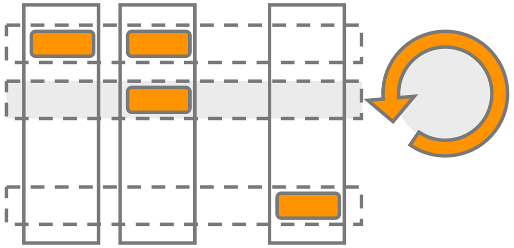
NII Shonan Meeting Seminar 052
Engineering Adaptive Software Systems (EASSy)



Dynamic Software Composition for Run-time System Evolution

(Context-oriented Programming at HPI)

Robert Hirschfeld

Hasso Plattner Institute

University of Potsdam

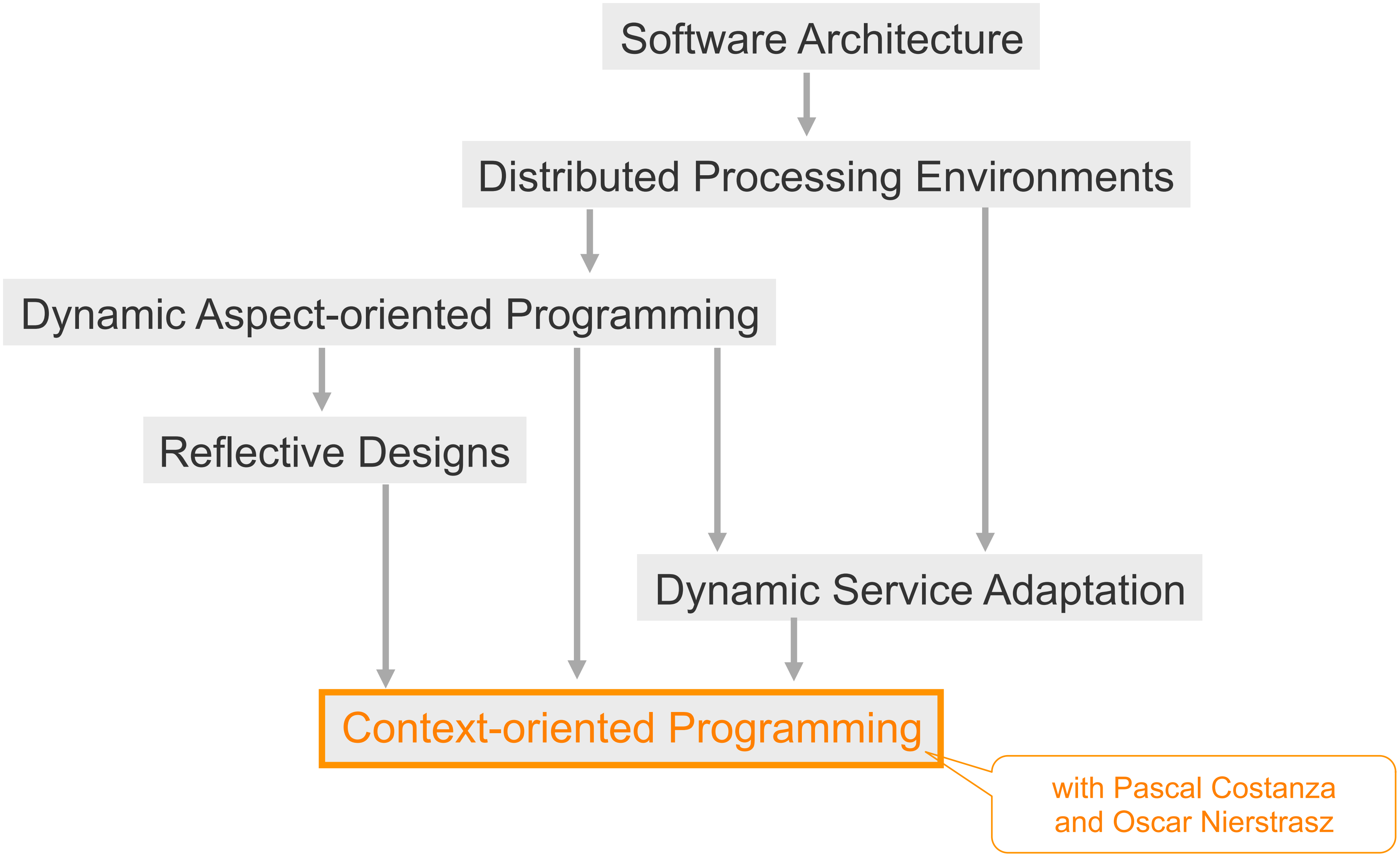
Germany

http://www.hpi.de/swa/

Shonan Village Center, Hayama, Kanagawa, Japan

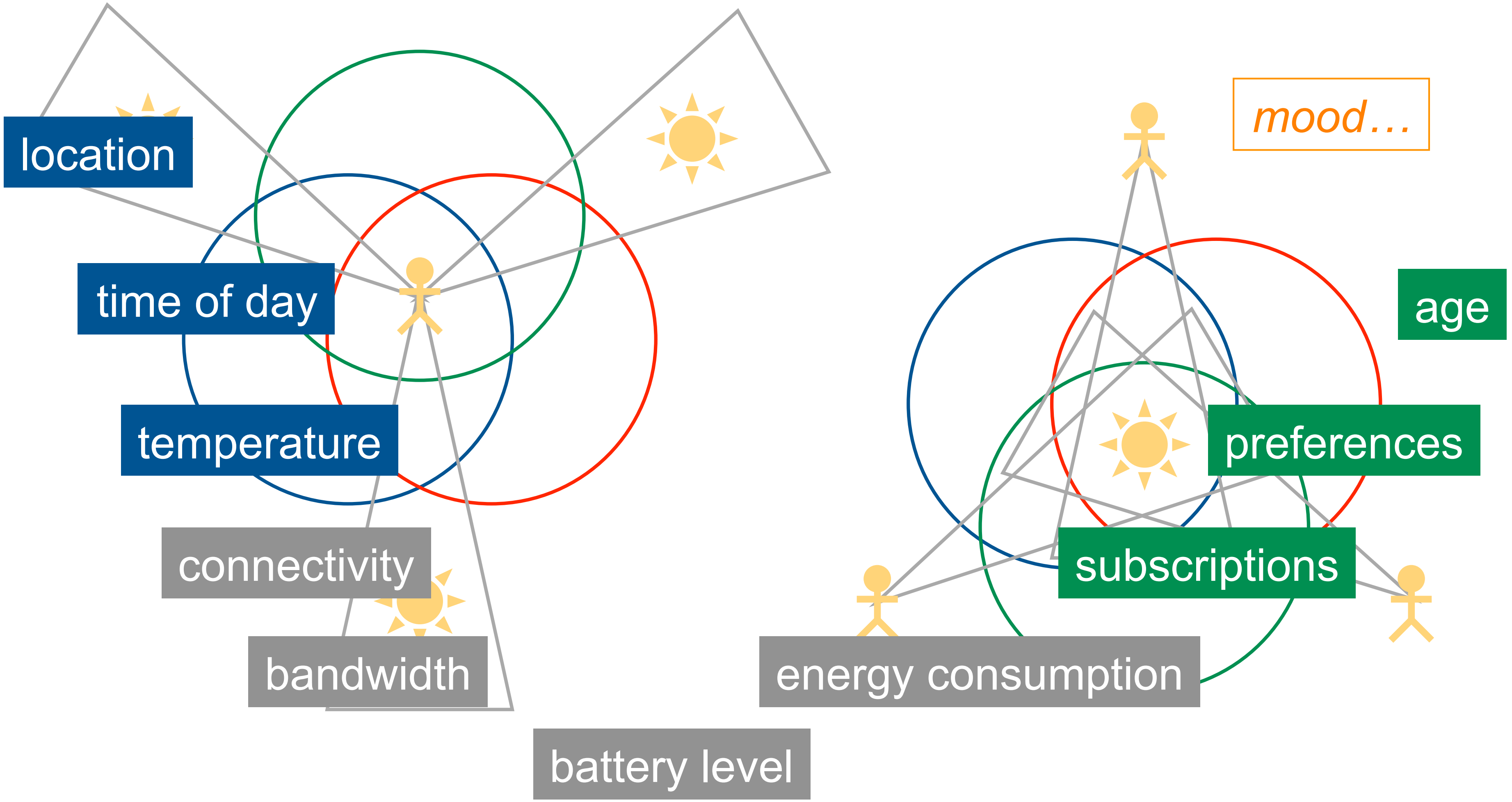
2015-09-07

Some History…

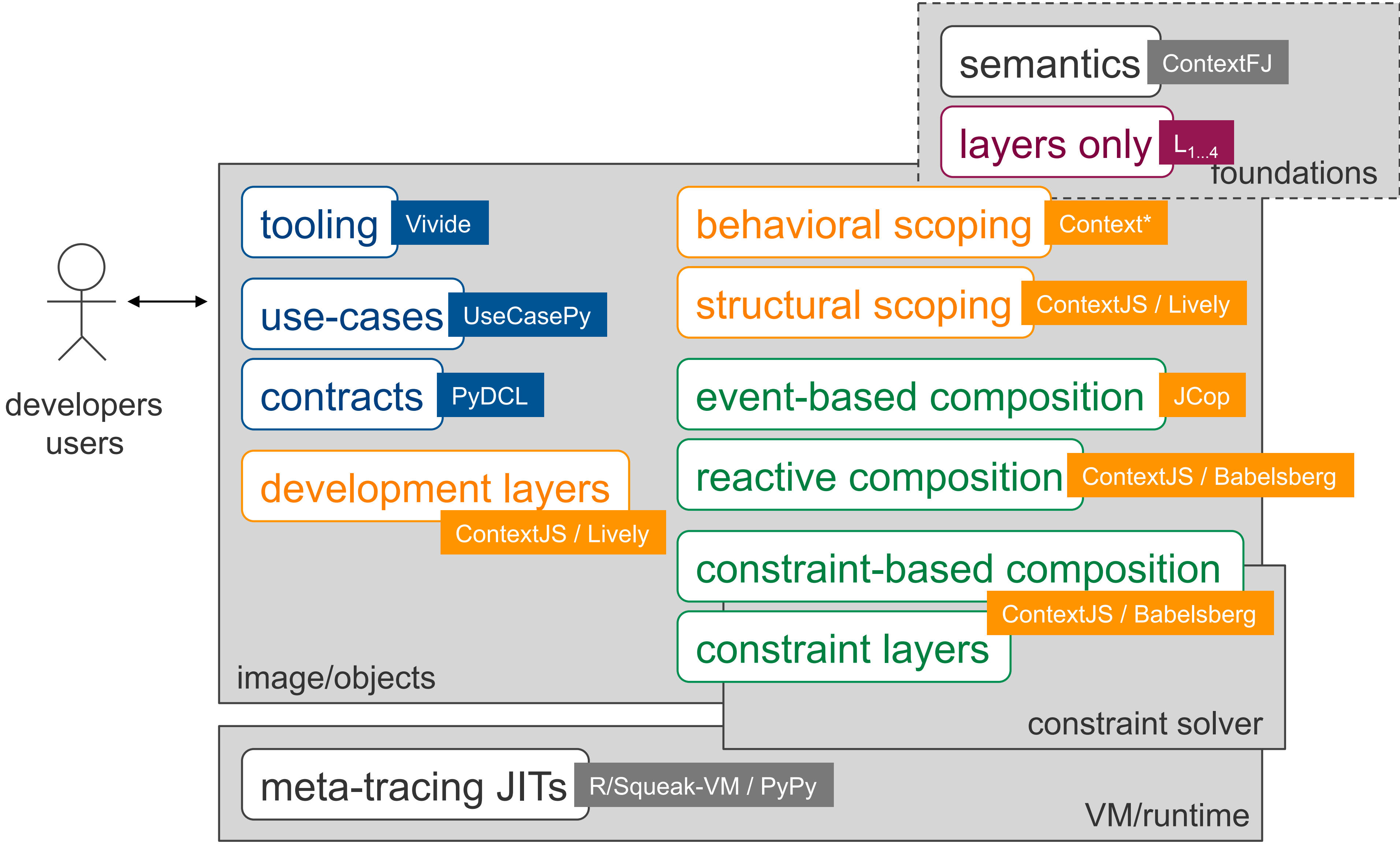


Context

context = everything computationally accessible



Outline


semantics
ContextFJ
layers only
$L_{1...4}$
foundations
tooling
Vivide
use-cases
UseCasePy
contracts
PyDCL
development layers
ContextJS / Lively
behavioral scoping
Context*
structural scoping
ContextJS / Lively
event-based composition
JCop
reactive composition
ContextJS / Babelsberg
constraint-based composition
ContextJS / Babelsberg
constraint layers
constraint solver
image/objects
developers
users
meta-tracing JITs
R/Squeak-VM / PyPy
VM/runtime

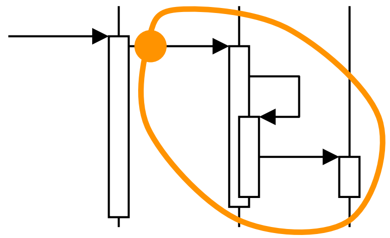
Behavioral Variations

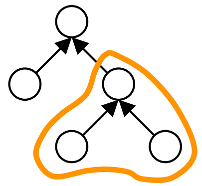
- Behavioral (dynamic) scoping
  - Dynamic extent of execution
  - Almost all COP extensions
- Structural (topological) scoping
  - ContextJS
  - Development layers
- Open implementation (OI) for scoping strategies
  - Allows for domain-specific scoping
  - Mainly applied to UI framework structures
    - Lively: Morphic
    - Webwerkstatt : Parts

behavioral scoping — Context*

structural scoping — ContextJS / Lively

development layers — ContextJS / Lively





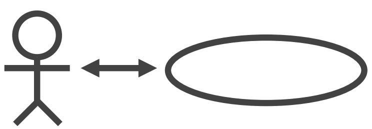
Development Support

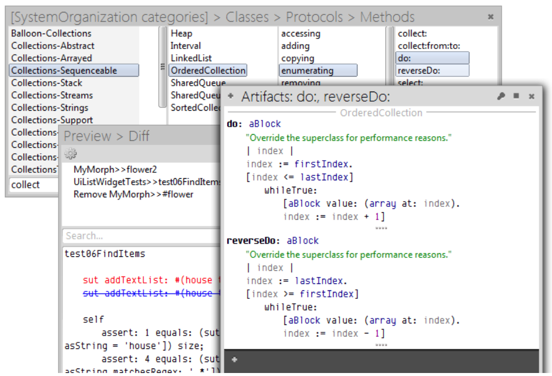
tooling | Vivide

use-cases | UseCasePy

contracts | PyDCL

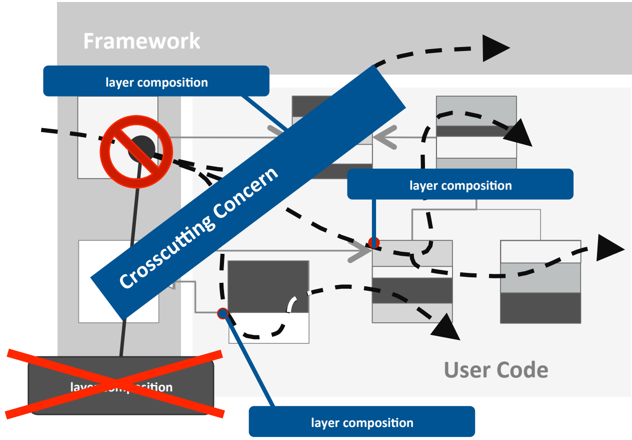
- More applied → more useful
- In PL work tool support often neglected
  - Usually too expensive, especially early…
    → Need for explorative tool building support
    - Vivide
- Crosscutting nature of layers lends itself nicely to crosscutting software engineering concerns
  - Explicit use-cases representation
    - UseCasePy
  - Dynamic contract layers
    - PyDCL

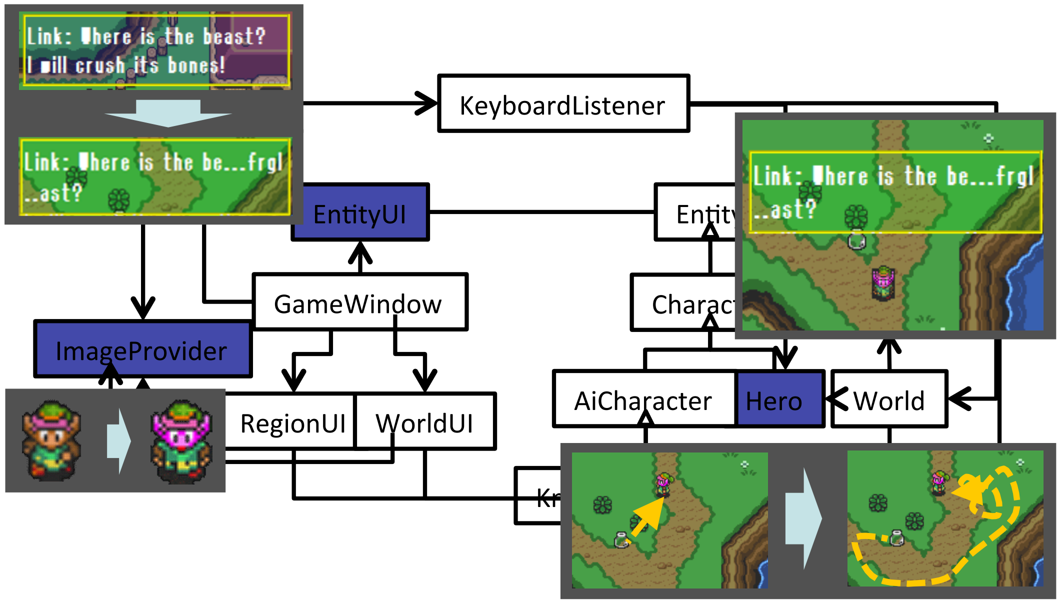




Reactive Approaches

- event-based composition — JCop
- reactive composition — ContextJS / Babelsberg
- constraint-based composition — ContextJS / Babelsberg
- constraint layers





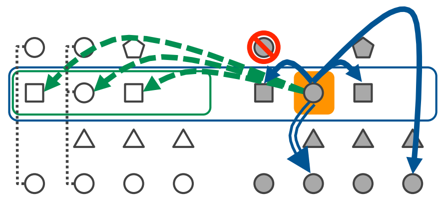
Foundations

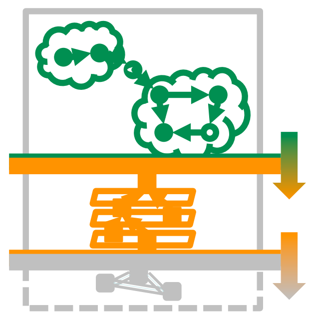
semantics ContextFJ

layers only $L_{1...4}$

meta-tracing JITs R/Squeak-VM / PyPy

- Semantics and types
  - ContextFJ
- Symmetry
  - No classes, only layers
  - No base system
    - $L_{1..4}$
- Sideways composition very expensive
  - Runtime support for optimizations
  - Meta-tracing JITs
    - R/Squeak-VM
  - Higher performance → more (meta-level) flexibility





$$\frac{PT(\mathtt{m}, \mathtt{C}, \mathtt{L}_0)\ \text{undefined} \qquad mbody(\mathtt{m}, \mathtt{C}, \overline{\mathtt{L}}', \overline{\mathtt{L}}) = \overline{\mathtt{x}}.\mathtt{e}\ \mathtt{in}\ \mathtt{D}, \overline{\mathtt{L}}''}{mbody(\mathtt{m}, \mathtt{C}, (\overline{\mathtt{L}}'; \mathtt{L}_0), \overline{\mathtt{L}}) = \overline{\mathtt{x}}.\mathtt{e}\ \mathtt{in}\ \mathtt{D}, \overline{\mathtt{L}}''}$$

Behavioral Scoping

COP

w/ Pascal Costanza and Oscar Nierstrasz

```
class Person {
  private String name, address;
  private Employer employer;

  Person(String newName,
         String newAddress,
         Employer newEmployer) {
    this.name = newName;
    this.employer = newEmployer;
    this.address = newAddress;
  }

  String toString() {return "Name: "+name;}

  layer Address {
    String toString() {
      return proceed()+"; Contact: "+address;
    }
  }

  layer Employment {
    String toString() {
      return proceed()+"; [Employer] "+employer;
    }
  }
}
```

```
class Employer {
  private String name, address;

  Employer(String newName,
           Stringe newAddress) {
    this.name = newName;
    this.employer = newEmployer;
  }

  String toString() {return "Name: "+name;}

  layer Address {
    String toString() {
      return proceed()+"; Visitors: "+address;
    }
  }
}
```

```
class Person {
  priva
  priva

  Pers

    this
    this
    this
  }

  String toString() {return "Name: "+name;}

  layer Address {
    String toString() {
      return proceed()+"; Contact: "+address;
    }
  }

  layer Employment {
    String toString() {
      return proceed()+"; [Employer] "+employer;
    }
  }
}
```

```
class Employer {

                                   er;

  String toString() {return "Name: "+name;}

  layer Address {
    String toString() {
      return proceed()+"; Visitors: "+address;
    }
  }
}
```

```
Employer hpi = new Employer(”HPI", ”14440 Potsdam");
Person robert = new Person(”Robert Hirschfeld", ”14471 Potsdam", hpi);

  System.out.println(robert);
```

Output: Name: Robert Hirschfeld

```
class Person {
  priva
  priva

  Pers

    this
    this
    this
  }

  String toString() {return "Name: "+name;}

  layer Address {
    String toString() {
      return proceed()+"; Contact: "+address;
    }
  }

  layer Employment {
    String toString() {
      return proceed()+"; [Employer] "+employer;
    }
  }
}
```

```
class Employer {

  String toString() {return "Name: "+name;}

  layer Address {
    String toString() {
      return proceed()+"; Visitors: "+address;
    }
  }
}
```

```
Employer hpi = new Employer(”HPI", ”14440 Potsdam");
Person robert = new Person(”Robert Hirschfeld", ”14471 Potsdam", hpi);

 with (Address) {
   System.out.println(robert);
 }

            Output:    Name: Robert Hirschfeld; Contact: 14471 Potsdam
```

```
class Person {
  priva
  priva

  Pers

    this
    this
    this
  }

  String toString() {return "Name: "+name;}

  layer Address {
    String toString() {
      return proceed()+"; Contact: "+address;
    }
  }

  layer Employment {
    String toString() {
      return proceed()+"; [Employer] "+employer;
    }
  }
}
```

```
class Employer {

  String toString() {return "Name: "+name;}

  layer Address {
    String toString() {
      return proceed()+"; Visitors: "+address;
    }
  }
}
```

```
Employer hpi = new Employer(”HPI", ”14440 Potsdam");
Person robert = new Person(”Robert Hirschfeld", ”14471 Potsdam", hpi);

with (Employment) {
  with (Address) {
    System.out.println(robert);
  }
}
```

Output: Name: Robert Hirschfeld; Contact:14471 Potsdam; [Employer] Name: HPI; Visitors: 14440 Potsdam

```
class Person {
  priva
  priva

  Pers

    this
    this
    this
  }

  String toString() {return "Name: "+name;}

  layer Address {
    String toString() {
      return proceed()+"; Contact: "+address;
    }
  }

  layer Employment {
    String toString() {
      return proceed()+"; [Employer] "+employer;
    }
  }
}
```

```
class Employer {

  String toString() {return "Name: "+name;}

  layer Address {
    String toString() {
      return proceed()+"; Visitors: "+address;
    }
  }
}
```

```
Employer hpi = new Employer(”HPI", ”14440 Potsdam");
Person robert = new Person(”Robert Hirschfeld", ”14471 Potsdam", hpi);

with (Address) {
  with (Employment) {
    System.out.println(robert);
  }
}
```

Output: Name: Robert Hirschfeld; [Employer] Name: HPI; *Visitors: 14440 Potsdam; Contact:14471 Potsdam*

```
Employer hpi = new Employer(”HPI", ”14440 Potsdam");
Person robert = new Person(”Robert Hirschfeld", ”14471 Potsdam", hpi);
```

Thread-1

```
with (Address) {
  with (Employment) {
    System.out.println(robert);
  }
}
```

Output-1: Name: Robert Hirschfeld<u>; [Employer] Name: HPI</u>*; Visitors: 14440 Potsdam; Contact:14471 Potsdam*

Thread-2

```
with (Employment) {
  with (Address) {
    System.out.println(robert);
  }
}
```

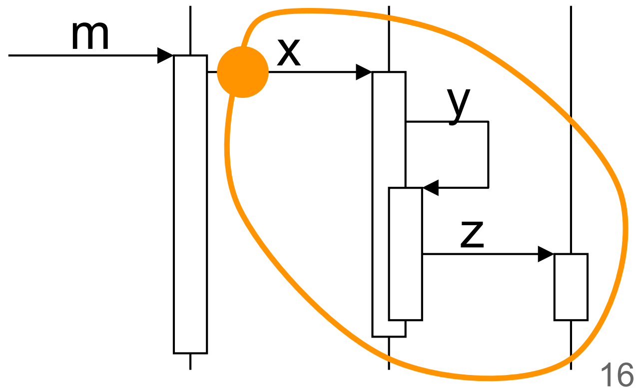
Output-2: Name: Robert Hirschfeld*; Contact:14471 Potsdam*<u>; [Employer] Name: HPI</u>*; Visitors: 14440 Potsdam*

```
class Person {
  priva
  priva

  Pers

    this
    this
    this
  }

  Strin

  laye
    Stri
      re
    }
  }

  laye
    Stri
      return proceed()+"; [Employer] "+employer;
    }
  }
}
```

```
class Employer {
;}
ss;
```

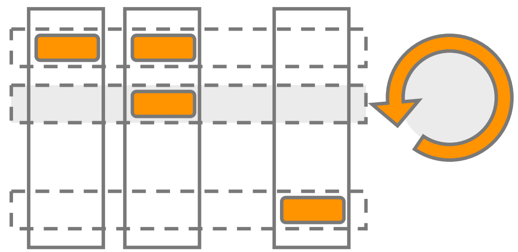
Dynamically-scoped Layer Activation

- Constructs
  - with (…) {…}
  - without (…) {…}
  - next (…)
- Activate (deactivate) layers for the current thread
  - No interference with other layer activations/deactivations in other threads
  - Layers are activated/deactivated only for the dynamic extent of the associated code block (dynamic scoping)
  - Activation order determines method precedence



COP Basics Summary

- Behavioral variations
  - Partial class, method, and layer definitions
- Layers
  - Groups of related context-dependent behavioral variations
- Activation
  - Activation and deactivation of layers at run-time
- Context
  - Anything computationally accessible
- Scoping
  - Well-defined explicitly-controlled scopes



COP Extensions (Some…)

- ContextS
- ContextS2
- ContextJS
- JCop (ContextJ)
- ContextPy
- PyDCL
- UseCasePy
- PyContext
- ContextR
- ContextG
- ContextAmber
- $L_{1\ldots4}$

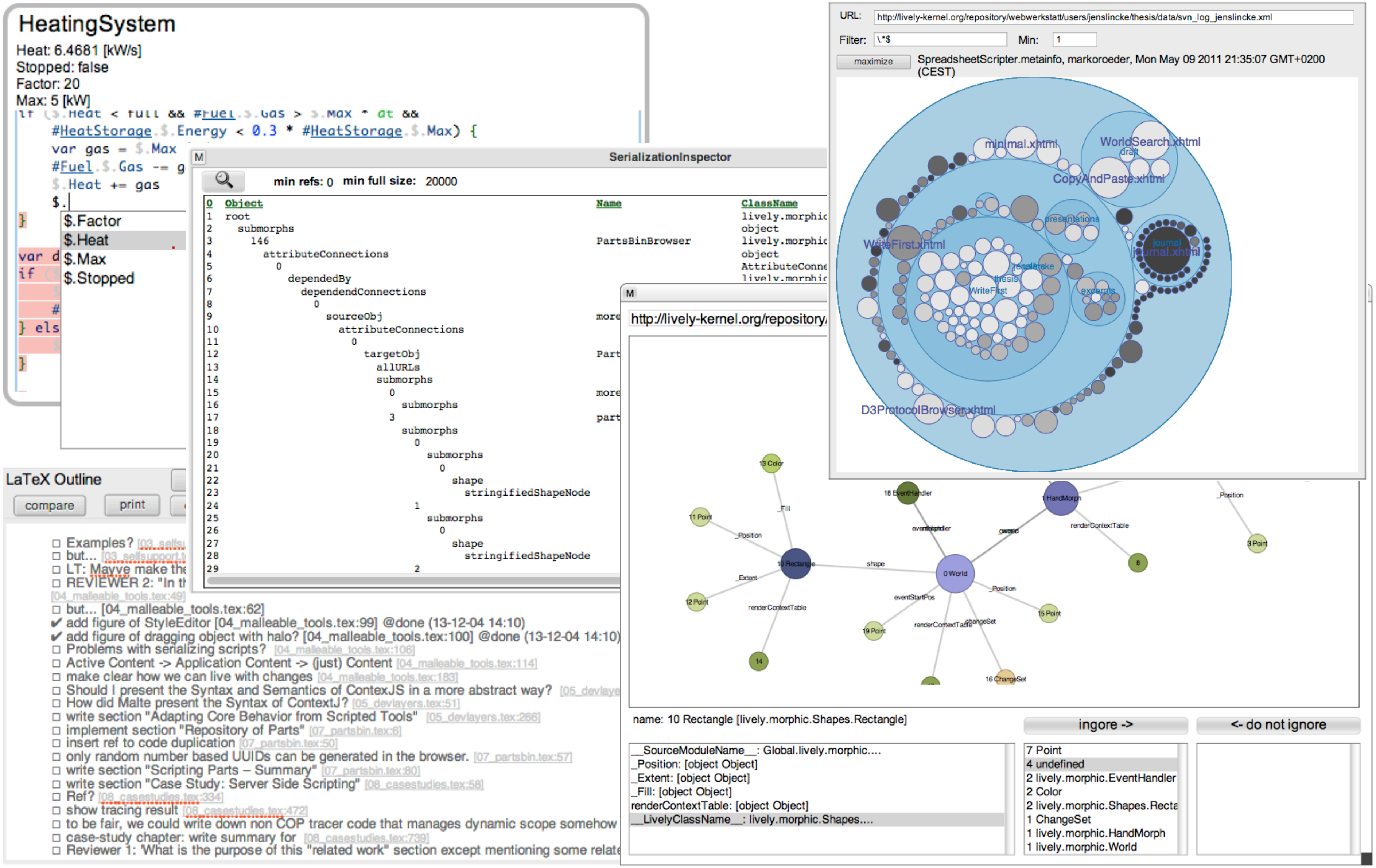
- ContextL
- ContextScheme
- ContextJ*
- ContextErlang
- EventCJ
- Lambic
- Ambience
- COP.JS
- delMDSCO/cj
- Phenomenal Gem
- Subjective-C
- Context Petri Nets

Structural Scoping & Development Layers

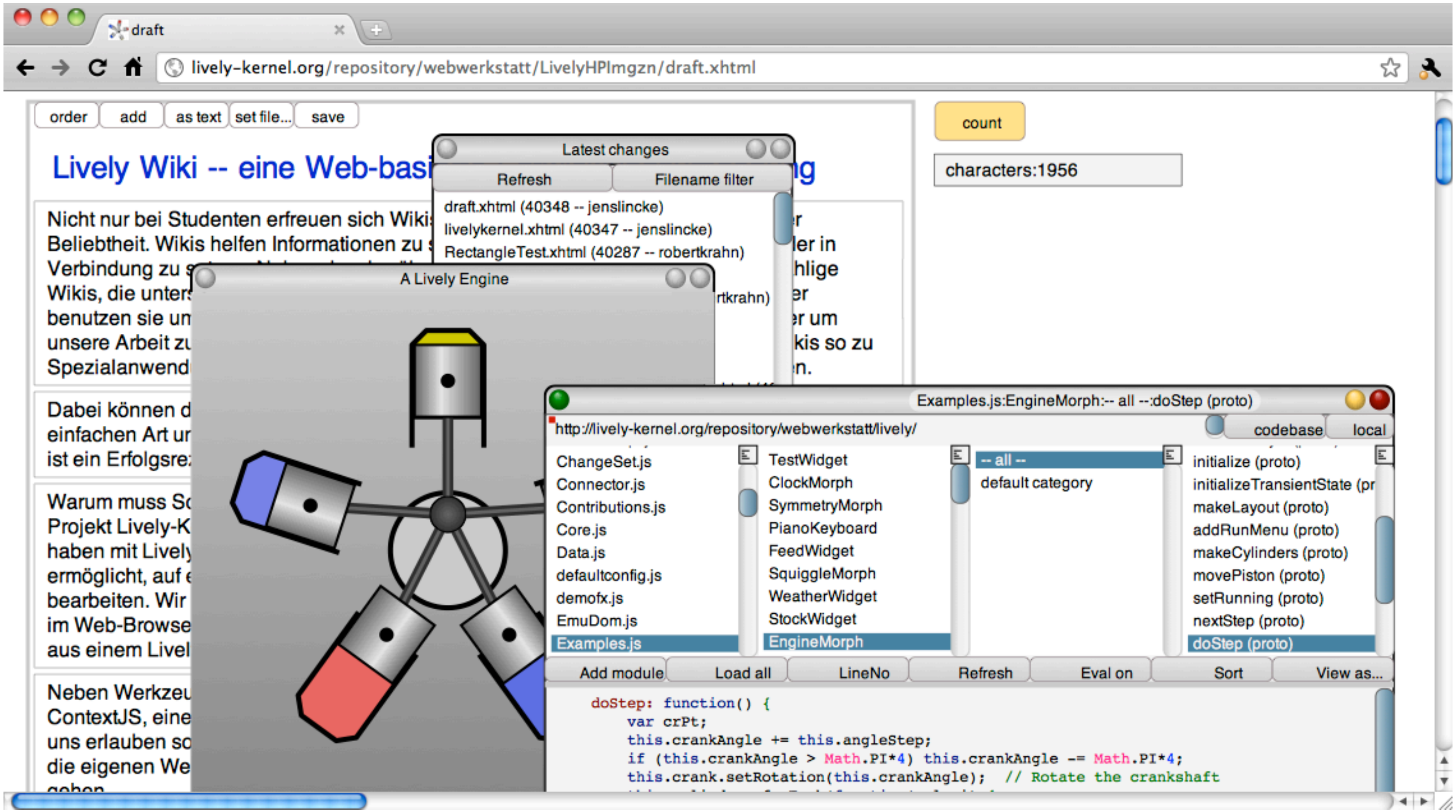
Strct

w/ Jens Lincke

Lively Webwerkstatt



Lively Kernel and Lively Wiki



Self-supporting Development Environments

- Collaboratively evolve tools and environment
  - Adapt tools while using them
  - From within
  - Share easily

- Design goals for self-supporting development environments (SSDEs)
  - Innovative repair
  - Short feedback loops → immediacy

- Technical problem
  - Changes to core functionality might break the environment (also for everyone)

ContextJS

- Library-based COP extension to JavaScript
- Open implementation (OI) for layer composition
  - Behavioral scoping
  - Structural scoping

```
EventCounter.count = function(evt) {
        alert("evt: " + evt);
        this.n = this.n + 1;
}
```

```
EventCounter = {
        n: 0,
        count: function(evt) {
                this.n = this.n + 1;
        }
}
```

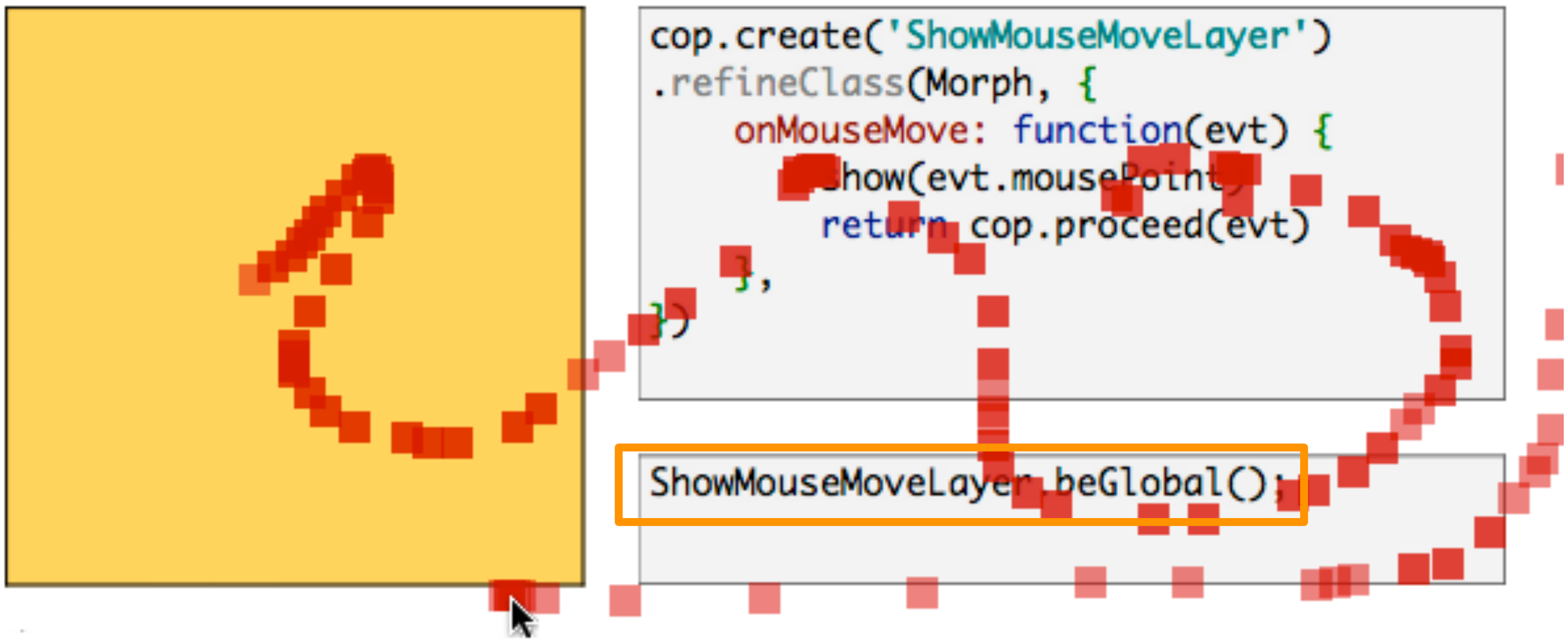
```
cop.create("DevLayer").refineObject(EventCounter, {
        count: function(evt) {
                alert("evt: " + evt);
                this.n = this.n + 1;
        }
})
```

```
cop.proceed(evt);
```

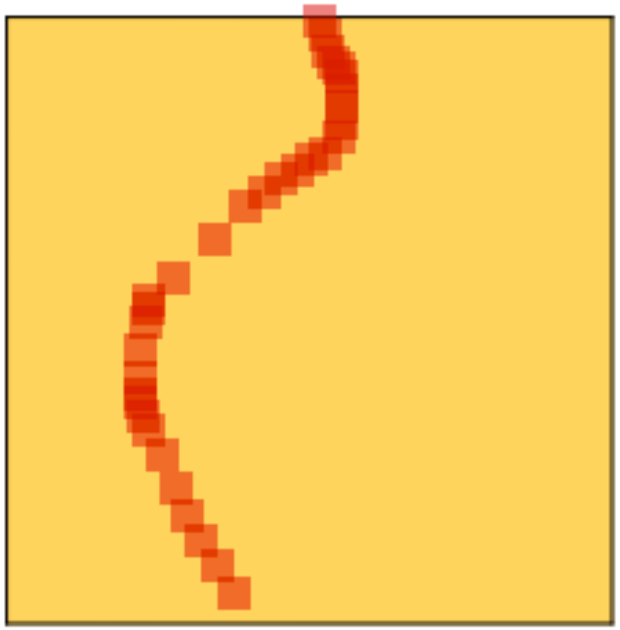
```
DevLayer.beGlobal();
```

```
debugArea.setWithLayers([DevLayer])
```

Example 1: Visualizing Events



Example 1: Visualizing Events



```
cop.create('ShowMouseMoveLayer')
.refineClass(Morph, {
    onMouseMove: function(evt) {
        show(evt.mousePoint)
        return cop.proceed(evt)
    },
})
```

```
$morph('DebugArea').setWithLayers([ShowMouseMoveLayer])
```

HPI

Example 2: Text Coloring

Hello World

Workspace

```
cop.create('TextColorLayer').refineClass(lively.morphic.Text, {
    processCommandKeys: function(evt) {
        var key = evt.getKeyChar();
        if (key) key = key.toLowerCase();
        if (evt.isShiftDown()) {  // shifted commands here...
            switch (key) {
                case "5": { this.emphasizeSelection({color: Color.black}); return true; }
                case "6": { this.emphasizeSelection({color: Color.red}); return true; }
                case "7": { this.emphasizeSelection({color: Color.green}); return true; }
                case "8": { this.emphasizeSelection({color: Color.blue}); return true; }
            }
        }
        return cop.proceed(evt);
    }
})
```

`this.setWithLayers([...])`

`TextColorLayer.beGlobal()`

Example 3: Developing Auto-completion

Workspace

```
cop.create('WordCompletionLayer').refineClass(lively.morphic.Text, {
    onKeyPress: function(evt) {

        var key = evt.getKeyChar();
        if (!key.match(/\w/)) {
            this.hideWordCompletionMorph();
            return;
        }
        var range  = this.getSelectionRange()
        var cursor = range[0];

        // ....
```

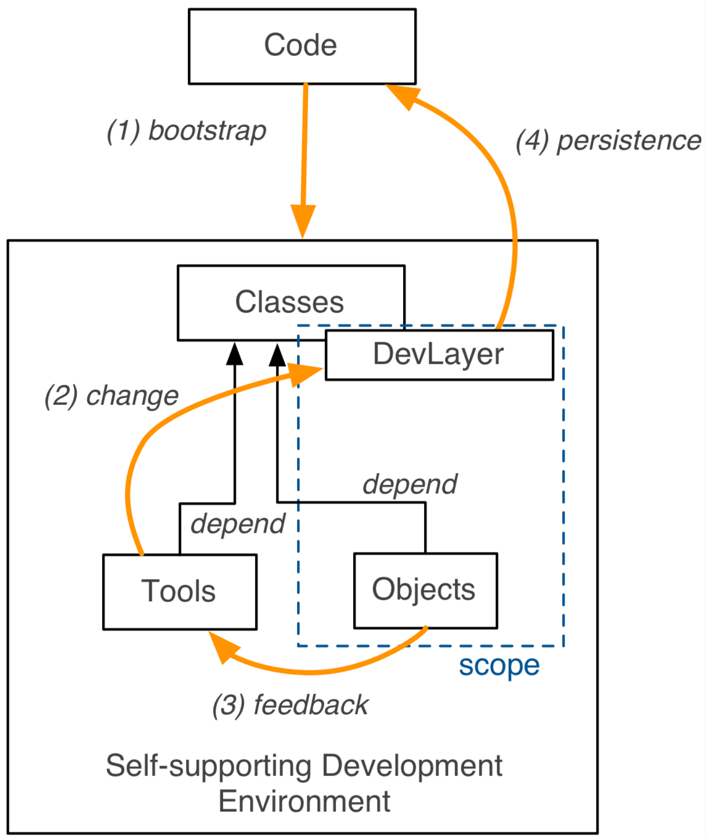
```
$morph('DevArea').setWithLayers([WordCompletionLayer]);
```

Structural Scoping Summary

- Application of COP to SSDEs
  - Organize changes into layers
  - Apply changes during development to only objects of interest
    → Structural scoping
    → Development layers

- Evolution of tools can be direct, interactive, and safe

- Future work
  - Refactoring of layers back into base



Explicit Use-case Representation

UC

w/ Michael Perscheid

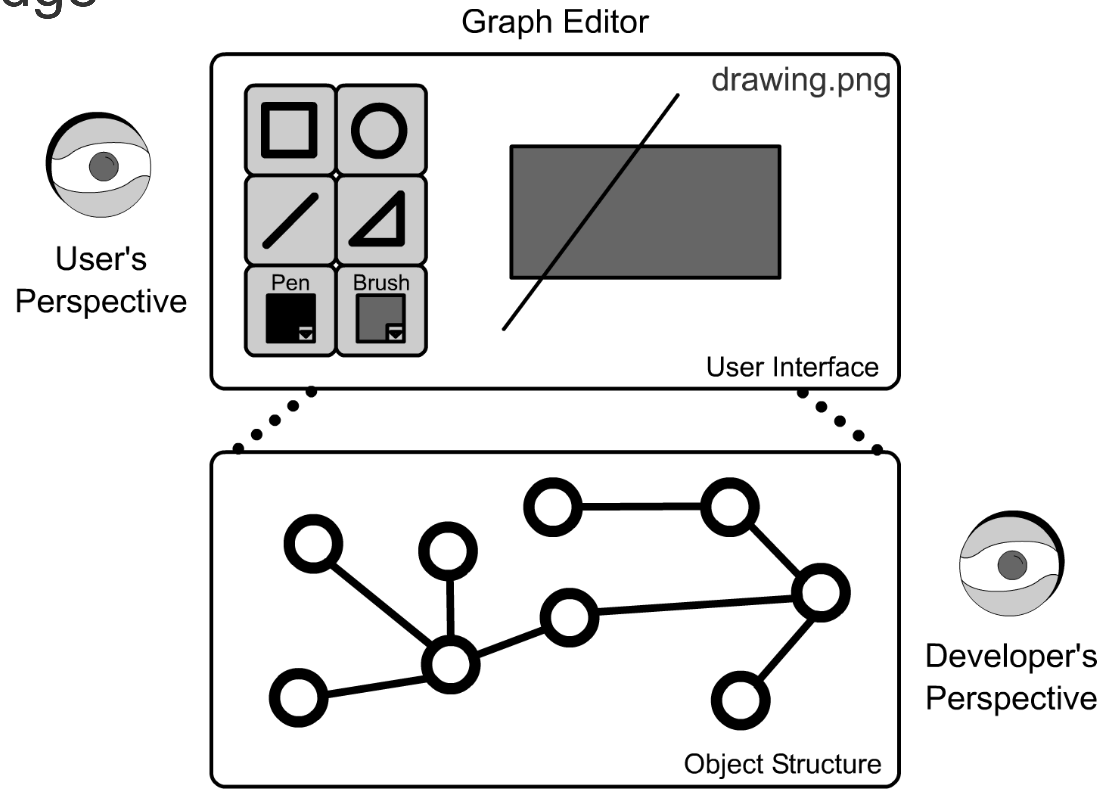
Use-cases in Software Development

- Users perceive program behavior without implementation knowledge

- Developers also know internals and implementation details

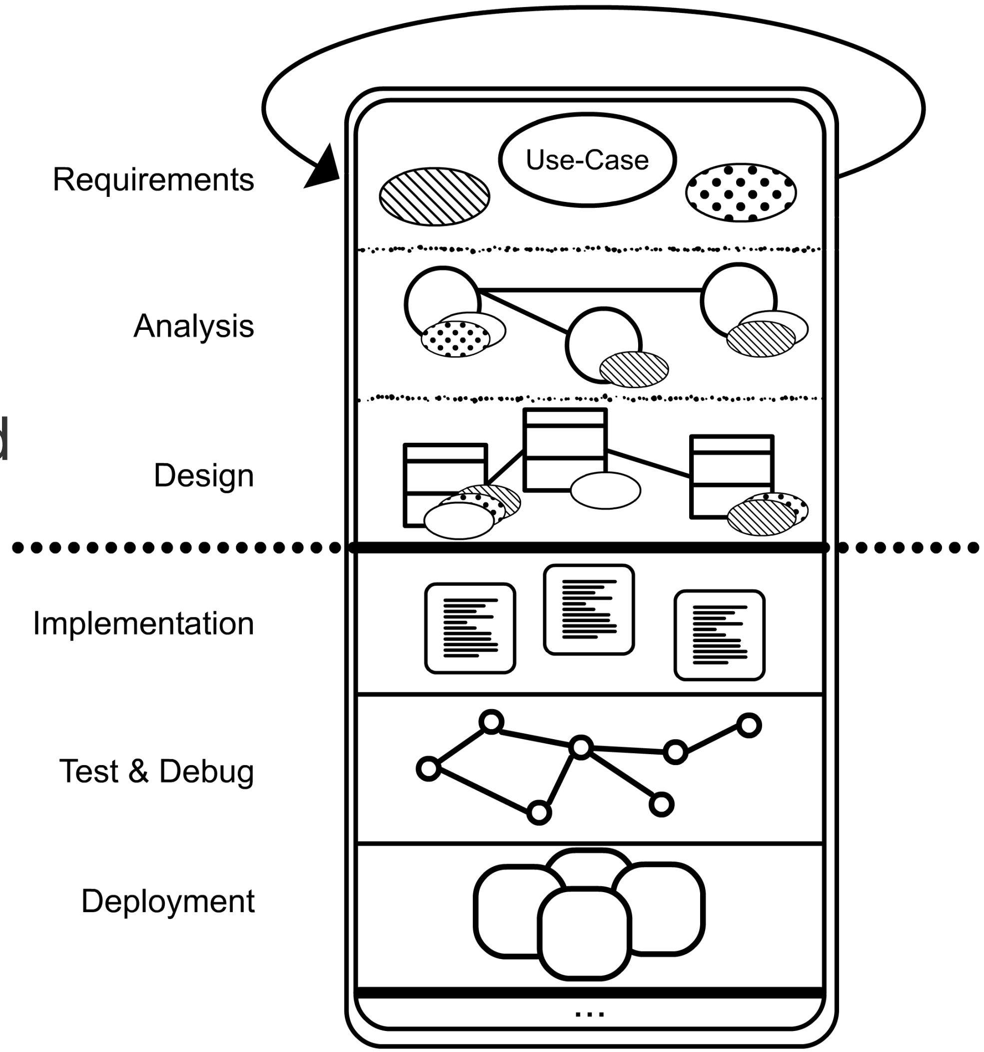
- Use-cases describe interaction at system boundary

- Use-cases link both perspectives



Use-cases in Software Development

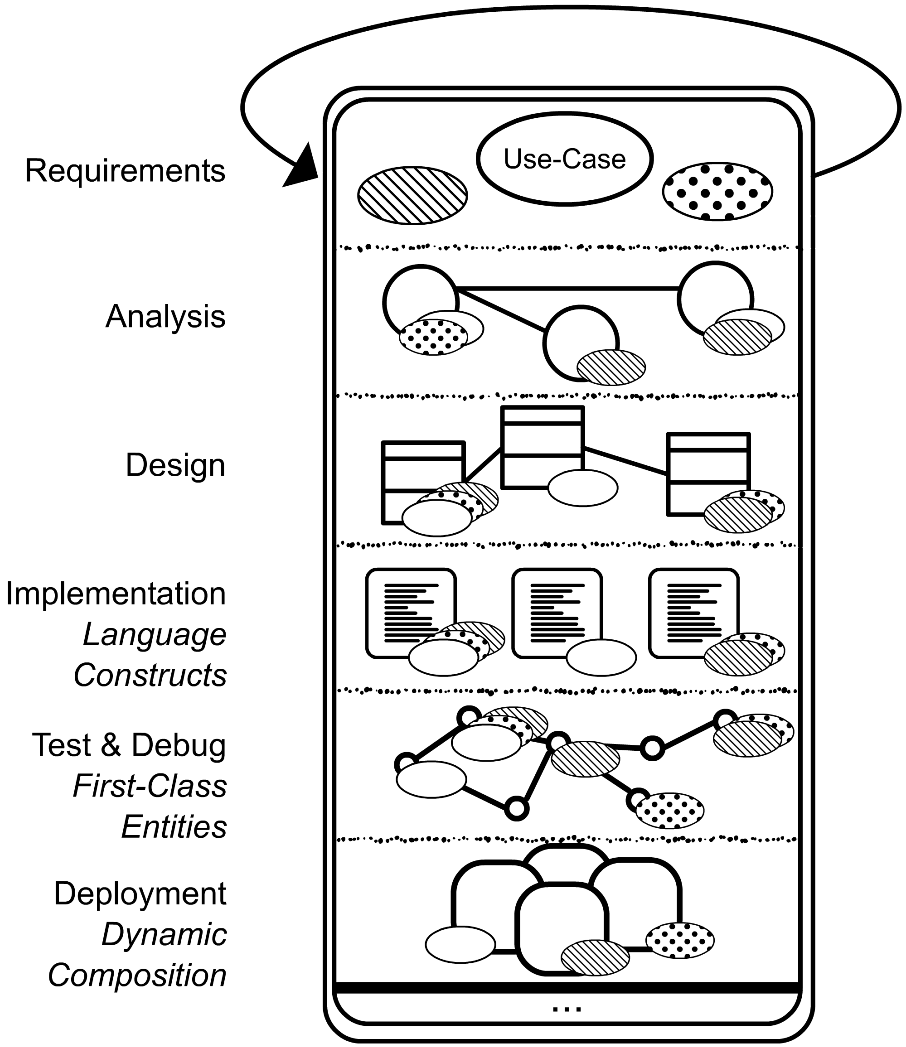
- Use-cases and variants are integral part of most contemporary development processes
- Traceability to use-cases lost in later more code- and deployment-centric development activities
- → Lack of understanding about which parts of the system contribute to which use-case



Use-case-centered Development

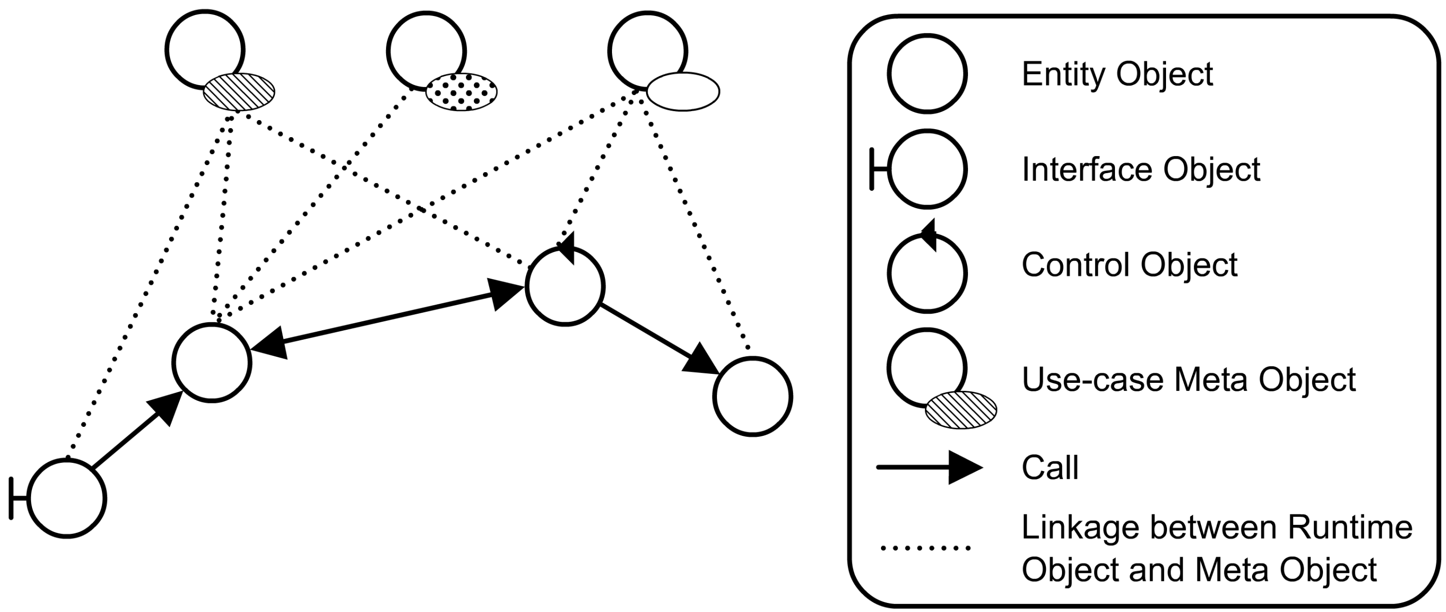
- Explicit use-case representation in object-oriented languages

- Use-cases in source code, as deployment units, and at run-time



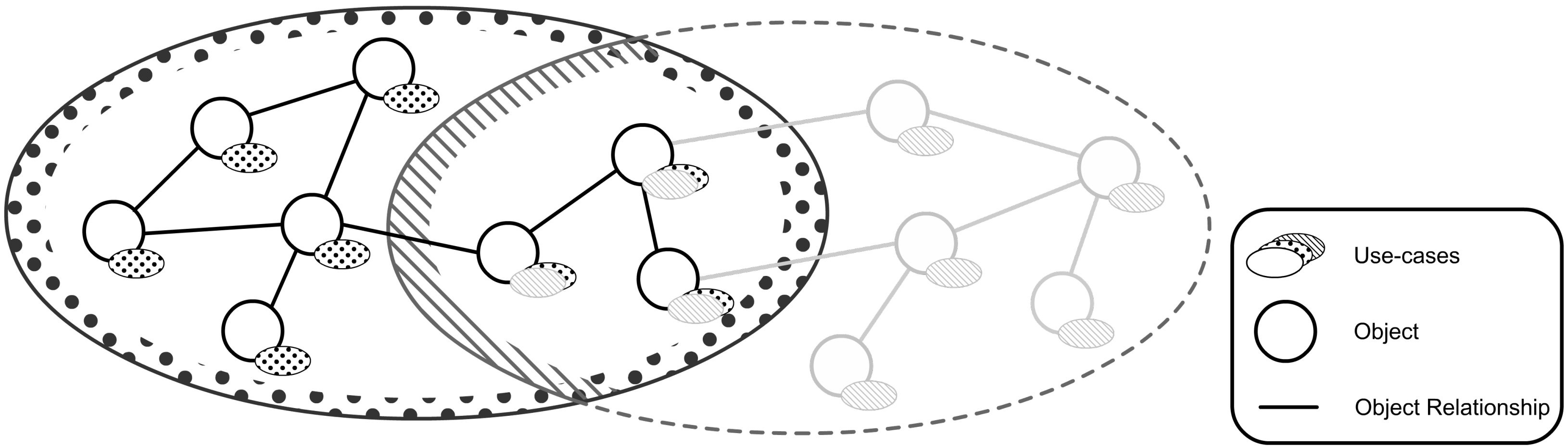
First-class Entities at Run-time

- Based on source code annotations
- Use-cases as meta objects
- Central registry of use-case descriptions
- Available at run-time for introspection and intercession



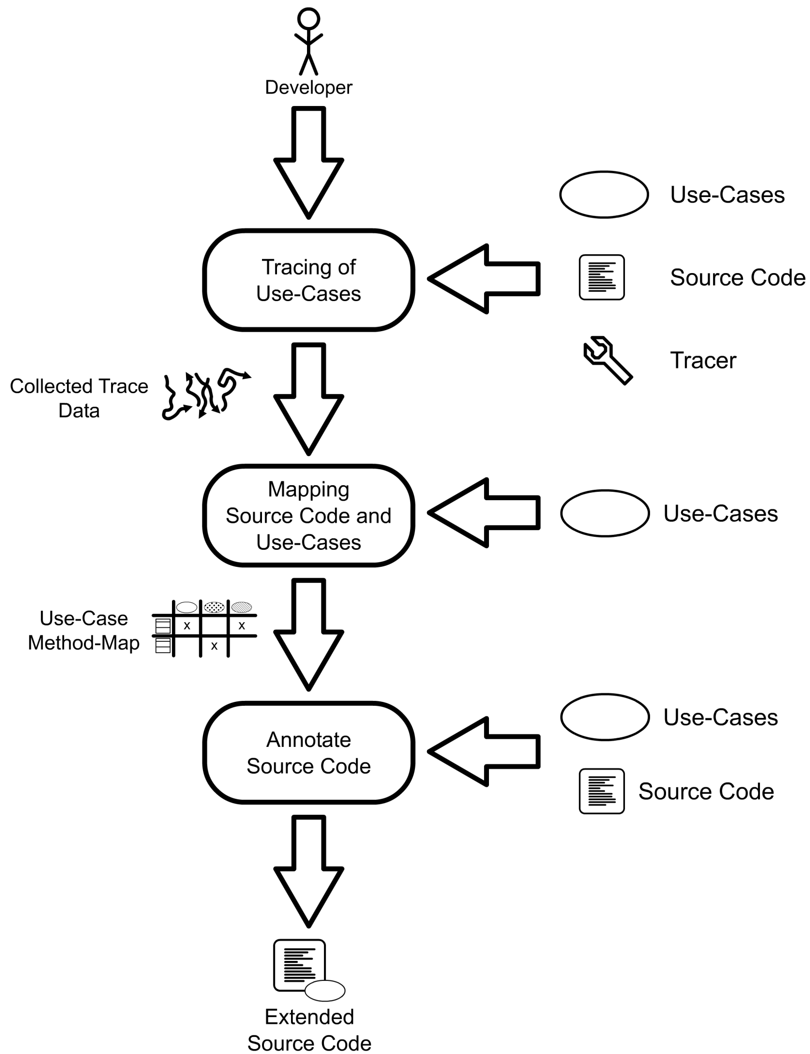
Dynamic Composition

- Based on selection of a set of desired use-cases
- Requires use-case-aware method dispatch
- Allows for use-cases as deployment units



Use-case Discovery

- Introduce use-case-centered development to existing systems
- Based on feature location techniques
- Tracer observes execution of use-cases from the users' point of view
- Semi-automatic and automatic implementations



Use-case Layers Summary

- Use-case-centered development allows for explicit representation of use-cases in code and at run-time
  - Available during implementation, testing, and deployment
  - Use-case discovery migrates existing systems to use-case-centered development

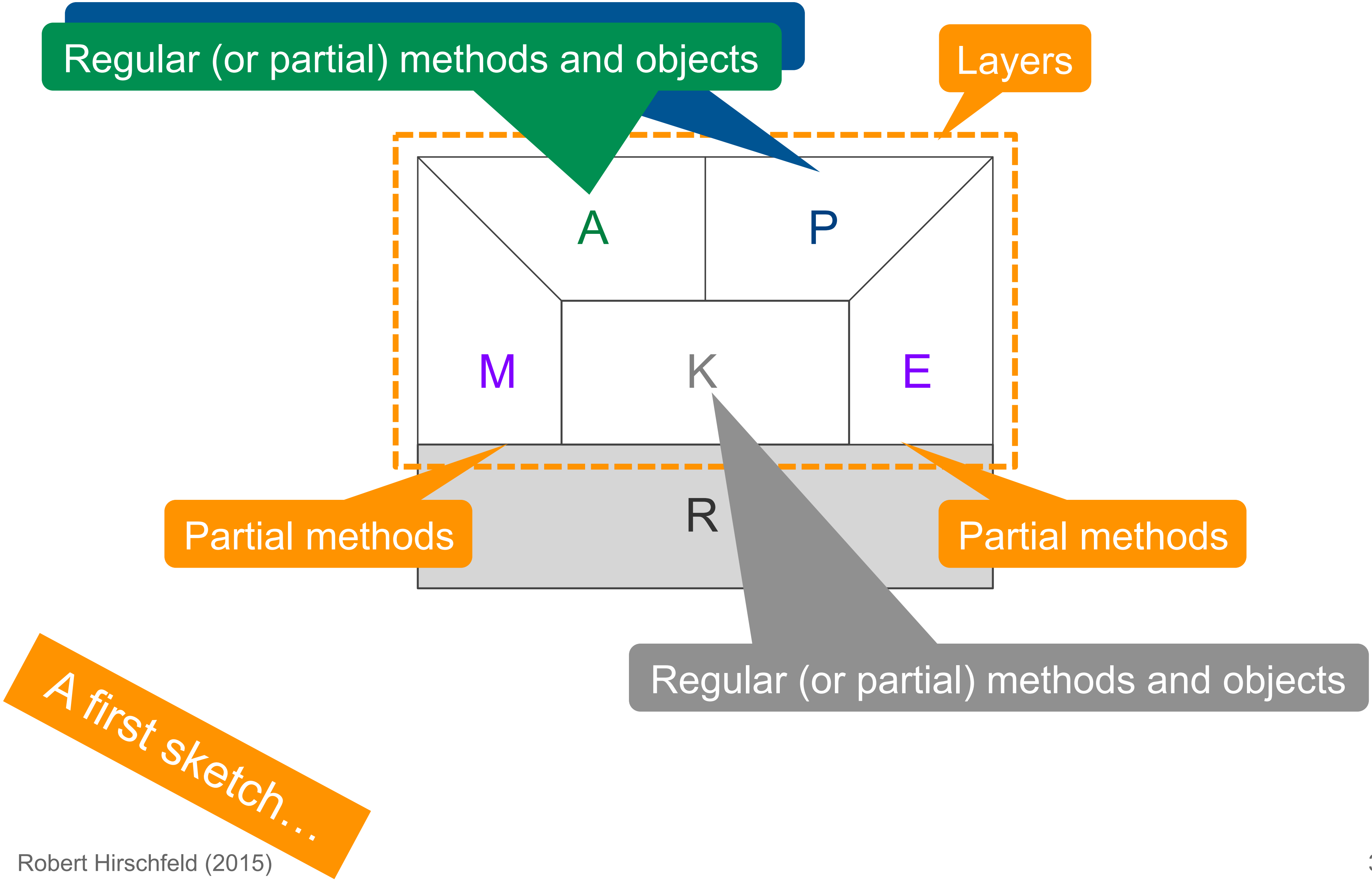
- Future work
  - User studies
  - Improved analysis techniques
  - Better tool support

Monitor Analyze Plan Execute-Knowledge

MAPE-K

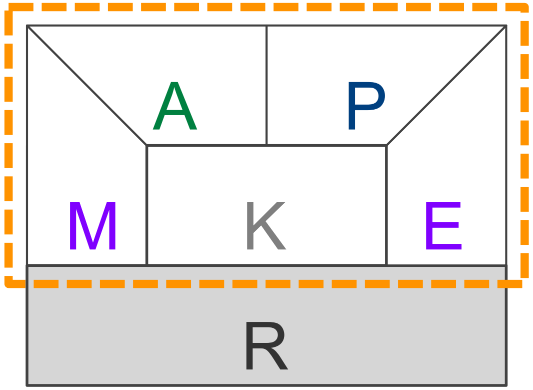
NII Shonan Seminar 052 EASSy

COP & MAPE-K



Regular (or partial) methods and objects
Layers
A
P
M
K
E
R
Partial methods
Partial methods
Regular (or partial) methods and objects
A first sketch…

COP & MAPE-K



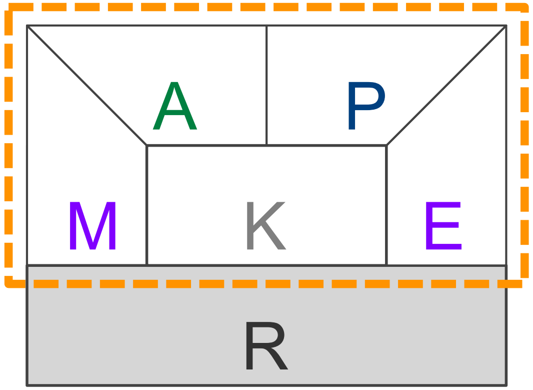

```
class ResourceManager {
  // …
  layer MAPE { // not necessary
    static mapeBefore(resource, in, ...) {
      // note = ...;
      monitorBefore(...);
      analyzeBefore(...);
      planBefore(...);
      executeBefore(…);
      return note;
    }
    static mapeAfter(resource, in, note, out, ...) {
      // newOut = ...
      monitorAfter(...); // executeAfter(...);
      analyzeAfter(...); // planAfter(...);
      planAfter(...); // analyzeAfter(...);
      executeAfter(…); // monitorAfter(…);
      return newOut;
} } }
```

```
class ManagedResource {
  // …
  process(in) {
    // out := ...;
    return out;
  }
  layer MAPE {
    process(in) {
      note := ResourceManager.mapeBefore(self, in, ...);
      out := next(in);
      out := ResourceManager.mapeAfter(self, in, note, out, ...);
      return out;
} } }
```

A first sketch…

COP & MAPE-K

```
class ResourceManager {
  // …
  layer MAPE { // not necessary
    static mapeBefore(resource, in, ...) {
      // note = ...;
      monitorBefore(...);
      analyzeBefore(...);
      planBefore(...);
      executeBefore(…);
      return note;
    }
    static mapeAfter(resource, in, note, out, ...) {
      // newOut = ...
      monitorAfter(...); // executeAfter(...);
      analyzeAfter(...); // planAfter(...);
      planAfter(...); // analyzeAfter(...);
      executeAfter(…); // monitorAfter(…);
      return newOut;
} } }
```



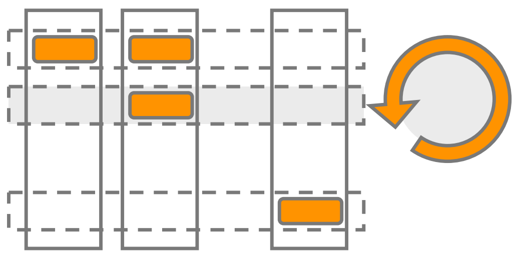

```
class ManagedResource {
  // …
  process(in) {
    // out := ...;
    return out;
  }
  layer MAPE {
    process(in) {
      note := ResourceManager.mapeBefore(self, in, ...);
      out := next(in);
      out := ResourceManager.mapeAfter(self, in, note, out, ...);
      return out;
} } }
```

1 2 3 4

A first sketch…

Acknowledgements

Pascal Costanza, Hidehiko Masuhara, Atsushi Igarashi, Michael Haupt, Malte Appeltauer, Michael Perscheid, Bastian Steinert, Jens Lincke, Marcel Taeumel, Tobias Pape, Tim Felgentreff, Robert Krahn, Carl Friedrich Bolz, Marcel Weiher, Hans Schippers, Tim Molderez, Oscar Nierstrasz, Shigeru Chiba, Hiroaki Inoue, Tobias Rho, Stefan Udo Hanenberg, Dick Gabriel, Dave Thomas, Gilad Bracha, Alan Kay, Dan Ingalls, Alan Borning, Jeff Eastman, Christopher Schuster, Christian Schubert, Gregor Schmidt, Stefan Lehmann, Matthias Springer, …



Web References

- COP-related publications
  - HPI/SWA
    http://www.hpi.uni-potsdam.de/swa/publications/
- Selected systems
  - JCop
    https://github.com/hpi-swa/JCop/
  - ContextJS and Lively Webwerkstatt
    http://lively-kernel.org/repository/webwerkstatt/webwerkstatt.xhtml
  - EventCJ
    http://prg.is.titech.ac.jp/projects/eventcj/
  - Lively Kernel
    http://lively-kernel.org/