

Document Engineering's Future and What it Means for the Web

Ethan V. Munson

University of Wisconsin-Milwaukee

ACM SIGWEB

Outline

- Documents and Document Engineering
- Document Technology and the Web
- Key Research Directions
- Example: Versioning for Software Product Line

Documents

- Documents are:
 - “Representations of information designed for consumption by people.”
 - Persistent or ephemeral
 - Text often dominant, but any medium is possible
- The human aspect is central to their design

Document Engineering

- Document engineering is the field that studies the creation, manipulation and distribution of documents
- Subdomains include:
 - Multimedia documents
 - Document analysis
 - Document representation and authoring
 - Document management

3 Foundations of the Web



HTML

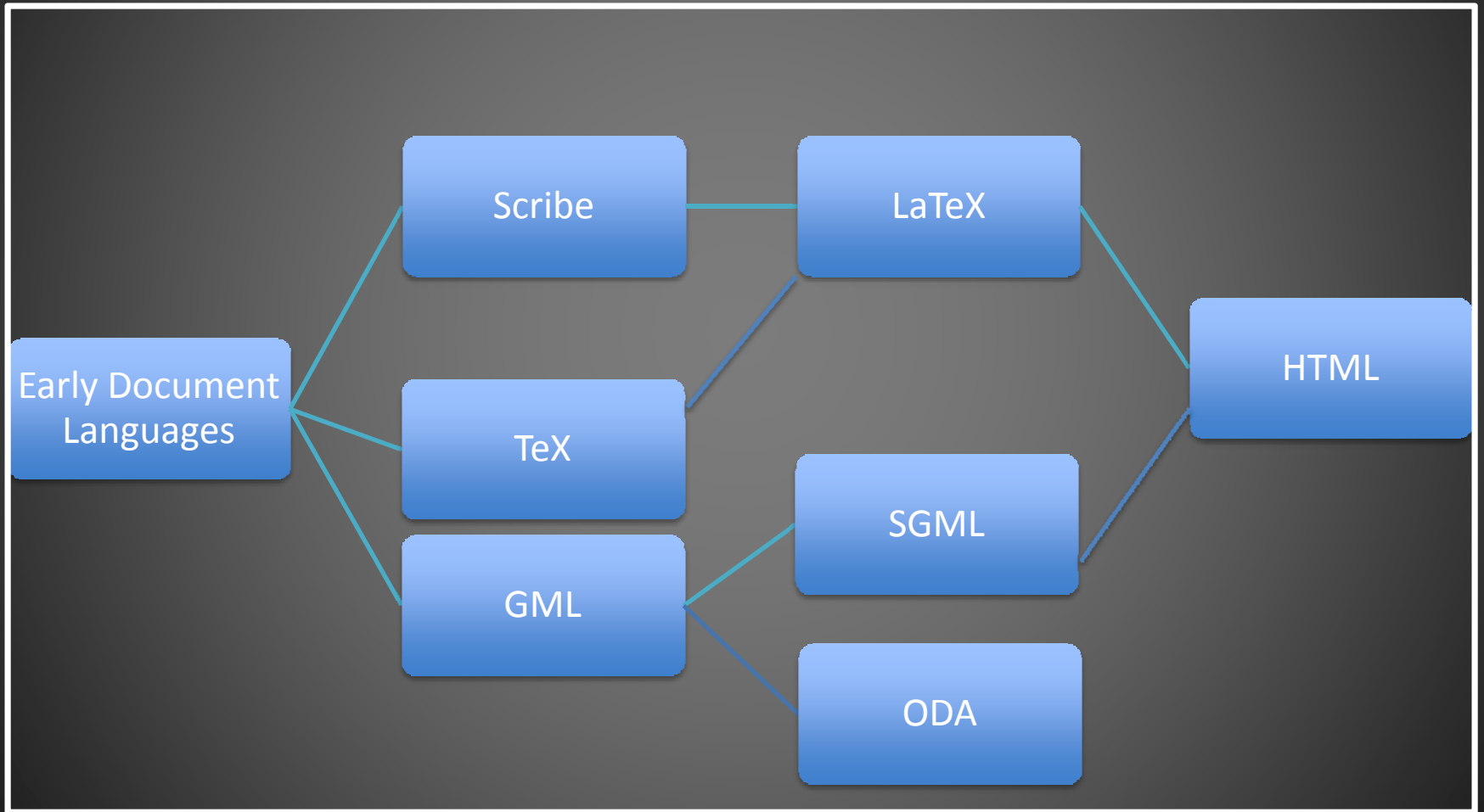
URL

HTTP

What made HTML successful?

- Human-readable
- Simple syntax for humans
 - Acceptable complexity for machines
 - Error tolerant
- Enough structure for its application
 - Typed elements and useful formatting effects
 - Simple hypertext model
- Easily extended with new elements and attributes

HTML's Family Tree



The Web's Gang of Three

- HTML, CSS and XML
- All three are:
 - Human-readable
 - Syntactically simple (but not always elegant)
 - Extensible and error-tolerant
- XML, in particular, supports solutions whose complexity matches the complexity of the problem
 - While freeing us from compiler design problems

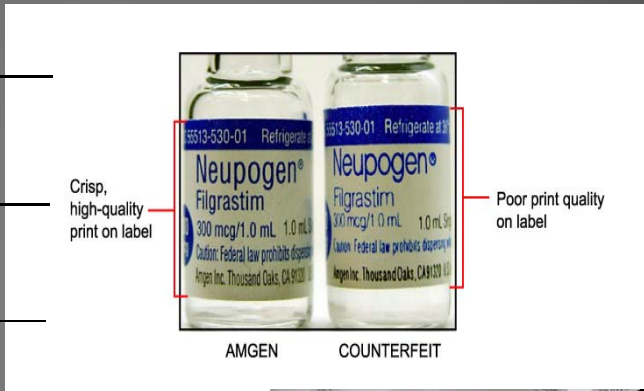
The Success of XML

- XML is generally not used as anticipated
 - XML documents are not published directly
- But XML data has become pervasive, e.g.
 - Dialects: SMIL, SVG, GML, MathML
 - SOAP used to support Remote Procedure Call
 - ad hoc XML dialects
- No one envisioned this

Current DocEng Research

- Security printing and variable-data printing
- Hybrid document representations
- Supporting diverse devices
- Document versioning

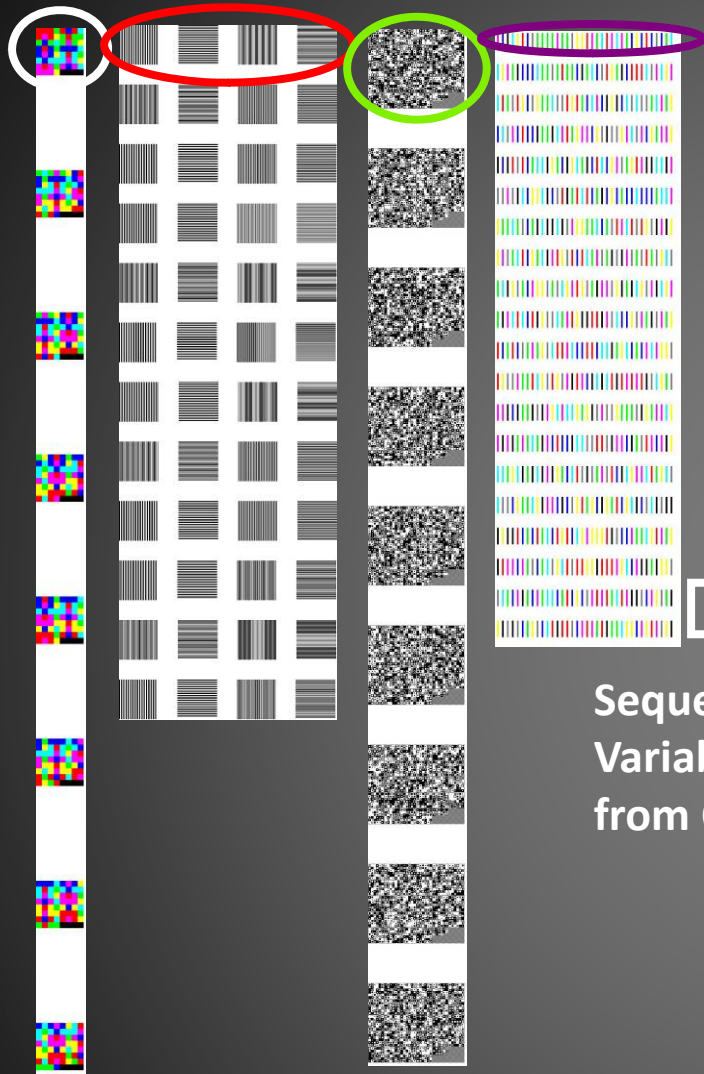
Counterfeiters Are Sophisticated and Have Large R&D Budget (no need for QA and Marketing!)

	Type	Description	Example
Fake Drugs	Drugs without APIs	<ul style="list-style-type: none"> Drugs missing active ingredients 	<ul style="list-style-type: none"> Neupogen, a cancer drug, containing only saline solution (2001) Fake Viagra (considered to be the most counterfeited drug) Combivir labels placed on Ziagen tablets and vice versa (2002)
	Diluted Drugs	<ul style="list-style-type: none"> Diluted products 	
	Accurate Knock-offs	<ul style="list-style-type: none"> Drugs with accurate compositions made through reverse engineering 	
	Contaminated Drugs	<ul style="list-style-type: none"> Drugs with unintentional, lethal impurities Drugs with intentional, lethal contaminants 	
Fake Labels	Fake Labels	<ul style="list-style-type: none"> Labels with wrong drug name Labels misrepresenting product potency Labels extending the expiration dates 	
	Diverted	<ul style="list-style-type: none"> Parallel trading due to differential pricing and/or drug shortages in certain regions <ul style="list-style-type: none"> drugs intended for export to foreign charitable organizations samples 	

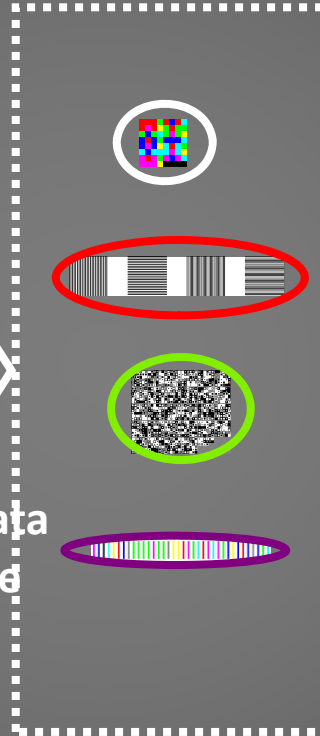
Solution: Variable-Data Printing

- VDP mixes traditional fixed content with regions of varying material
- Challenge: high-speed production when every item (package, page) is somewhat different
 - “High-speed” means “faster than 1 per second”
- Requires an expensive digital press

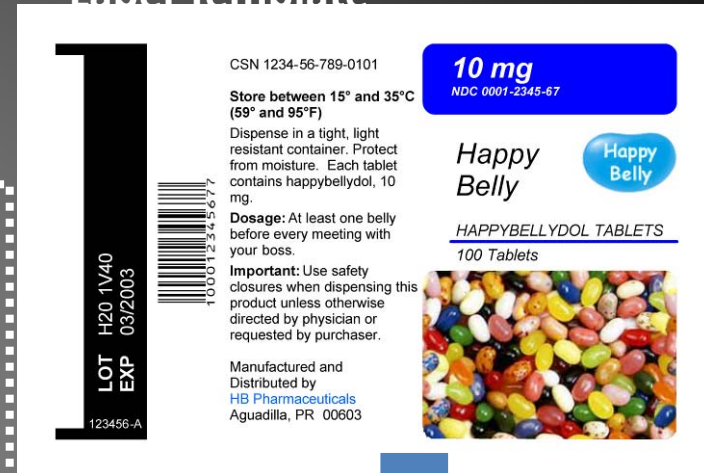
Security Variable Data Printing is:



Sequence
Variable Data
from Queue



Label Template



Merge



Variable Data Security Marks

Security VDP Label

Recent Research

- Lumley et al (HP): Document Description Framework
 - Uses functional programming techniques to represent partially evaluated layouts
 - Allows flexible recomposition in various contexts
- Harrington et al (Xerox): Metrics for aesthetics in document layout
 - Used to drive optimization algorithms for VDP

Applications to the Web?

- Uses of VDP for brand/product protection are very pragmatic
 - Goal is loss reduction, not absolute prevention
 - Can a similar pragmatism help the Web?
- Can physical documents be used to help validate Web sites?
- Can data make the “round trip” from virtual to print to virtual?

Hybrid document representations

Hybrid document representations

- Source form and final form are considered distinct
- Final form (PDF) has advantages
 - Less mutable
 - Less device dependent
- And disadvantages
 - Hard to reuse or repurpose
 - Suffers from information loss

Enriching PDF

- Tagged PDF: an existing, under-used feature
 - Document chunks are labeled with structure tags
 - Allows semantic search and document analysis
- Component Object Graphics (COGs) (Bagley, MacDonald et al, Nottingham U.)
 - Independent chunks of PDF that can be rearranged or repurposed
 - Object independence is a hard problem in printer languages

Other research

- Balinsky: extracts structure that is implicit in documents (fonts, spacing) to support automated repurposing

Supporting diverse devices

Device adaptation

- Documents are viewed on diverse devices
 - HTML was one approach to dealing with this
 - And the problem is worse now
- Devices differ in
 - Display type
 - Processor performance
 - Network bandwidth
- Authors can't support all combinations

Current Research

- SMIL State (Bulterman and Jannsen): adds support for interaction to SMIL and SVG presentations
- Scalable MSTI (Pellan and Concolato): allows progressive representations on multiple axes (Spatial, Temporal, Interactive)
- Marriott et al: Flexible, constraint-based formatting for wildly varying screen sizes and aspect ratios

Document versioning

Documents are alive

- Few documents are truly static
- Versioning technology exists
 - RCS, CVS, Subversion
- But it has failed to serve non-technical users
 - Evolving documents are fundamentally complex
 - Documents are rarely standalone objects
 - Failure to represent variants
 - Versioning systems don't support end-user models

Software Product Line (SPL)

- Promising software development paradigm
- Intended to improve product variability management
- Uses a manufacturing analogy
 - Interchangeable parts (components)
 - Assembled into final products
- Used by some electronics and telecom vendors

Product Derivation

- Construction of a software product from a base set of core assets
 - Selecting, pruning, extending, and modifying copies of the core assets
- Everything is evolving
 - Both core assets and derived products
- Change propagation would be a good thing
 - Forward: from core assets to products
 - Backward: from products to core assets

Services required for SPL

- Component selection
 - Specify components to use/reuse in products
 - Adding product-specific components
- Overriding components
 - Replace provided components with alternative implementation
- Managing modified components
- Maintaining derivation relations
- Modifying product architectures

Related Work

- Versioning tools (RCS, CVS, Subversion)
 - Designed to manage single product projects
 - No support for change propagation
 - Variants resemble products in SPL
- SPL Tools
 - Generally, they treat components as unvarying
 - Focus on managing build process or on documenting variation

Two SPL Approaches

- Koala: requires all new component versions to be backward compatible
 - Components are developed separately and configuration managed independently
- Krueger: Evolution is managed entirely on core assets
 - Products are regenerated when changes are made to the components
 - No product specific code

Molhado

- Uses the Fluid Persistence Model
- Supports version-aware editing of Java, XML, UML, and other representations
- Structure-oriented: not fundamentally based on files and lines of text
- Product versioning: a single change creates a new version of entire project
 - Better human model
 - Efficient version differencing

Molhado SPL

- MoSPL provides:
 - Support for required services
 - Component creation, maintenance, and override
 - Derivation management
 - Relationship management
 - selection rules for components
 - representing derivation
 - Propagation of changes
 - Merge and conflict resolution

Versioned Data Model

- Implements the product versioning model
- Intermediate Representation (IR)
 - Node : a unique identity
 - Attribute : mapping from nodes to slots
 - Slot : storage location
 - Version: point in discrete time

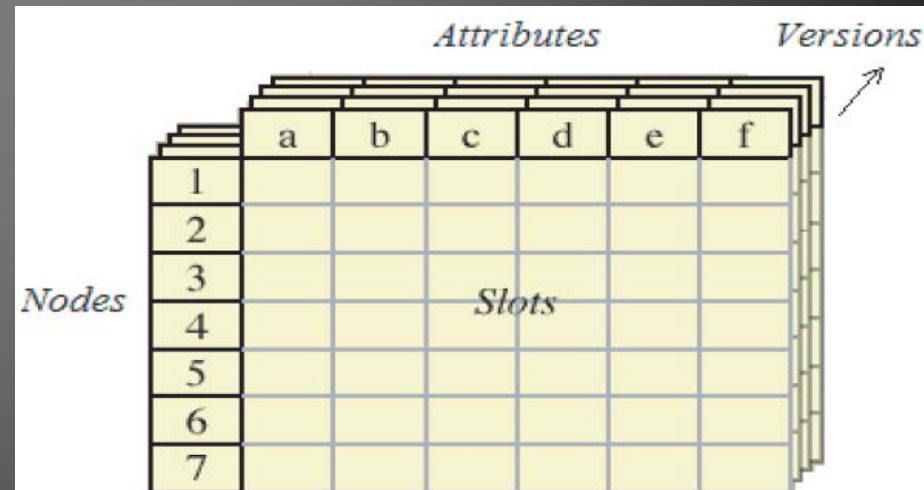
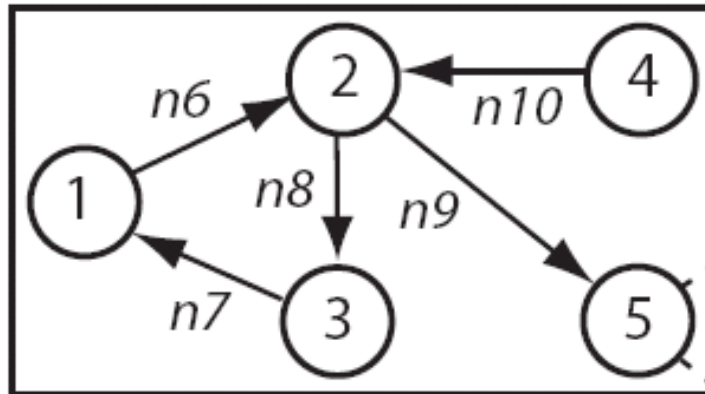


Figure 5. Data Model

Structured versioning representation

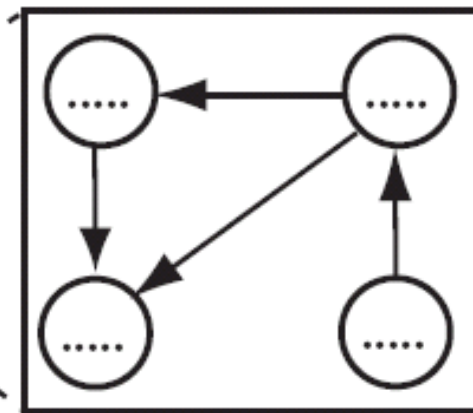
- Component represented by a directed graph
- Use *edge table* for representing directed graph
 - Each IR node representing an edge has
 - *Source* slot
 - *Sink* slot
 - Each IR node representing a node has
 - *Children* slot which identifies its outgoing edges
 - *Ref* attribute refers to nested component
- Common structures are shared among versions

Initial Version



Component C

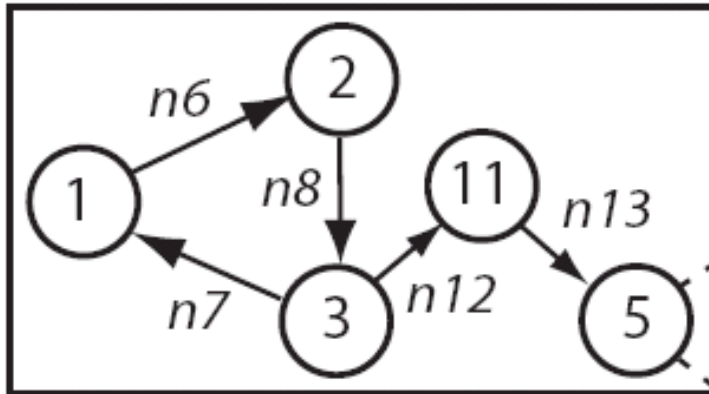
Component A



Attribute table for C

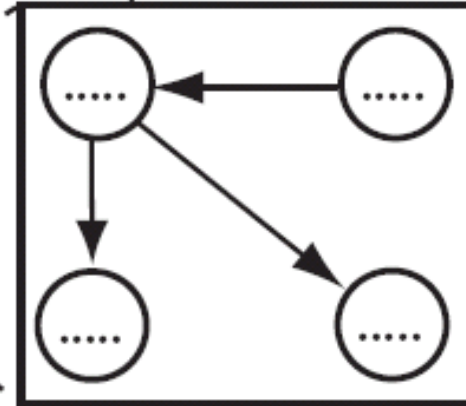
node	"type"	"source"	"sink"	"children"	"ref"	"attr1"
n1	node	undef	undef	[n6]	null	
n2	node	undef	undef	[n8,n9]	null	
n5	node	undef	undef	null	comp_A	
n6	edge	n1	n2	undef	null	

After Modification



New version of component C

New version of component A



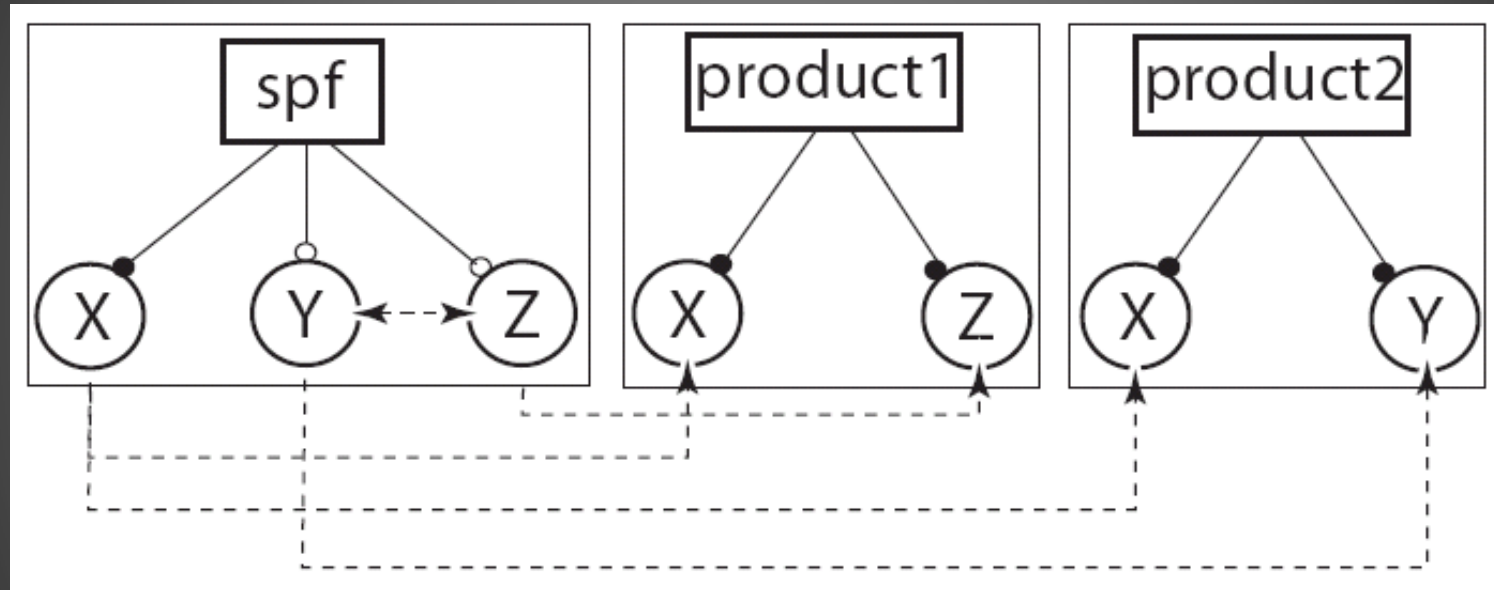
Attribute table for C

node	"type"	"source"	"sink"	"children"	"ref"	"attr1"
n1	<i>node</i>	<i>undef</i>	<i>undef</i>	[n6]	null	
n4	<i>undef</i>	<i>undef</i>	<i>undef</i>	<i>undef</i>	<i>undef</i>	<i>undef</i>	
n11	<i>node</i>	<i>undef</i>	<i>undef</i>	[n13]	null	
n12	<i>edge</i>	n3	n11	<i>undef</i>	null	
n13	<i>edge</i>	n11	n5	<i>undef</i>	null	

Relationship Model

- Adopted feature model from Benavides et al.
- Constraints between components
 - Mandatory
 - Optional
 - Or
 - Alternative
 - Implies
 - Excludes

Motivating Example (v1)



Legends:

—● mandatory

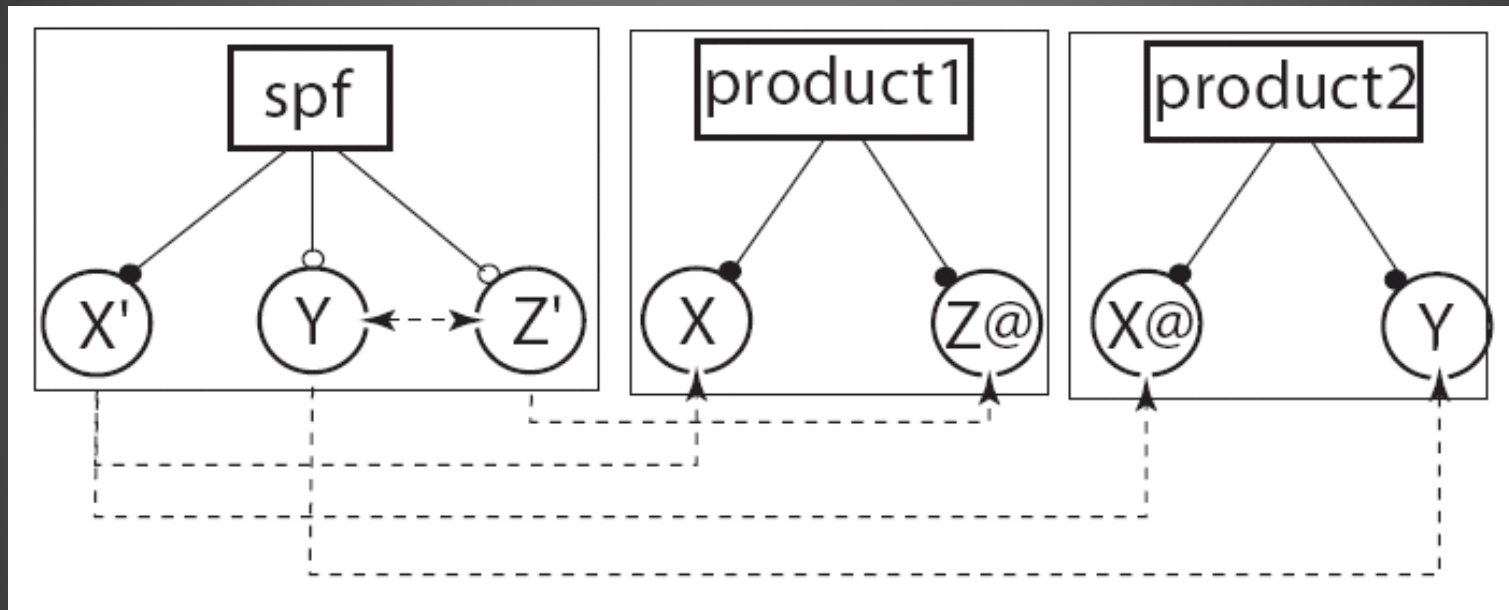
—○ optional

-----> derivation

<-----> exclusion

====> dependence

Motivating Example (v2)



Legends:

—● mandatory

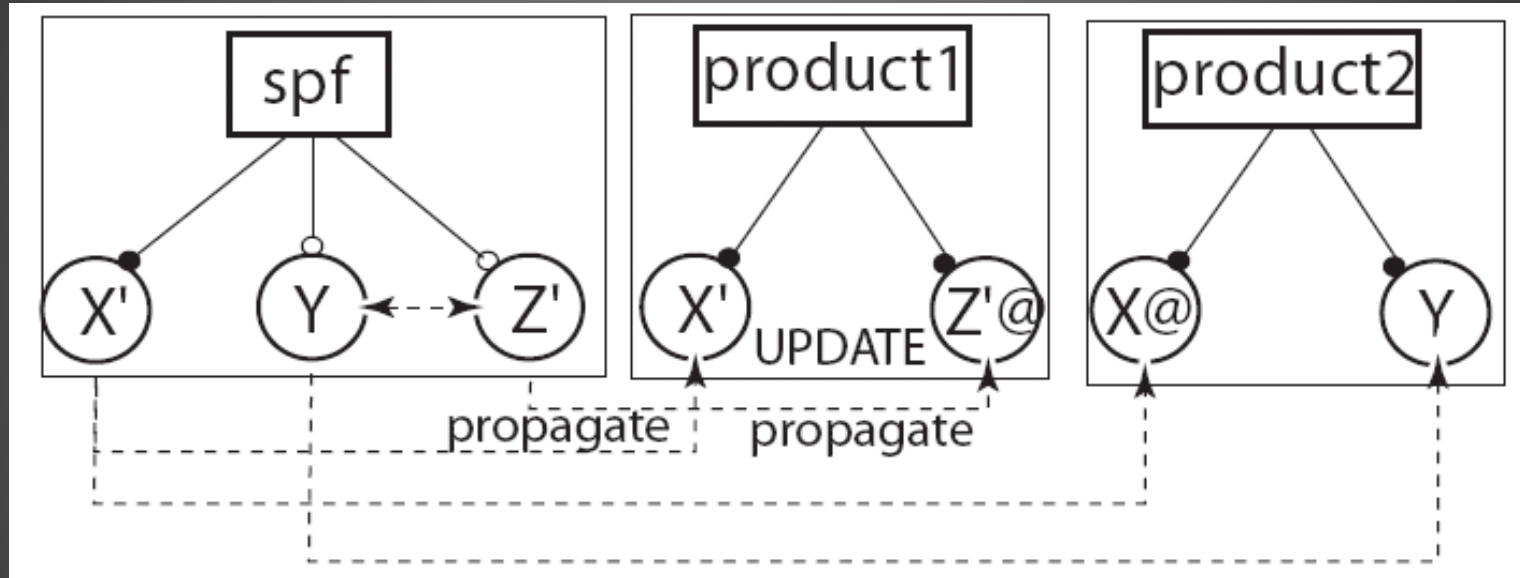
—○ optional

-----> derivation

<-----> exclusion

====> dependence

Motivating Example (v3)



Legends:

—● mandatory

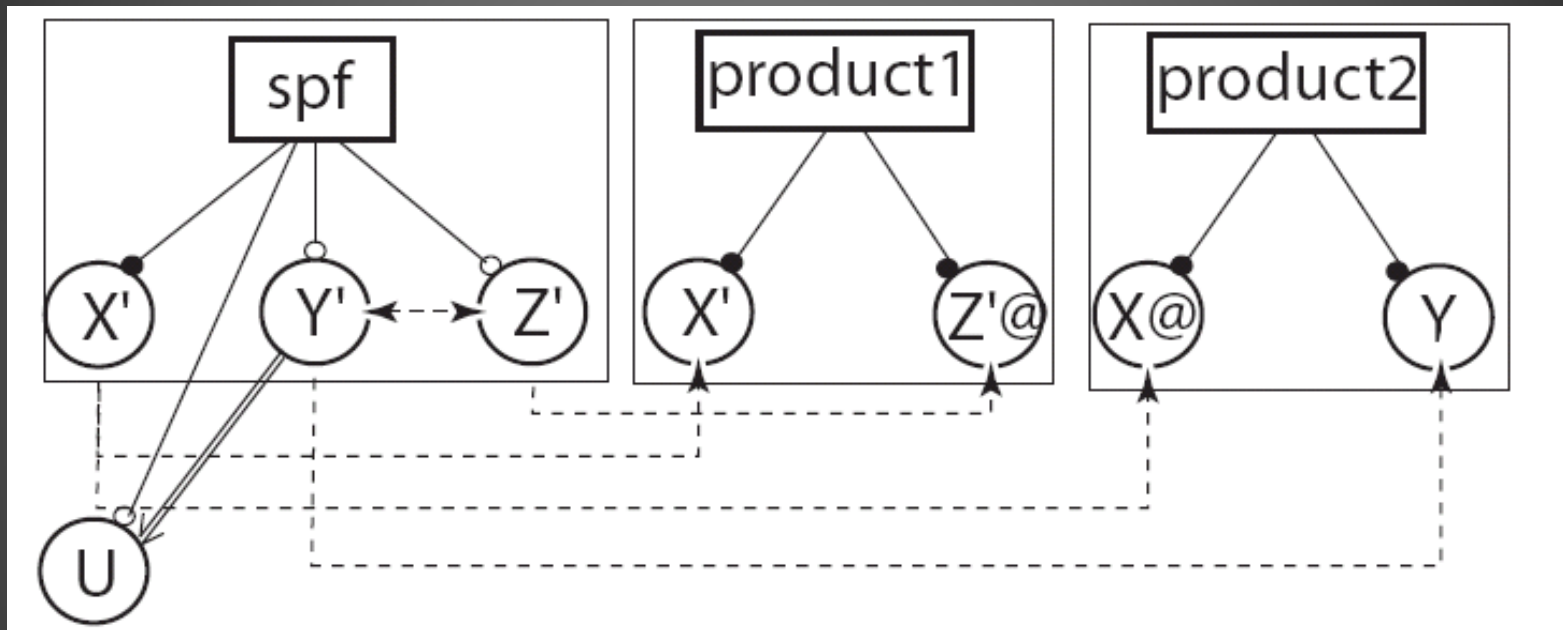
—○ optional

-----> derivation

<-----> exclusion

====> dependence

Example Revisit (v4)



Legends:

—● mandatory

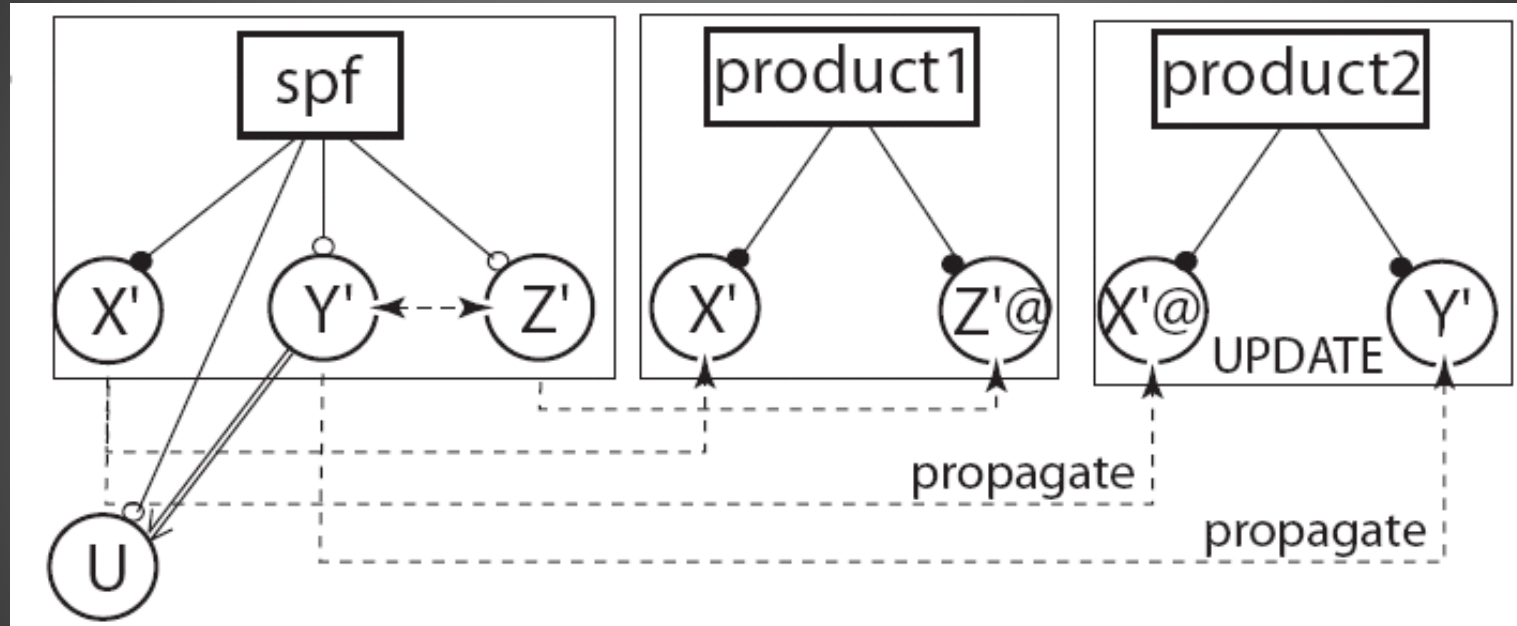
—○ optional

-----> derivation

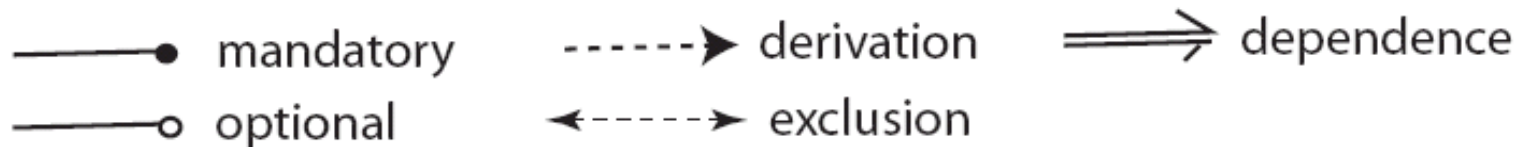
<-----> exclusion

====> dependence

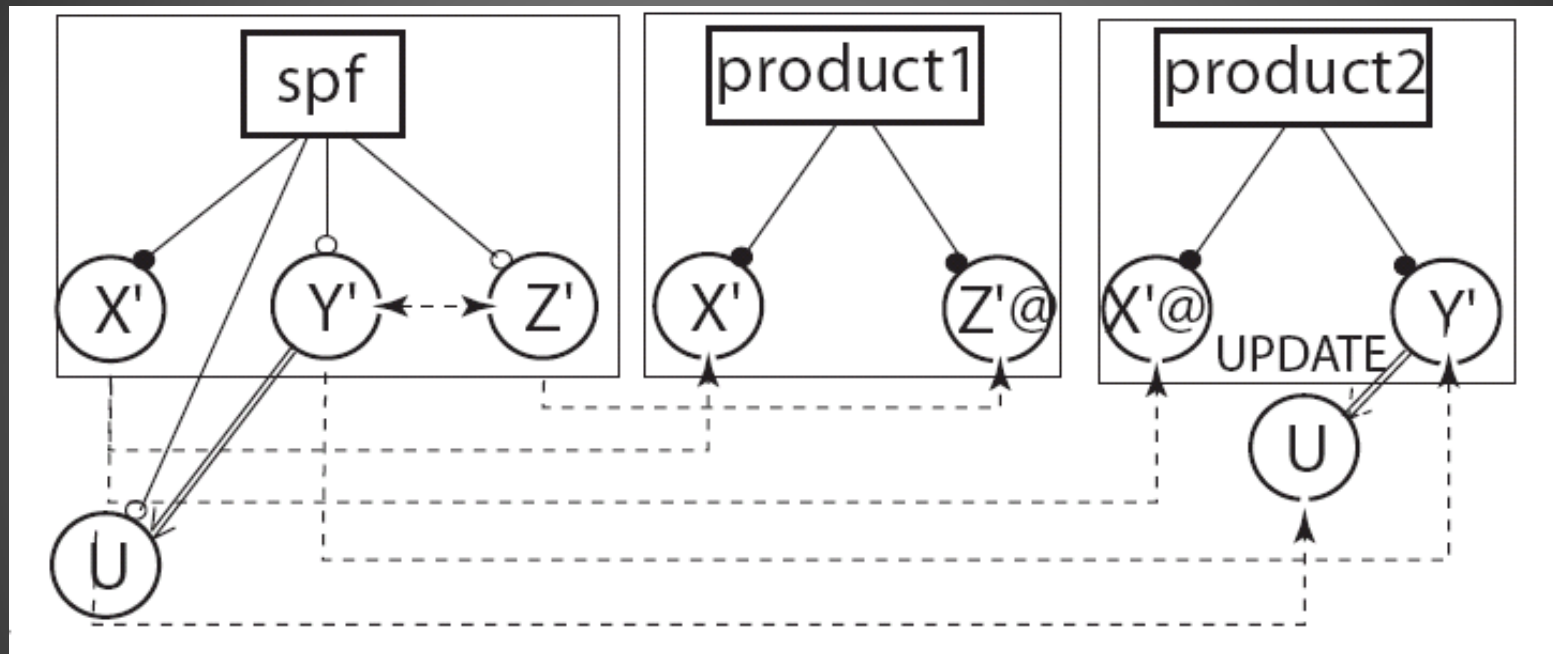
Motivating Example (v5)



Legends:



Motivating Example (v6)



Legends:

—●— mandatory

—○— optional

-----> derivation

<-----> exclusion

====> dependence

Implementation

- Implemented in Java reusing and extending the Molhado framework
- Supports direct editing on components and relationships
- Components are mapped to documents, code, packages
- Tracks shared components through derivation relations
- Maintains constraint relationships

Prototype Snapshot

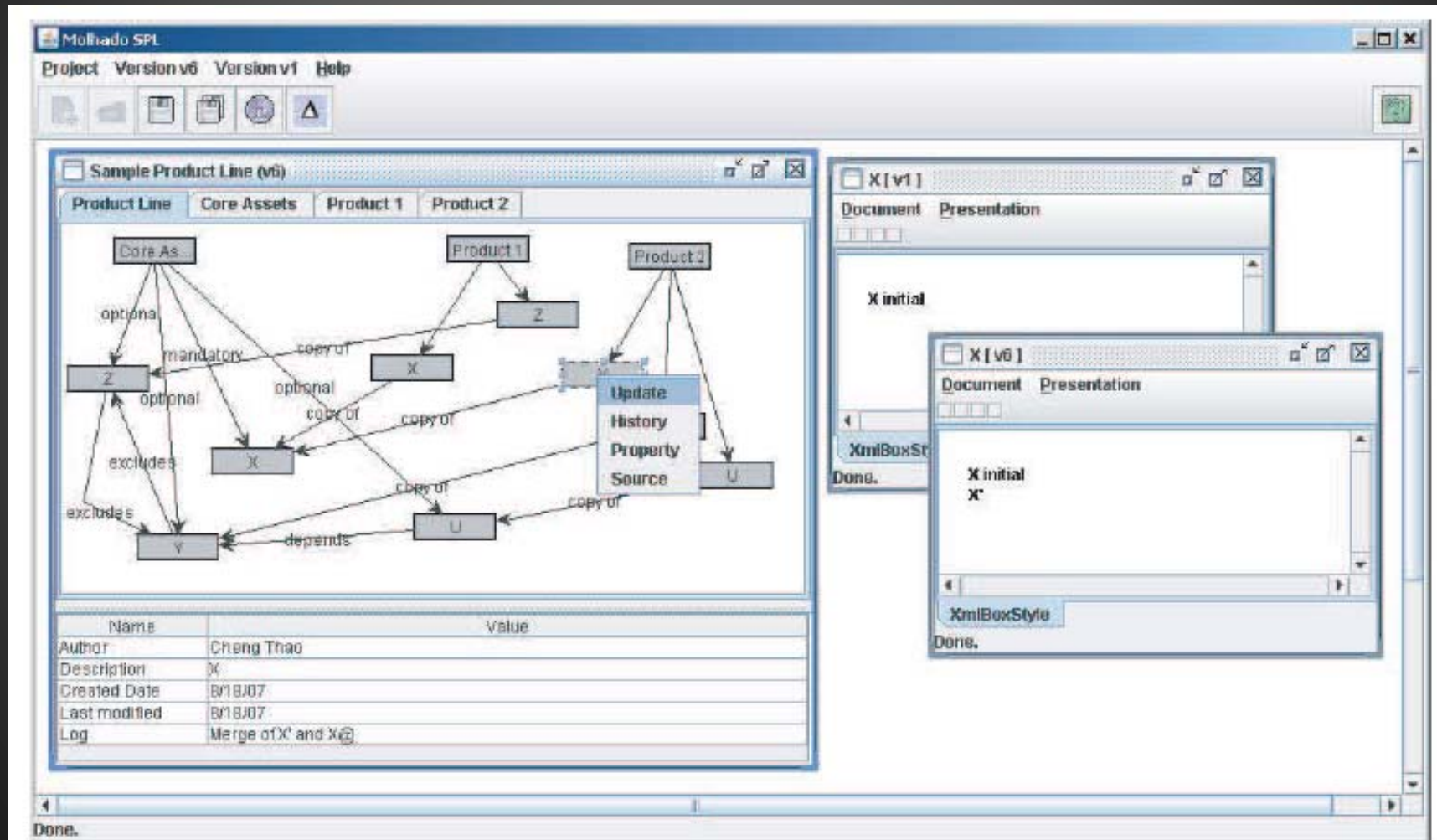


Figure 9. MoSPL Snapshot

Research Directions

- Current implementation is quirky and brittle
 - In-memory data structures limit scalability
 - Odd performance bottlenecks
- Continuing work on supporting SPL
- Starting new project based on standard database technology

And what about the Web?

- Many classes of documents have both variants and versions
 - People and organizations have trouble managing them
- A solution for SPL might well serve for other domains
 - eGovernment
 - Medical records
 - Scholarly documents

Questions?

Ethan Munson

University of Wisconsin-Milwaukee

munson@uwm.edu