- Introduction
- Knowledge Representation
- **Which Components are relevant for a Configurable Product?**
  - Algorithm
  - Complexity
- Which Components are not relevant for any Configurable Product?
- Which Components are "Reachable"?
- Conclusion

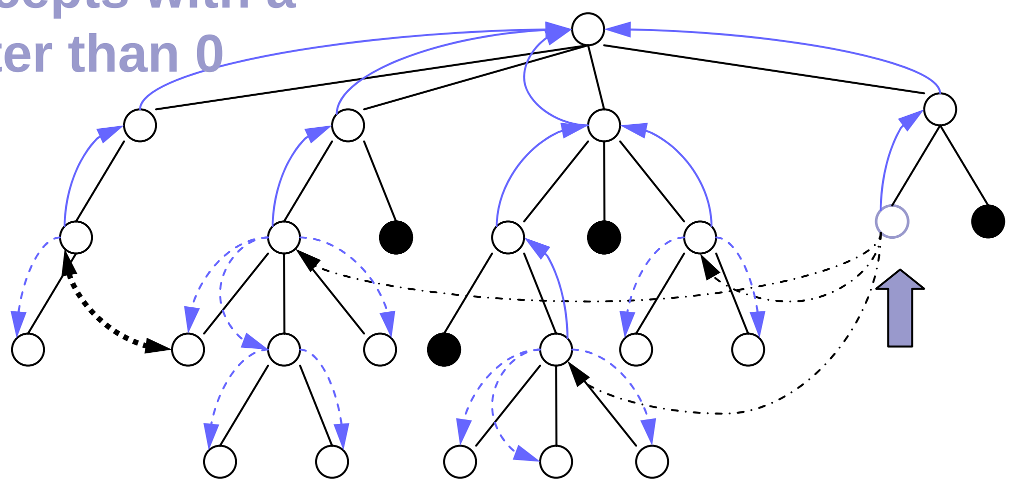
Which Components are relevant for a Configurable Product?

- **Algorithm**
  - Creating a product-specific *segment*

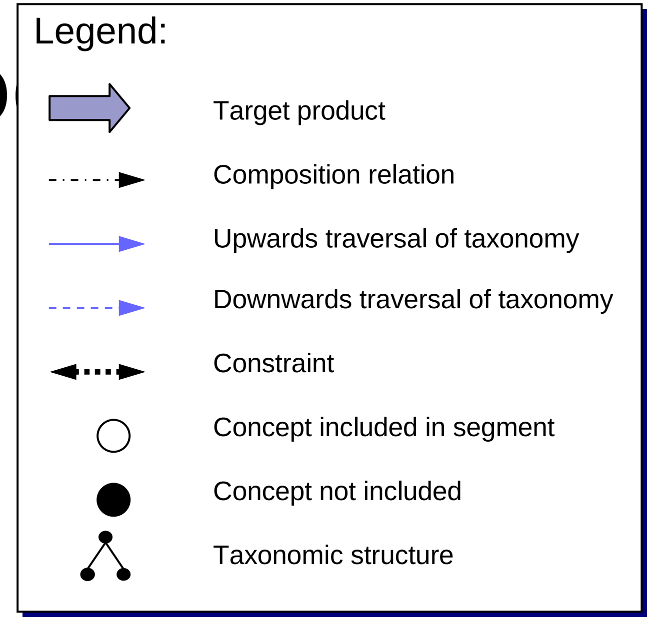
Which Components are relevant for a Configurable Pro



- **Algorithm**
  - Creating a product-specific *segment*
  1. Downwards Traversal of the Partonomy
     Including all part concepts with a max. cardinality greater than 0



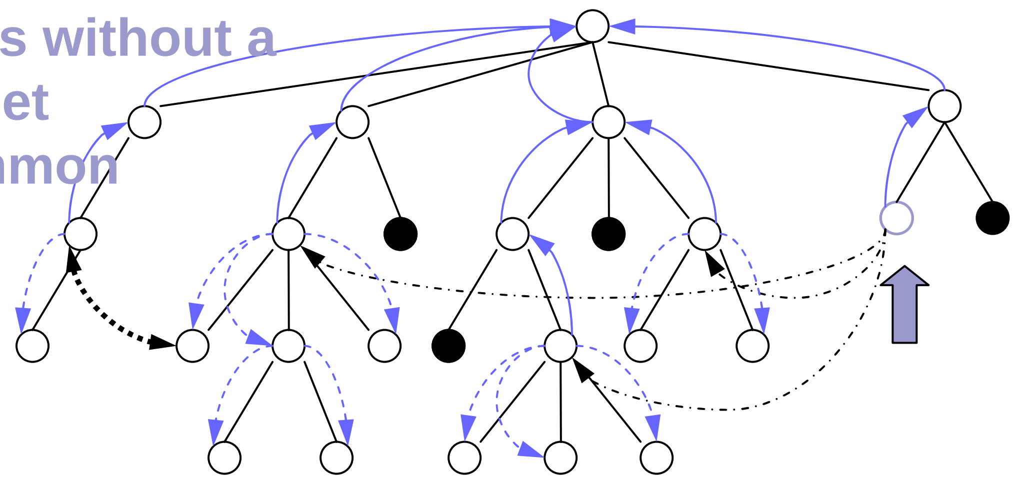
Which Components are relevant for a Configurable Pro



- **Algorithm**
  - Creating a product-specific *segment*
    1. Downwards Traversal of the Partonomy
    2. Comparing Attribute Values

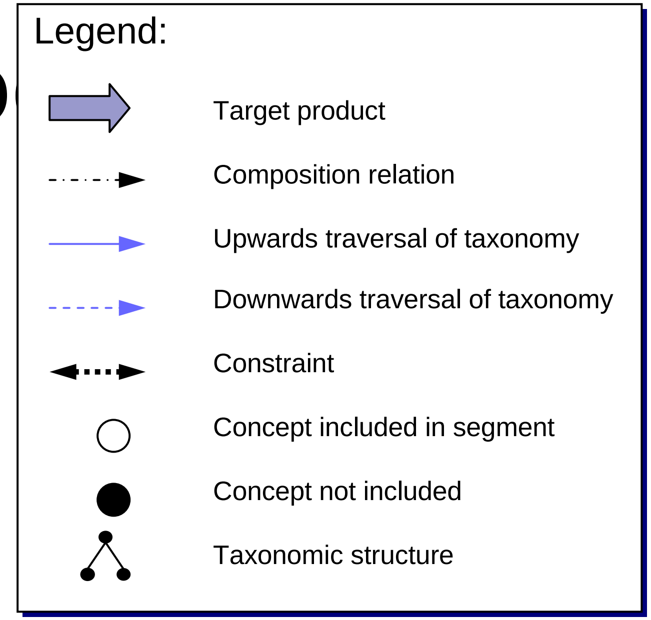
       Filter 1: omit concepts without a common value subset

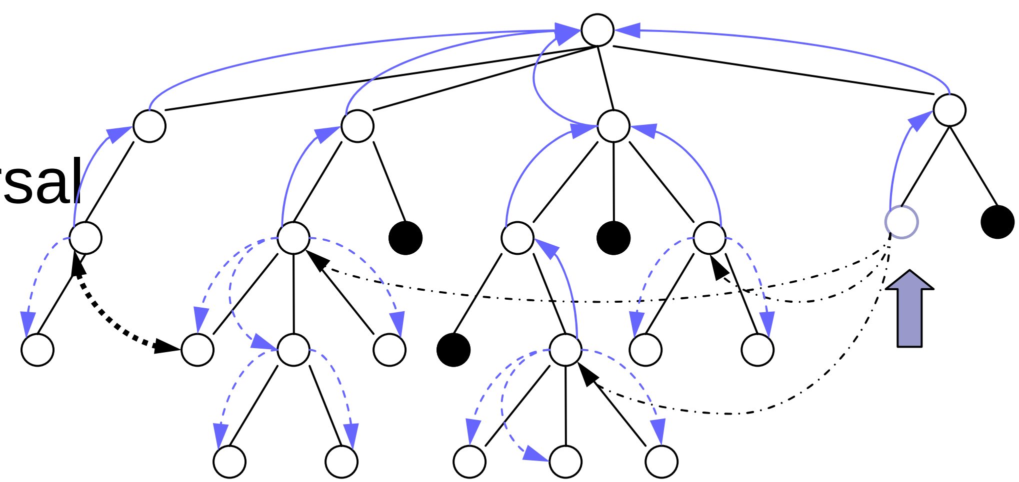
       Filter 2: use only common value subset



Which Components are relevant for a Configurable Pro

- **Algorithm**
  - Creating a product-specific *segment*
  1. Downwards Traversal of the Partonomy
  2. Comparing Attribute Values
  3. Upwards Traversal of the Taxonomy
  4. Downwards Traversal of the Taxonomy
  5. Sibling Concepts
  6. Constraints



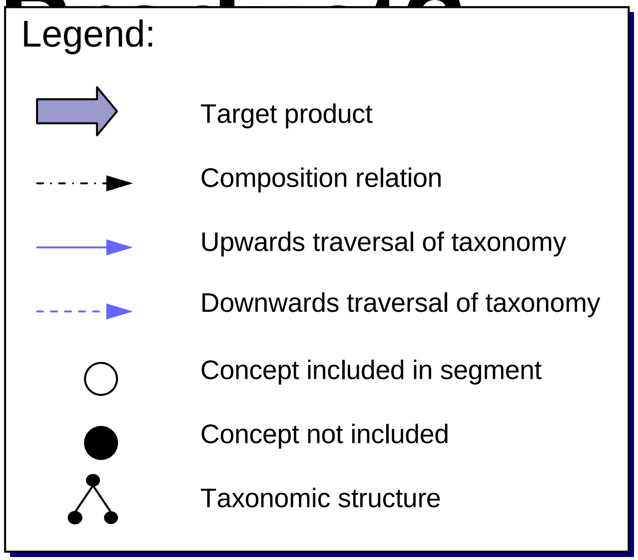


Which Components are relevant for a Configurable Product?

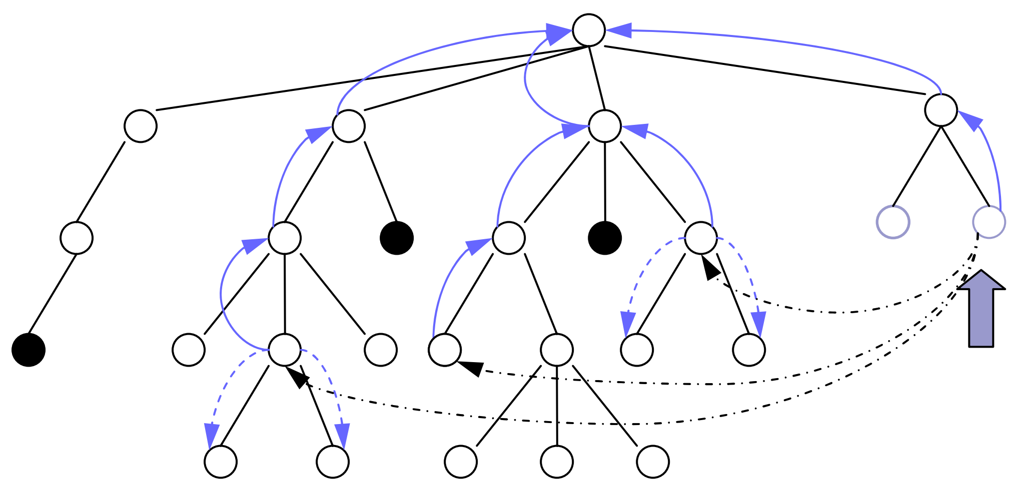
- **Complexity**
  - n concepts in total
  - $k < n$ components
  - $l < k$ product parts
  - $i < k$ max. number of taxonomic levels
  - $j < k$ max. number of children in a taxonomic level
  - $i \times j < n$
  - m number of constraints
  - a max. arity of constraints
  - The worst case complexity is $O(n^2+(a-1)m)$

Contents

Which Components are not relevant for any Conf.



- **Algorithm**
  - Executing the *segmentation* for all configurable products
- **Complexity**
  - *p* configurable products
    - p time the previous complexity
  - The worst case complexity is $O(p(n^2+(a-1)m))$



Contents

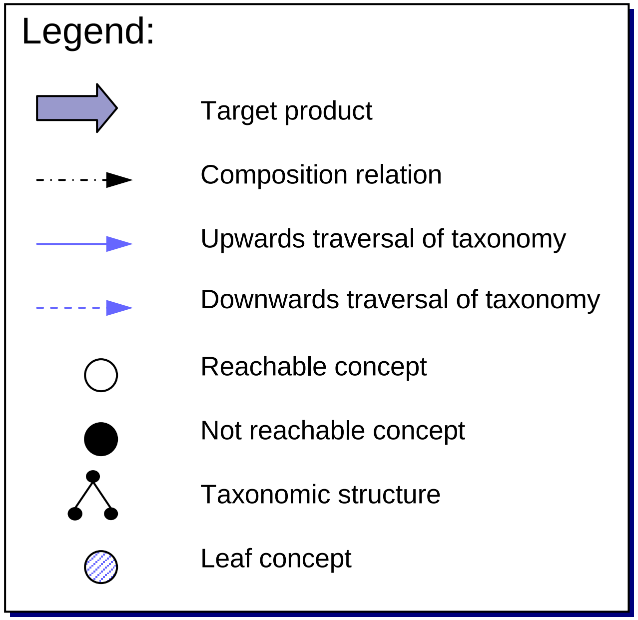
- Introduction
- Knowledge Representation
- Which Components are relevant for a Configurable Product?
- Which Components are not relevant for any Configurable Product?
- **Which Components are “Reachable”?**
  - Algorithm
  - Complexity
- Conclusion

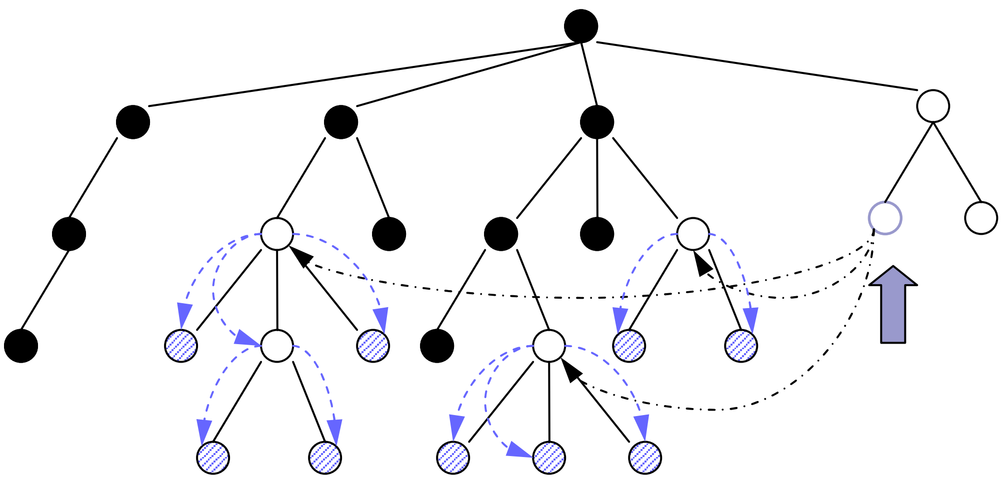
Which Components are "Reachable"?

- **Reachability**
  - A *reachable* concept can in fact be instantiated during product configuration
  - This includes two aspects:
    1. Taxonomy-based reachability
    2. Constraint-based reachability

Which Components are “Reachable”?

- **Algorithm**
  - Taxonomy-based Reachability
    - Downwards Traversal from part concepts
- **Complexity**
  - *l* < *k* product parts
  - *i* < *k* max. levels
  - *j* < *k* max. children
  - *i* x *j* < *k*
  - The worst case complexity is $O(k^2)$





Which Components are "Reachable"?

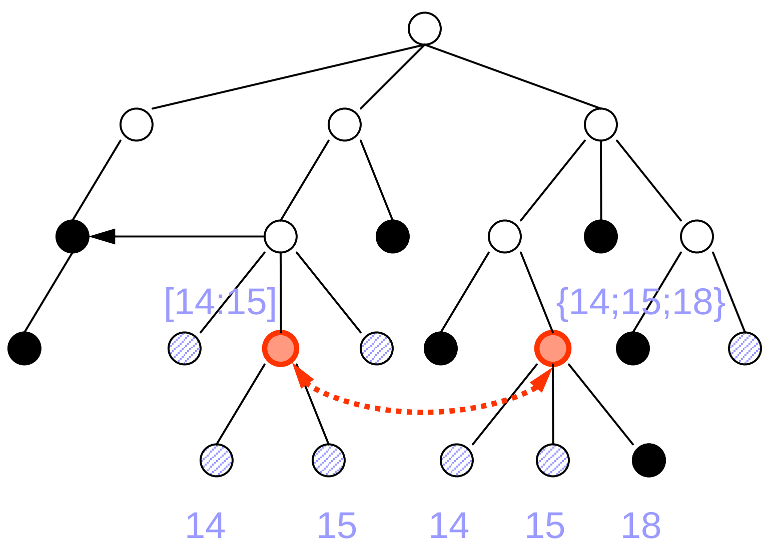
- **Algorithm**
  - Constraint-based Reachability
    - Constraint Satisfaction is known to be NP-hard
    - Approximations are not sufficient
    - Try to get as far as possible from the worst case:
    1. <u>Node Consistency</u>
       - Solving unary constraints first
       - Plays the role of a pre-processor for subsequently solving the constraint net
       - Eliminates local inconsistency that would otherwise be stumbled upon later

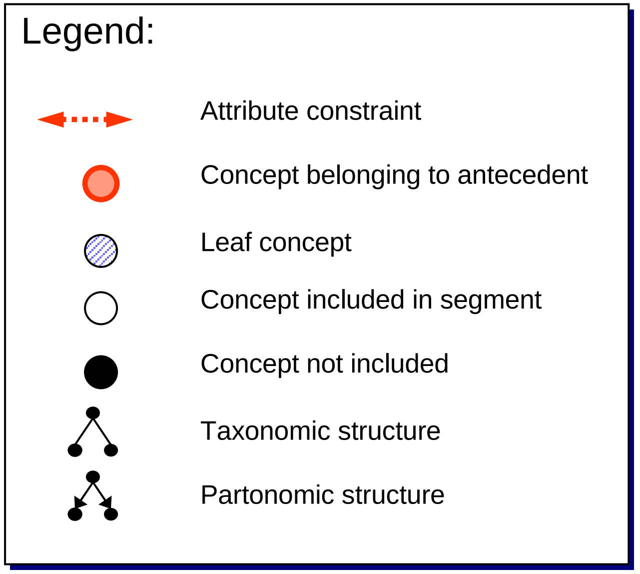
Which Components are "Reachable"?

- **Algorithm**
  - Constraint-based Reachability
    2. <u>Reducing the Search Space</u>
       - Specialization constraints and composition constraints rule out instances
         - Fewer instances of leaf concepts need to be addressed
       - When value ranges are specified:
         - Not all specified property values are covered by leaf concepts
         - Merging property values of leaf concepts to a common value
         - Need not consider infinite value domains
         - When a value is modified, leaf concepts can be pruned!

Which Components are "Reachable"?

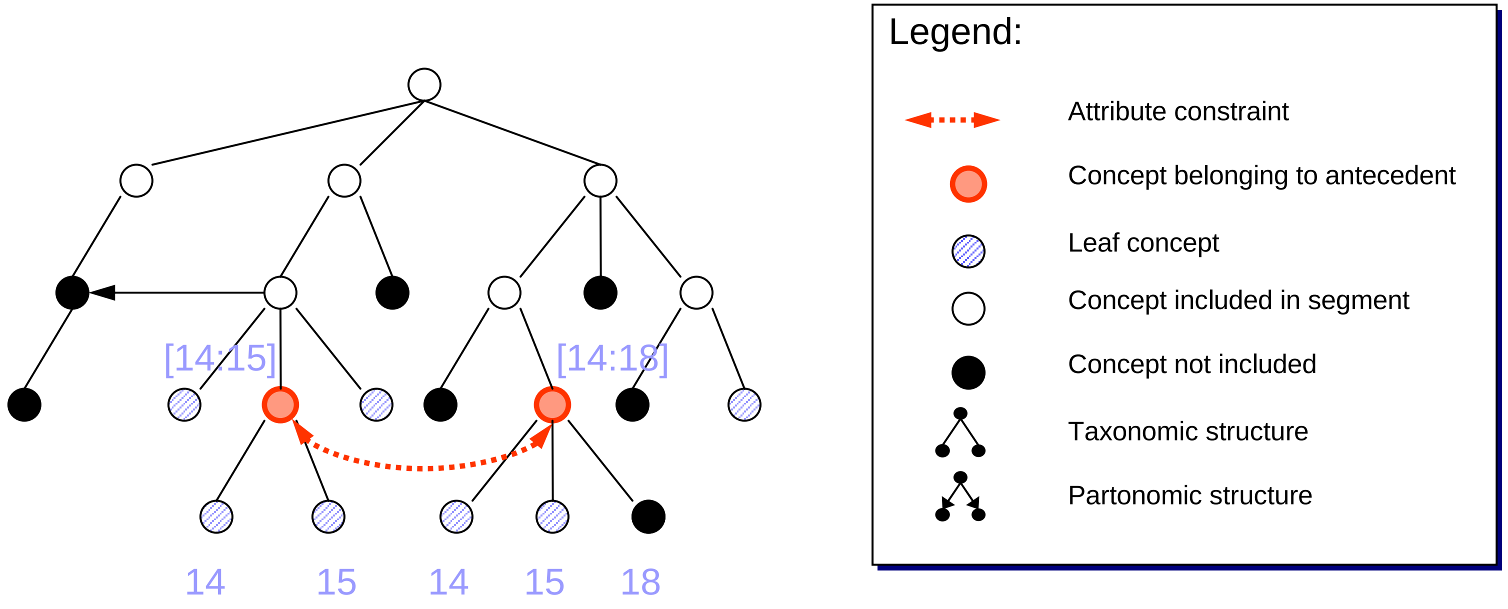
- Example





Which Components are "Reachable"?

- Example



Which Components are "Reachable"?

- **Algorithm**
  - Constraint-based Reachability
    3. Evaluating Constraints on the Conceptual Level
       - The constraint net need not be evaluated for every potential combination of instances – instead:
         - Evaluate for instances of the constraint concepts,
         - Traverse downwards in taxonomy, and
         - Evaluate emerging new constraints
    4. Independent Constraint Subnets
       - The whole constraint net needs to be evaluated once
       - But after that mutually independent subnets can be discarded
- **Complexity**

Contents

Conclusion

- **Proof of Concept**
  - Prototype implementation
  - Managing a configuration model
  - Impacts that changes to components have on configurable products
    - Evaluation
    - Visualization
  - Experiments regarding
    - Reasonable computation time
    - Scalability

Conclusion

- **Experiments Setup**
  - Goals: validate
    - Reasonable computation time
    - Scalability
  - Input data – 3 test models:

Domain	Concepts	Attributes	Compositions	Constraints
CPS (EngCon)	72	107	34	62
Mercedes (LiMEd)	115	134	49	8
Sartorius (EngCon)	805	6794	222	145

Conclusion

- **Experiment Results**

Measurement	Average	Min	Max
Execute Change	1.966	0.371	6.177
Validate Product	35.850	7.786	159.114
- **Validate Conceptually**	10.072	6.714	15.921
- **Validate Constraints**	38.548	2.869	153.053

Configuration Model	Average	Min	Max
CPS	0.235	0.187	0.429
Mercedes	0.242	0.216	0.445
Sartorius	0.191	0.100	0.594

Conclusion

- **Experiment Results**
  - Reasonable computation time
    - Executing a change takes between 0.1 and 25 ms
    - Validating product consistency takes between 8 and 160 ms
  - Scalability
    - Execution time does not increase according to size
    - But only according to complexity of knowledge
- **Yet to be evaluated**
  - Comparing
    - Simple G&T constraint satisfaction algorithm
    - Constraint satisfaction algorithm incl. defined improvements

Conclusion

- **Summary**
  - Knowledge management framework that supports the evolution of configurable products
  - Consistency-preserving evolution process
  - Product-centered approach
    - Semantic differentiation between component representation and product representation
    - Impacts that changes to components have on products
  - Experiments with prototype show feasibility