Practical syntactic and extra-functional compatibility for black-box software components

Přemek Brada, Katedra informatiky a výpočetní techniky
FAV ZČU Plzeň

21.5.2012

Agenda

- Software Components: Origins and Core Concepts
- What is Software Component Anyway:
  From Concepts to Visualization
- Software Component Substitutability:
  What and How

- Service to People, Institution and Community
- Conclusions

Component Compatibility: Two streams in confluence

- What is Software Component
- How to Evaluate Substitutability

- The Roots and Manifestations of Practicality
  - software architecture research as response to growing complexity of real-world software
  - desire to provide answers that have both formal backing and near-term realistic applicability
  - mostly validating on current cutting edge technology

hmm, isn't this just low-hanging fruit?

Software Components: Origins and Core Concepts

Foundations (1)

black box

> Modularity + Information Hiding
  - separate compilation units
  - public interface / private implementation

> Assume-Guarantee
  - postconditions / functionality guaranteed
    <=> assumptions (preconditions / dependencies) valid

> Contract
  - syntax (signature)
  - semantics (ADT), behaviour (traces)
  - extra-functional (QoS)

```
set_hour (a_hour: INTEGER)
        -- Set `hour' to `a_hour'
    require
        valid_argument: a_hour >= 0
    do
        hour := a_hour
    ensure
        hour_set: hour = a_hour
    end
```

Foundations (2)

> Compositionality

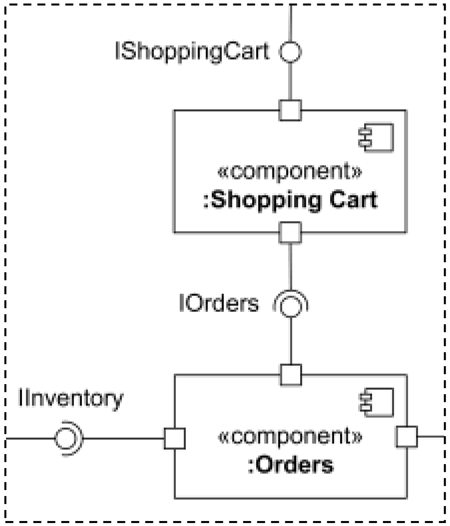
- [build systems so that] if property valid on parts
- it holds on the whole as well

> Software Architectures

- specify parts
- compose system by rules (styles)

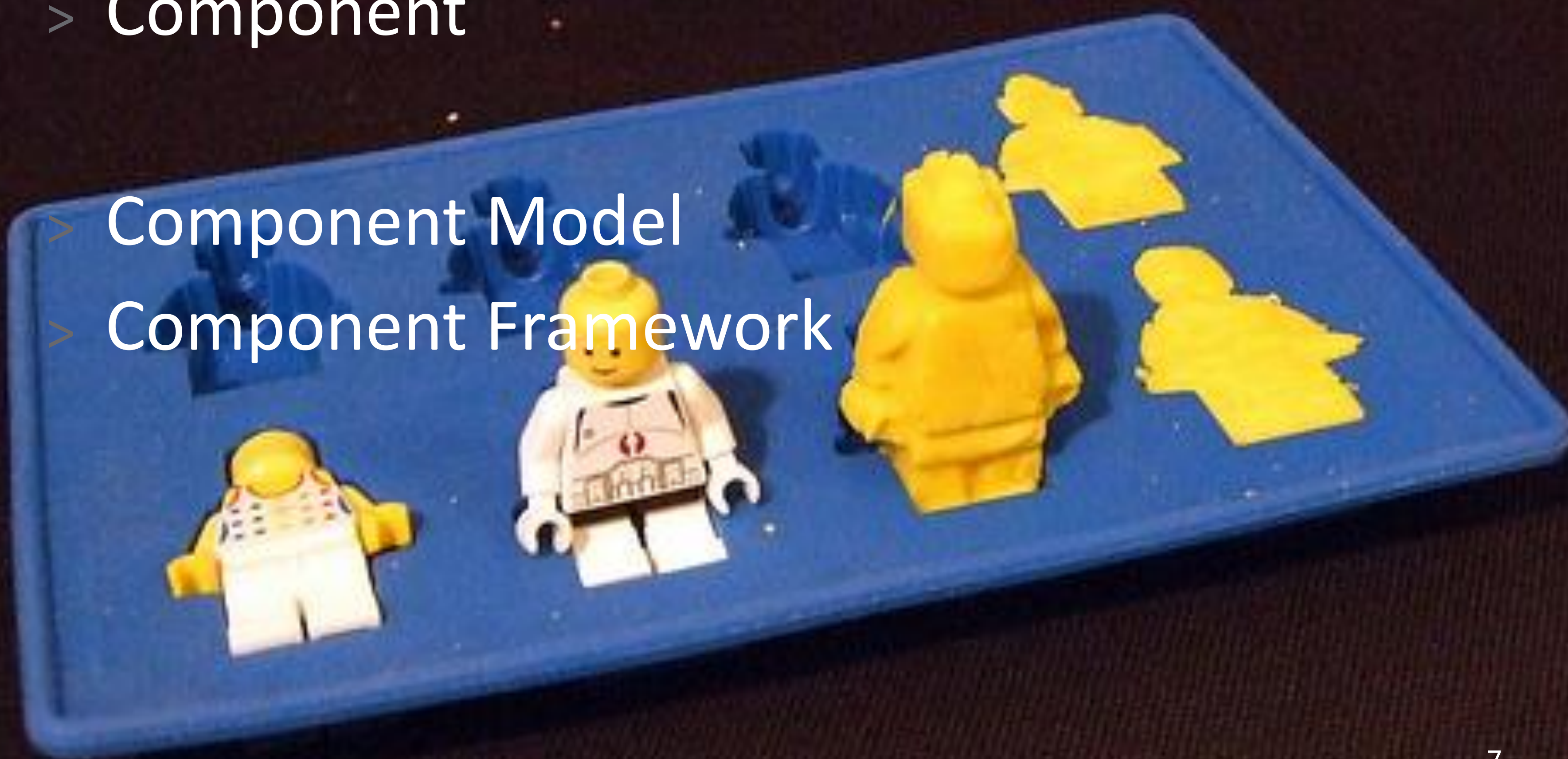
> Dependency Injection

- part declares dependency
- container injects supplier



Core Concepts

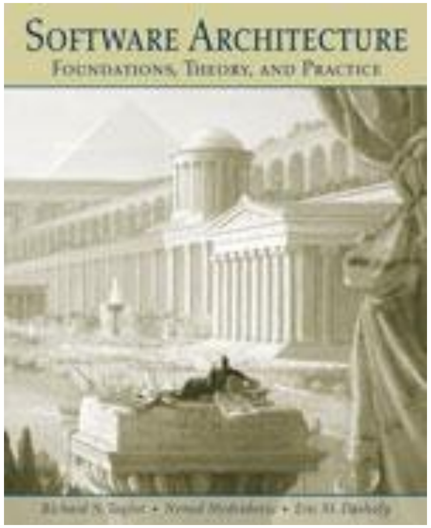
- Software Architecture
- Component
- Component Model
- Component Framework



http://www.flickr.com/photos/dorkomatic/5360309441/

Software Architecture

- "The set of principal design decisions made about a system."

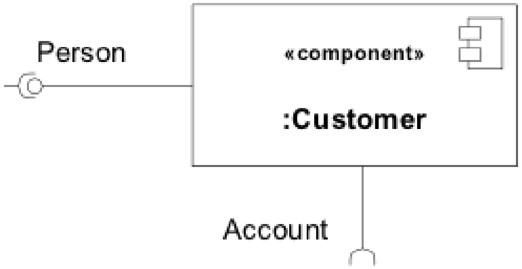


- Component
- Connector
- Composition rules / styles

Software Component

“A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.”

WCOP 1996 → Szyperski 2002



- Technical part
  - independence, contractual interfaces, composition
- Market-related part
  - third parties, [distribution and] deployment

Component Model

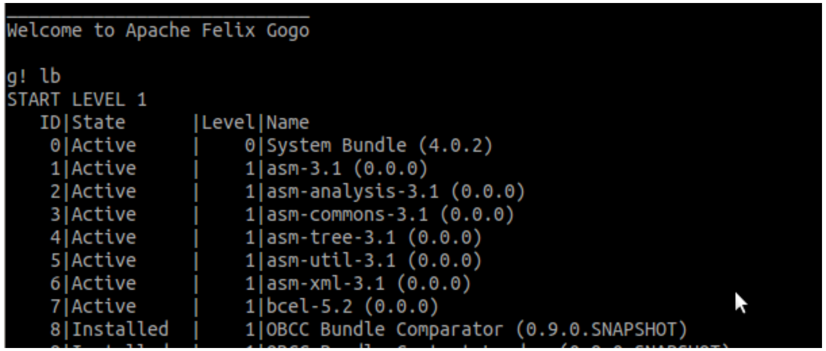
- “A component model specifies the standards and conventions imposed on developers of components.” [Bachmann 2000]
  - component types (incl. surface features)
  - interaction scheme
  - composition rules

Static x Dynamic

- “Compliance with a component model is one of the properties that distinguish components (…) from other forms of packaged software.“

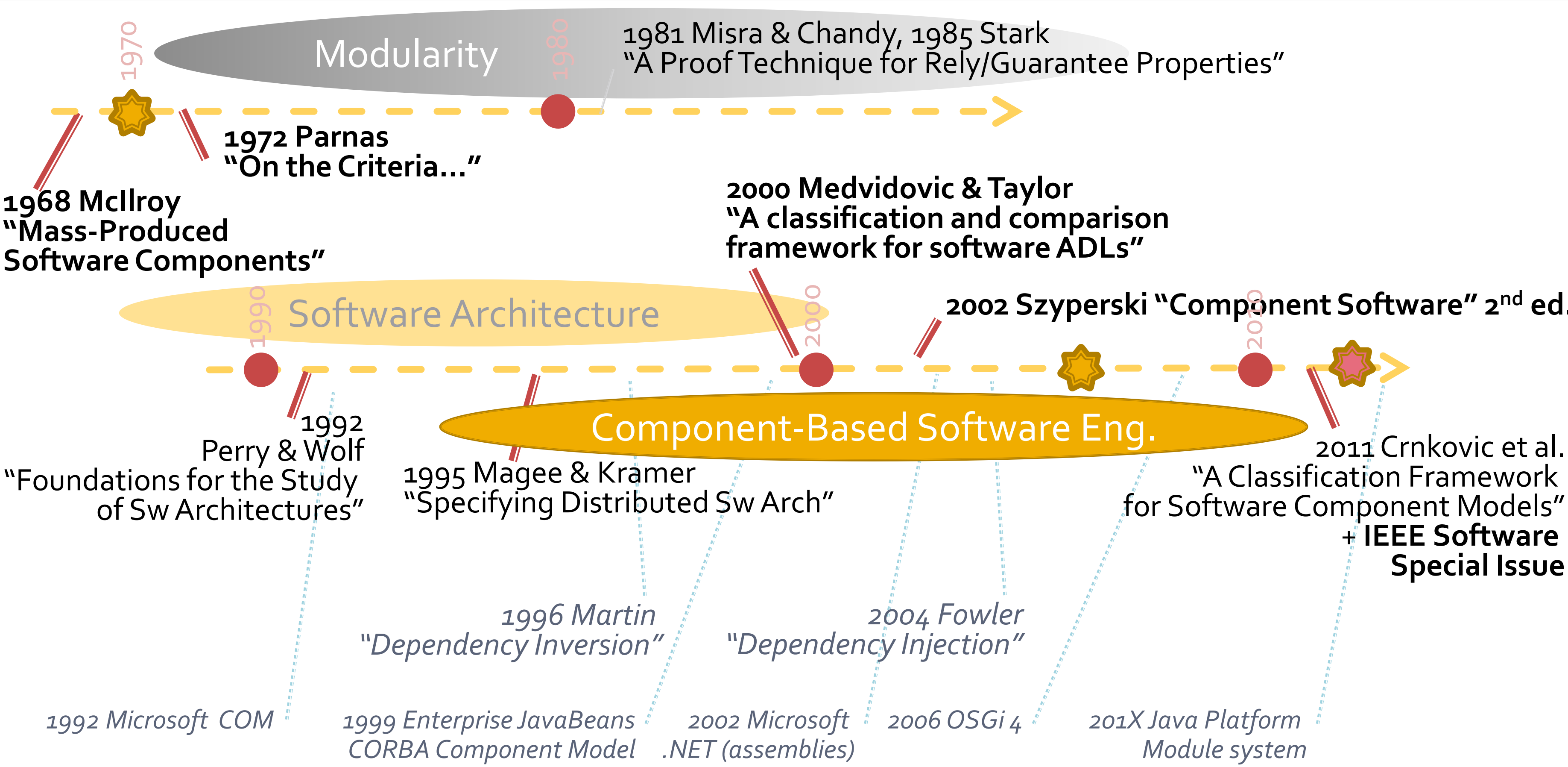
Component Framework

- "A component framework is an implementation of services that support or enforce a component model."
  - deployment
  - lifecycle
  - resource management



What is Software Component Anyway: From Concepts to Visualization

A Historical Perspective / Components



Motivation

- If we don’t understand concepts
  - we cannot communicate
  - things cannot be modeled
  - models cannot be manipulated and visualized
- If we don’t capture an aspect (of module interface)
  - we suffer from hidden dependencies
  - modularity breaks

http://www.flickr.com/photos/tobiasschlitt/2736153191/

Motivation: What Is a "Component"?

Szyperski book 2nd edition, preface

The endless struggle for perfect terminology

What is a software component? As with the first edition, this book has many pages on that fundamental question. It contains three different definitions that adopt different levels of abstraction: a first one is found at the very beginning of the original Preface; a second in Chapter 4; and a final one in Chapter 20. The existence of more than one definition in this book – and quite a few more cited from related work (see Chapter 11) – has led to some turbulence. … in their excellent book … the term "component" … for a brief discussion see

"CBSE is a coherent engineering practice, but we still haven't fully identified just what it is." [Brown, IEEE Software 1998]

Motivation: What Is a "Component"?

About 17 other definitions 1987-2007

A definition: software component

From the above characterization, the following definition can be formed:

> "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

This definition was first formulated at the 1996 European Conference on Object-Oriented Programming (ECOOP) as one outcome of the Workshop on Component-Oriented Programming (Szyperski and Pfister, 1997).

A Compo
- an op
- subje
- confo

components, ar

ic, *software compon*
and deployment th
pendence and bin
dors and robust in

element
indepen
according to a composition standard."

ose
der-
used
the
d in
this section includes *component model*, *component-based system*, *component*, and *binding*. A component itself is defined relatively to a specific component model [6].

A Definition

- We talk about deployable architectural components
- A piece of software called a component is
  - black-box (opaque) software element
  - with completely and precisely specified features crossing its encapsulation barrier (machine readable way)
  - 3rd party composable and deployable in ways not foreseeable by the developer
  - model conformant (features etc. not arbitrary)
  - + small enough to be reusable, large enough to be business interesting

Why black-box matters

- Software Engineering core concept:
  modules -> interfaces -> components
- Information hiding enforced

- Goals
  - prevent property leaks
  - localize change effects
  - make software comprehensible
  - make software composable, interchangeable

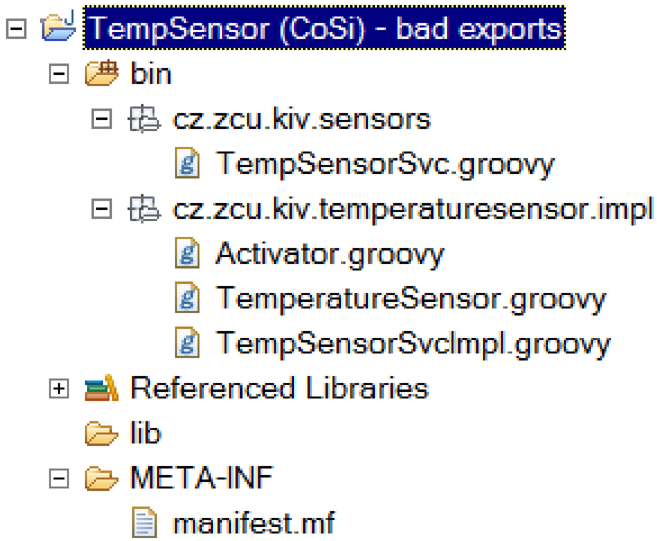
Case study: OSGi

```
Bundle-SymbolicName: org.openwms.core.integration
Bundle-Vendor: org.openwms
Bundle-Version: 0.0.1.SNAPSHOT
Export-Package: org.openwms.core.integration;version="0.0.
 uses:="org.openwms.core.domain,org.openwms.core.domai
 agement",org.openwms.core.integration.exception;version="
Import-Bundle: org.openwms.core.domain;version="[0.0.1.SN
 .SNAPSHOT]",com.springsource.javax.persistence;version="[
```

- Explicit required role
- In-completeness of specification
  - core: don’t declare services
  - declarative services: good & complete, not universal
- Weak specification-implementation consistency
  - core: package resolving only
- Moderate enforcement of black box
  - bind to declared packages and registered services only
  - class leaks from packages deprecated but easy

Outcome: The CoSi Component Model

- Component (bundle)
  - flat, dynamic model
  - OSGi-like packaging
  - Java, Groovy
- Rich base features
  - services, events, EFPs, etc.
- Container
  - simple run-time framework
  - no horizontal services (75kB .jar)
  - lifecycle interceptors, AOP and DI support



Guarantee of Conceptual Properties

- Enforce information hiding on surface
  - no export/import at runtime unless declared
- Prevent unresolvable dependencies
  - statically on install, update

```
public void start(BundleContext context) {
    Integer numValues = (Integer) context.getAttributeValue("sensor.nu
    sensor = new TemperatureSensor(numValues.intValue());
    service = context.getService("cz.zcu.fav.kiv.sensorregistry.Sensor
    service.registerSensor(sensor);

    TempSensorSvc tsvc = new TempSensorSvcImpl();
    context.registerService("cz.zcu.kiv.sensors.TempSensorSvc",tsvc);
}
```

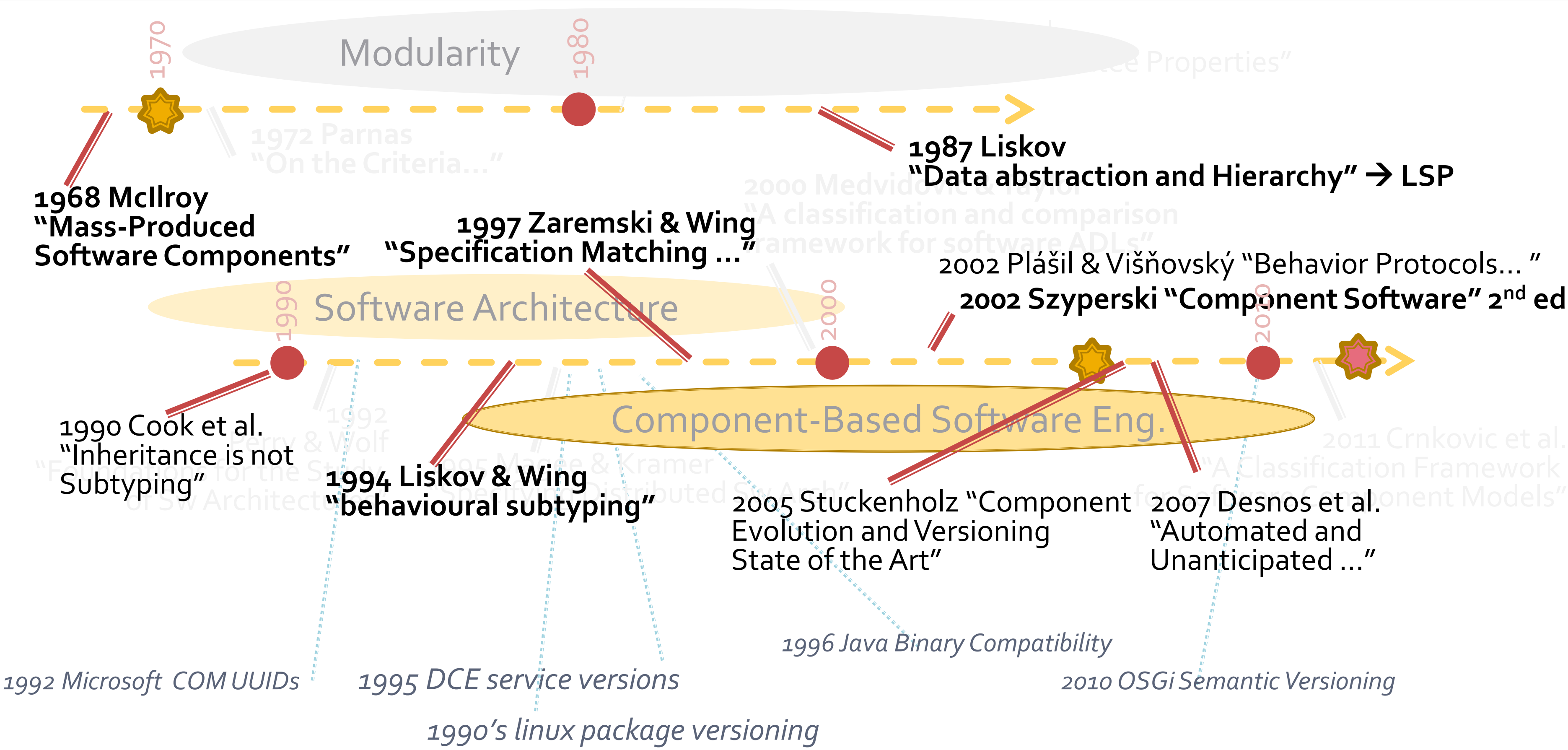
```
>install tempsensor-bad-exports.jar
>ps
Id      State           Name
-----------------------------------------------
0       Started         systembundle (1.0.0)
1       Started         messagebundle (1.0.0)
2       Started         simpleshell.jar (1.0.0)
3       Started         sensorregistry.jar (1.0.0)
4       Started         windspeedsensor.jar (1.0.0)
5       Started         temperaturesensor.jar (1.0.0)
6       Started         measuringstation.jar (1.0.0)
8       Installed       tempsensor-bad-exports.jar (1.0.0)
>start 8
Bundle tempsensor-bad-exports.jar cannot register service with type cz.zcu.kiv.s
ensors.TempSensorSvc since it is not exporting proper interface
>
```

Selected Published Papers

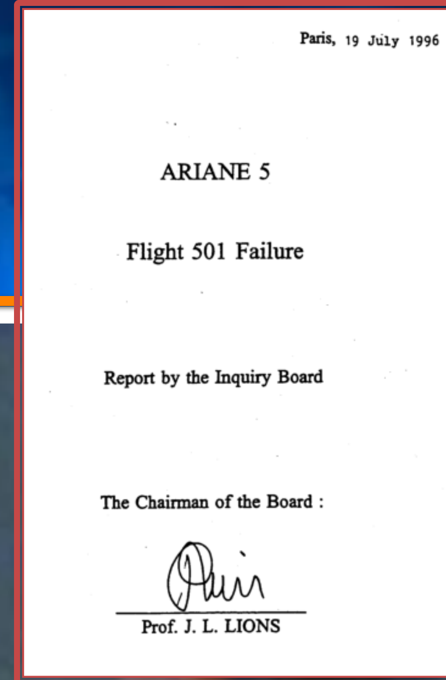
- Brada, Přemysl. *The CoSi Component Model: Reviving the Black-Box Nature of Components*. In Component-Based Software Engineering. Heidelberg : Springer, 2008, s. 318-333. ISBN: 978-3-540-87890-2
- Brada, Přemysl. *A Look at Current Component Models from the Black-box Perspective*. In 2009 35th Euromicro conference on software engineering and advanced applications. Los Alamitos : IEEE Computer Society, 2009, s. 388-395. ISBN: 978-0-7695-3784-9
- Šnajberk, Jaroslav; Brada, Přemysl. *ENT: A Generic Meta-Model for the Description of Component-Based Applications*. Electronic Notes on Theoretical Computer Science, 2011, vol.279, pp.59-73, ISSN 1571-0661
- Ježek, Kamil; Brada, Přemysl. 6th International Conference on Evaluation of Novel Approaches to Software Engineering – Revised Selected Papers, chapter *Formalisation of a Generic Extra-functional Properties Framework*. Accepted for publication in Communications in Computer and Information Science (CCIS), vol 275, ISSN: 1865-0929. Springer-Verlag, 2012.

Software Component Substitutability: What and How

A Historical Perspective / Compatibility



Motivation



Paris, 19 July 1996

ARIANE 5

Flight 501 Failure

Report by the Inquiry Board

The Chairman of the Board :

Prof. J. L. LIONS

On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure. Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded.

…

3.1 FINDINGS

…

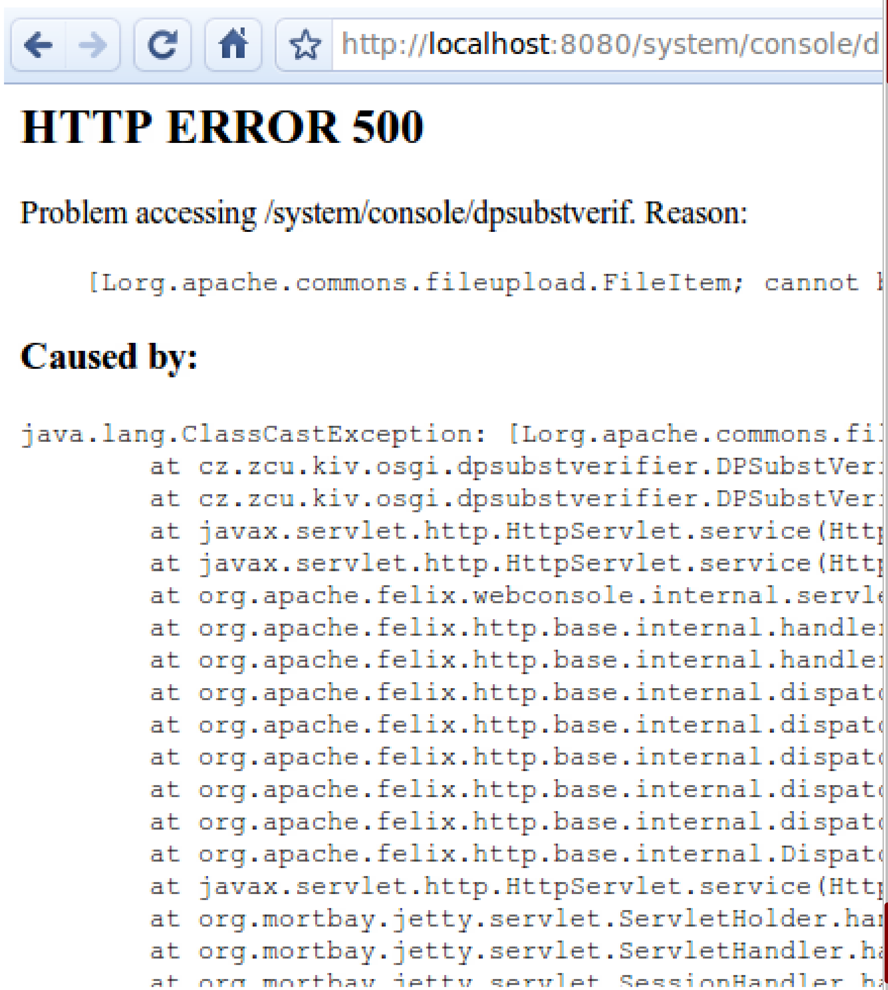
m) The inertial reference system of Ariane 5 is essentially common to a system which is presently flying on Ariane 4. (...) [Its] realignment function (…) was [retained and allowed] to operate for approx. 40 seconds after lift-off.

...

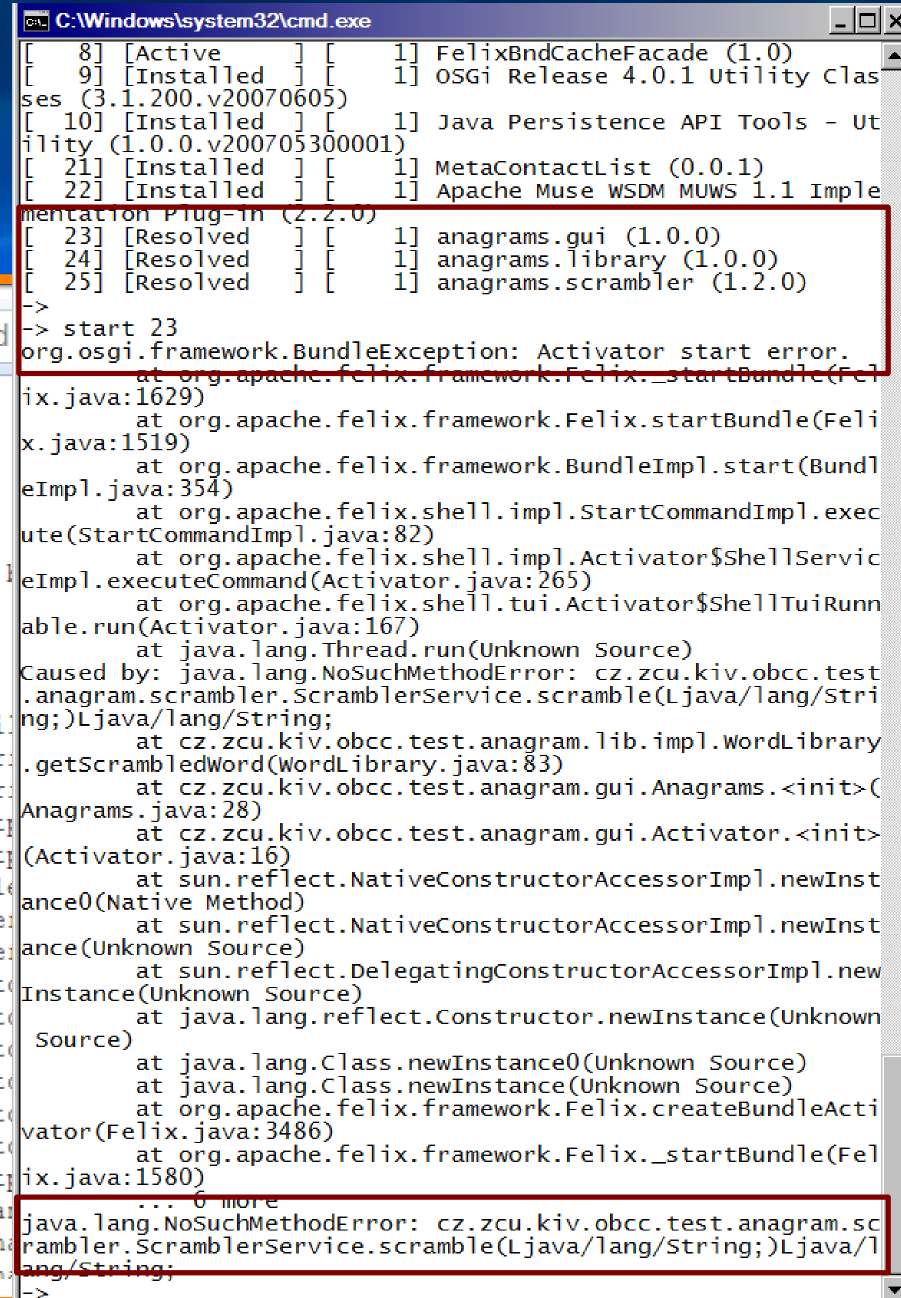
p) Ariane 5 has a high initial acceleration and a trajectory which leads to a build-up of horizontal velocity which is five times more rapid than for Ariane 4. [This generated] the excessive value which caused the inertial system computers to cease operation.

http://www.esa.int/esapub/bulletin/bullet89/dalma89.htm

Motivation



http://localhost:8080/system/console/d
HTTP ERROR 500
Problem accessing /system/console/dpsubstverif. Reason:
[Lorg.apache.commons.fileupload.FileItem; cannot
Caused by:
java.lang.ClassCastException: [Lorg.apache.commons.fi
at cz.zcu.kiv.osgi.dpsubstverifier.DPSubstVer
at cz.zcu.kiv.osgi.dpsubstverifier.DPSubstVer
at javax.servlet.http.HttpServlet.service(Htt
at javax.servlet.http.HttpServlet.service(Htt
at org.apache.felix.webconsole.internal.servl
at org.apache.felix.http.base.internal.handle
at org.apache.felix.http.base.internal.handle
at org.apache.felix.http.base.internal.dispat
at org.apache.felix.http.base.internal.dispat
at org.apache.felix.http.base.internal.dispat
at org.apache.felix.http.base.internal.dispat
at org.apache.felix.http.base.internal.dispat
at org.apache.felix.http.base.internal.Dispat
at javax.servlet.http.HttpServlet.service(Htt
at org.mortbay.jetty.servlet.ServletHolder.ha
at org.mortbay.jetty.servlet.ServletHandler.h



C:\Windows\system32\cmd.exe
[8] [Active] [1] FelixBndCacheFacade (1.0)
[9] [Installed] [1] OSGi Release 4.0.1 Utility Classes (3.1.200.v20070605)
[10] [Installed] [1] Java Persistence API Tools - Utility (1.0.0.v200705300001)
[21] [Installed] [1] MetaContactList (0.0.1)
[22] [Installed] [1] Apache Muse WSDM MUWS 1.1 Implementation Plug-in (2.2.0)
[23] [Resolved] [1] anagrams.gui (1.0.0)
[24] [Resolved] [1] anagrams.library (1.0.0)
[25] [Resolved] [1] anagrams.scrambler (1.2.0)
->
-> start 23
org.osgi.framework.BundleException: Activator start error.
at org.apache.felix.framework.Felix._startBundle(Felix.java:1629)
at org.apache.felix.framework.Felix.startBundle(Felix.java:1519)
at org.apache.felix.framework.BundleImpl.start(BundleImpl.java:354)
at org.apache.felix.shell.impl.StartCommandImpl.execute(StartCommandImpl.java:82)
at org.apache.felix.shell.impl.Activator$ShellServiceImpl.executeCommand(Activator.java:265)
at org.apache.felix.shell.tui.Activator$ShellTuiRunnable.run(Activator.java:167)
at java.lang.Thread.run(Unknown Source)
Caused by: java.lang.NoSuchMethodError: cz.zcu.kiv.obcc.test.anagram.scrambler.ScramblerService.scramble(Ljava/lang/String;)Ljava/lang/String;
at cz.zcu.kiv.obcc.test.anagram.lib.impl.WordLibrary.getScrambledWord(WordLibrary.java:83)
at cz.zcu.kiv.obcc.test.anagram.gui.Anagrams.<init>(Anagrams.java:28)
at cz.zcu.kiv.obcc.test.anagram.gui.Activator.<init>(Activator.java:16)
at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
at sun.reflect.NativeConstructorAccessorImpl.newInstance(Unknown Source)
at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(Unknown Source)
at java.lang.reflect.Constructor.newInstance(Unknown Source)
at java.lang.Class.newInstance0(Unknown Source)
at java.lang.Class.newInstance(Unknown Source)
at org.apache.felix.framework.Felix.createBundleActivator(Felix.java:3486)
at org.apache.felix.framework.Felix._startBundle(Felix.java:1580)
... 6 more
java.lang.NoSuchMethodError: cz.zcu.kiv.obcc.test.anagram.scrambler.ScramblerService.scramble(Ljava/lang/String;)Ljava/lang/String;
->

Component Compatibility: Challenges

- *“Independent composition by 3rd parties”*
  => need to check compatibility

- Very late binding
  - static architecture: deployment-time
  - dynamic architectures: run-time (or reconfiguration time)
- Can work with distribution form only
  - no sources
  - weak specifications
- Target platform ≠ development one
  - performance, access issues

Checking Compatibility: Options

- Formal – how much from contract to include
  - syntax → type checks
  - semantics (behaviour) → model checking
  - extra-func properties → function evaluation
- Informal – how good is the data
  - version numbers
  - compatibility meta-data

Terminology
- substitutability: A <-> B
- compatibility: A -> A'

Informal checks: Analysis of a high-profile OSS OSGi project

"When you only import packages and require some minimal or maximal version it means that the developer of the library has to do very good version management. If he changes an API and does not change the major number it can affect an already deployed application.

"With maven you already have the same problems it compile time but with OSGi it can crash at runtime. We need a whole set of new tools for this problem."

-- comment on Peter Kriens' blog, 6/2009

"Marketing" vs "Semantic" version IDs

Original	Semantic	Surface changes
1.0.1	1.0.0	n/a
1.0.3	1.0.1	(none)
1.0.4	1.0.2	(none)
1.2.0	2.0.0	modification
1.2.1	2.0.1	(none)
1.2.2	2.0.2	(none)
1.4.0	2.1.0	extension
1.4.1	2.2.0	extension

Type-Based Compatibility for Software Components

Reconciling

- formal strength
- practical aspects

(Semantic and Behavioural)

- intensive research
- “state explosion problem” → $O(e^n)$

Formal checks: Type-Based

> Assignment analogy

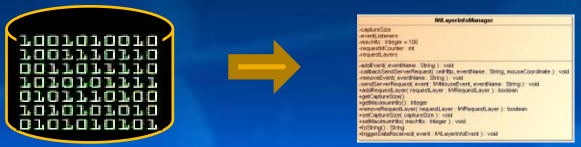
```
Vehicle v := (Car) ford;
```

- triggers type check
- dynamic type checking allows unforeseen subtypes

Instances of type $\boldsymbol{T_1}$ can be bound to variables declared to be of type $\boldsymbol{T}$ if $\boldsymbol{T_1 <: T}$

- short **<:** long
- Car **<:** Vehicle

Component as a Type



- Formalized (and simplified) model
- Captures
  - collection of elements (their types)
  - element role (provided, required)

- $C = (E^P, E^R)$; $E = \{ e_i \mid e=(n, T, r, o, a) \}$
  - declared component type
  - elements – name, type, role, opt, arity
- Instance $\gamma : C$; $\gamma.P \subseteq E^P$ etc.
  - not all elements may be always present

Type-Based Compatibility: Variations

- Strict
  - on component type representations
  - used for 1:1 any-time compatibility

- Contextual
  - consider actual use of component instance
  - include in comparison with *A′* or *B*

Strict Compatibility

- Standard subtyping

$A^r <: A^c$ when

$$\forall p^r \in A^r.P,\ \forall p^c \in A^c.P \quad \bullet \quad p^r <: p^c$$

$$\forall r^r \in A^r.R\ ,\ \forall r^c \in A^c.R \quad \bullet \quad r^r :> r^c$$

  - covariance for provided part
  - contravariance for required one

- "A substitute component should be usable whenever the current one was expected, without the client noticing it." [Wegner, Zdonik, 1988]

$e_i <: e_j$

$\Leftrightarrow$

$T_i <: T_j$

$r_i = r_j$

o_i vs o_j, a_i vs a_j depends on r

Subtype Checks: Expressing the Results

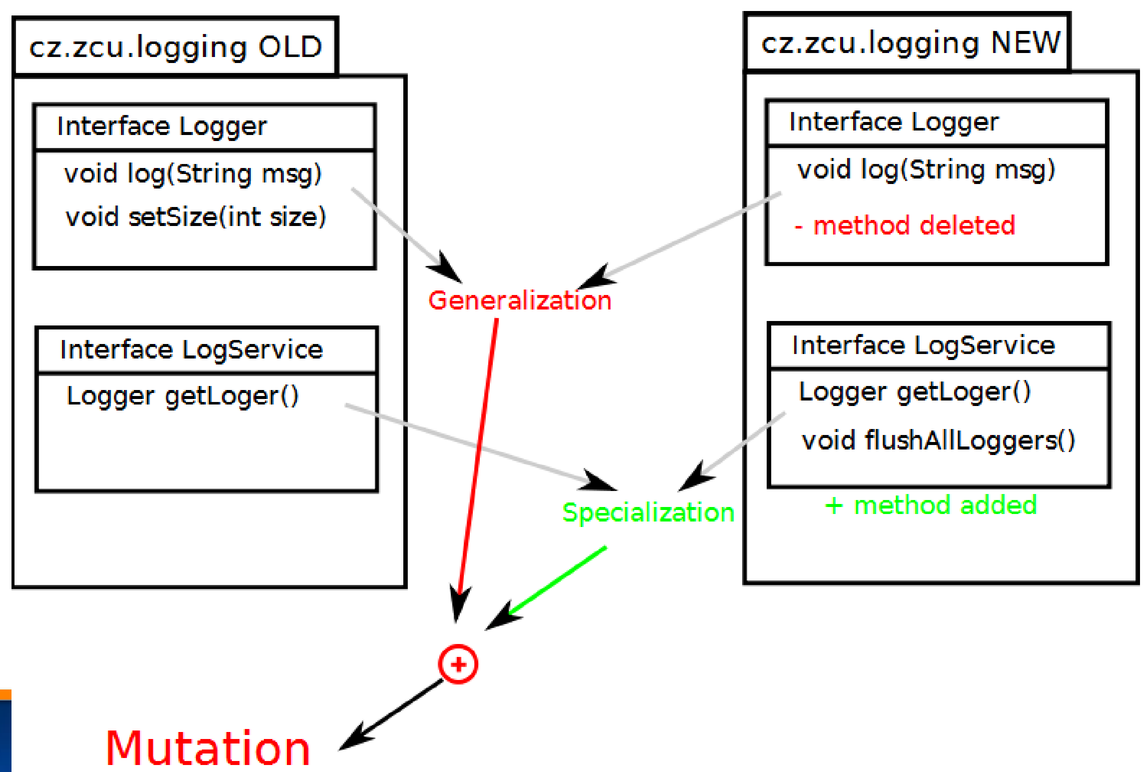
> ## Goal

- evaluate through recursion
- store for future reference
- read

$$diff(a,b) = spec \iff b <: a$$

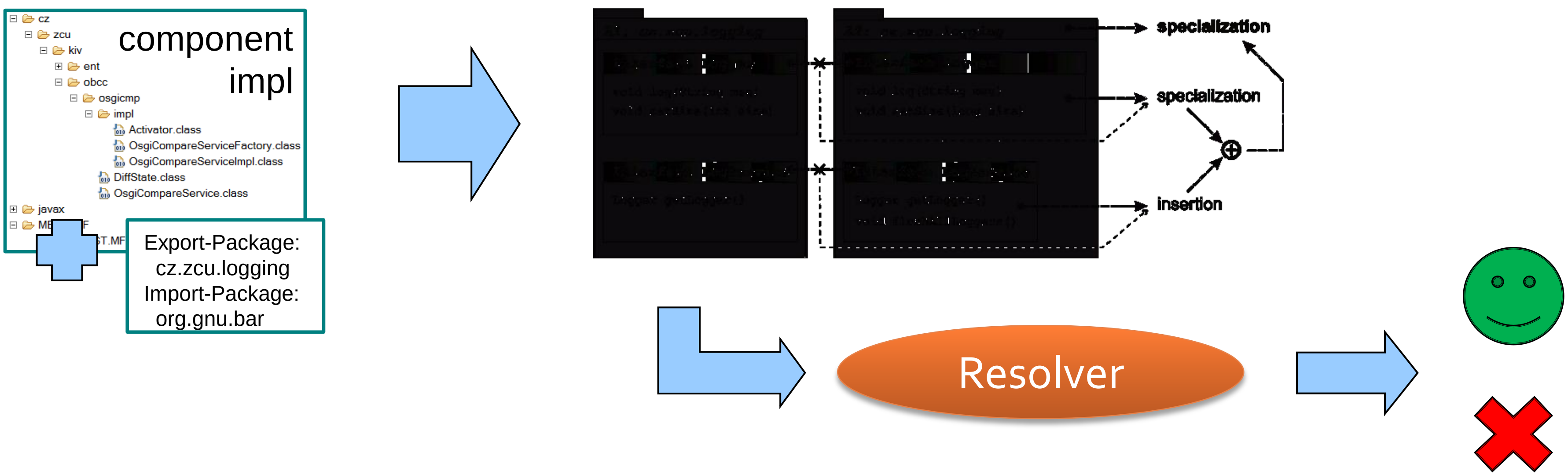
> ## Rules

- *none* $<$ *ins* | *del* $<$ *spec* | *gen* $<$ *mut*
- $ins \oplus del \rightarrow mut$, $ins \oplus spec \rightarrow spec$, …



Outcome: Type-Safe Component Updates

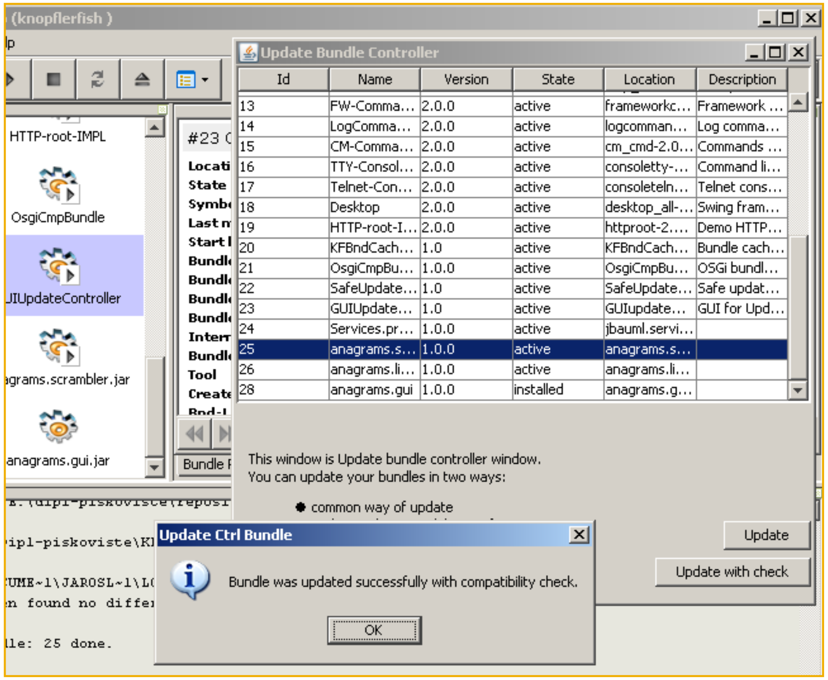
- Initial bindings and Updates that ensure application consistency
- Relation to framework checks (lifecycle)



Application and Measurements: OSGi Resolving Integration

	Null app	Parking	CoCoME
Meta-data	28	58	234
Type check	256	1345	10437

- Applications
  - Null = 2 bundles
  - Parking = 6 bundles
  - CoCoME = 15/37 bundles
- Felix resolver hook
- Desktop + Android implementation



Industrial Applications

- Luminis (NL)
  - conceptual cooperation
  - Apache ACE project enhancements
- Openmatics s.r.o.
  - strict compatibility for API + 3rd party OSGi application verification
  - simulation tests of extra-functional property limits

Type-Based Compatibility: Variations

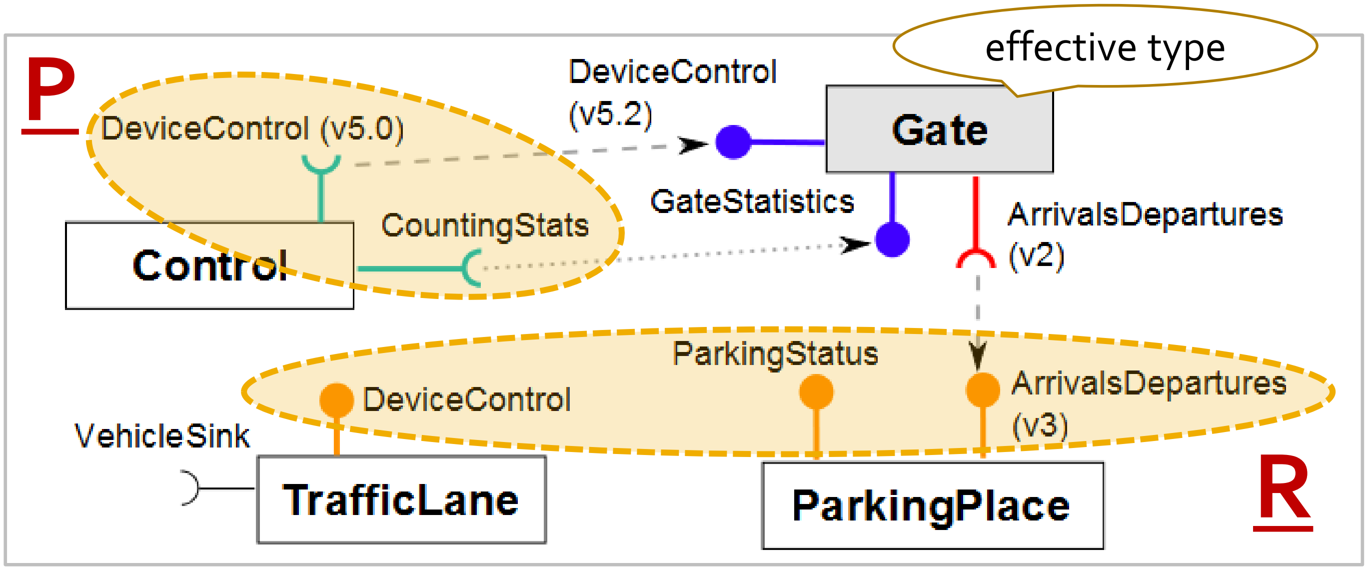
> Strict
  - on component type representations
  - used for 1:1 any-time compatibility

> Contextual
  - consider actual use of component instance
  - include in comparison with A' or B

> Prerequisites
  - deployment context as a type

Deployment Context as a Type

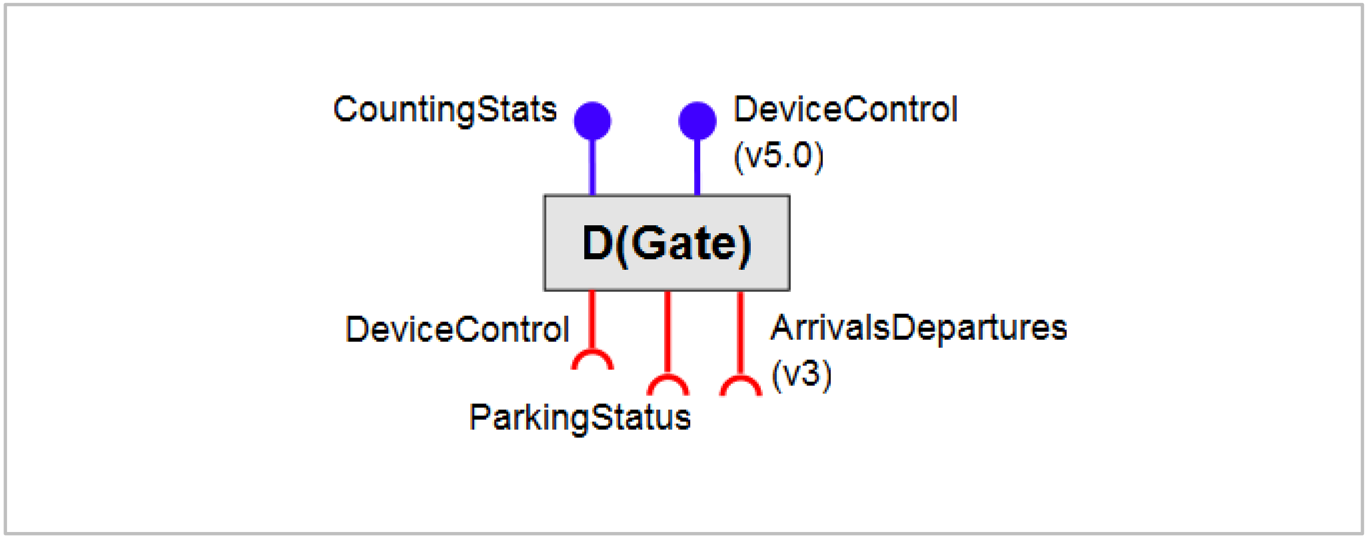
- Deployment Context (D) = rest of architecture
  - components, bindings (relations, actual types)
  - architectural consistency
- Contextual Complement of α:A = α's view of D as a type
  1. obtain actual usage
  2. invert



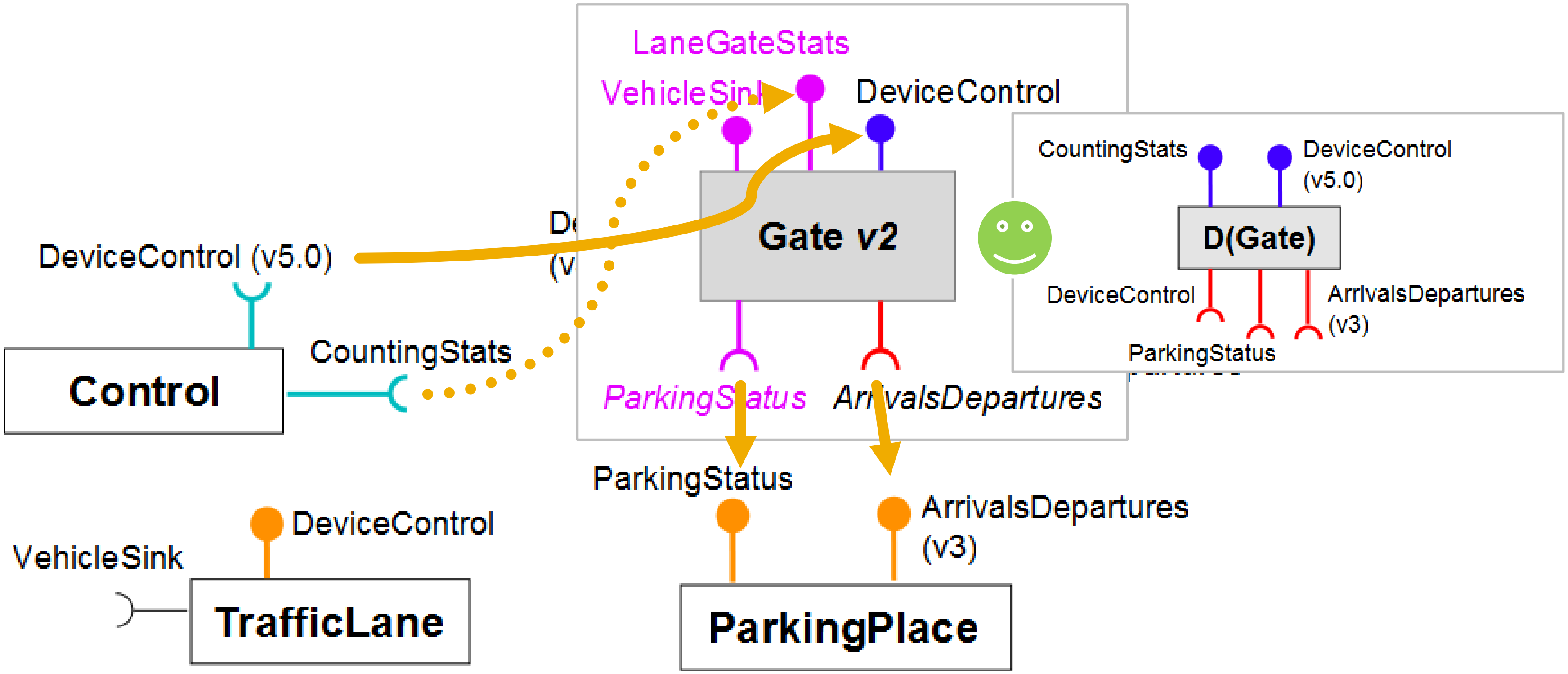
Deployment Context as a Type

- Deployment Context (*D*) = rest of architecture
  - components, bindings (relations, actual types)
  - architectural consistency
- Contextual Complement of α:*A* = α's view of *D* as a type

1. obtain actual usage
2. invert



Contextual Substitutability



$$\forall \underline{x}^p \in \underline{P}\ \exists p \in P \ \bullet\ p <: \underline{x}^p$$

$$\forall r \in R\ \exists \underline{x}^r \in \underline{R} \ \bullet\ \underline{x}^r <: r$$

Gate v2

$$\Rightarrow A^r <:_D A^c$$

Gate

Context and Substitutability Properties

- Contextual complement: may provide less, require more
  - $\underline{A}_D.\underline{P} \subseteq \alpha.P$ – not all provisions need be used, etc.
- Effective type & context time-dependent (!)

- Substitutability: strict ensures contextual in any context
  - $A' <: A$ will always fit into $D(\alpha)$

Type-Based Substitutability: Practical Considerations

- How to obtain replacement component type
  - binary package, e.g. bytecode [Bauml,Brada 2010]
  - E^r not included, obviously
- How to obtain context (-> complement)
  - query component framework
- Subtyping vs. language rules
  - e.g. Java binary compatibility

Compatibility of Extra-Functional Properties

EFPs = qualitative characteristics

- performance, resource consumption, reliability
- maintainability, security, usability

Motivation and challenge

- properties “same” but values “context dependent”
- lack of normalization (esp. in CBSE)
- rudimentary support beyond RT and HA domains

Closest model: Palladio

Based on Generic EFP Meta-Model (K.Ježek)

> Component-model independent

- generic meta-model
- primitive, complex and derived properties

  $E = \{e \mid e = (n, T, E_d, \gamma, M)\}$

- assignment and evaluation framework

> Usage context independent

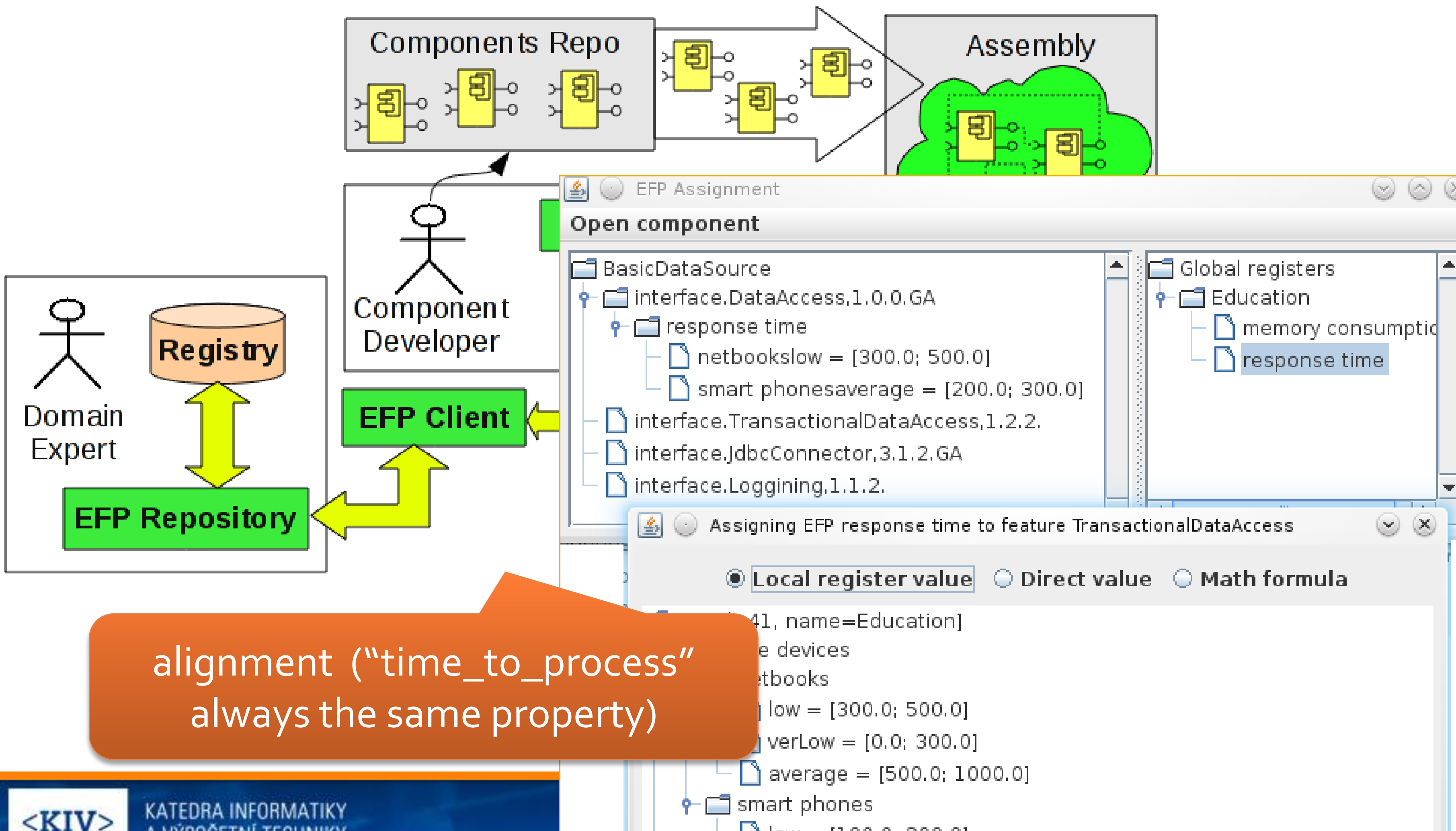
- declaration (type) – global repository

  ```
  (time_to_process, integer, {unit:‘‘ms’’, names: {low, avg, high}} )
  ```

- definition (value) – local repositories

  ```
  mobile/GPRS:      time_to_process: low = 10, high=5000, ...
                    data_transferred: low = 1, high=100, ...
  desktop/10GEth:   time_to_process: low = 1, high=250, ...
                    data_transferred: low = 1000, high=1000000, ...
  ```

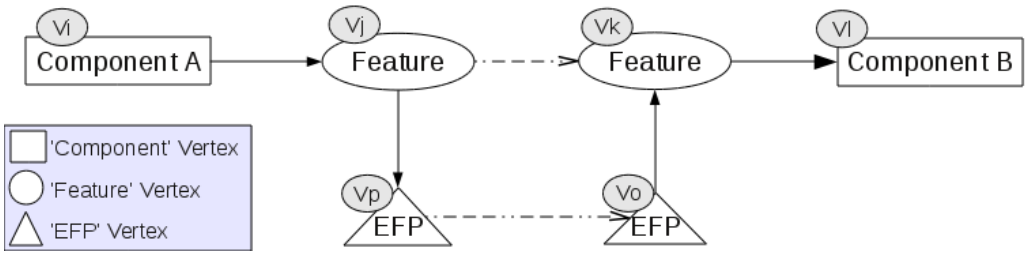
EFFCC: Extra-functional properties Featured Compatibility Checks



alignment ("time_to_process" always the same property)

EFP Compatibility Checks

> Works on complete (to be) composed architecture graph

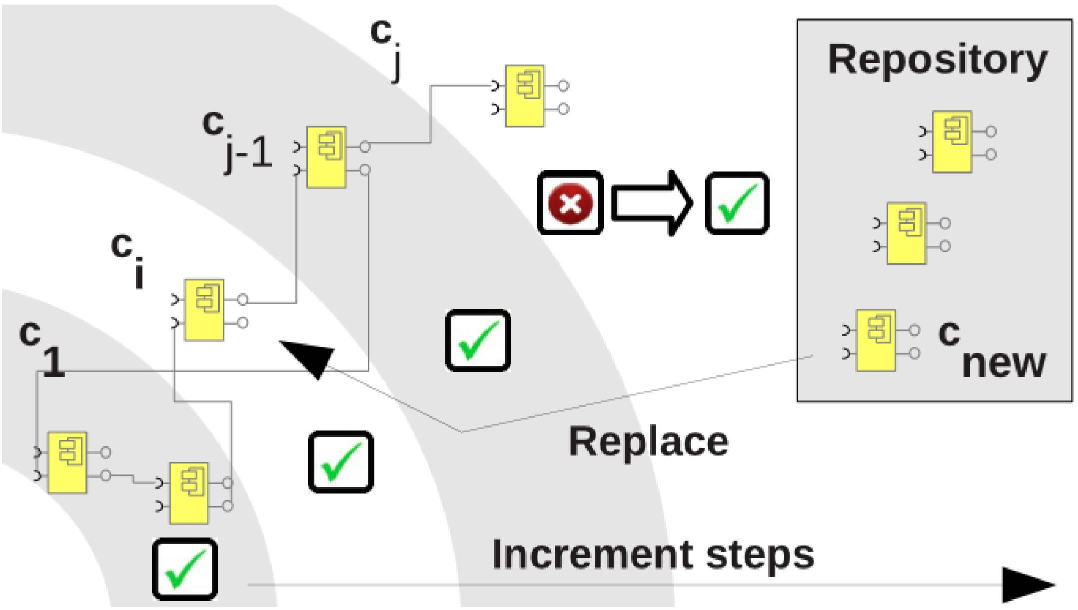


1. Create graph
   > uses element/feature meta-types, types, roles
2. Find values (depth-first)
   > assign direct values
   > compute derived and function-defined values (recursion on "R" features)
3. Compare and evaluate
   > quality vector using the γ function
   > results pair-wise and for the whole composition

Can always say "low < high"

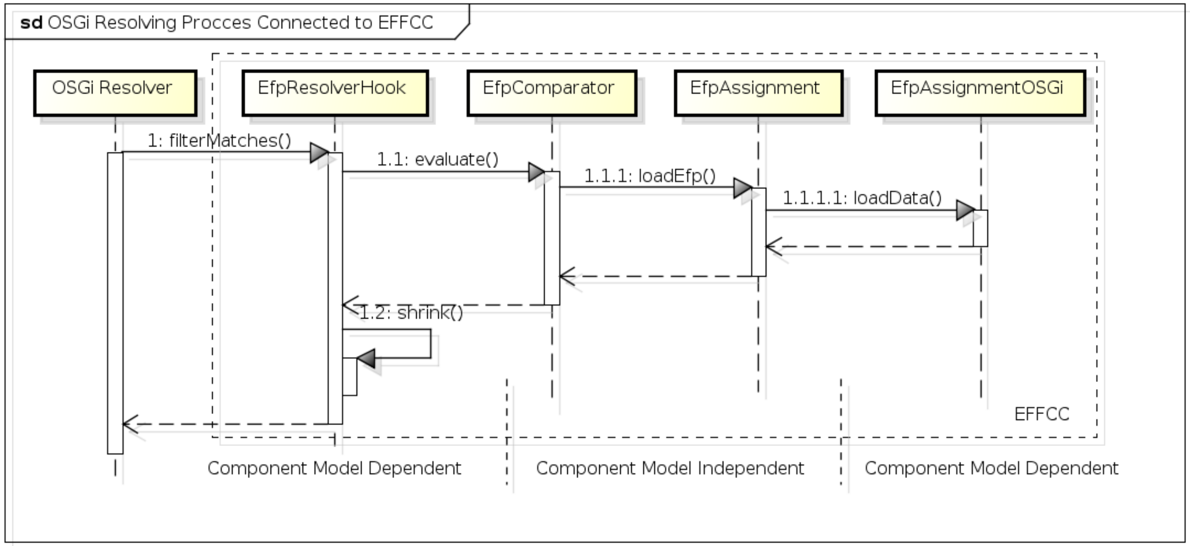
Dealing with Complexity

> Use incremental evaluation for
  - large applications
  - architecture reconfiguration (substitution)



Application: OSGi Enhancements

- Goal
  - integrate EFFCC to OSGi => provide EFP support
- Implementation
  - framework-tied “assignment module”
  - metadata extensions



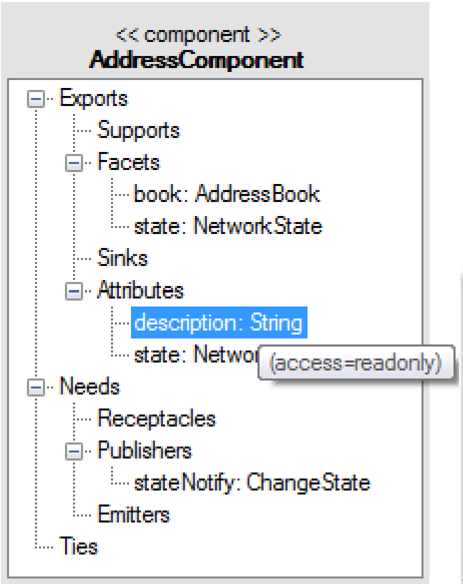
Selected Published Papers

- Brada, Přemysl. *Component Change and Version Identification in SOFA*. In Pavelka, J. and Tel, G. (Eds.): Proceedings of SOFSEM'99, LNCS 1725, Springer-Verlag, 1999. ISSN 0302-9743
- Brada, Přemysl. *Metadata support for safe component upgrades*. In Proceedings of COMPSAC'02, the 26th Annual International Computer Software and Applications Conference, Oxford, England, August 2002. IEEE Computer Society Press. ISBN: 0-7695-1727-7
- Brada, Přemysl; Valenta, Lukáš. *Practical verification of component substitutability using subtype relation*. In 32nd Euromicro conference on software engineering and advanced applications (SEAA). Los Alamitos : IEEE Computer Society, 2006, s. 38-45. ISBN: 0-7695-2594-6
- Bauml, Jaroslav; Brada, Přemysl. *Automated Versioning in OSGi: a Mechanism for Component Software Consitency Guarantee*. In 2009 35th Euromicro conference on software engineering and advanced applications. Los Alamitos : IEEE Computer Society, 2009, s. 428-435. ISBN: 978-0-7695-3784-9
- Brada, Přemysl. *Enhanced type-based component compatibility using deployment context information*. Electronic Notes on Theoretical Computer Science, 2011, vol.279, pp.17-31, ISSN 1571-0661
- Ježek, K.; Brada, P. *Correct Matching of Components with Extra-functional Properties -- A Framework Applicable to a Variety of Component Models.* Proceedings of the Evaluation of Novel Approaches to Software Engineering (ENASE 2011) conference, SciTePress 2011, s. 155-166. ISBN: 978-989-8425-57-7

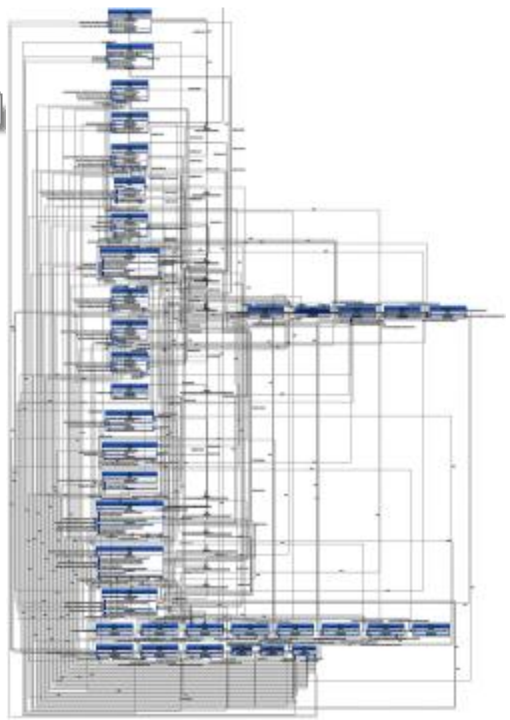
“… were omitted for brevity”

Components

> Meta-modeling
  - The ENT meta-model



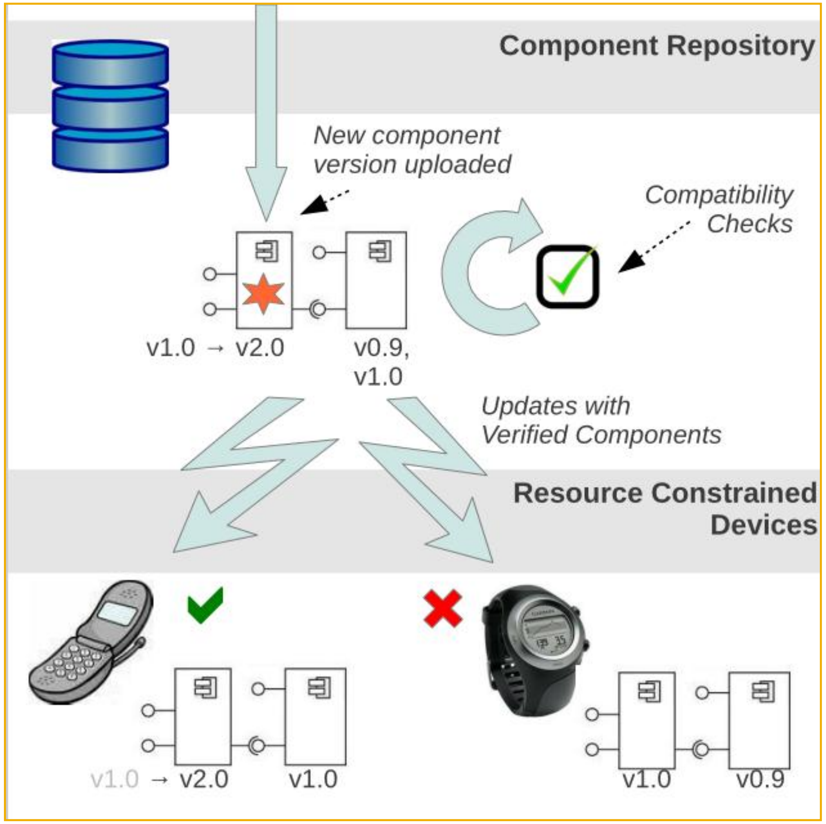
> Visualization
  - Advanced Interactive Visualization Approach
  - Vieport in Diagrams



Compatibility

> Simulation-based Approaches
  - componentized simulations
  - EFP verification

> Meta-data for Resource Constrained Scenarios
  - the CRCE repository



Service to People, Institution and Community

Grants

- (GAČR 1999-2001 „Developing software components for distributed environment“)
- GAČR 2008-2010 „Methods and models for consistency verification of advanced component-based applications“
- GAČR 2011-2013 „Methods of development and verification of component-based applications using natural language specifications“

- PhD student grants: Kamil Ježek M/cr, TALENT

Education and Institutional

- Master-level courses (2001-today)
  - Principles of / Advanced Software Engineering, Modern Trends in Software Engineering (seminar)
  - Programming Internet Applications, Java Enterprise Technologies
- Bc-level courses, master theses
- PhD students (2006+)
  - Kamil Ježek „Extra-Functional Properties Support For a Variety of Component Models“ (thesis submitted)
- Institutional involvement
  - Head of Software Engineering and Info Systems (2011)

Community Involvement

- Program Committees
  - Euromicro SEAA (2007+)
  - SOFSEM (2011+)
  - QUASOSS, CNSI, Objekty
- Industry Liaisons
  - guest lectures, CZJUG
  - Enterprise Software Engineering Competence Center (2011)

Conclusion

The Road(s) Ahead

- Making compatibility possible in resource-constrained scenarios
  - algorithm optimizations
  - rich meta-data use
  - pre-computed results of computationally expensive checks
- Visualization of complex software architectures
  - data-supported graph layouts and interaction
  - interaction and usability aspects

With a big Thank You to

Lukáš Valenta, Jarda Bauml
Kamil Ježek, Jarda Šnajberk, Lukáš Holý
+ all those Bc and Ing students

František, Širo, Ralf, Ivica,
Kung-Kiu, Tomáš, Petr
+ their colleagues

Jana + j + t + k

Him

http://www.flickr.com/photos/sushiraider/6893083210/