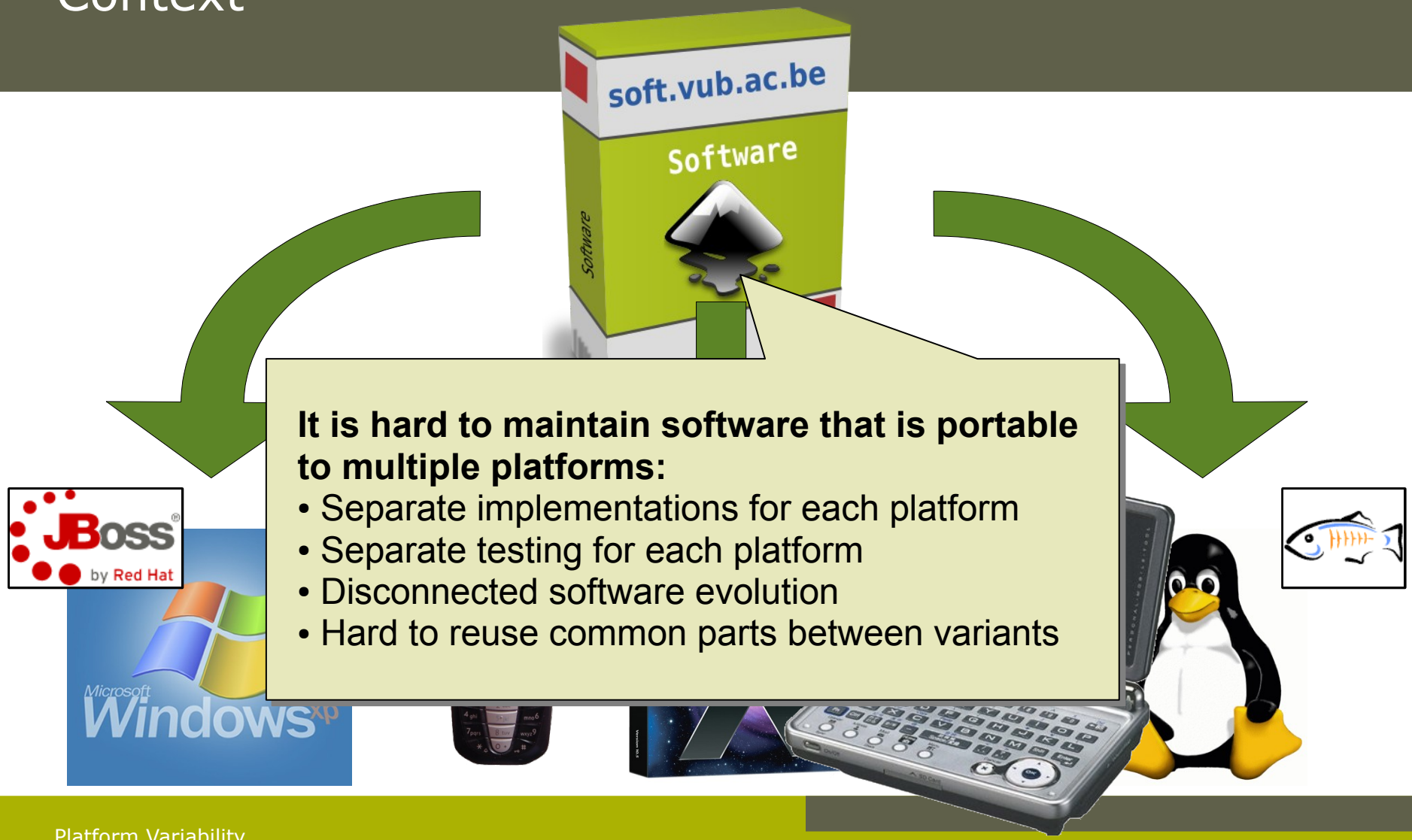
Platform Variability

Dennis Wagelaar
Software Languages Lab
✉ dennis.wagelaar@vub.ac.be

Vrije Universiteit Brussel

Platform Variability:
Context

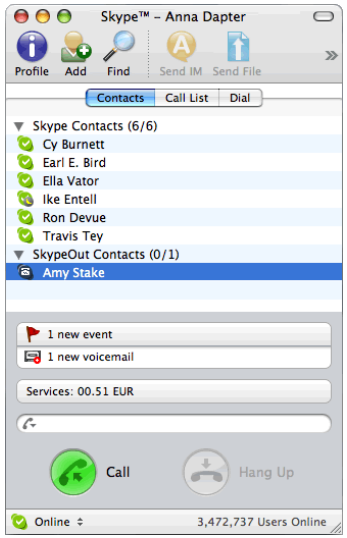
Platform Variability:
Context

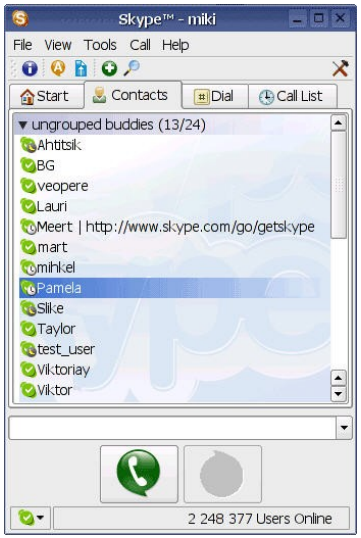


It is hard to maintain software that is portable to multiple platforms:

- Separate implementations for each platform
- Separate testing for each platform
- Disconnected software evolution
- Hard to reuse common parts between variants

Platform Variability example:
Skype (at a certain point in time)





- Audio/Video
- Conference
- SkypeCast
- PublicChat

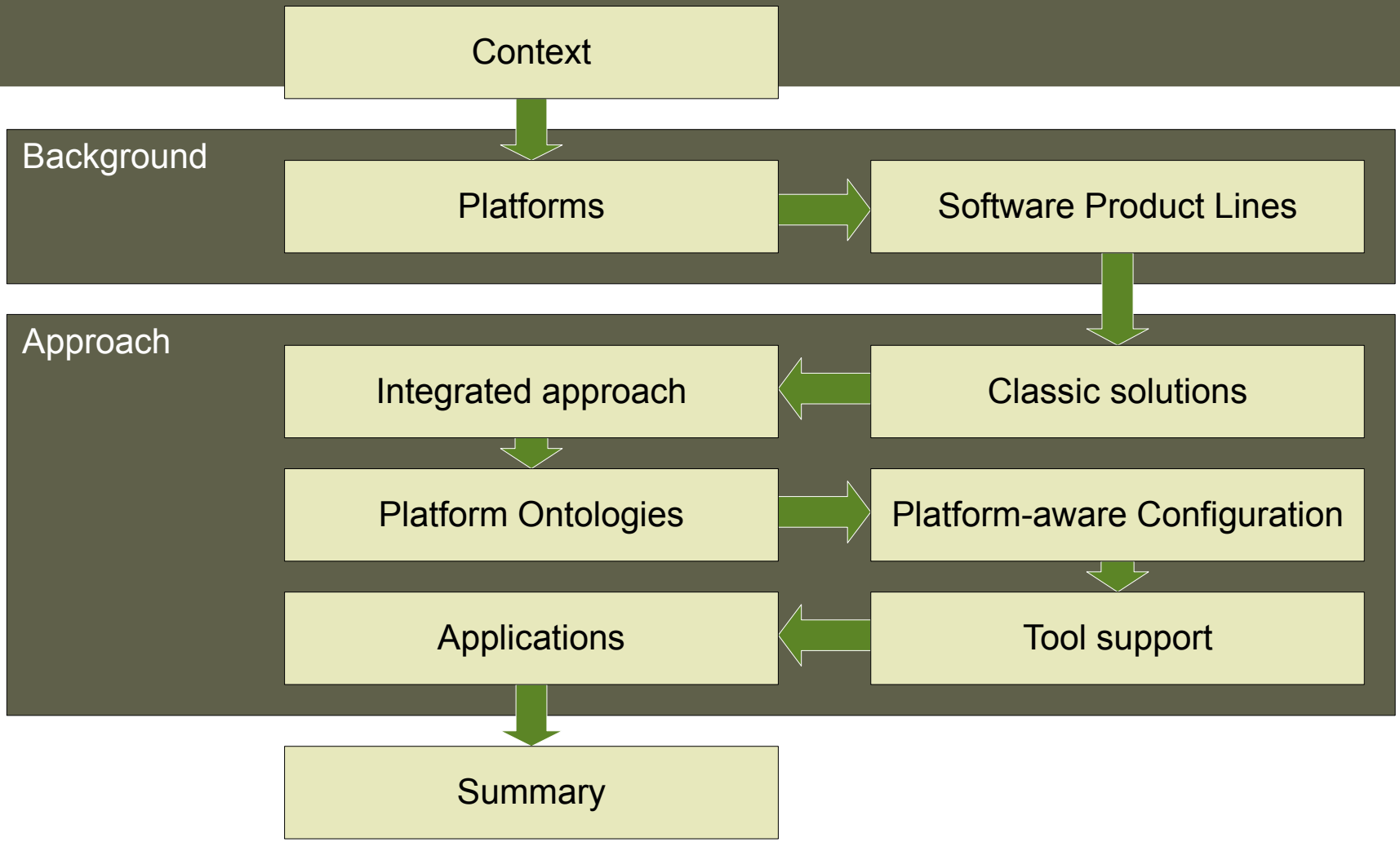
- Audio/video
- Conference

- Only audio

Platform Variability example:
Skype Mobile (at a certain point in time)

→Only audio

Outline



Platforms:
What is a platform?

- What is meant by “platform” in general?
  - “A platform is any base of technologies on which other technologies or processes are built.” [Pohl et al. 2005]
- More specific:
  - A platform is a combination of visible hardware and/or software technologies on top of which other hardware and/or software is built.
- Examples:
  - Java, Windows, WinTel, ARM, EJB, Eclipse, ...

Platforms:
What is the purpose of a platform?

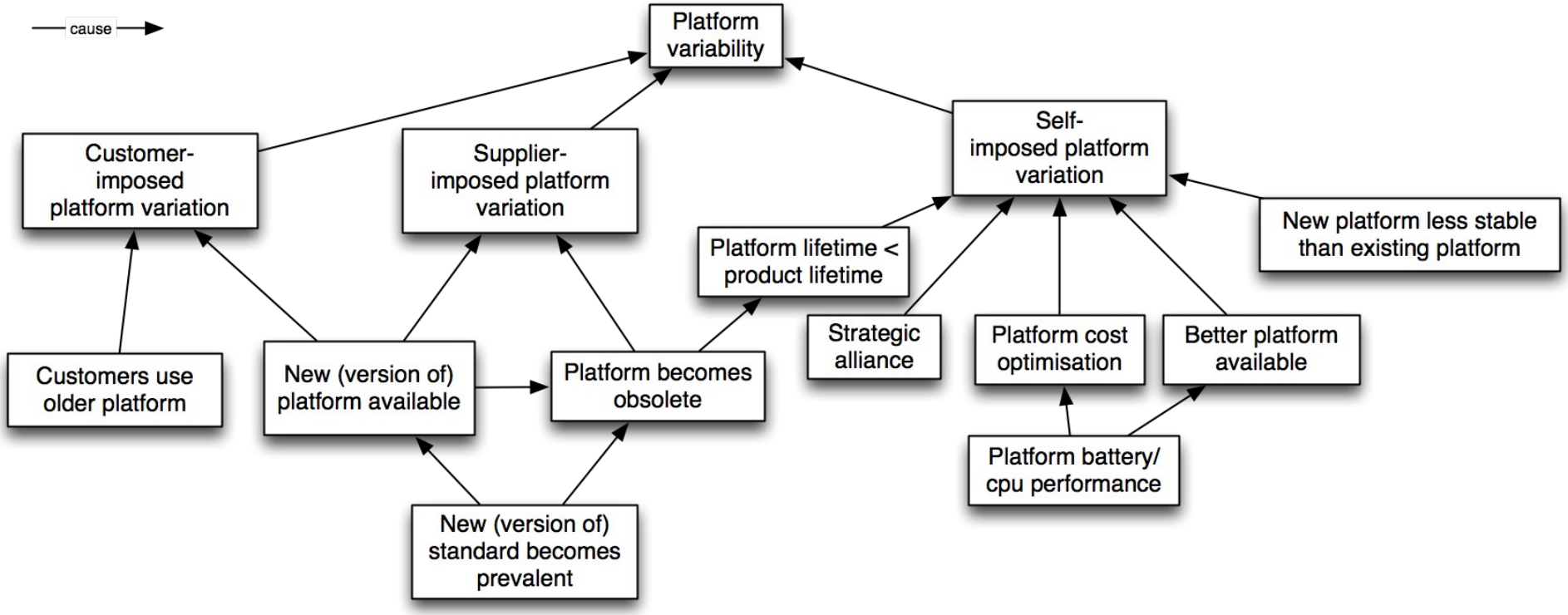
- Platforms originate from:
  - A desire to "wrap" technologies, meant to be used as a part of an end-user technology, as a standardised/stable platform.
  - In Software Product Lines, a desire to organise core assets of the product line into a platform that can safely be reused in all of the derived products.
- So the purpose is:
  - To provide a sound and stable basis for technology reuse.
  - To provide a repository in which to put reusable technology.

Platforms:
Why vary the platform?

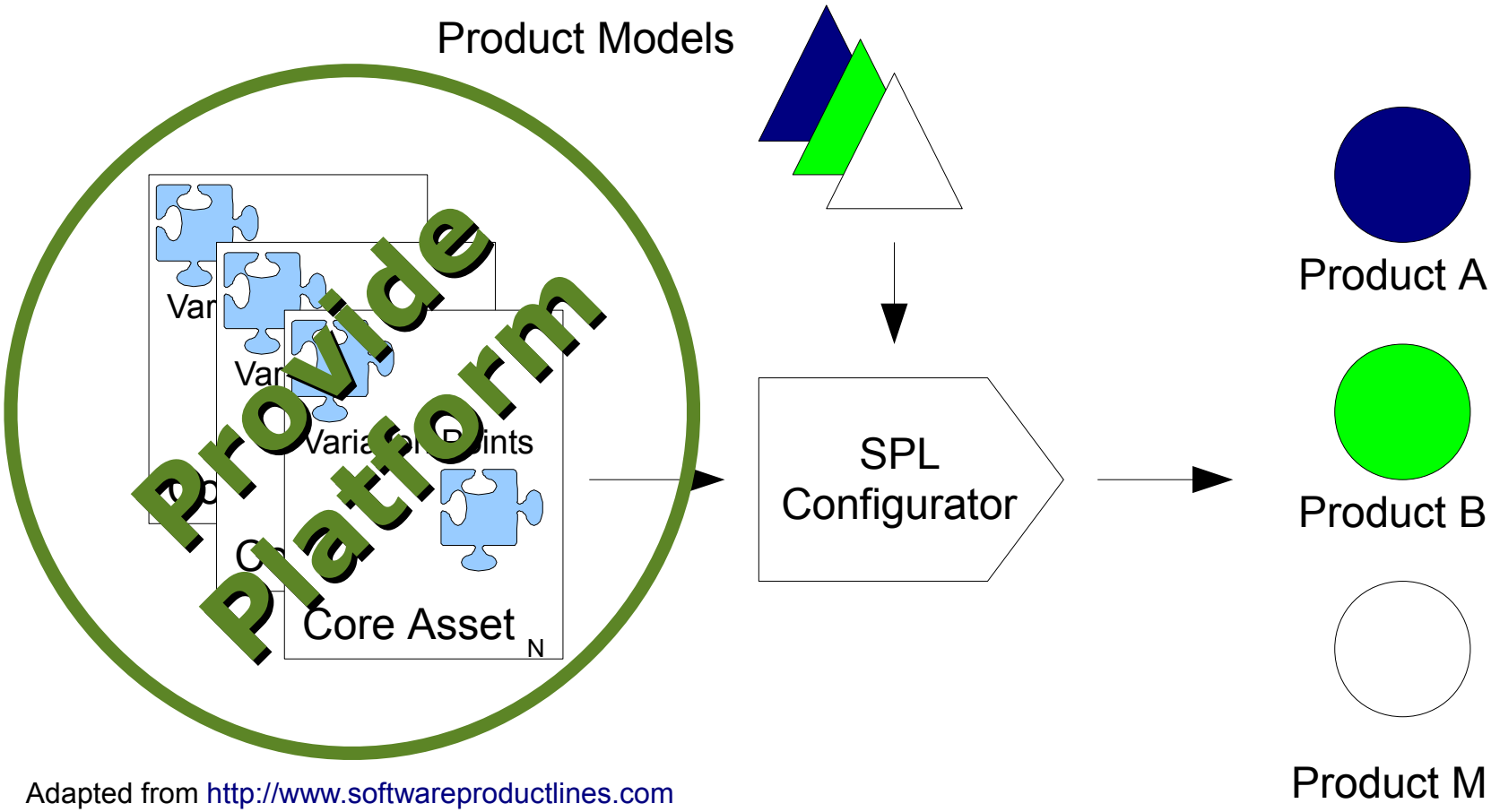
- Changing the platform goes against the purpose of the platform:
  - A platform was meant to be a “sound and stable basis”!
  - A platform provides a means for reuse!
- So, why do it??

Platforms:
Why vary the platform?

- Let's look at common causes of Platform Variability:



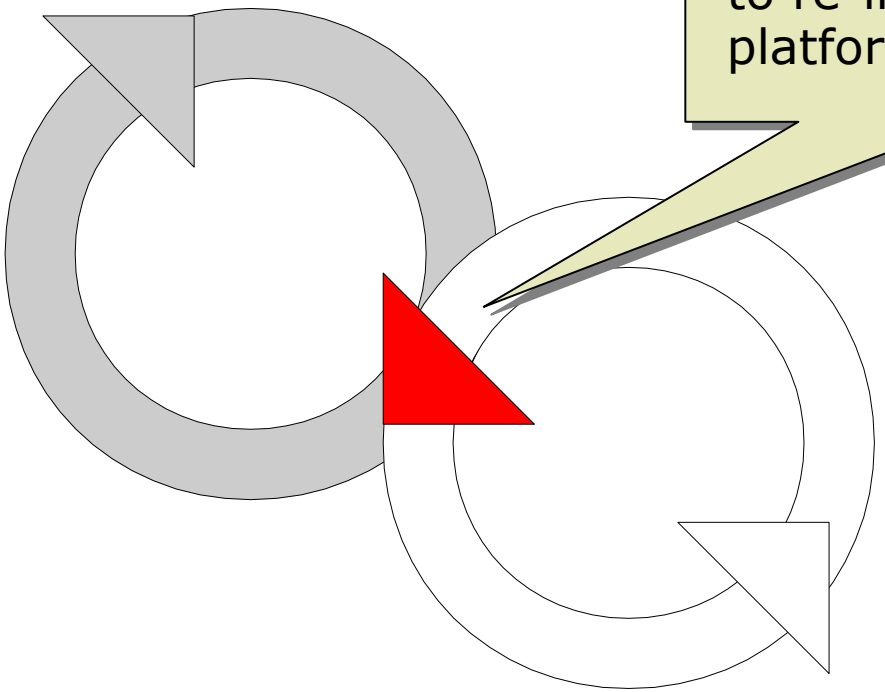
Software Product Lines & Platforms:
Overview



Adapted from http://www.softwareproductlines.com

Cause of Platform Variation in SPL:
The two lifecycles

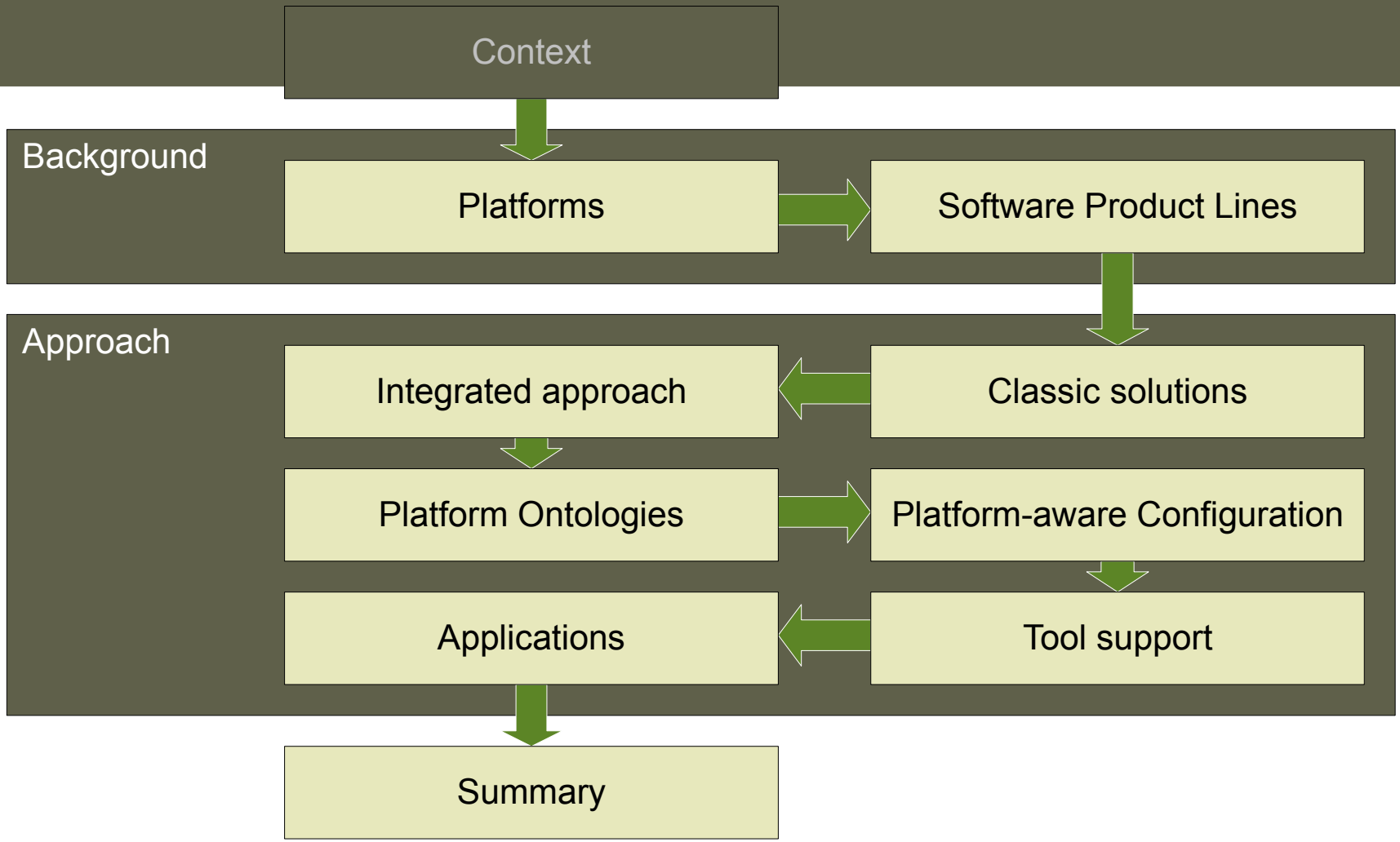
Platform lifecycle



This is where you have to re-integrate for each platform release

Product lifecycle

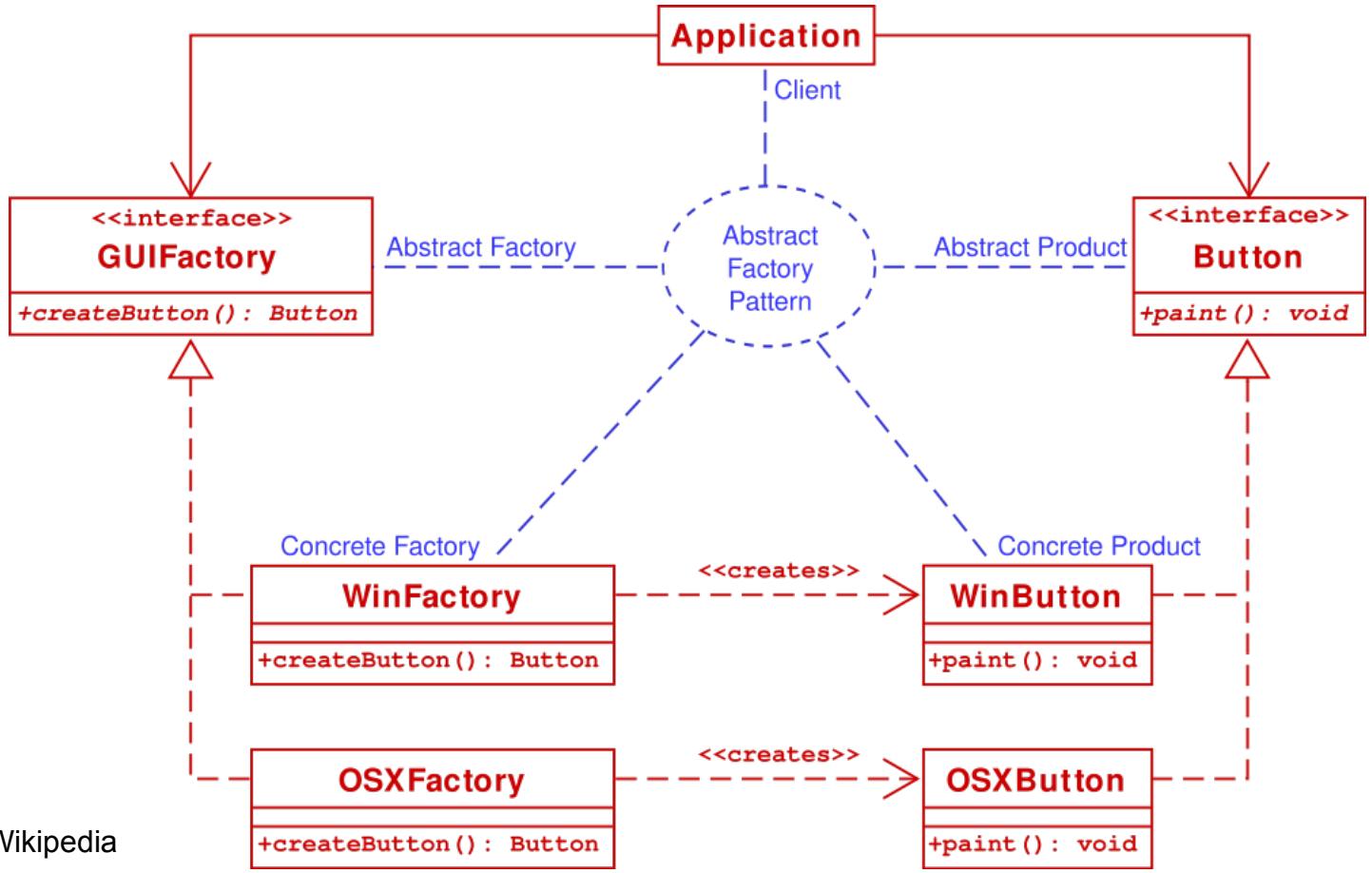
Outline



Classic solutions for Platform Variability:
Overview

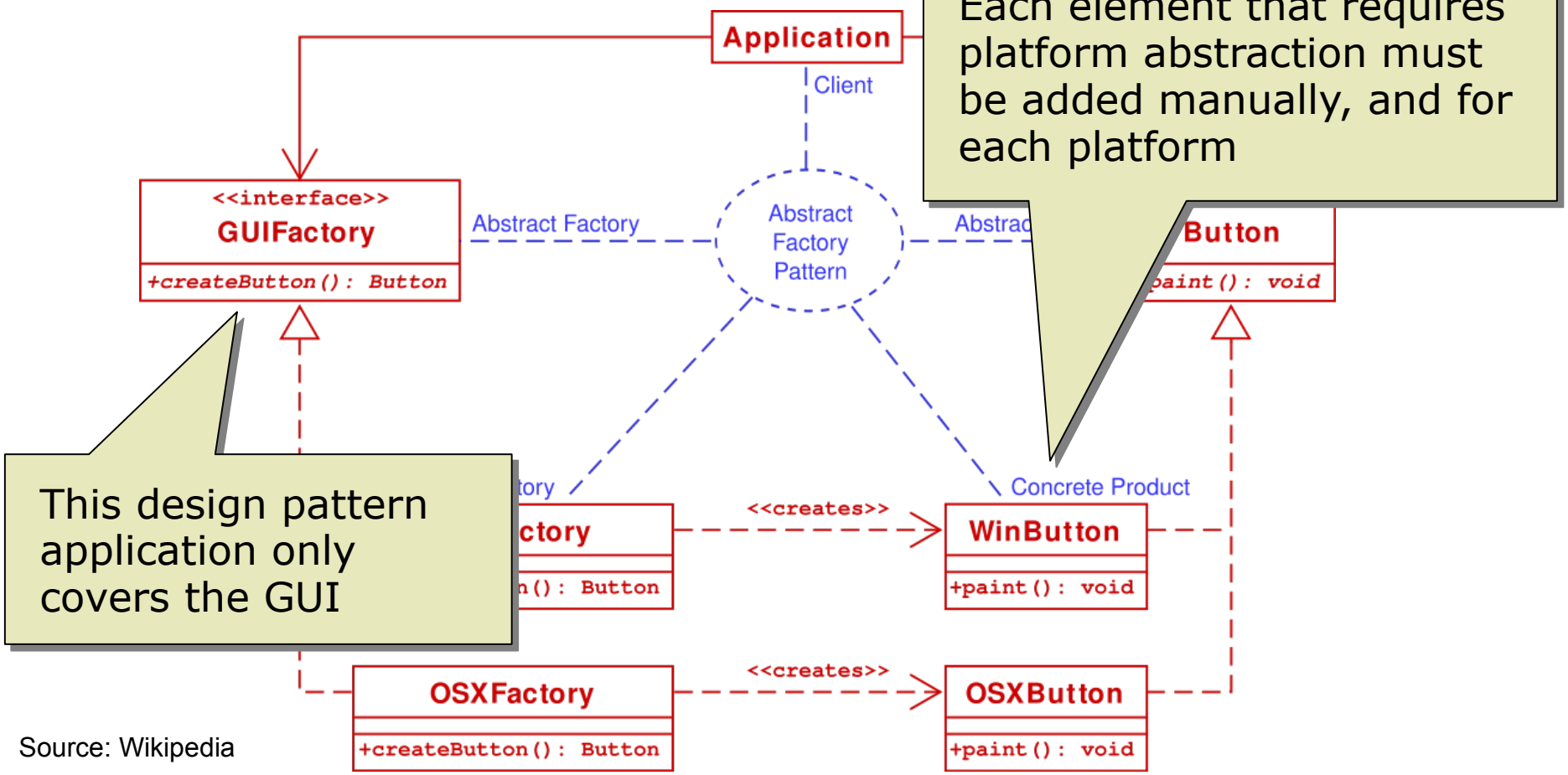
- Solutions for Platform Variability typically take the form of an **abstraction layer**:
  - Application of Design Patterns
    - Bridge, Abstract Factory, Adapter, ...
  - Cross-platform interpreter/virtual machine
    - Java, SmallTalk, JavaScript, Python, ...
  - Transformation-based approach
    - Model Driven Architecture, embedded DSLs (e.g. in Ruby), ...

Example Platform Variability solution:
Abstract Factory pattern



Source: Wikipedia

Example Platform Variability solution:
Abstract Factory pattern



Source: Wikipedia

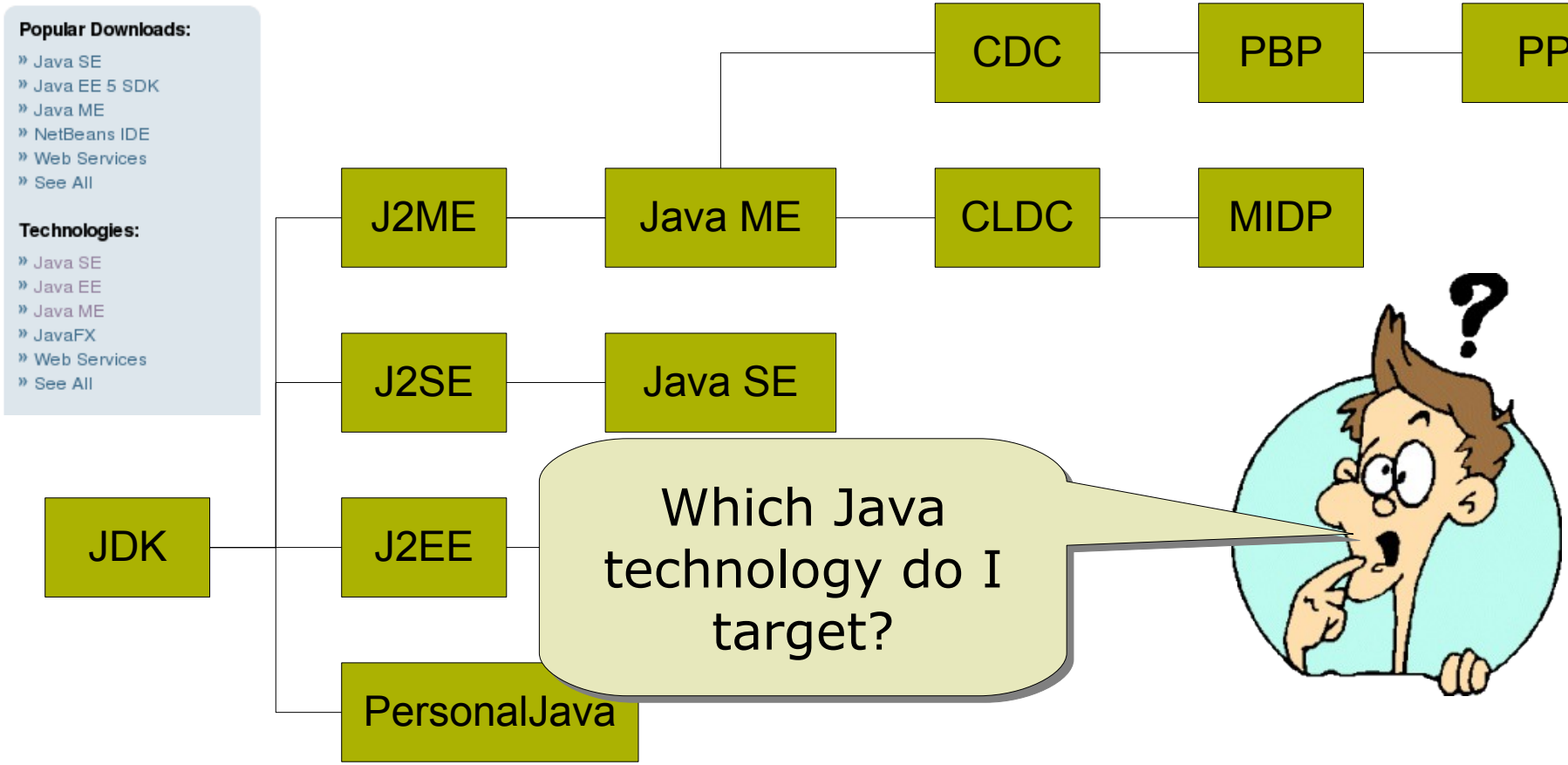
Example Platform Variability solution:
Java

Popular Downloads:
- » Java SE
- » Java EE 5 SDK
- » Java ME
- » NetBeans IDE
- » Web Services
- » See All

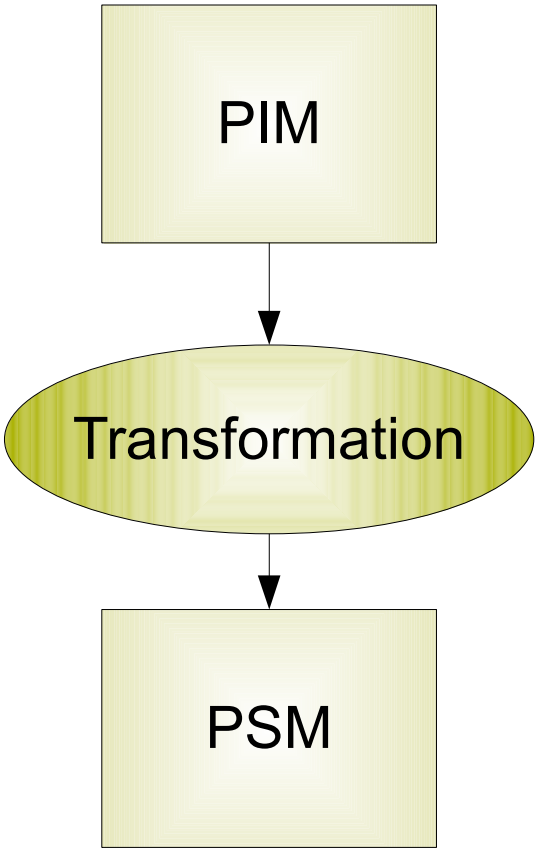
Technologies:
- » Java SE
- » Java EE
- » Java ME
- » JavaFX
- » Web Services
- » See All

"Write Once, Run Anywhere"

Example Platform Variability solution:
Java

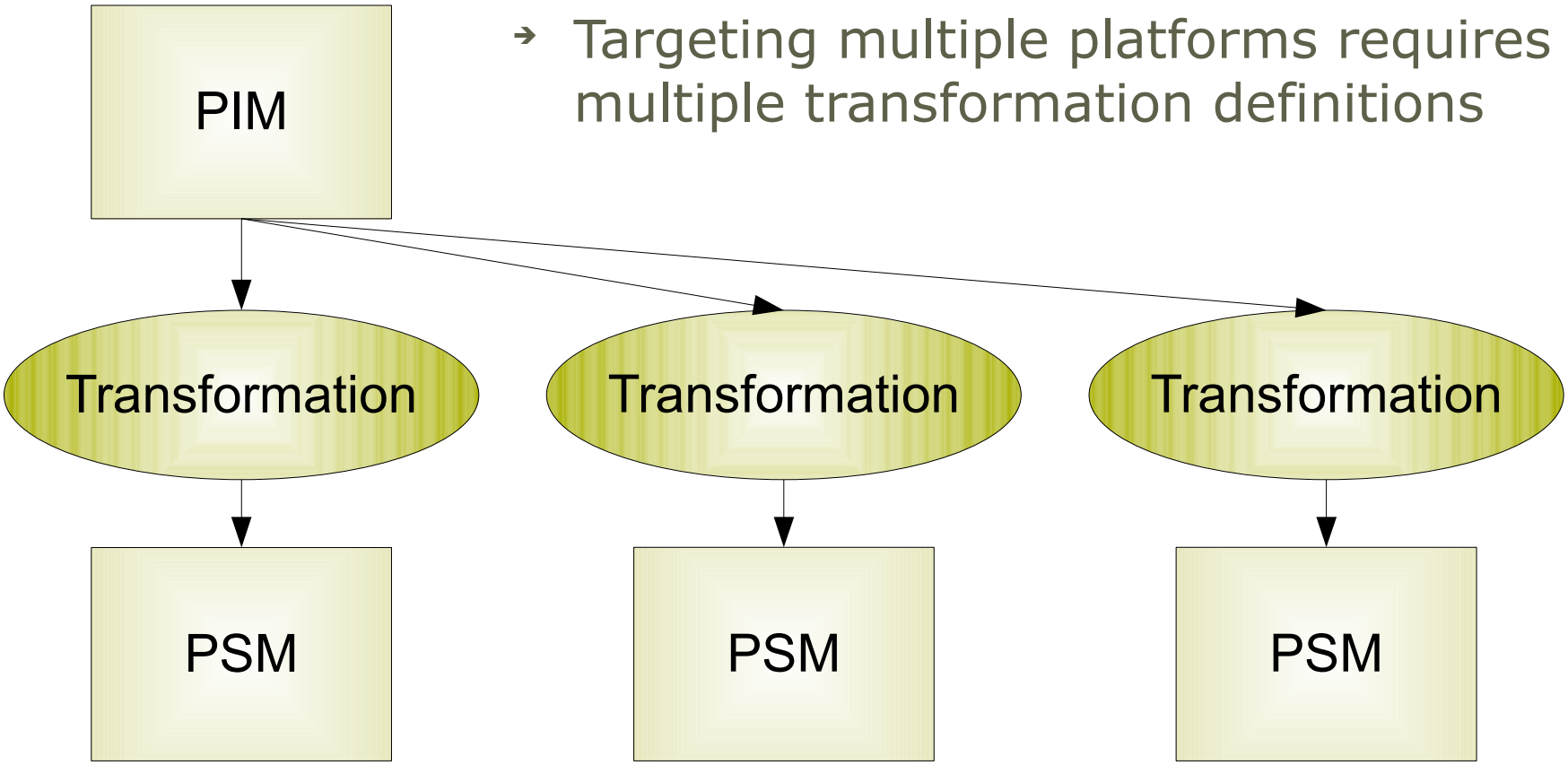


Example Platform Variability solution:
Model Driven Architecture



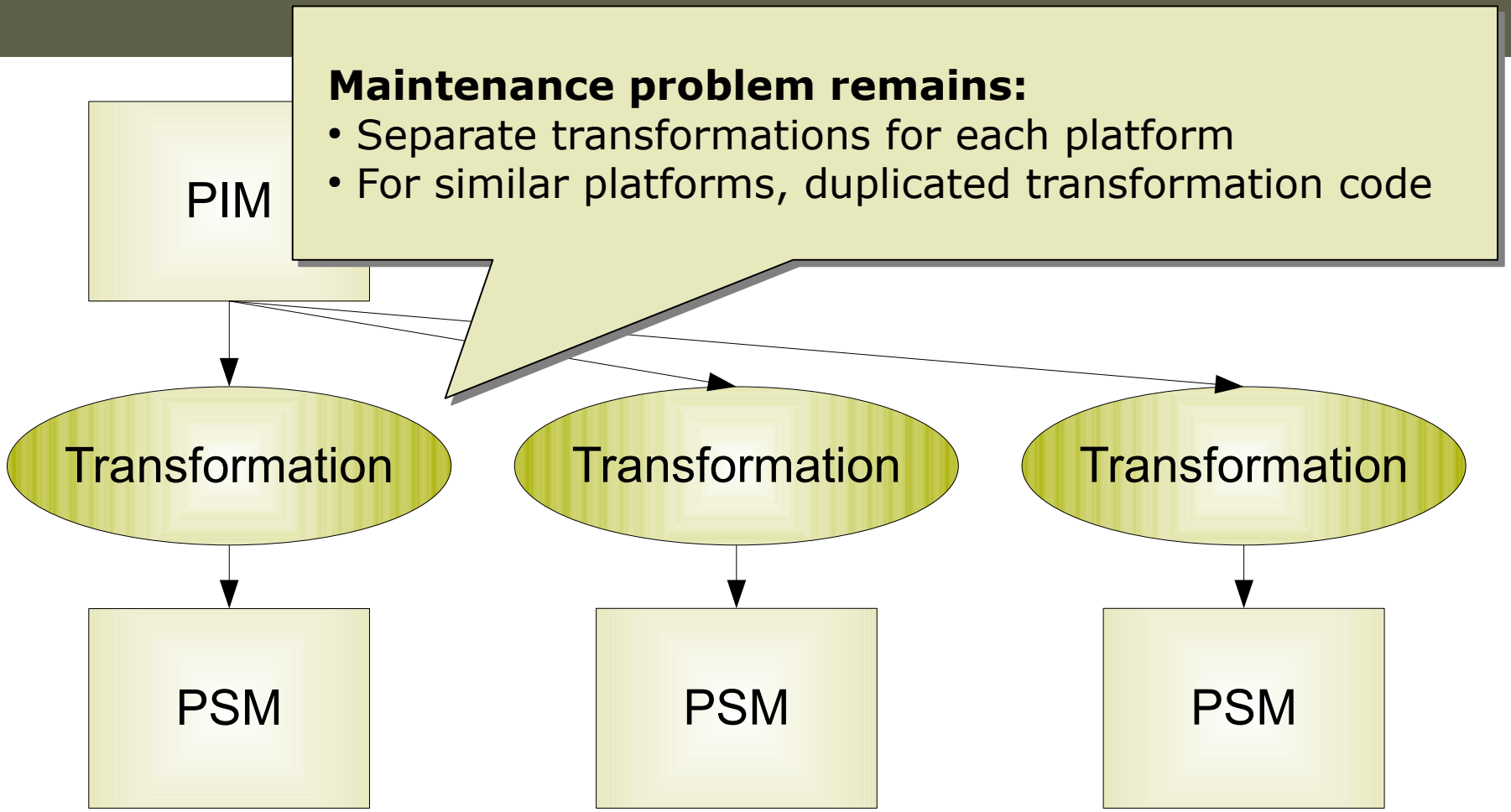
- Design a Platform Independent Model (PIM)
- Automatically transform to a Platform Specific Model (PSM)
- Repeat until you reach code
  - "Platform Independent" is relative:
    - Example: independent from J2SE, J2EE and J2ME, but specific to Java
    - Example: independent from OOP, RDBMS, but specific to data modelling

Example Platform Variability solution:
Model Driven Architecture



- Targeting multiple platforms requires multiple transformation definitions

Example Platform Variability solution:
Model Driven Architecture



Example Platform Variability solution:
Duplicated transformation code in the MDA

```
rule Property {
    from s : UML2!Property (...)
    to t : UML2!Property (
        ...,
        type <- if s.isSingle then
          s.type
          else
          'java::util::Vector'.type()
          endif)
}
```

This is where a platform binding/dependency is introduced

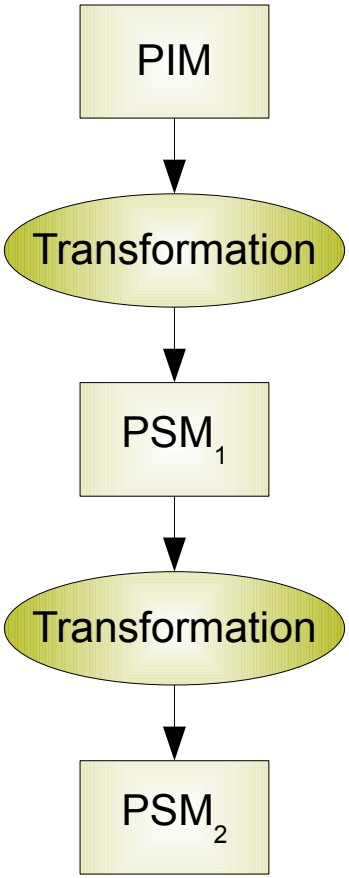
This platform dependency works for all platforms below

Java SE

Java ME MIDP

Java ME PP

Example Platform Variability solution:
Stepwise Refinement in the MDA



- Use stepwise refinement transformations:
  - Smaller, reusable transformation steps
  - Refinement transformations may be reused for multiple target platforms
    - E.g. Java SE, Java ME MIDP, Java ME PP, ...

Classic solutions for Platform Variability:
Application of Design Patterns

- Advantages:
  - Easy to apply, no extra technology required
    - Standard OO language features are sufficient
  - Platform Variability is bridged at run-time
    - Possibility to move platform

- Disadvantages:
  - **Must be applied manually and locally for each platform-specific context**
    - Maintenance effort for supporting multiple platforms is not much reduced
  - Platform Variability is bridged at run-time, even though the target platform may have been fixed before
    - Can impose unnecessary overhead

Classic solutions for Platform Variability:
Cross-platform interpreter/virtual machine

- Advantages:
  - Platform differences are bridged automatically
    - Low maintenance
  - Platform Variability is bridged at run-time
    - Possibility to move platform
- Disadvantages:
  - Platform Variability is bridged at run-time, even though the target platform may have been fixed before
    - Performance hit
  - **There may not exist general purpose abstractions for all platform differences**
    - Becomes especially clear in the domain of user interfaces

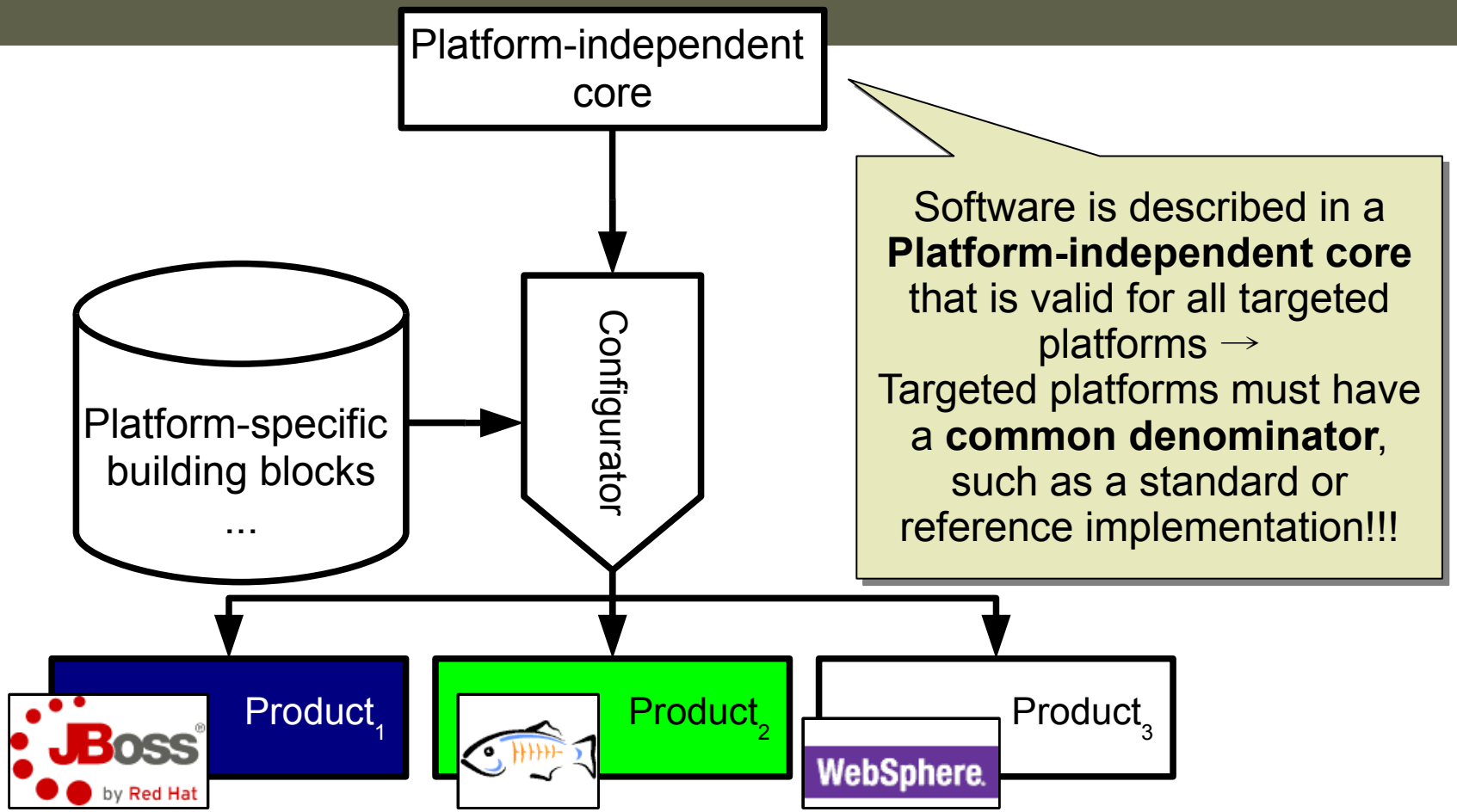
Classic solutions for Platform Variability:
Transformation-based approach

- Advantages:
  - Platform differences are bridged automatically
    - Low(er) maintenance
  - Platform Variability is bridged at compile-time
    - No performance hit
- Disadvantages:
  - Platform Variability is bridged at compile-time
    - Cannot move platform
  - Transformations are domain-specific
    - **Maintenance effort moved to transformations**

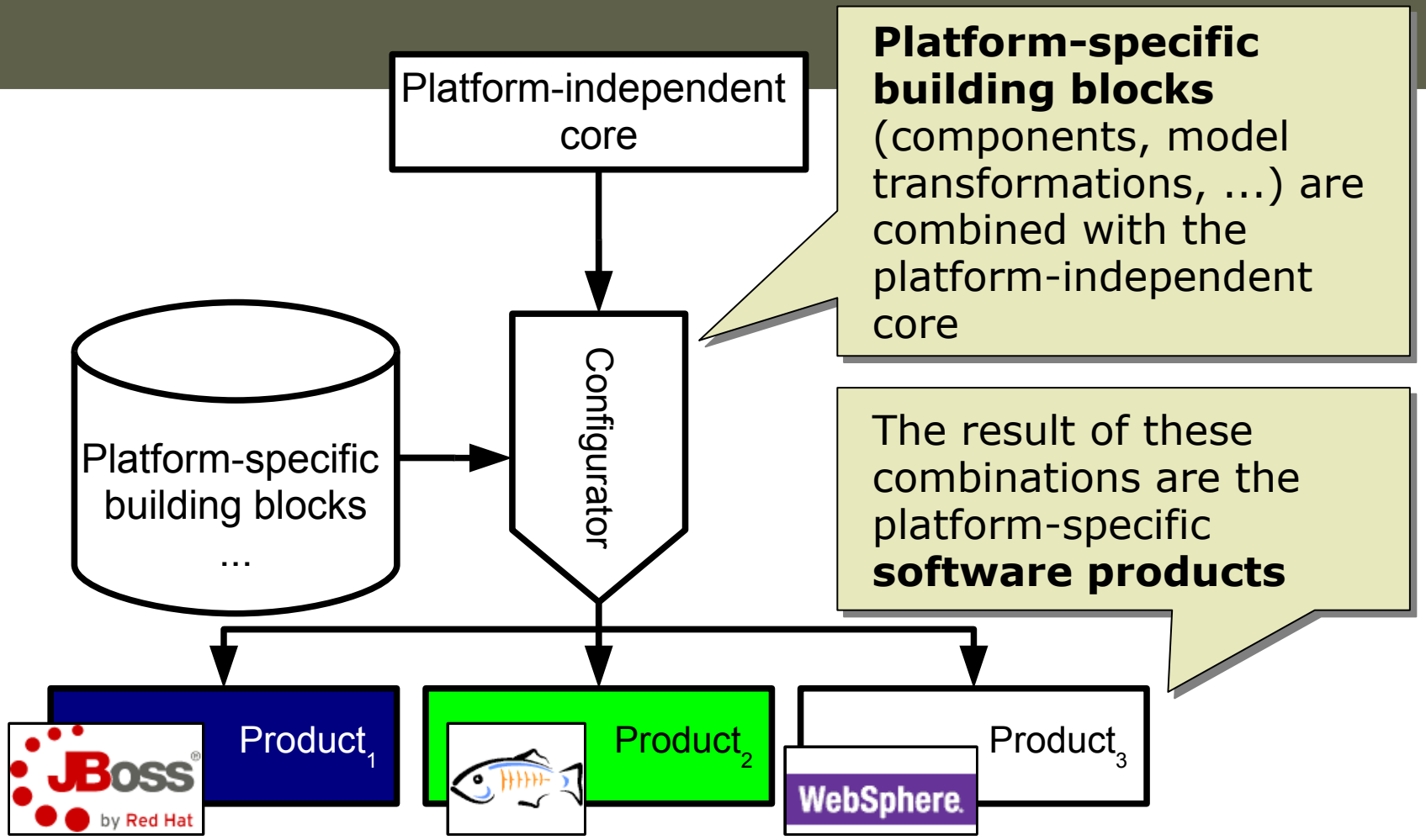
Classic solutions for Platform Variability: Conclusions

- None of the classic solutions solves all our problems
  - But each solution has strong and weak points!
- Platform Variability can be tackled using a combination of these solutions, e.g.:
  - If it is feasible to use an existing interpreter/virtual machine-based solution, apply it first
  - Smaller platform differences that occur many times can be covered using a transformation-based approach
  - Larger platform differences that occur locally can be covered using design patterns

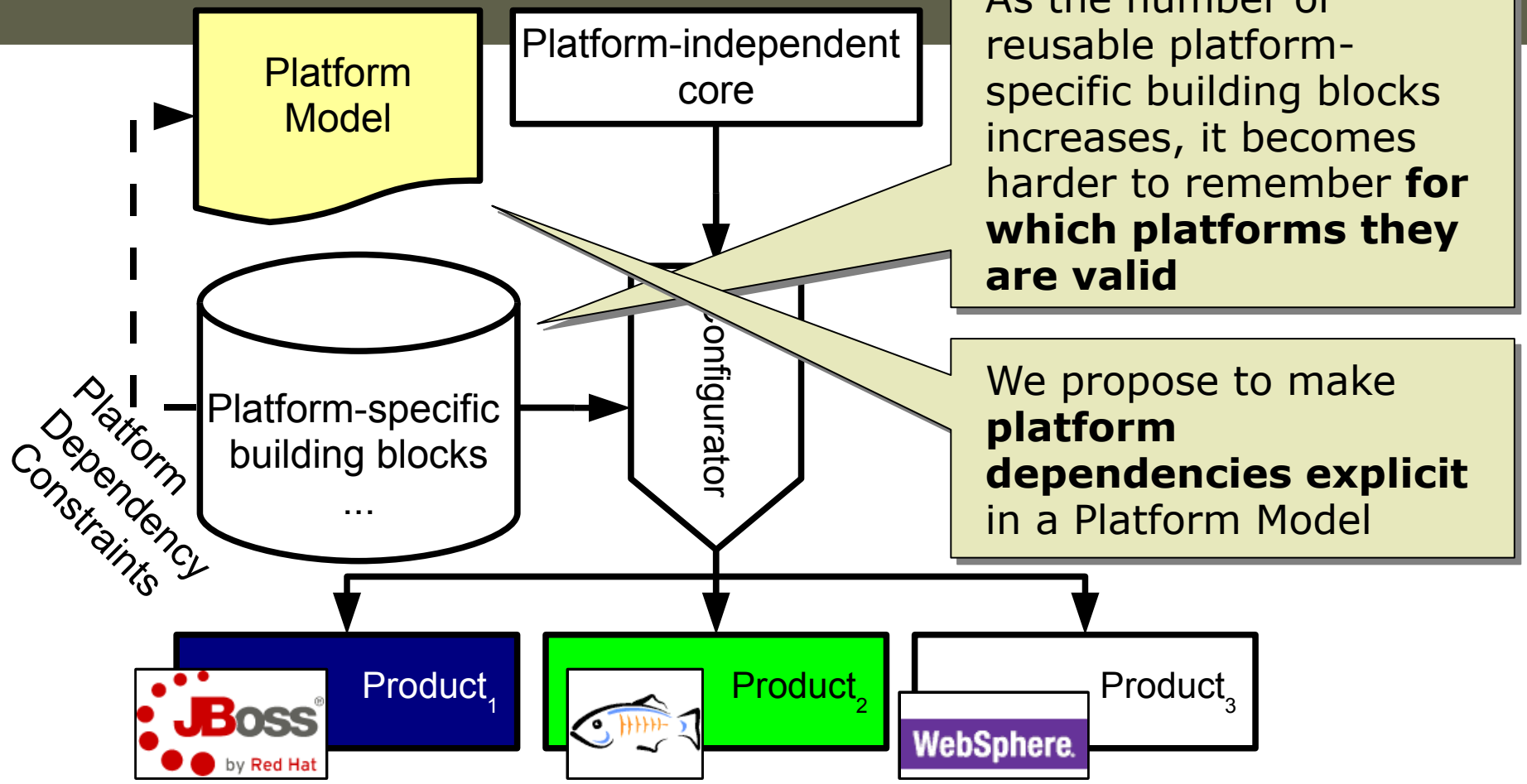
Integrated Platform Variability approach:
Overview



Integrated Platform Variability approach:
Overview



Integrated Platform Variability approach:
Platform Model



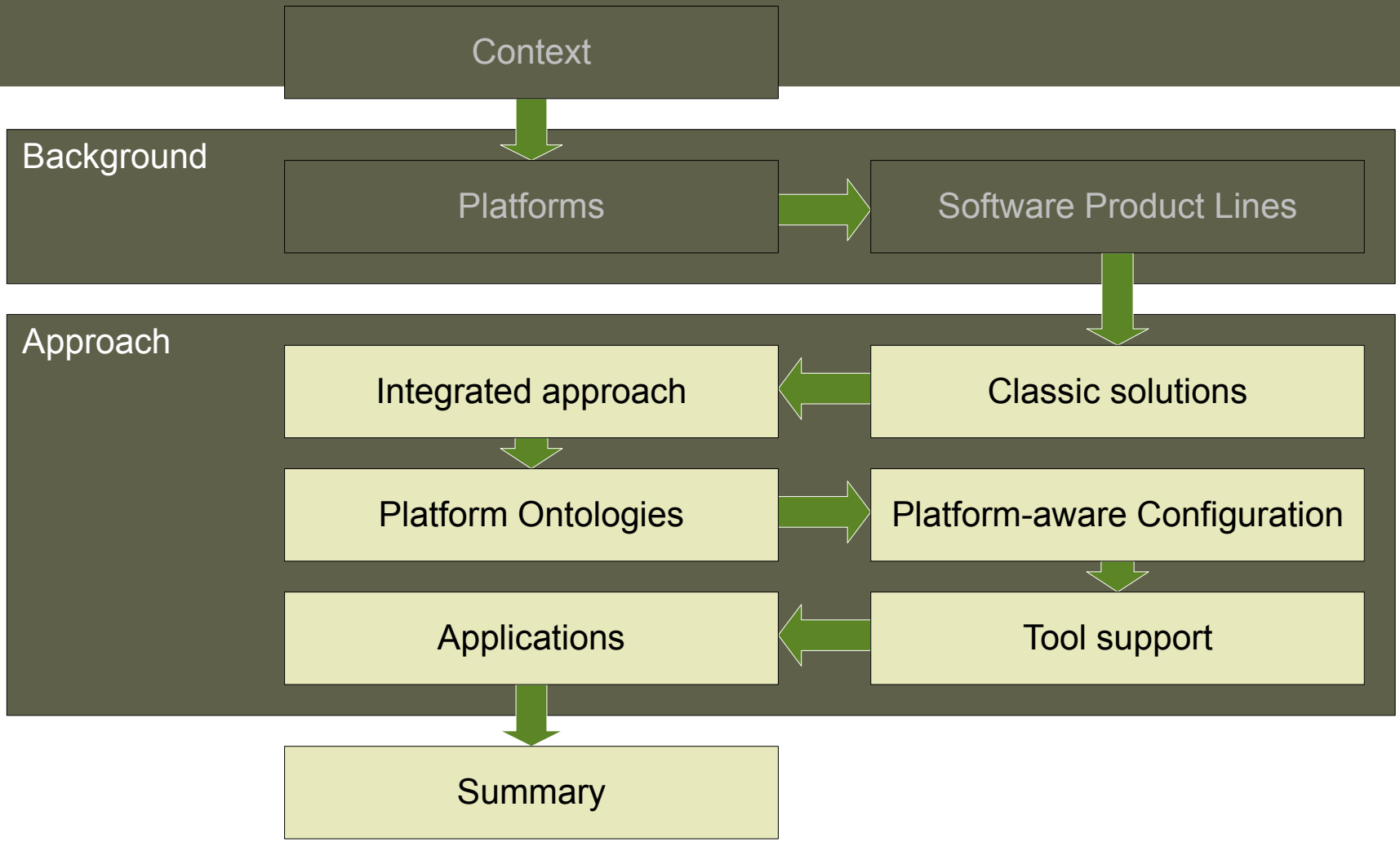
As the number of reusable platform-specific building blocks increases, it becomes harder to remember **for which platforms they are valid**

We propose to make **platform dependencies explicit** in a Platform Model

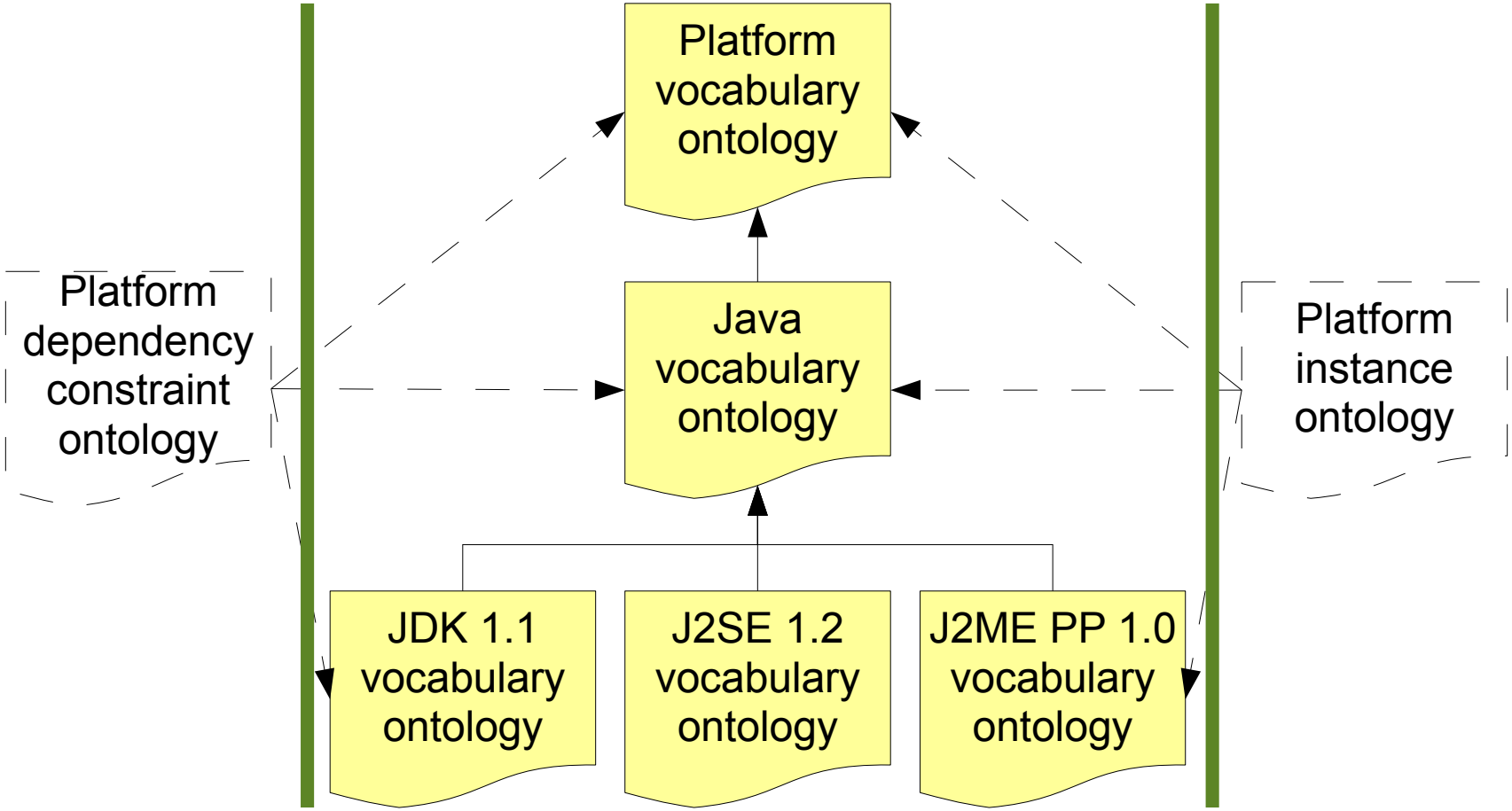
Platform Model:
Purpose and anatomy

- The Platform Model must be able to record platform dependencies in general
  - No technology-specific approach (Autoconf, ENVY, OSGi, …)
  - A general domain description language is required
- The Platform Model must support checking dependencies against specific platform instances
  - Logic-based languages provide a general framework for this
- Ontologies, and OWL DL in particular, fit the bill
  - The platform domain concepts are described first
  - Dependencies and instances are described on top of these

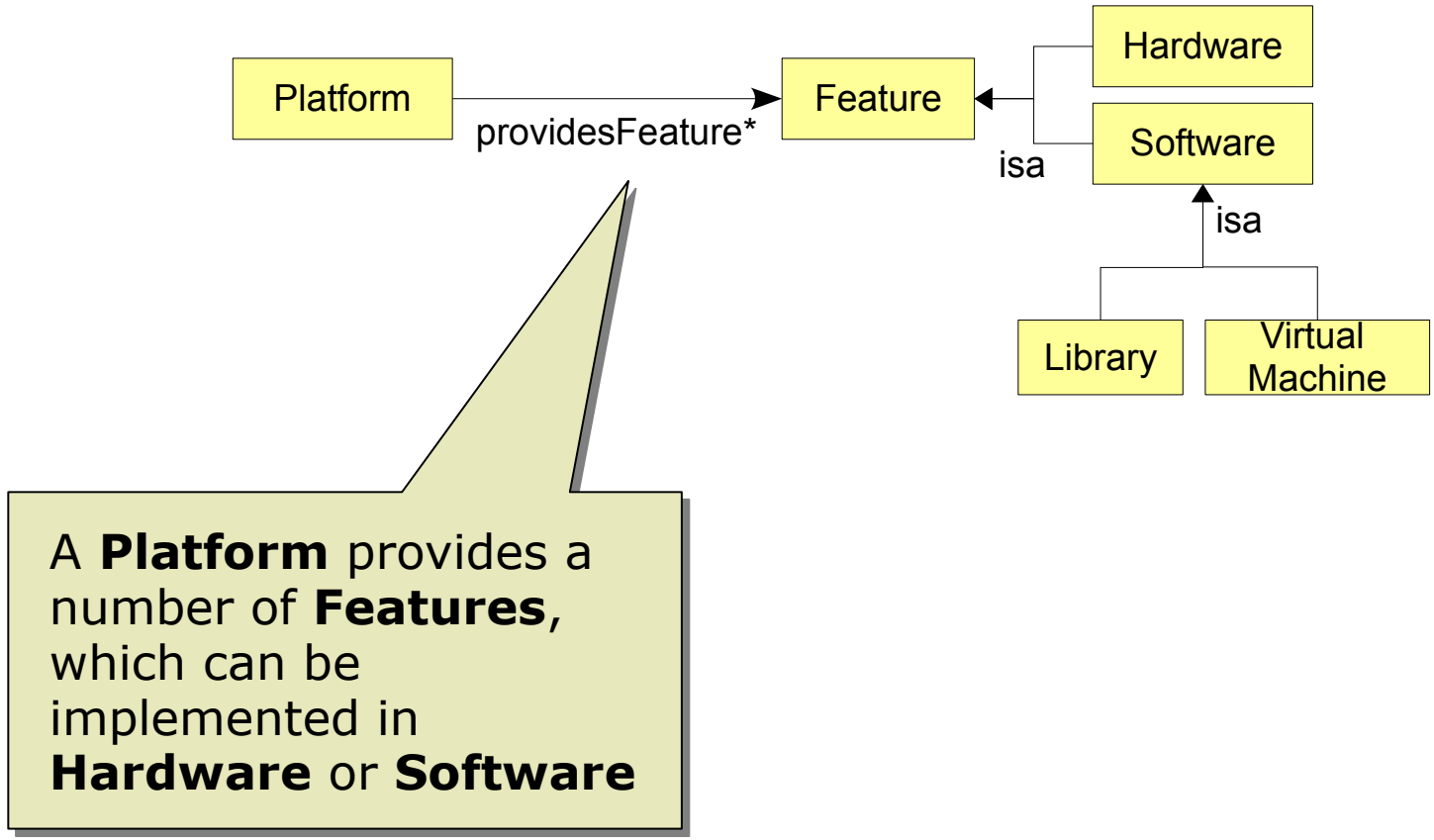
Outline



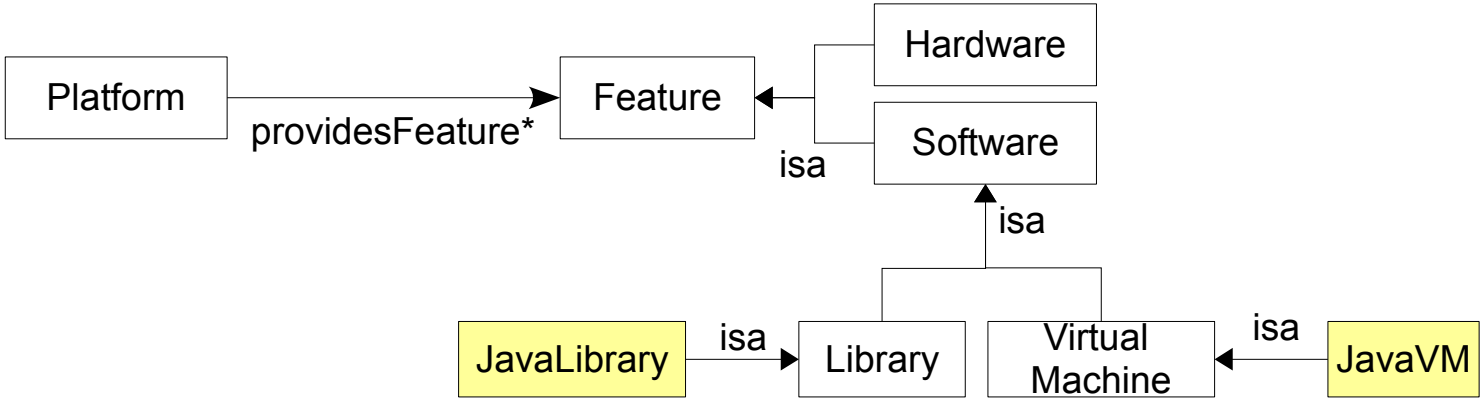
Platform Ontologies:
Overview



Platform Ontologies:
Platform vocabulary ontology

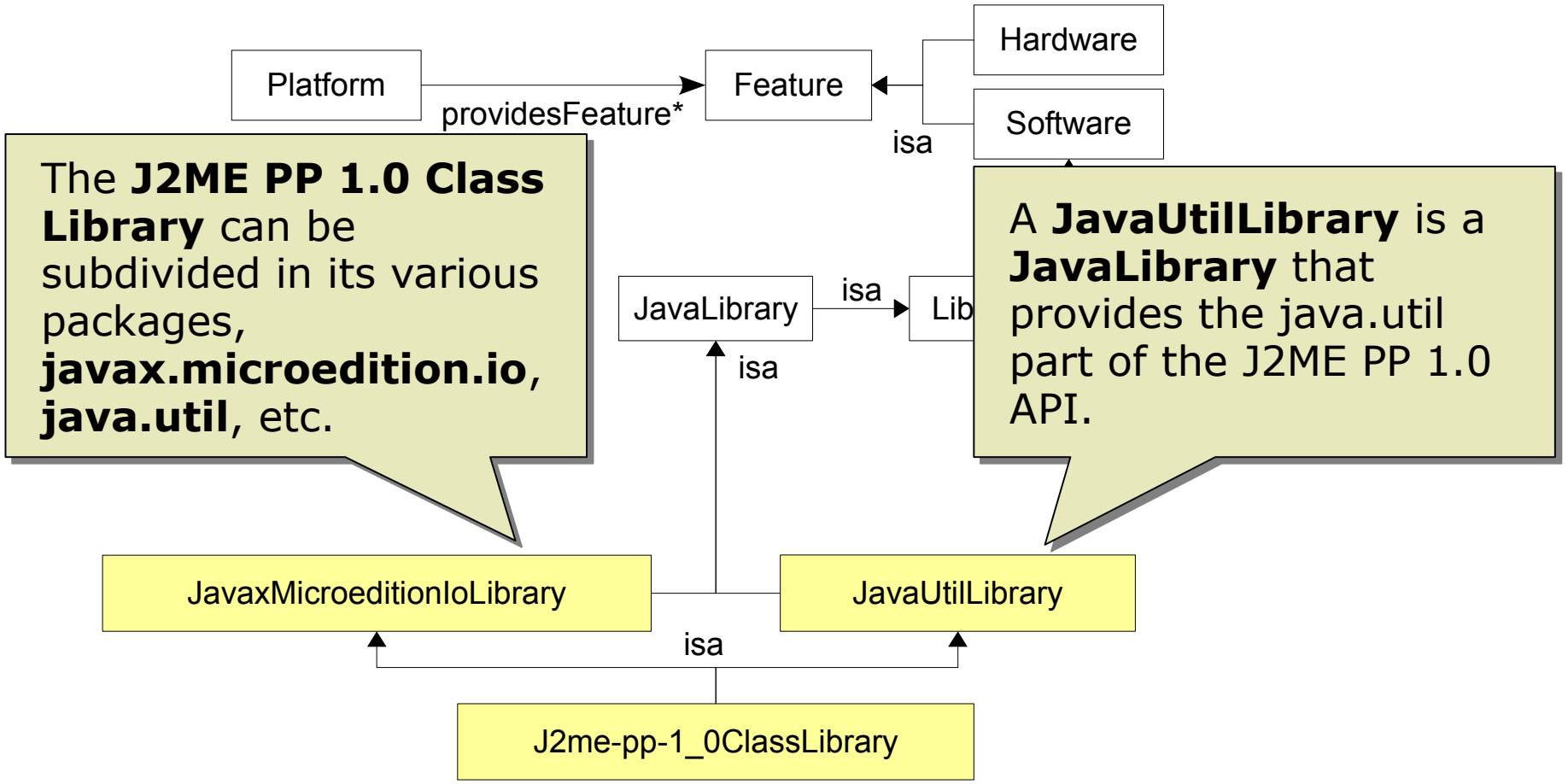


Platform Ontologies:
Java vocabulary ontology



A **JavaLibrary** is a kind of **Library**, and a **JavaVM** is a kind of **VirtualMachine**

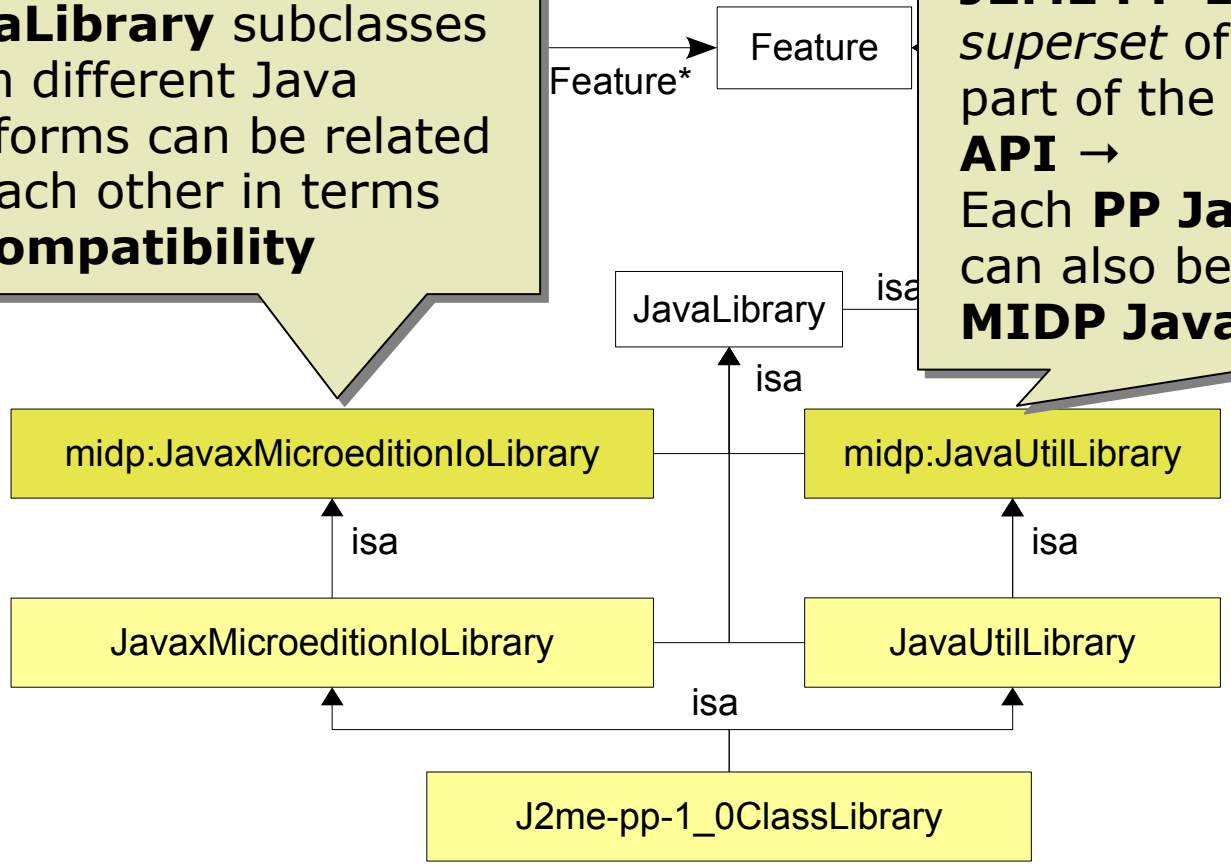
Platform Ontologies:
J2ME PP 1.0 vocabulary ontology



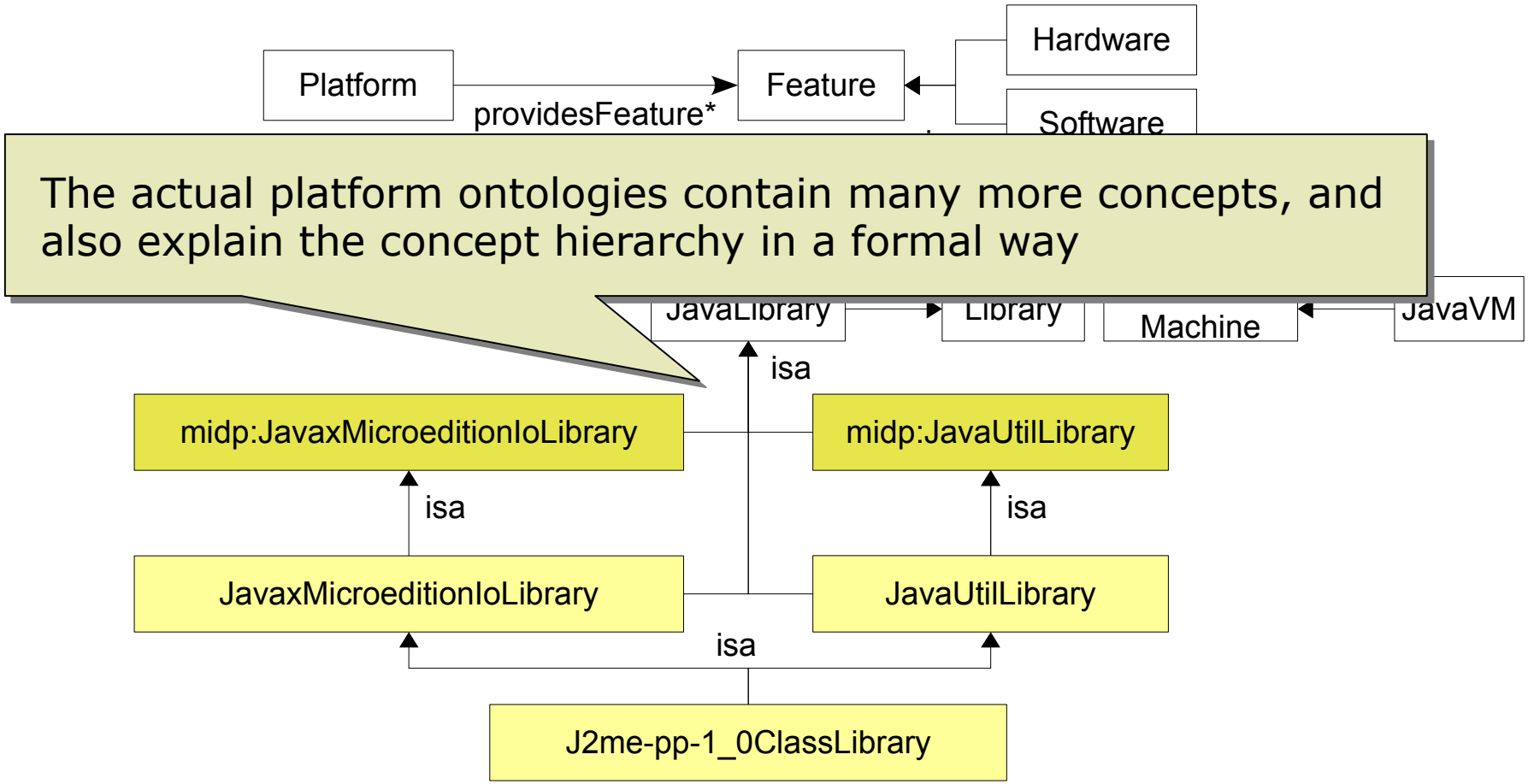
Platform Ontologies:
J2ME PP 1.0 vocabulary ontology

The different **JavaLibrary** subclasses from different Java platforms can be related to each other in terms of **compatibility**

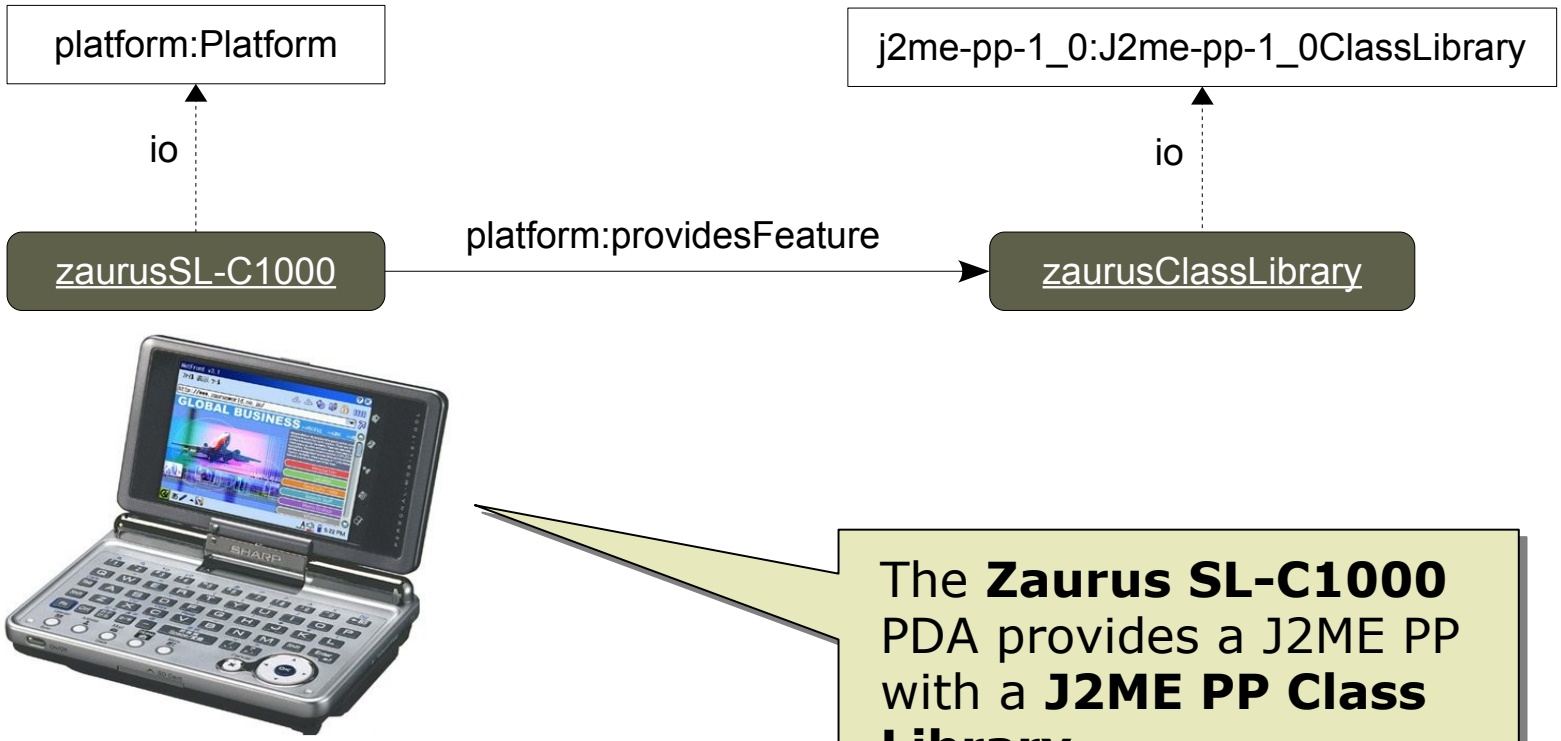
The **java.util** part of the **J2ME PP 1.0 API** is a *superset* of the **java.util** part of the **J2ME MIDP 1.0 API** →
Each **PP JavaUtilLibrary** can also be considered an **MIDP JavaUtilLibrary**



Platform Ontologies:
J2ME PP 1.0 vocabulary ontology



Platform Ontologies:
Platform instances

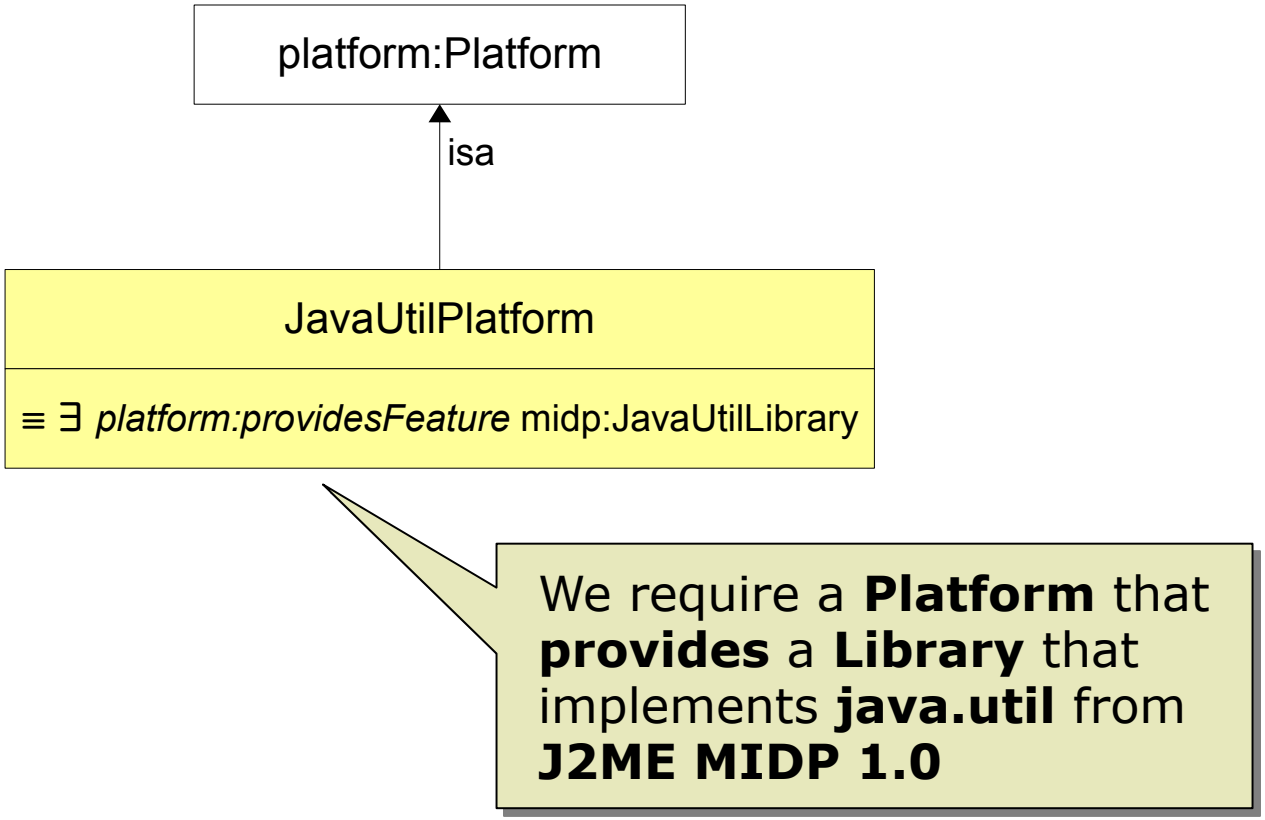


Platform Ontologies:
Platform dependencies

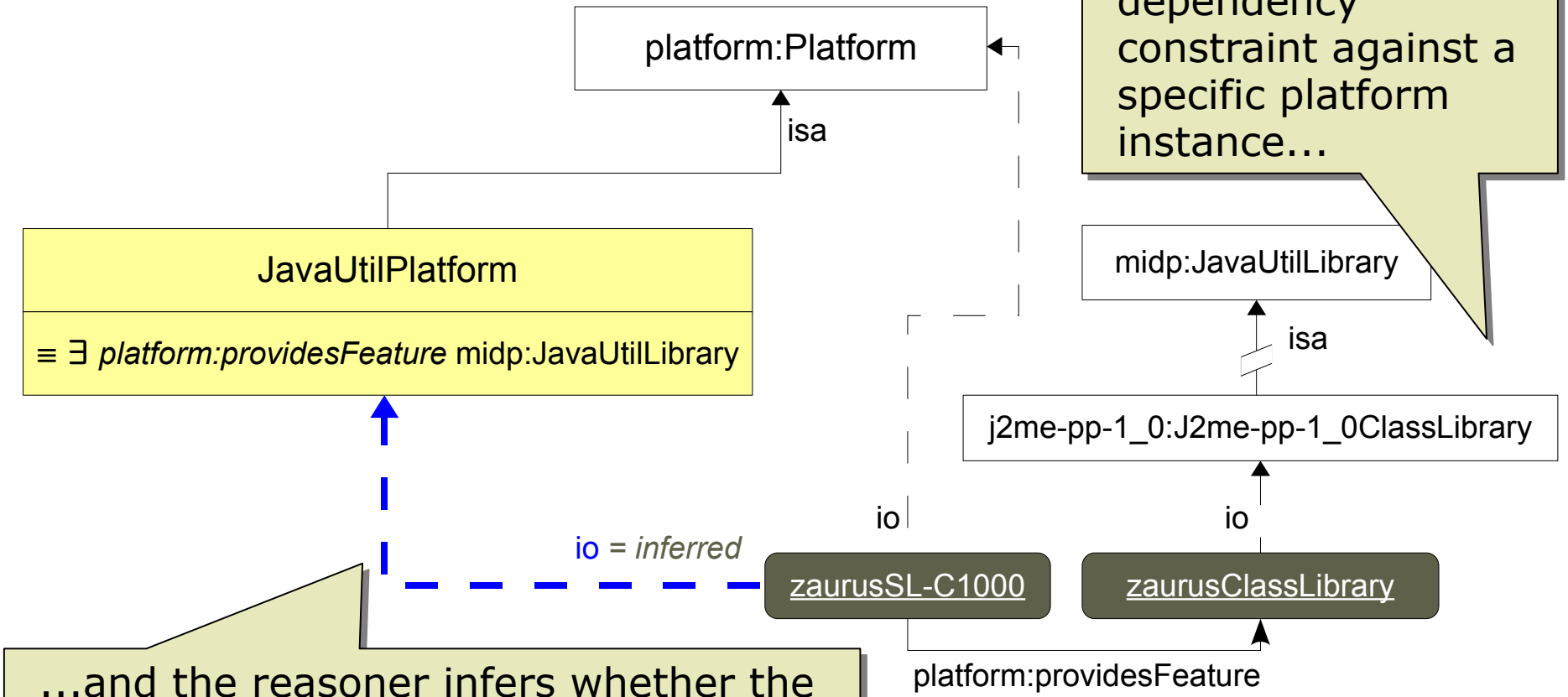
```
rule Property {
    from s : UML2!Property (...)
    to t : UML2!Property (
        ...,
        type <- if s.isSingle then
          s.type
          else
          'java::util::Vector'.type()
          endif)
}
```

The **java.util.Vector** class appears in the **java.util** part of the **J2ME MIDP API**.

Platform Ontologies:
Platform dependencies

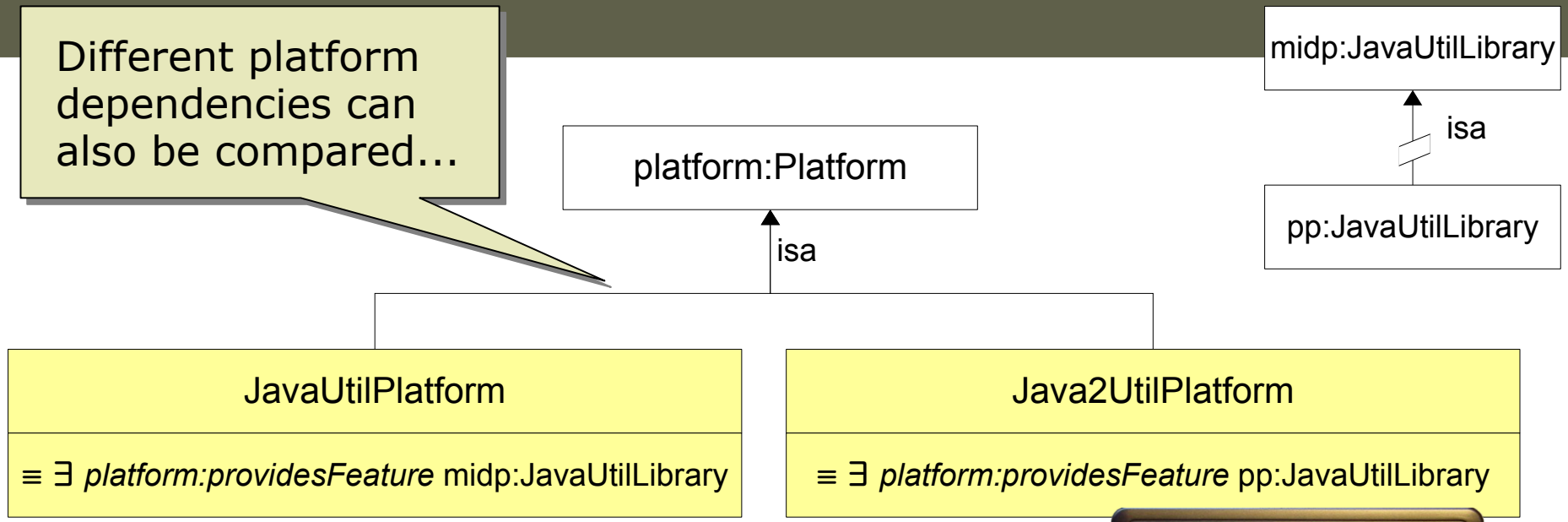


Platform Ontologies:
Platform dependencies

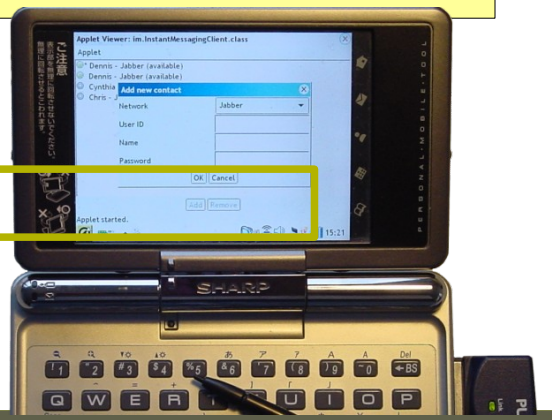


Platform Ontologies:
Platform dependencies

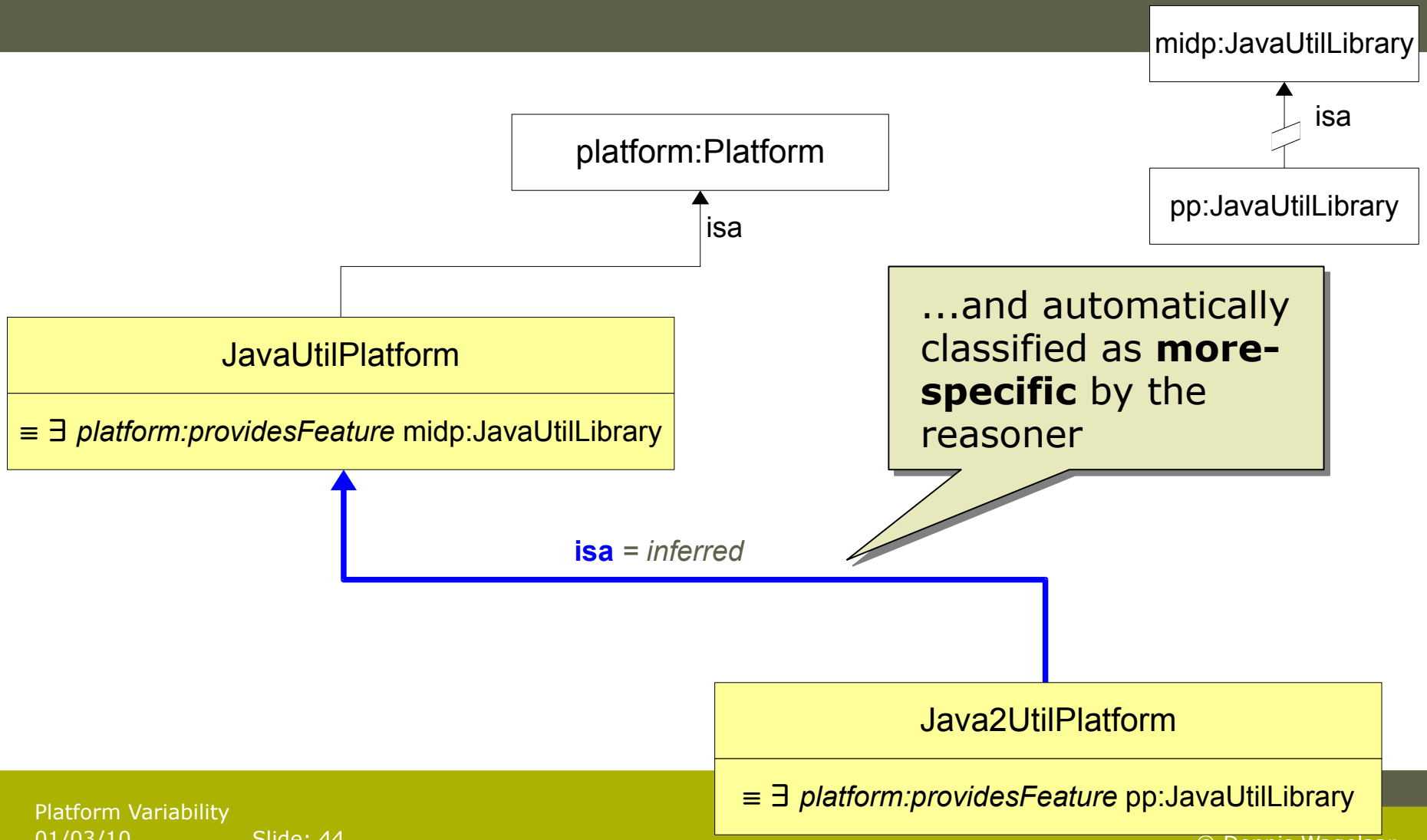
Different platform dependencies can also be compared...



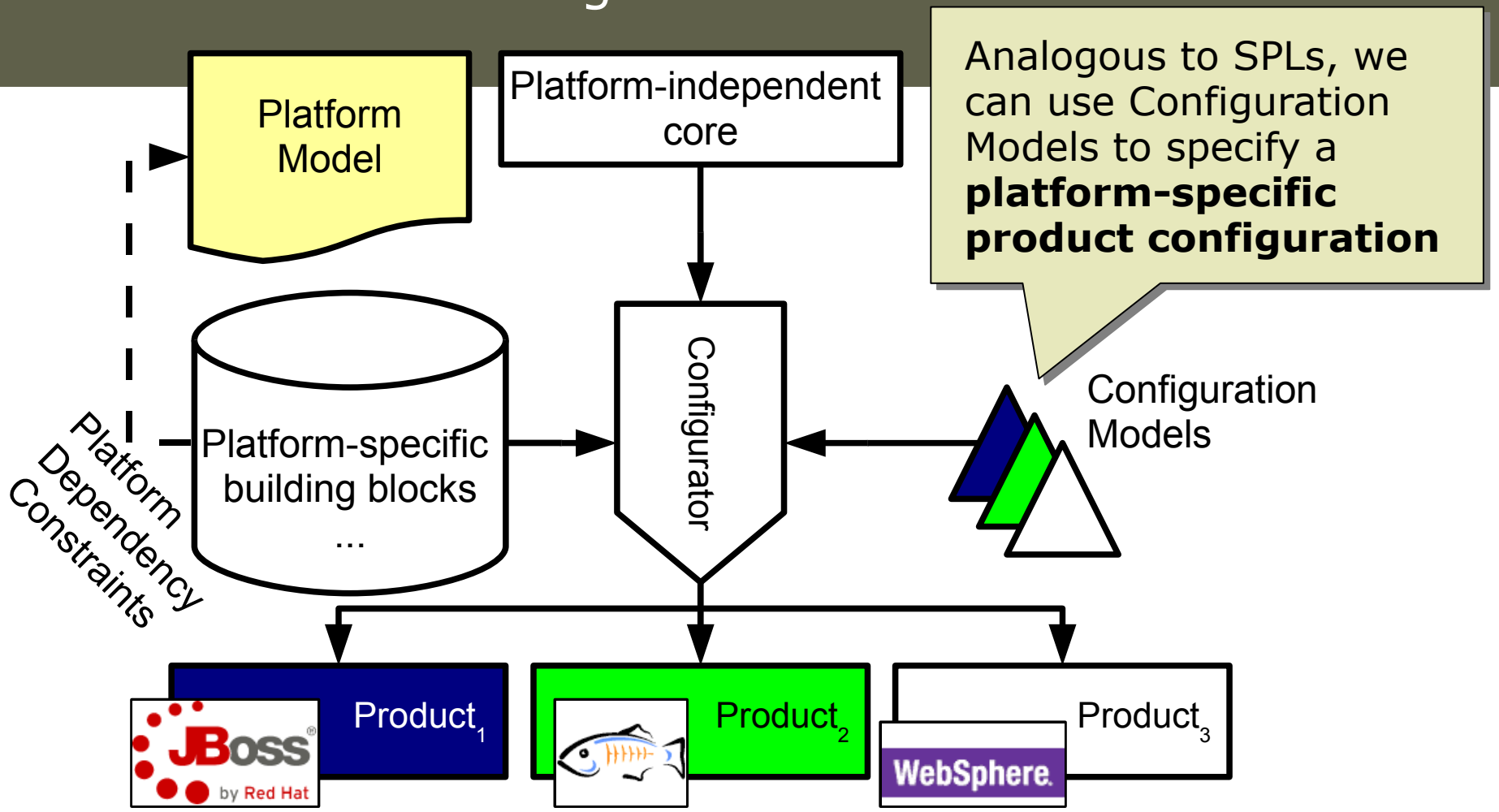

```
rule Property {
    ...
        'java::util::List'.type()
    ...
}
```



Platform Ontologies:
Platform dependencies

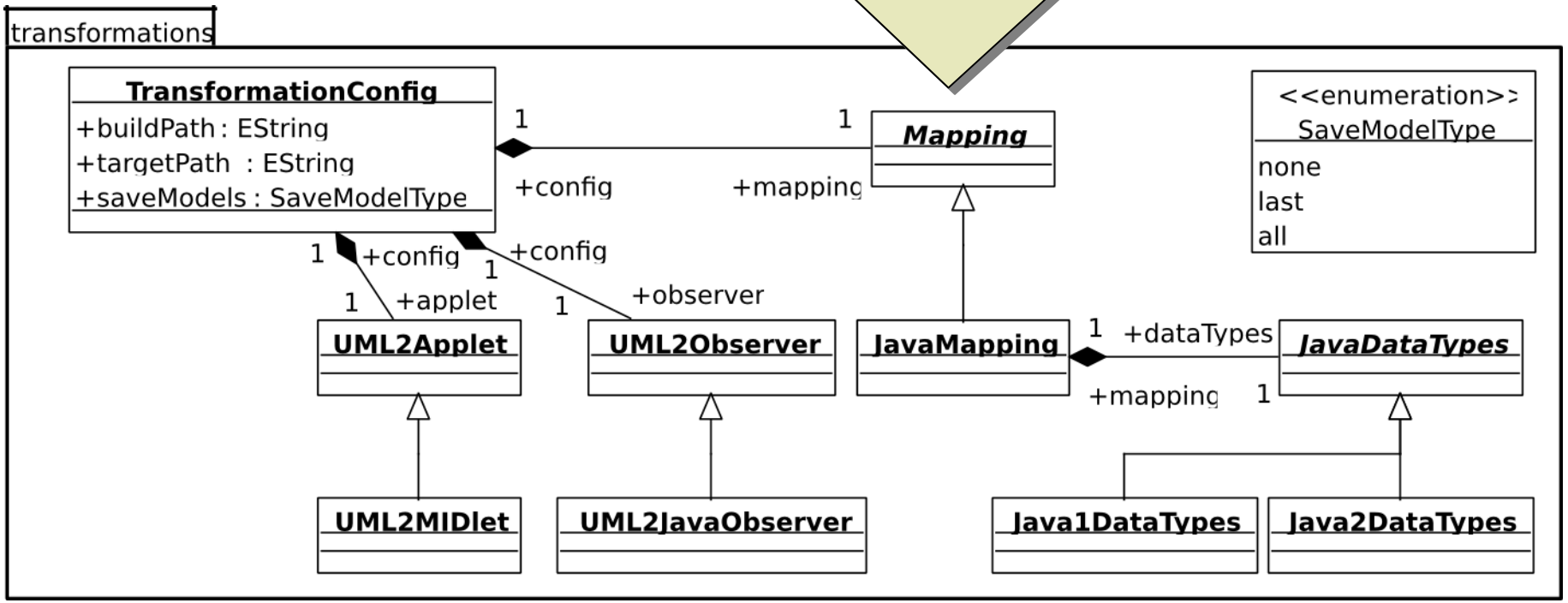


Integrated Platform Variability approach:
Platform-aware Configuration



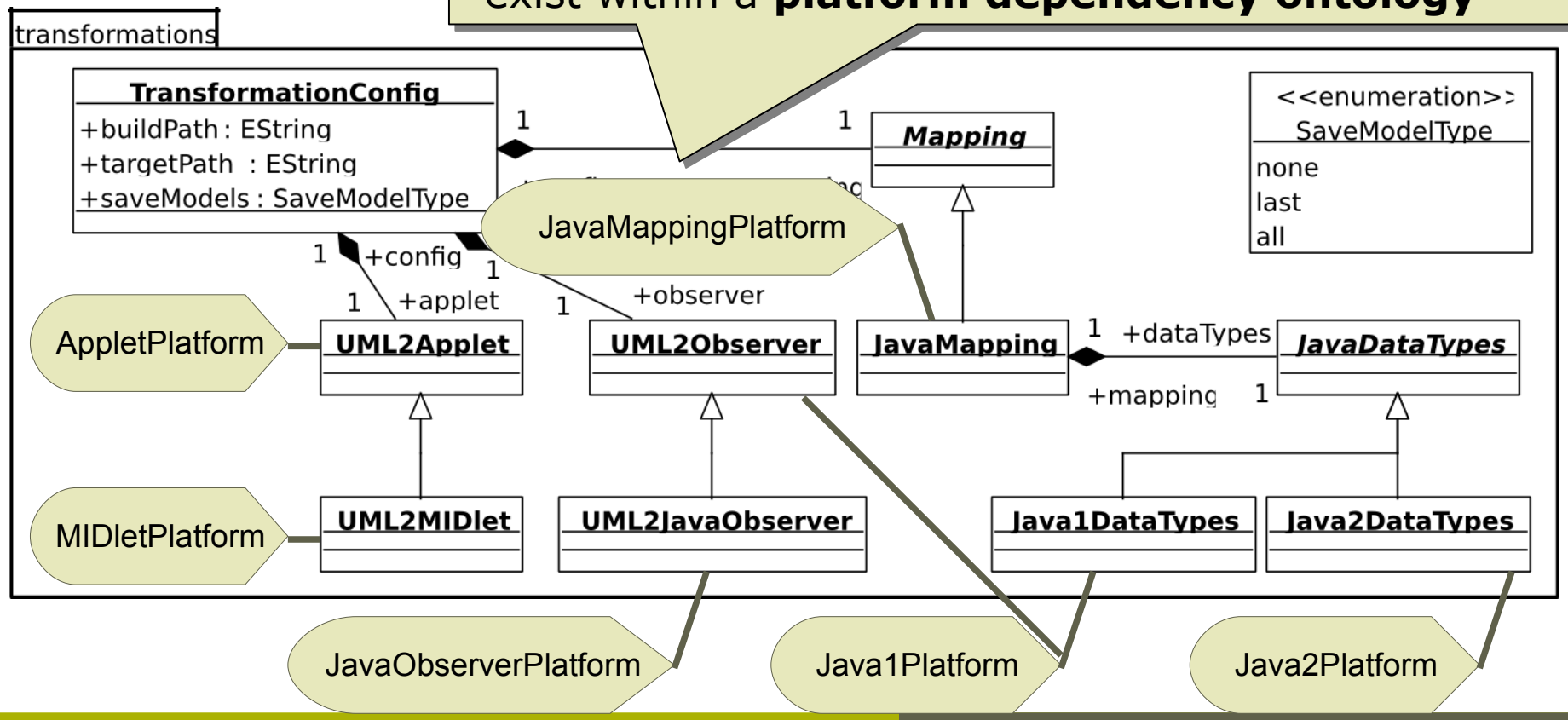
Platform-aware Configuration:
Configuration Language

A configuration language **meta-model** provides configuration **rules**

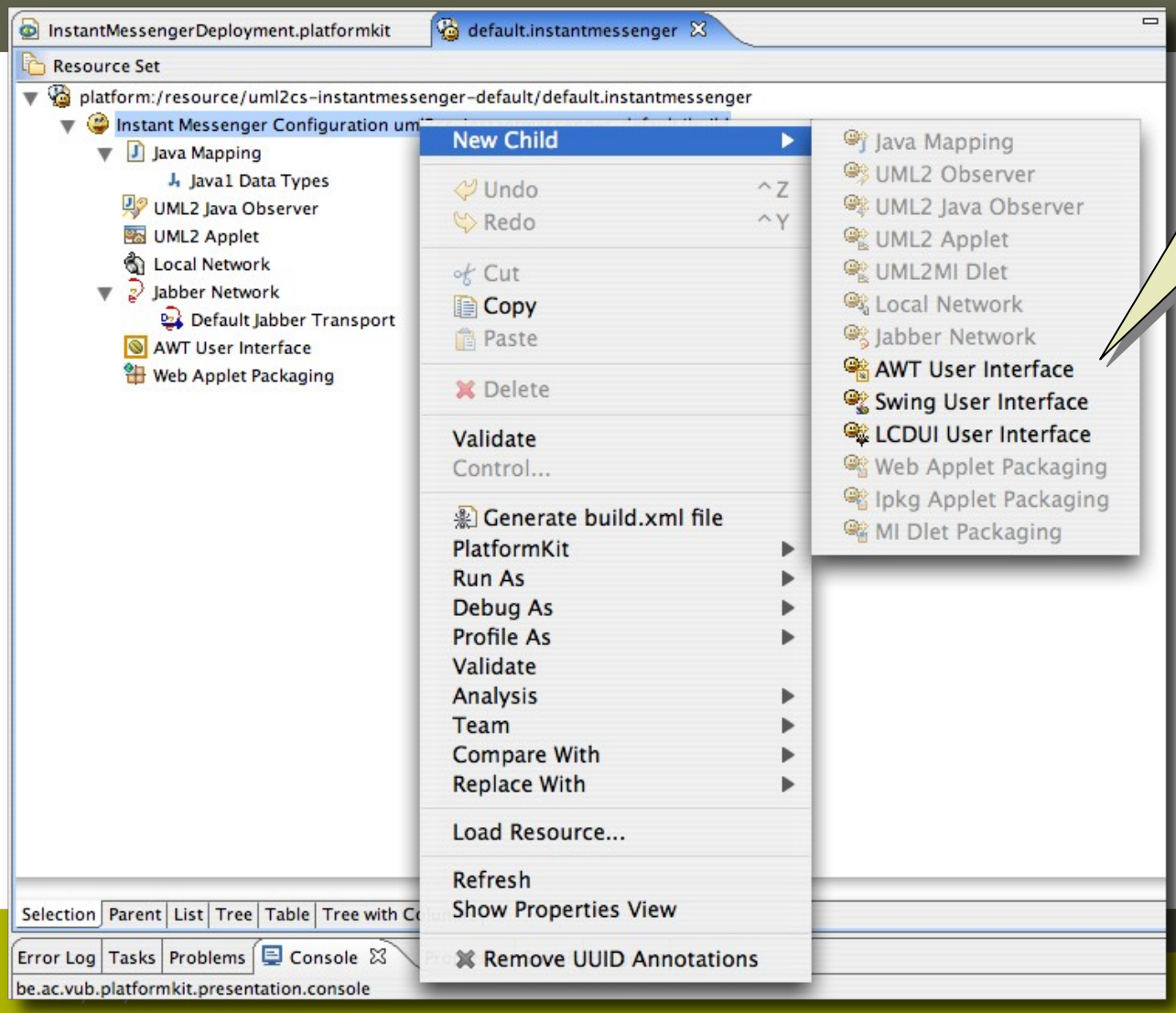


Platform-aware Configuration:
Configuration Language

Each **meta-class** can be **annotated** with one or more **platform dependency constraints** that exist within a **platform dependency ontology**

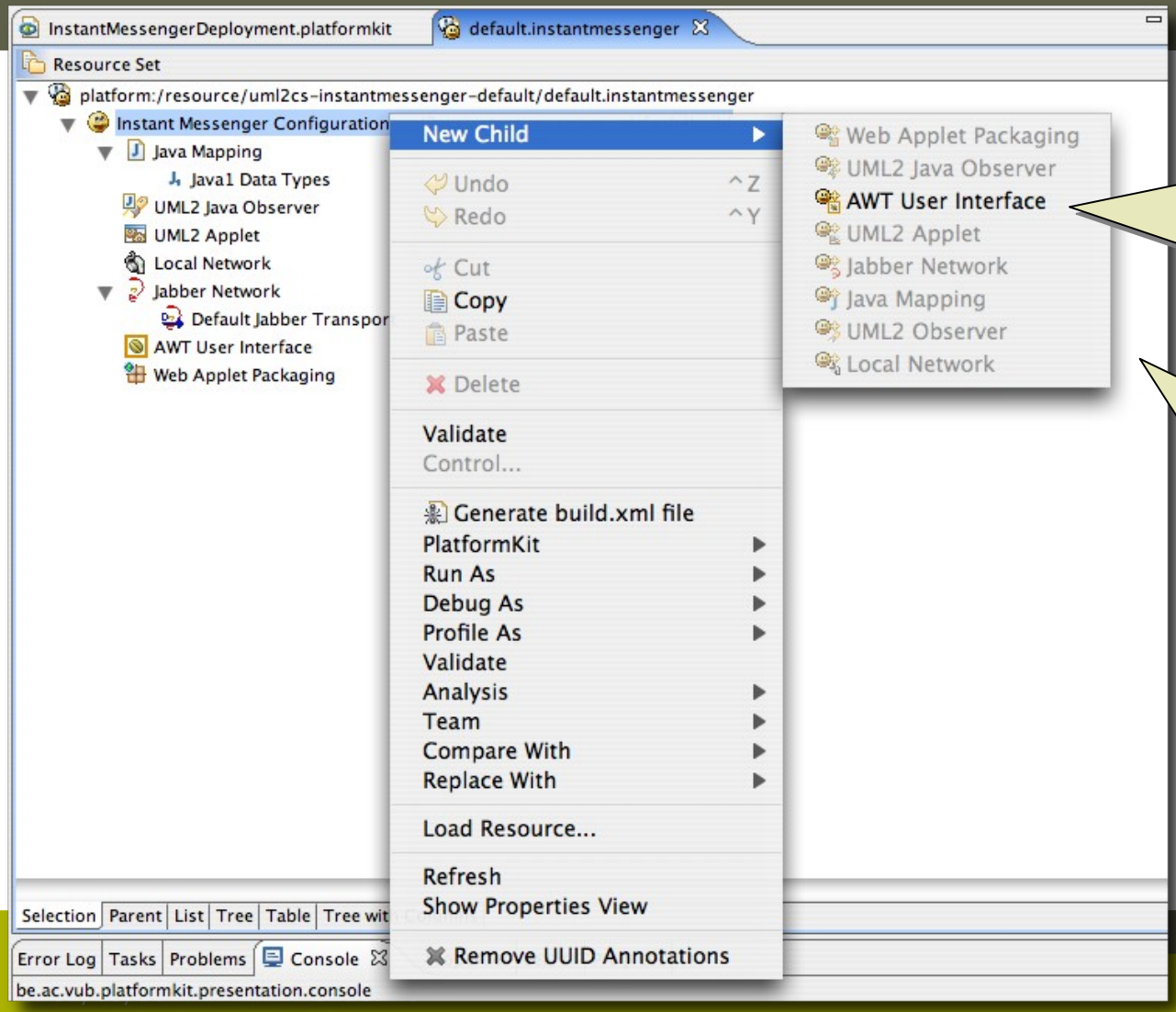


Platform-aware Configuration:
At development time



Limit the available configuration options to the ones valid for the target platform

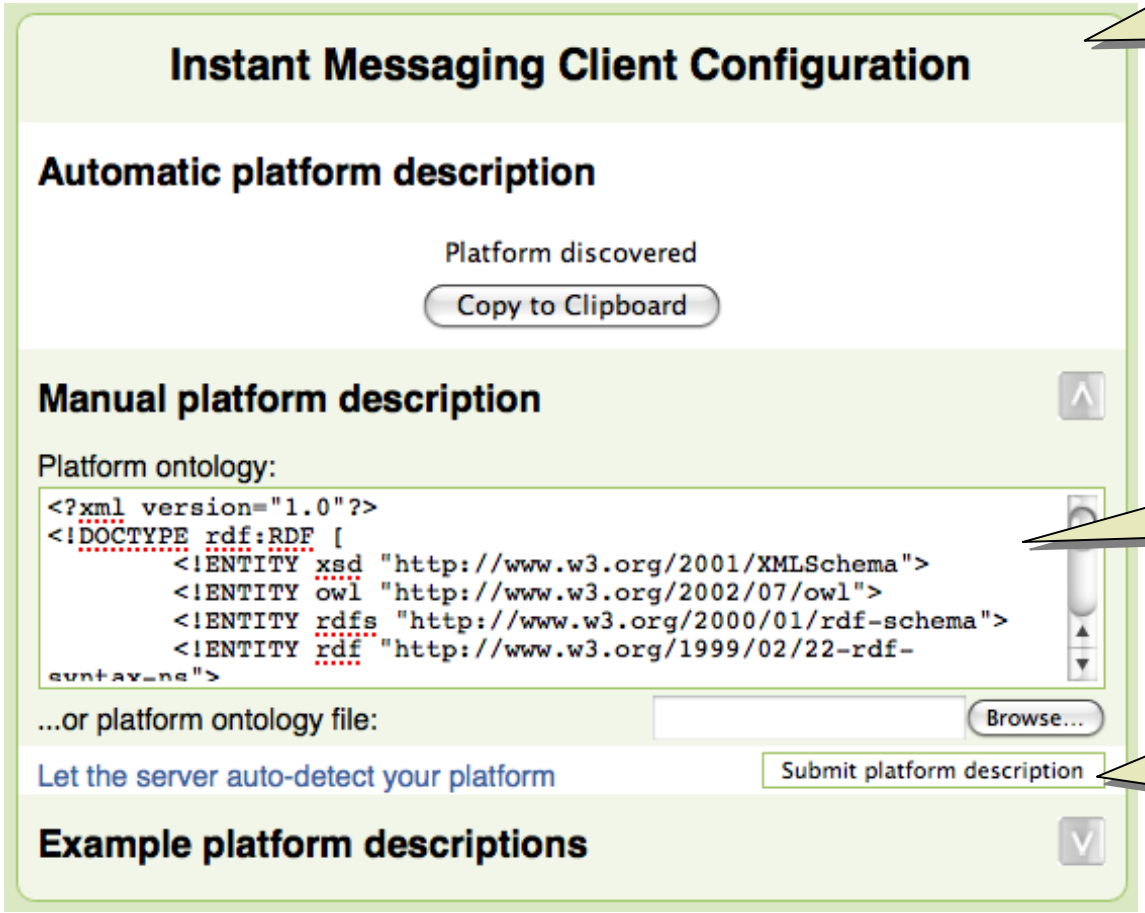
Platform-aware Configuration:
At development time



Limit the available configuration options to the ones valid for the target platform

Sort the remaining configuration options *most-specific-first*

Platform-aware Configuration:
At deployment time



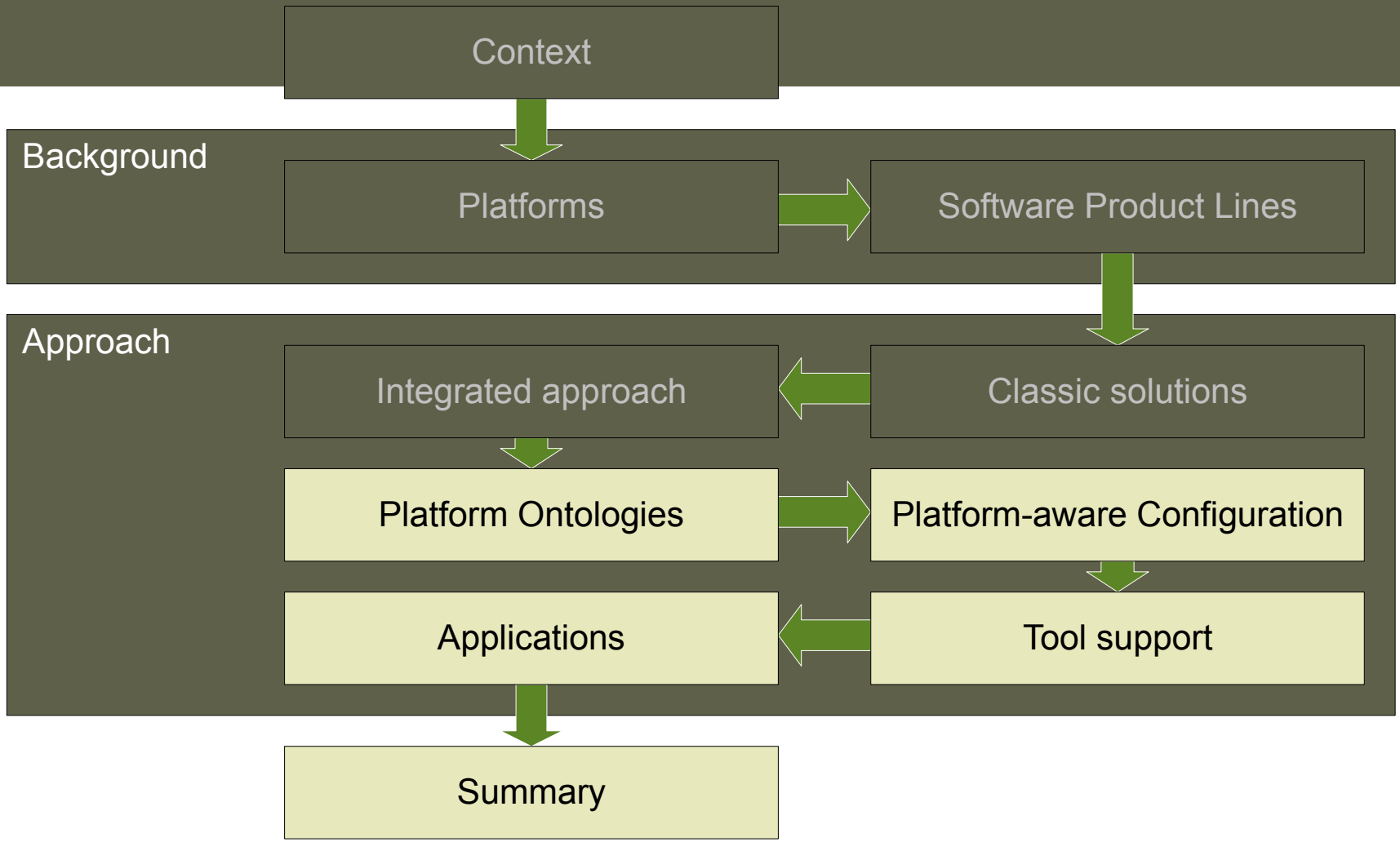
A number of pre-configured software products are made available for download

These products are sorted *most-specific-first*

The platform instance description of the client is provided

The most appropriate software configuration is returned for download/installation

Outline

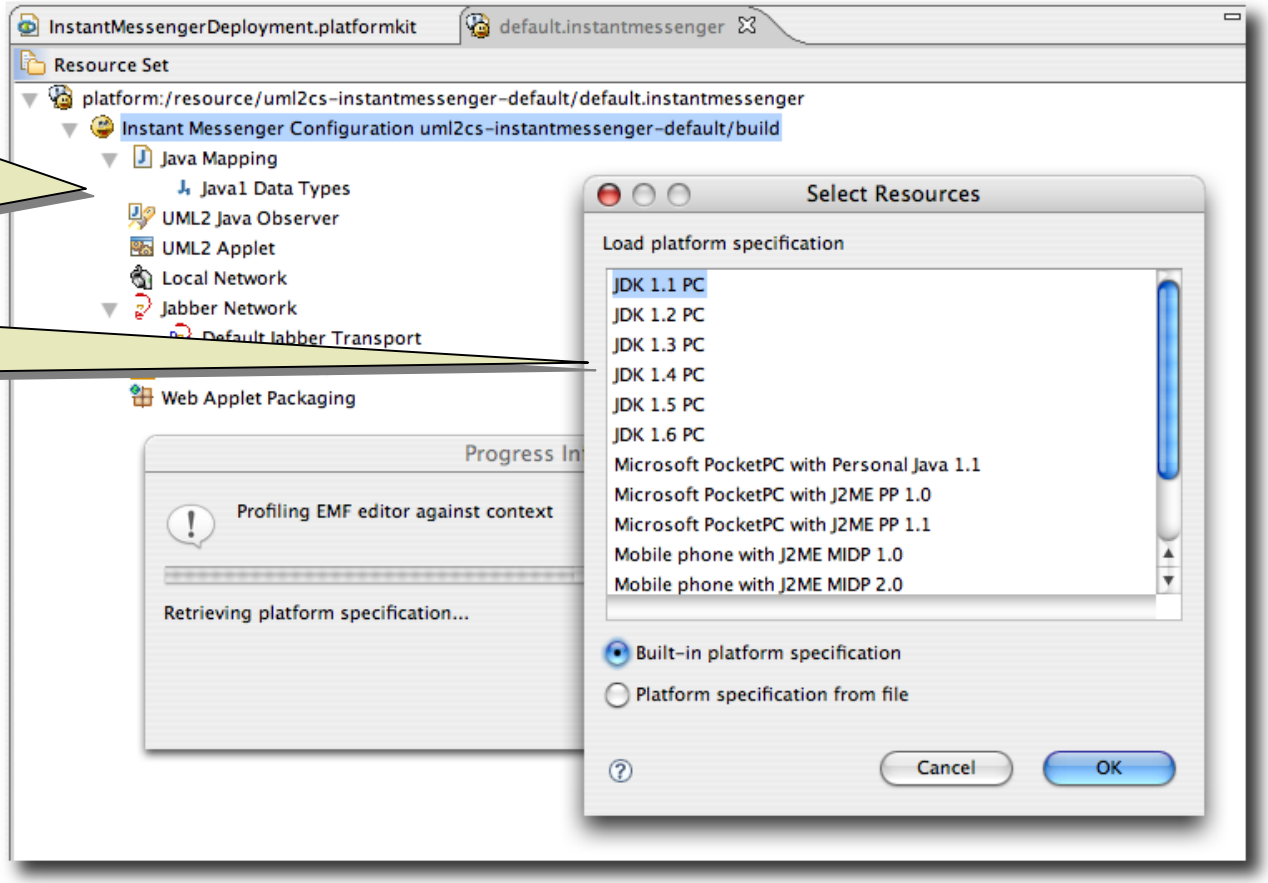


Tool support:
PlatformKit Eclipse plug-in

PlatformKit tool attaches to a **configuration language editor**

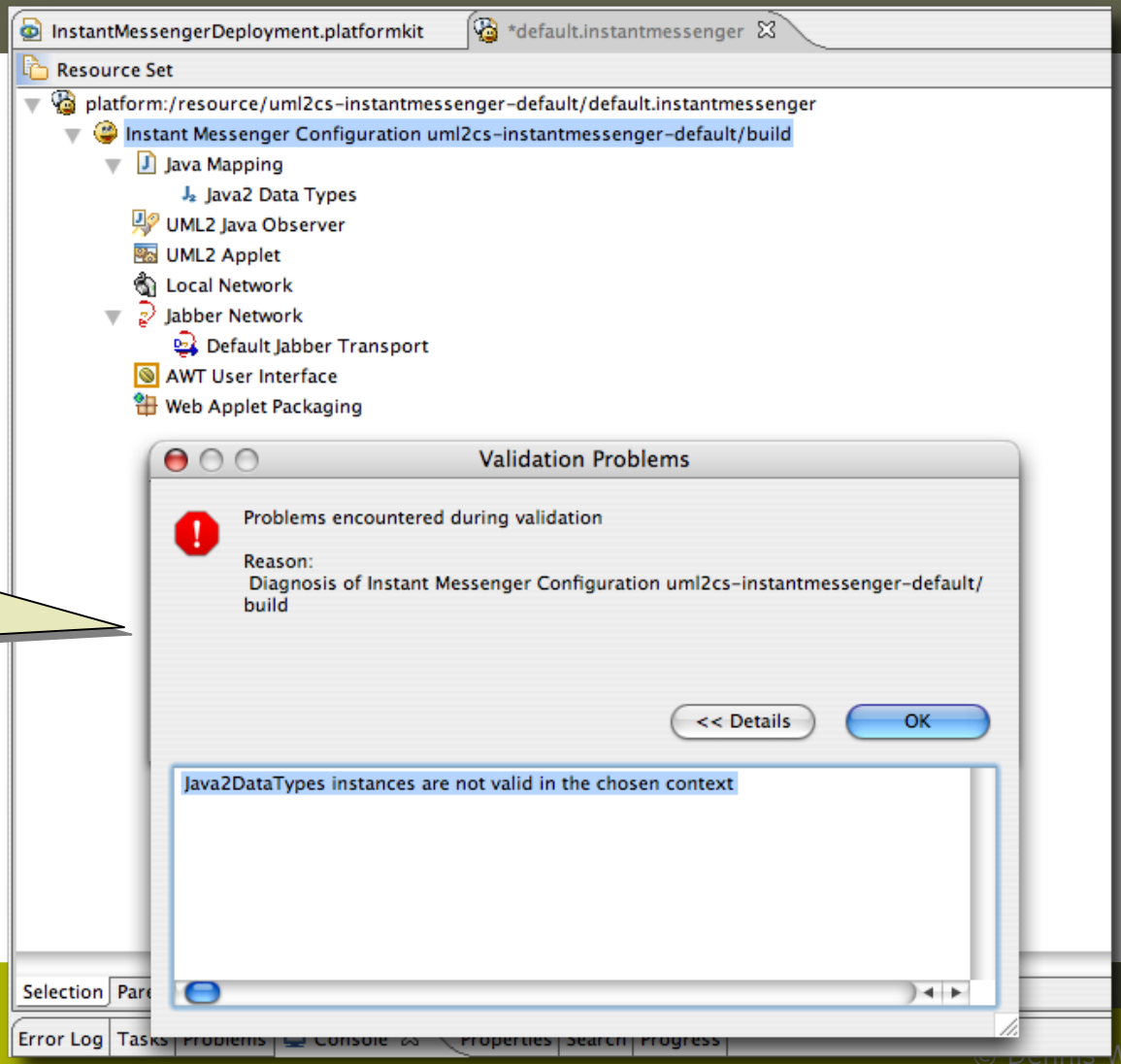
Several built-in prototype **platform models** are provided

Given a set of platform dependency constraints, PlatformKit can do its work



Tool support:
PlatformKit Eclipse plug-in

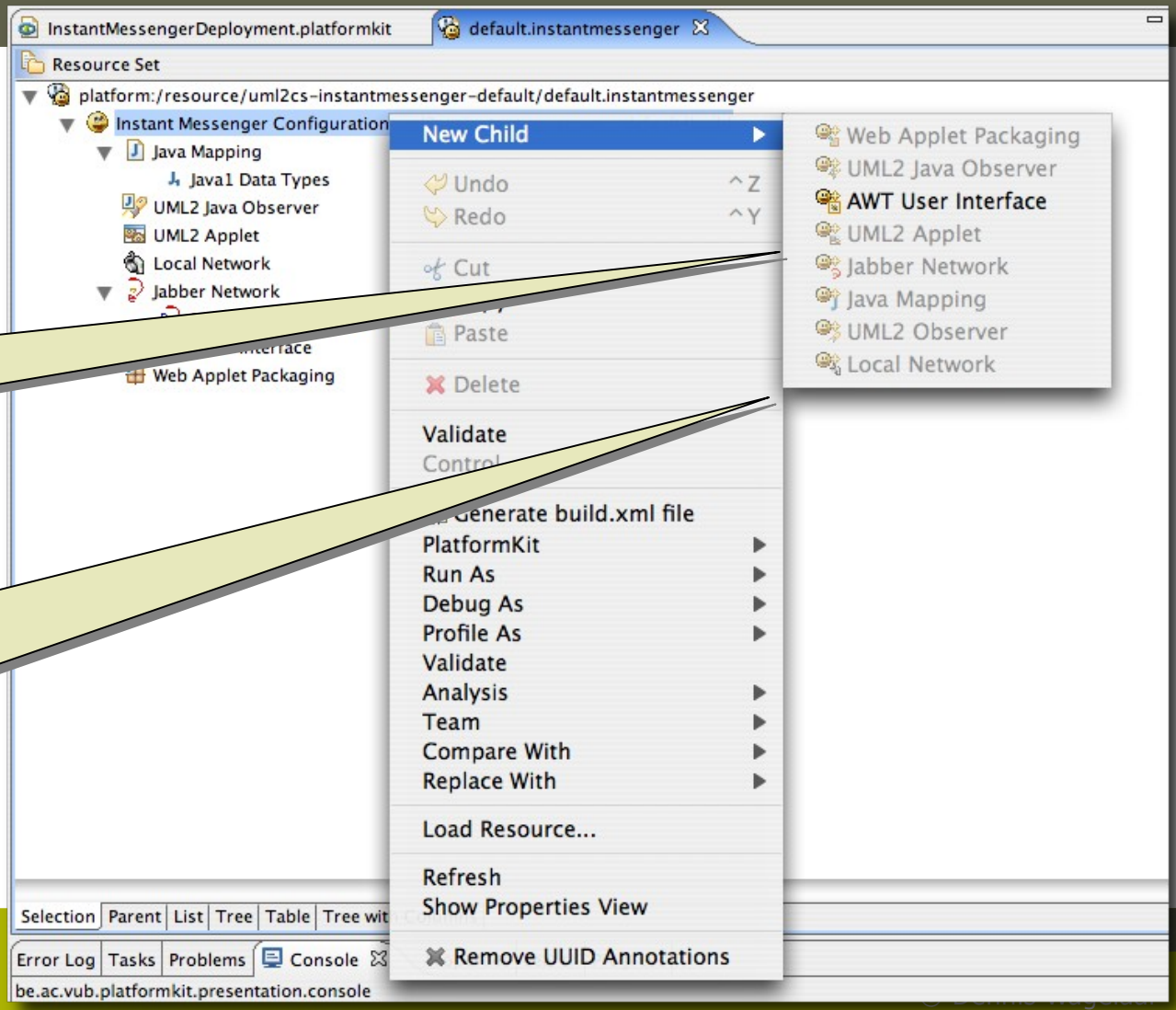
Configuration models can now be **validated** against the selected platform



Tool support:
PlatformKit Eclipse plug-in

Invalid choices are **removed** from the configurations options

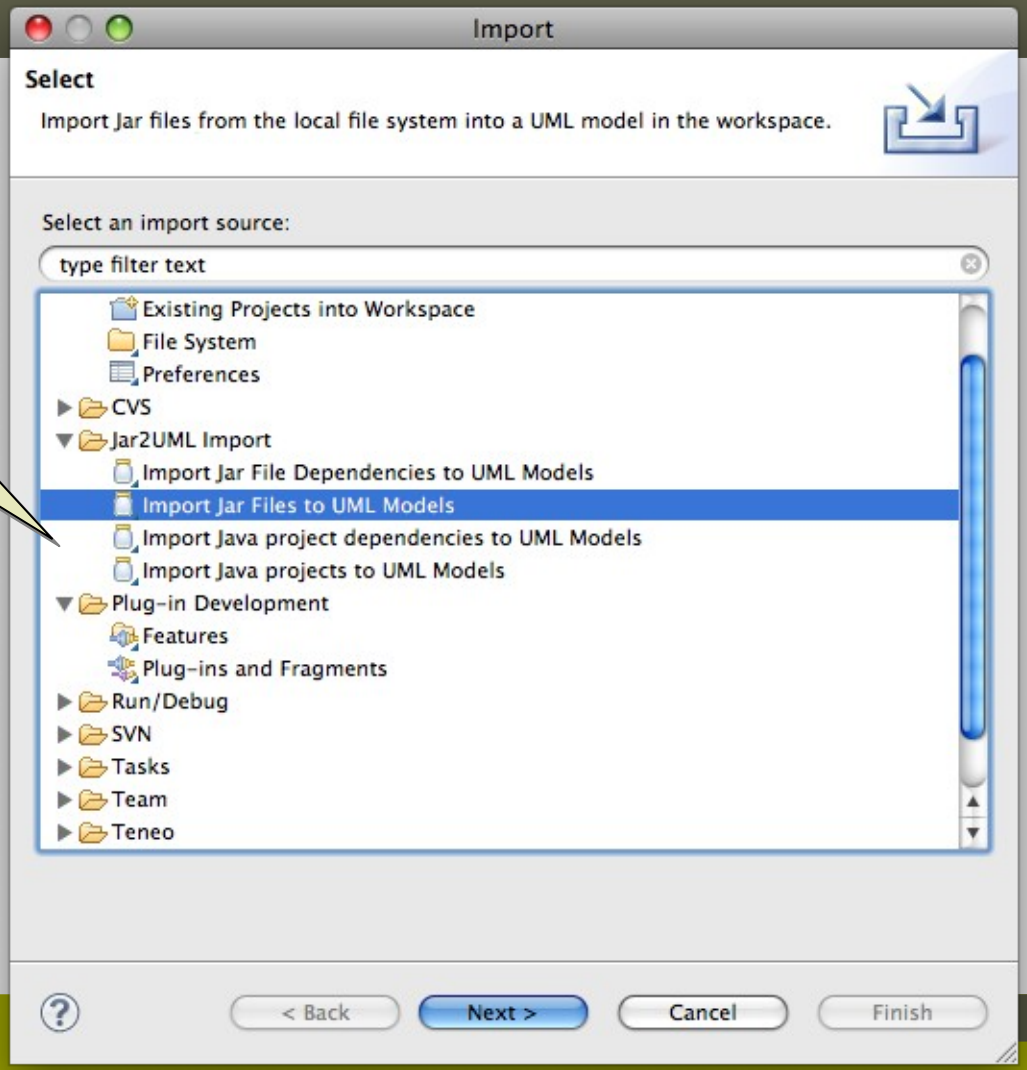
Configurations options are sorted **most-specific-first** → Optimise for maximum or minimum platform dependencies



Tool support:
PlatformKit Eclipse plug-in

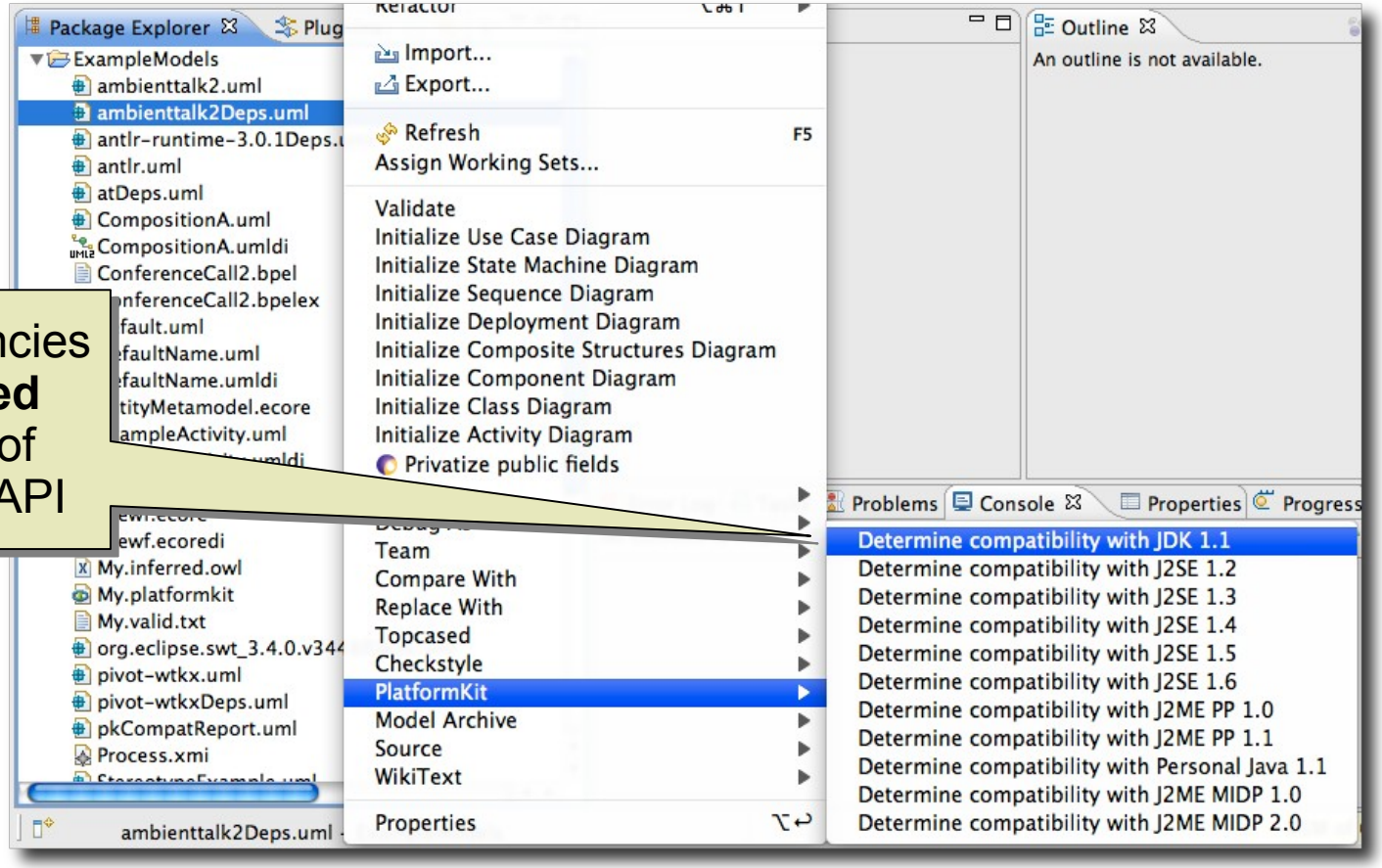
UML model of **platform dependencies** can be automatically extracted from **Java bytecode**

Support for **defining** platform dependency constraints is still limited



Tool support:
PlatformKit Eclipse plug-in

Platform dependencies can be **compared** against models of existing platform API

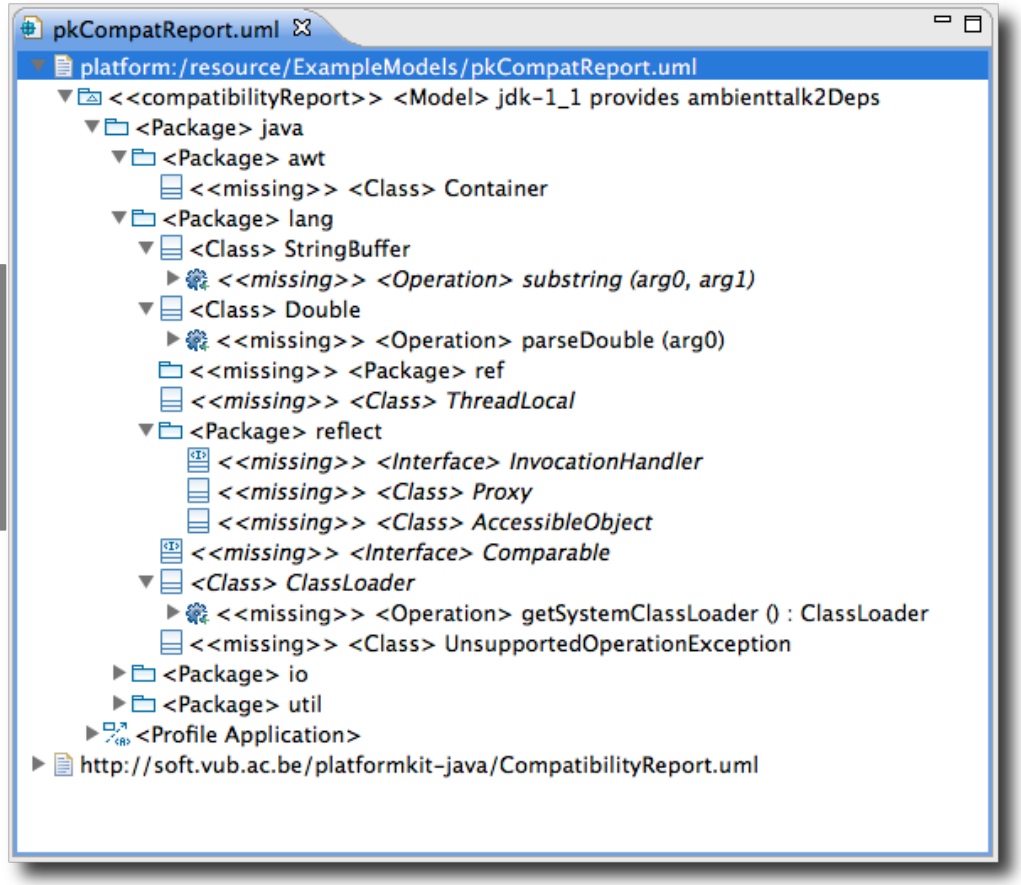


Tool support:
PlatformKit Eclipse plug-in

Missing: possibility to **automatically** derive the platform dependency constraints **in OWL DL**

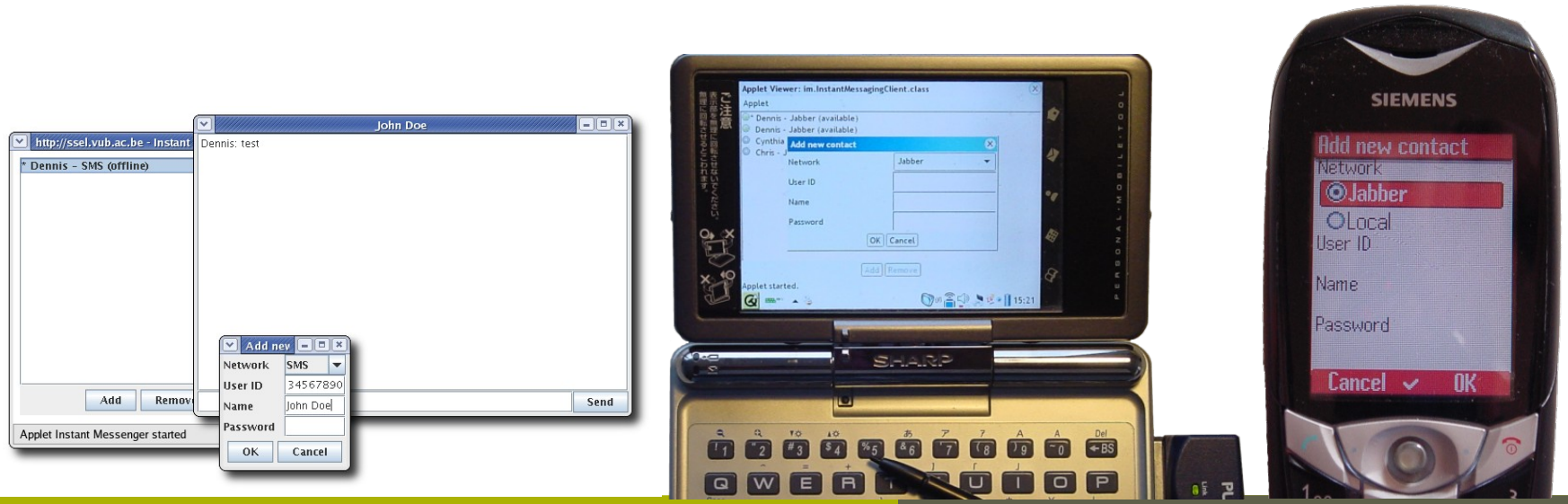
Missing: possibility to **directly** derive the platform dependency constraints

A full **compatibility report** is generated, which helps to determine the precise platform dependencies



Applications:
Instant messenger case study

- Instant messaging client
  - 11 PIM-to-PSM refinement transformations
  - One core PIM and 7 optional feature PIMs
  - Targets all Java client platforms

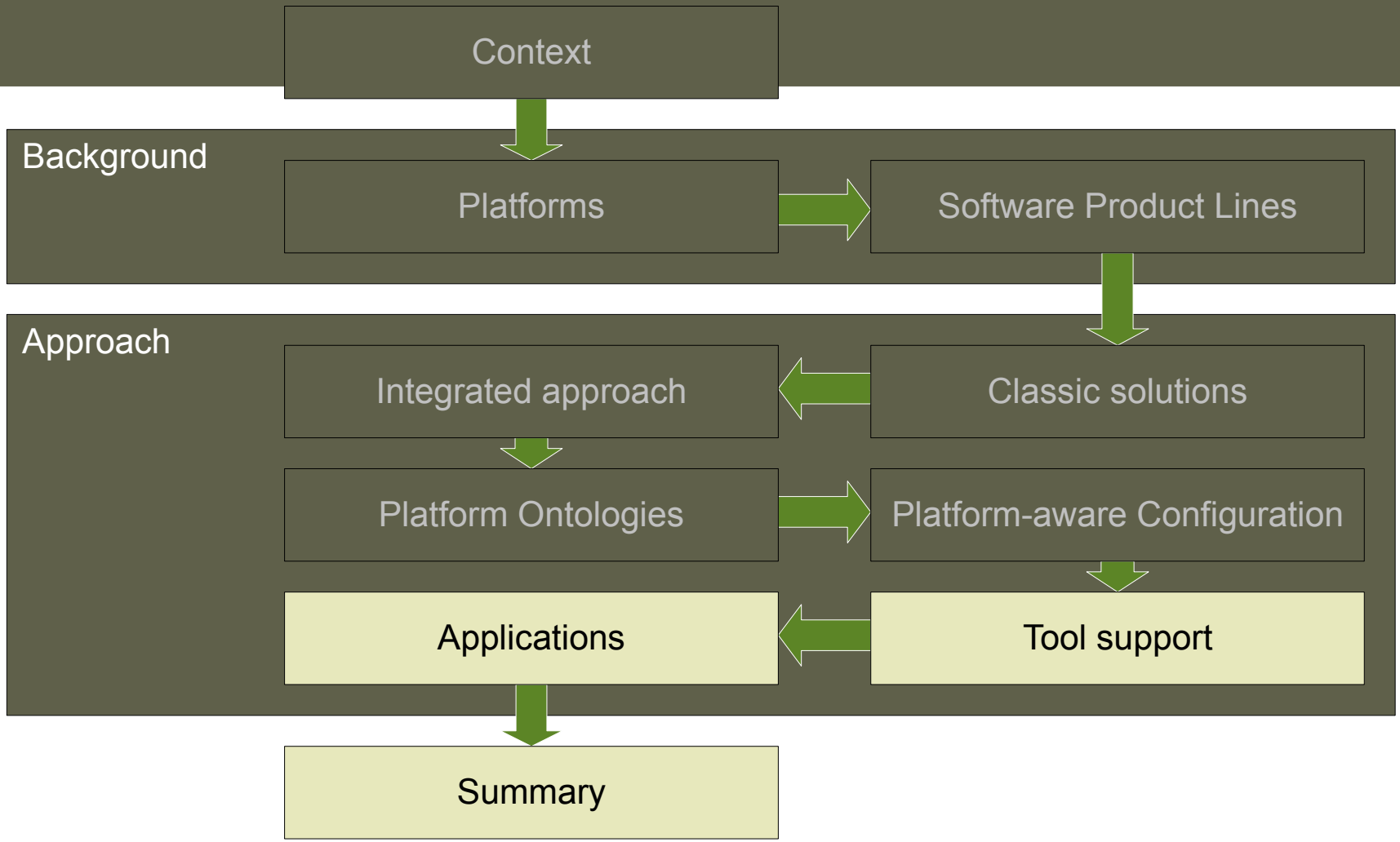


Applications:
Industry case studies

- Java EE case study with Inno.com (IWT OZM)
  - Introduce Platform Variability in the back-end of a medical imaging storage system
- Data interchange case study with Steria Belgium*
  - Apply Platform Variability approach to the adaptation of data translators for the required external data format
  - In the context of the VariBru project
- Telecom case study with Alcatel*
  - Apply Platform Variability approach to heterogeneous client devices
  - Introduce Platform Variability in telecom back-end systems

** Under consideration*

Outline



Summary:
Reasons for Platform Variability

- Platform Variability goes against the purpose of a platform
  - A platform was meant to be a "sound and stable basis"!
  - A platform provides a means for reuse!
- But from the viewpoint of a software product built on the platform, platform variation may be necessary
  - New platform version
  - Other (3rd party) platform is also relevant or even better
  - Old platform becomes obsolete
  - ...

Summary:
Solutions for Platform Variability

- Typically take the form of an **abstraction layer**:
  - Application of Design Patterns (e.g. Abstract Factory)
  - Cross-platform interpreter/virtual machine (e.g. Java)
  - Transformation-based approach (e.g. MDA)
- None of the classic solutions solves all our problems
  - But each solution has strong and weak points!
- Platform Variability can be tackled using a combination of these solutions
  - Integrated approach

Summary:
Management of Platform Dependencies

- As the number of reusable platform-specific building blocks increases, it becomes harder to remember for which platforms they are valid
  - We propose to make platform dependencies explicit through Platform Ontologies, expressed in OWL DL
    - Can record platform dependencies in general
    - Supports checking dependencies against specific platform instances
  - Analogous to SPLs, we can use Configuration Models to specify a platform-specific product configuration
    - Platform-aware configuration of software products
    - Platform-driven deployment of alternative software products

Summary:
Platform Variability tool support & applications

- PlatformKit supports:
  - Extracting platform dependencies from existing Java binaries
  - Platform-aware configuration
  - Platform-driven deployment
- Platform Variability applications:
  - Instant Messenger case study
  - Industrial case study with Inno.com (IWT OZM)
  - Industrial case studies under consideration with Steria Belgium (VariBru) and Alcatel

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/
- A. Kleppe, J. Warmer, W. Bast, *MDA Explained: The Model Driven Architecture : Practice and Promise* (2003)
  http://books.google.be/books?vid=ISBN032119442X
- S.J. Mellor, K. Scott, A. Uhl, D. Weise, *MDA Distilled: Principles of Model-Driven Architecture* (2004)
  http://my.safaribooksonline.com/0201788918
- D. Wagelaar, *Platform Ontologies for the Model-Driven Architecture* (2008)
  http://www.vubpress.be/article.aspx?article_id=PLATFO014U

Further reading:
Papers

- J. Coplien, D. Hoffman, D. Weiss, *Commonality and variability in software engineering*, IEEE Software **15**(6), pp. 37—45, 1998.
  http://doi.ieeecomputersociety.org/10.1109/52.730836
- T. Mens, P. Van Gorp, *A Taxonomy of Model Transformation.* Electr. Notes Theor. Comput. Sci. **152**, pp. 125—142, 2006.
  ftp://ftp.umh.ac.be/pub/ftp_infofs/2005/GraMOT-taxonomy.pdf
- D. Wagelaar and R. Van Der Straeten, *Platform Ontologies for the Model-Driven Architecture*, European Journal of Information Systems **16**(4), pp. 362-373, 2007.
  http://www.palgrave-journals.com/ejis/journal/v16/n4/abs/3000686a.html

Further reading:
Websites

- Software Product Lines website by BigLever:
  http://www.softwareproductlines.com
- Open Model CourseWare: http://www.eclipse.org/gmt/omcw/resources/
- VariBru - Variability in Software-Intensive Product Development:
  http://www.varibru.be/
- MDSE research within the Software Languages Lab:
  http://soft.vub.ac.be/soft/research/mdd
- PlatformKit tool: http://soft.vub.ac.be/soft/research/mdd/platformkit
- ENVY/Developer: http://c2.com/ppr/envy/
- OSGi reference: http://www.osgi.org/Specifications/Reference
- GNU Autoconf: http://www.gnu.org/software/autoconf/