

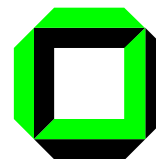
Integrated Deductive Software Design: The KeY Tool

W. Ahrendt T. Baar B. Beckert M. Giese

R. Hähnle W. Menzel W. Mostowski P. H. Schmitt



Chalmers University



Universität Karlsruhe

Linköpings Universitet — Department of Computer Science
Oct 25 2001

Formal Methods in SW Development



There are excellent reasons to use FM in SW Development!

Formal Methods in SW Development



There are excellent reasons to use FM in SW Development!

- **financial (Ariane desaster)**
- **quality (formalization exhibits bugs)**

Formal Methods in SW Development



There are excellent reasons to use FM in SW Development!

- **financial (Ariane desaster)**
- **quality (formalization exhibits bugs)**
- **legal (higher EALs in Common Criteria)**



There are excellent reasons to use FM in SW Development!

- **financial (Ariane desaster)**
- **quality (formalization exhibits bugs)**
- **legal (higher EALs in Common Criteria)**
- **productivity (formalization creates new ways to use tools)**



There are excellent reasons to use FM in SW Development!

- **financial (Ariane desaster)**
- **quality (formalization exhibits bugs)**
- **legal (higher EALs in Common Criteria)**
- **productivity (formalization creates new ways to use tools)**

Moreover:

- **Formal specification/verification of realistic applications feasible (documented in many case studies)**
- **Tools available (KIV, Isabelle, PVS, B-tool, Agda, Nuprl, Nqthm/ACL2, HOL)**



“Many of these techniques are capable of handling industrial-sized examples; in fact, in some cases these techniques are already being used on a regular basis in industry.”

— Ed Clarke & Jeanette Wing

ACM 50th Anniversary Overview Article on FM

**No significant use
of formal methods in
software industry**

? ? ?



Formal Methods must be integrated into commercially used **methodologies, tools, and languages** for software development





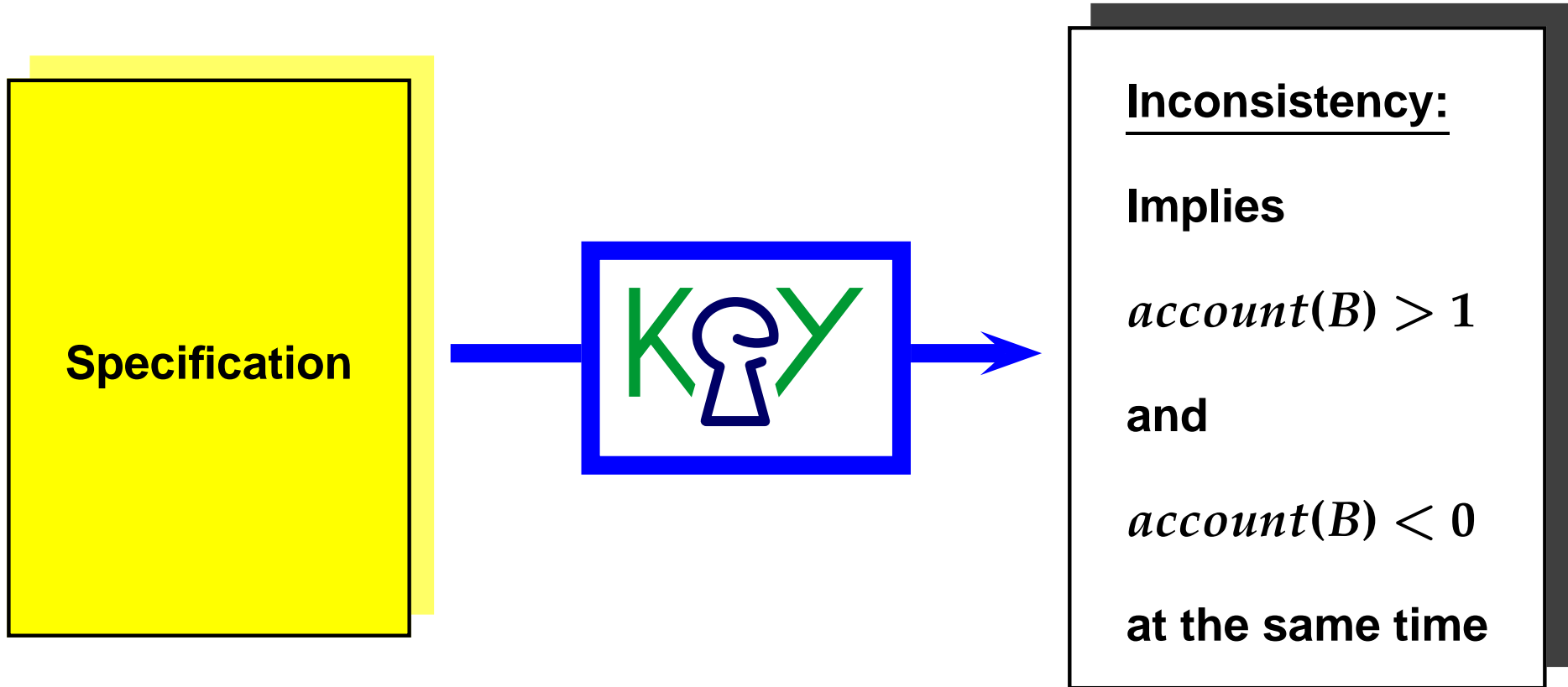
Formal Methods must be integrated into commercially used **methodologies, tools, and languages** for software development

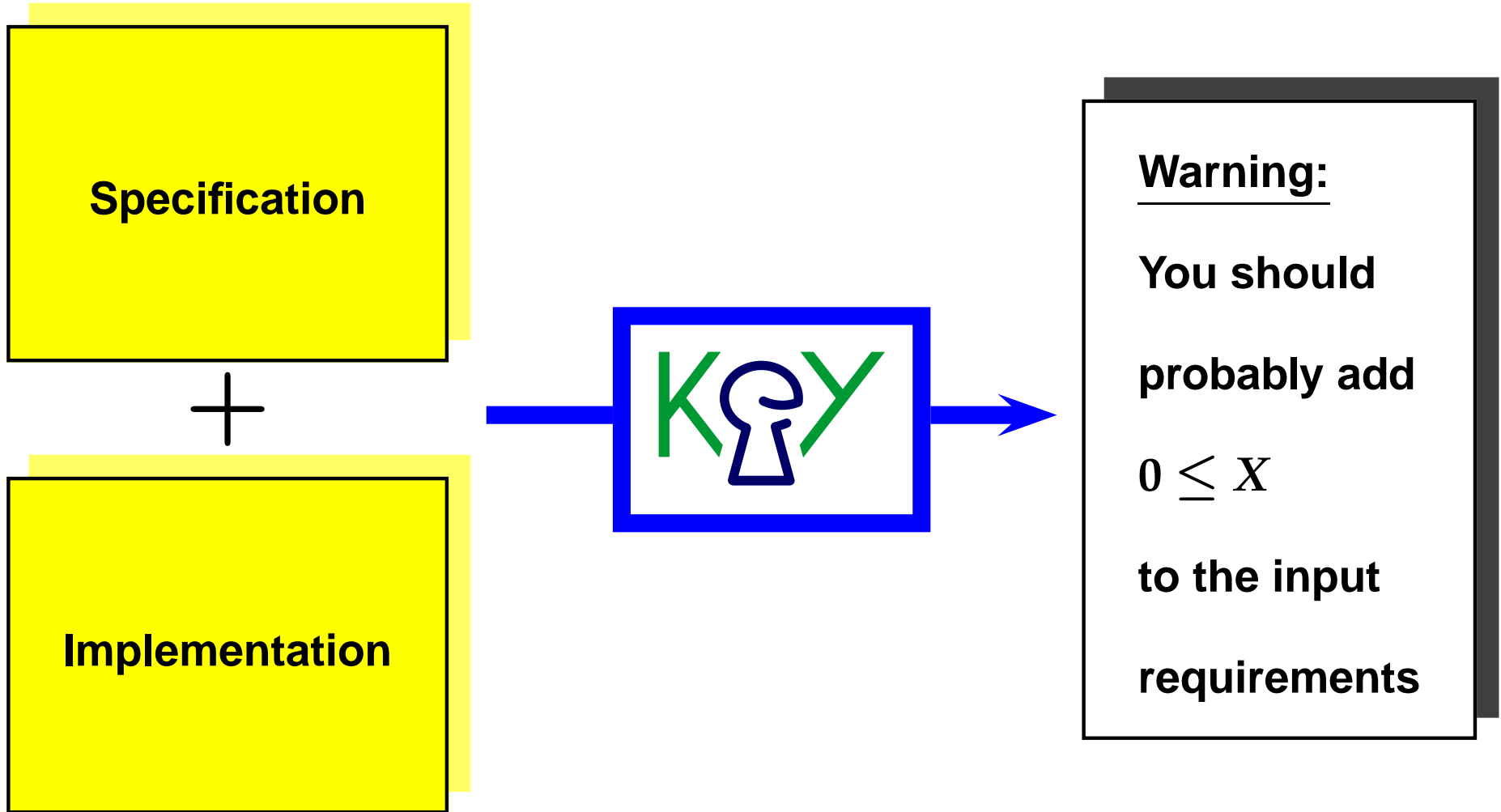
Integrated tool for

- Modeling
- Formal specification
- Verification

of object-oriented programs







State of the KeY Project



- Start:** November 1998
- Funding:** DFG, Vinnova, VR (requested)
- Our group:** Project leaders: R. Hähnle, W. Menzel, P. H. Schmitt
Researchers: One post-doc, four PhD students
Add'l Implementors: 8–10 undergraduate students

State of the KeY Project

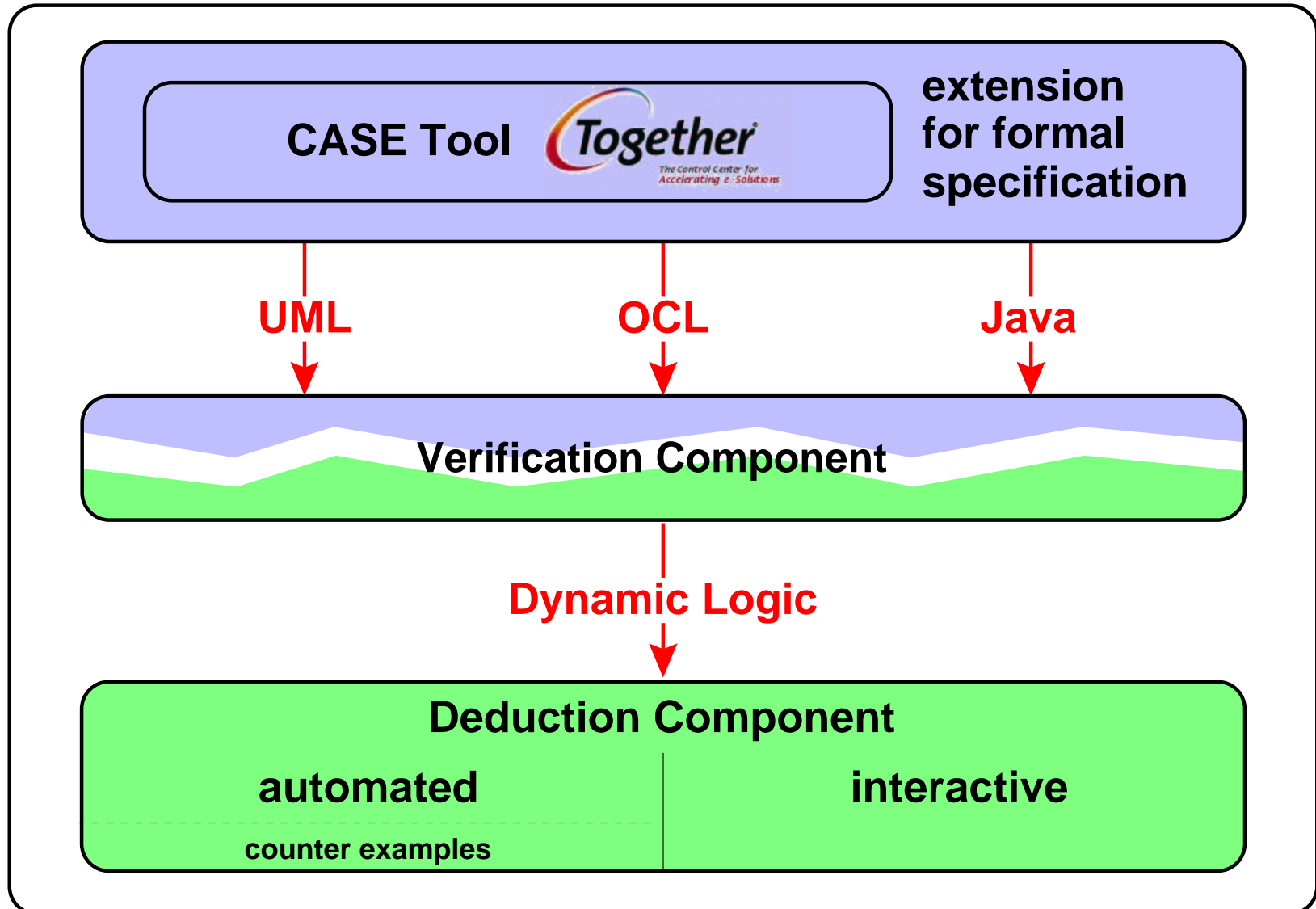


- Start:** November 1998
- Funding:** DFG, Vinnova, VR (requested)
- Our group:** Project leaders: R. Hähnle, W. Menzel, P. H. Schmitt
Researchers: One post-doc, four PhD students
Add'l Implementors: 8–10 undergraduate students
- First Prototype:** Ready (FME 2001 Tools Exhibition)
- ▶ Available for download
 - ▶ Demo prototype

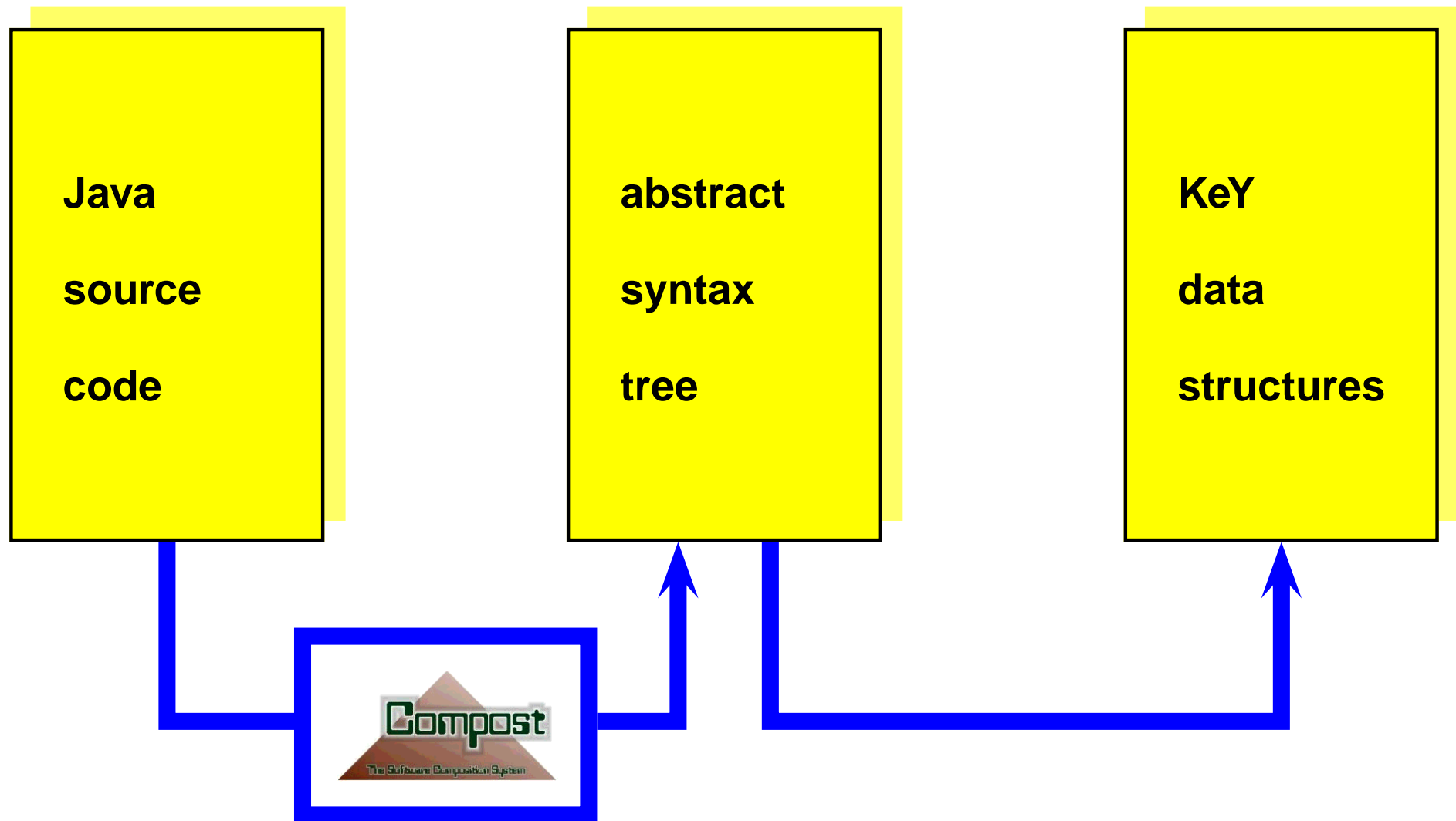
State of the KeY Project



- Start:** November 1998
- Funding:** DFG, Vinnova, VR (requested)
- Our group:** Project leaders: R. Hähnle, W. Menzel, P. H. Schmitt
Researchers: One post-doc, four PhD students
Add'l Implementors: 8–10 undergraduate students
- First Prototype:** Ready (FME 2001 Tools Exhibition)
- ▶ Available for download
 - ▶ Demo prototype
- Functional Tool:** By end of 2001



KeY Uses Real JAVA (JAVA CARD)



UML has semi-formal textual part: Object Constraint Language (OCL)

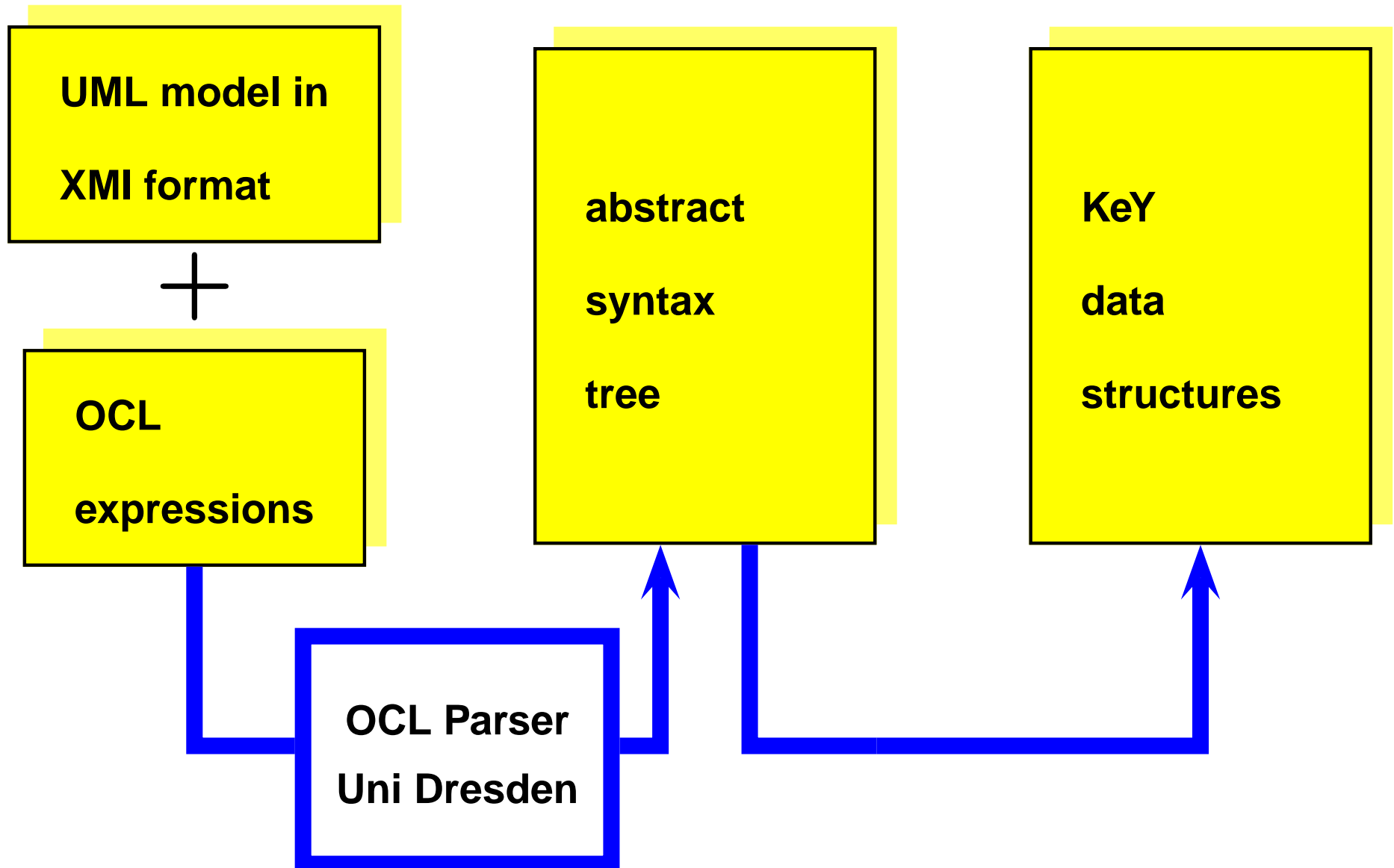
- ⇒ close to OOP syntax**
- ⇒ optimized for navigating in class diagrams, etc.**
- ⇒ typed language**
- ⇒ formal semantics by mapping into typed first-order logic**

OCL used to express restrictions on diagrams:

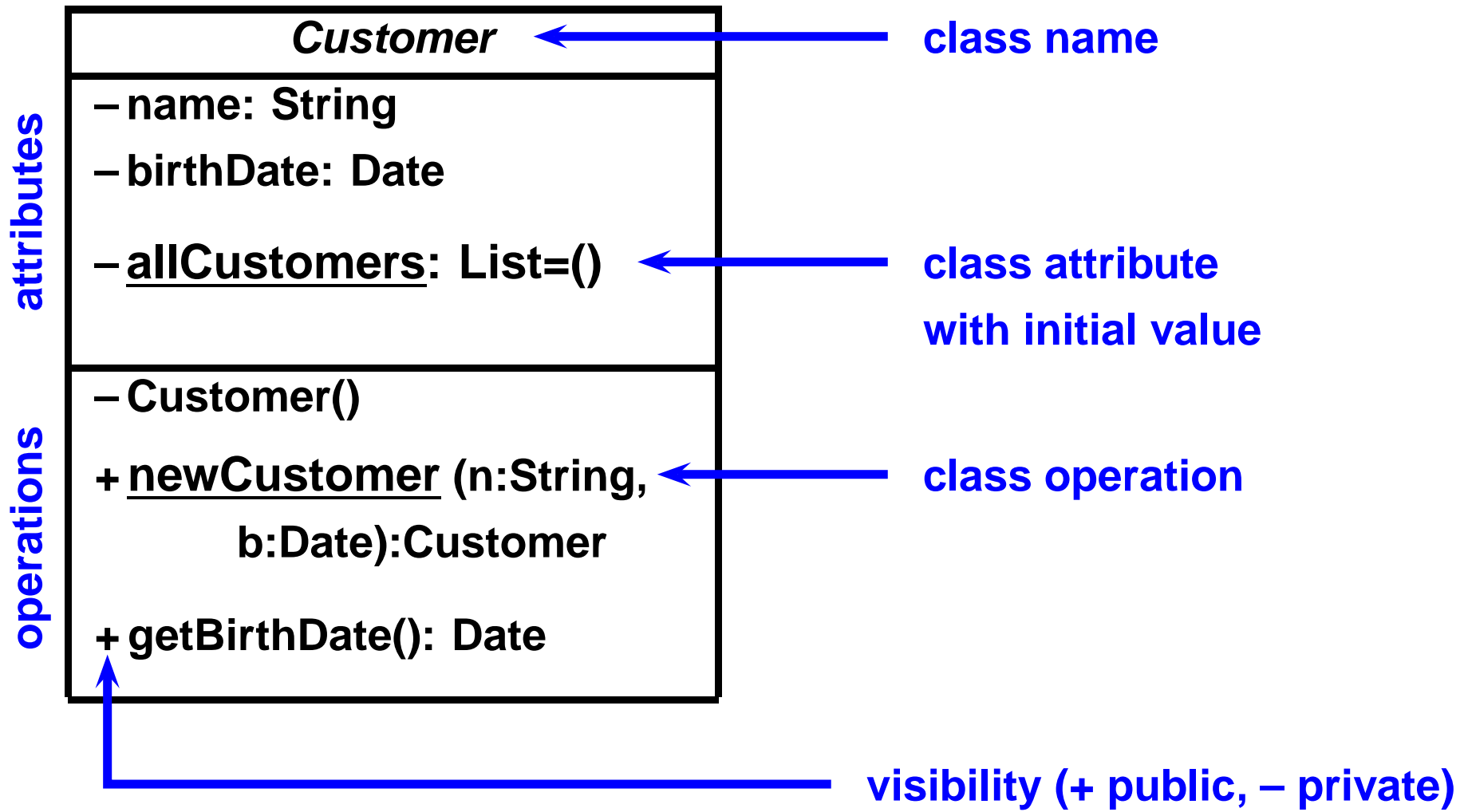
- ⇒ invariants of classes**
- ⇒ pre-/postconditions of operations**
- ⇒ functional specification**

Many further languages, not considered right now: *Alloy, Object-Z, ...*

Processing OCL Expressions



Example: UML Class Diagram



Example: OCL Invariant



“Uniqueness” constraint for the class Customer

context Customer **inv:**

```
Customer.allInstances —> forAll(c1, c2 | (c1.name = c2.name
                                     and
                                     c1.birthDate = c2.birthDate)
                               implies c1 = c2
                              )
```

OCL key words/predef'd ops

Syntax from UML model

Example: Java Program



```
import java.util.*;

public class Customer {

    private String name;
    private Date birthDate;
    private static List allCustomers = new List();

    private Customer(String n, Date b) {
        name = n;
        birthDate = b;
    };

    public static String newCustomer(String n, Date b) {
        ...
    }
}
```

user supplied parts

automatically generated by Together

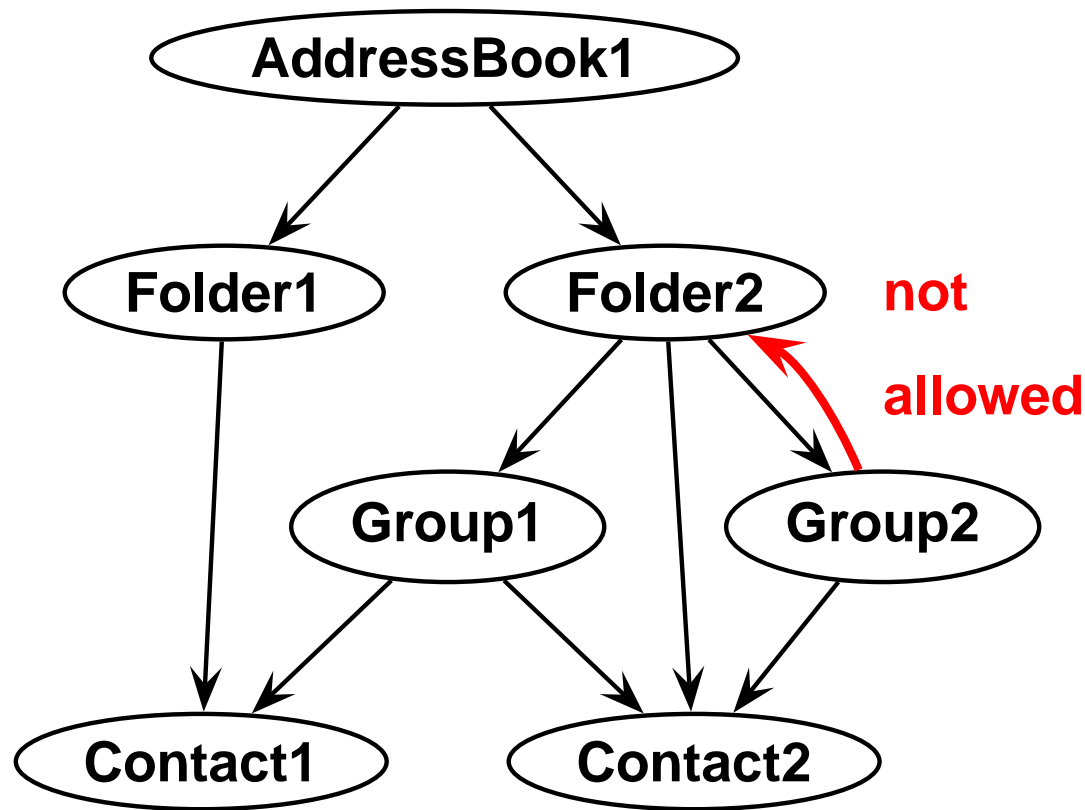


“Part of an email client is an AddressBook, which manages all addresses of email partners. Each user can have only one AddressBook. The AddressBook should be presented to the user in several views (long, short, etc.) and each view must be updated when the AddressBook was changed.

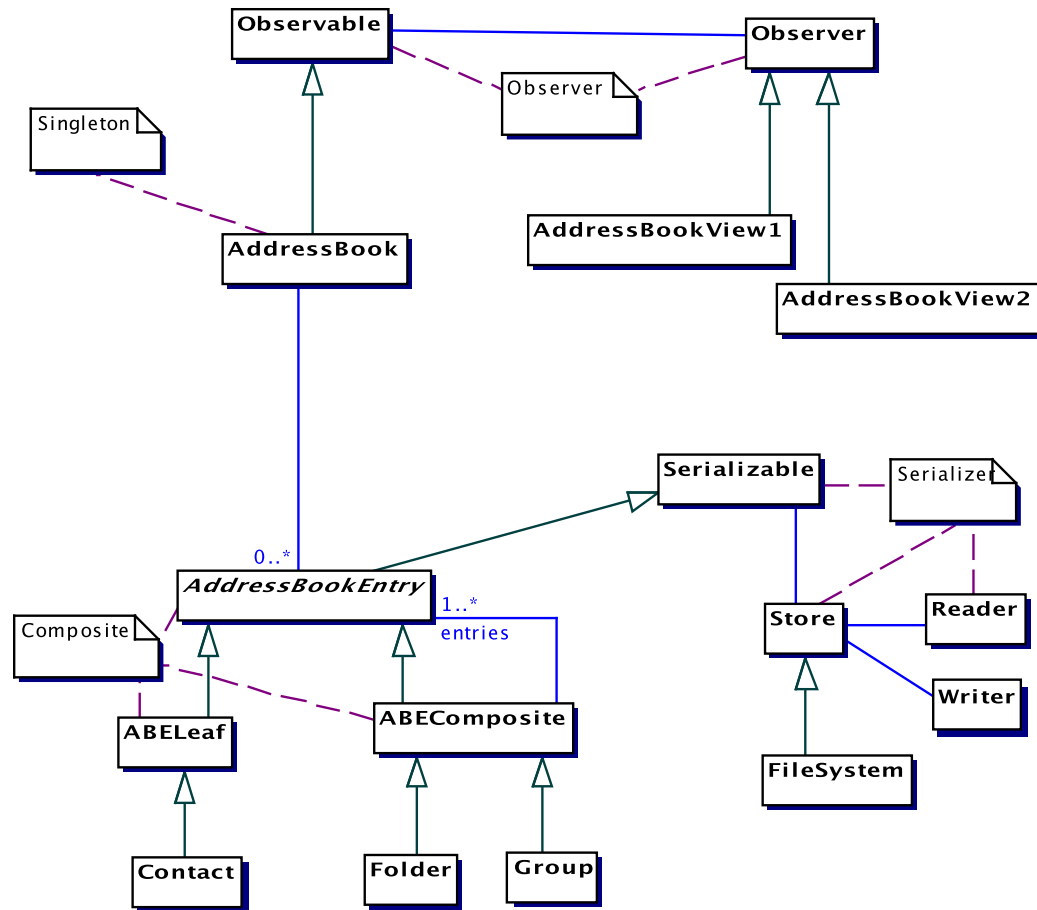
After finishing the session the AddressBook is stored in the FileSystem and reloaded when the next session starts.

An AddressBook has several kinds of hierarchically ordered entries: a Contact represents the address of a single email partner, a Group is a collection of Contacts, a Folder contains Groups or Contacts.”

A Case Study: The *keyMail/S* Project



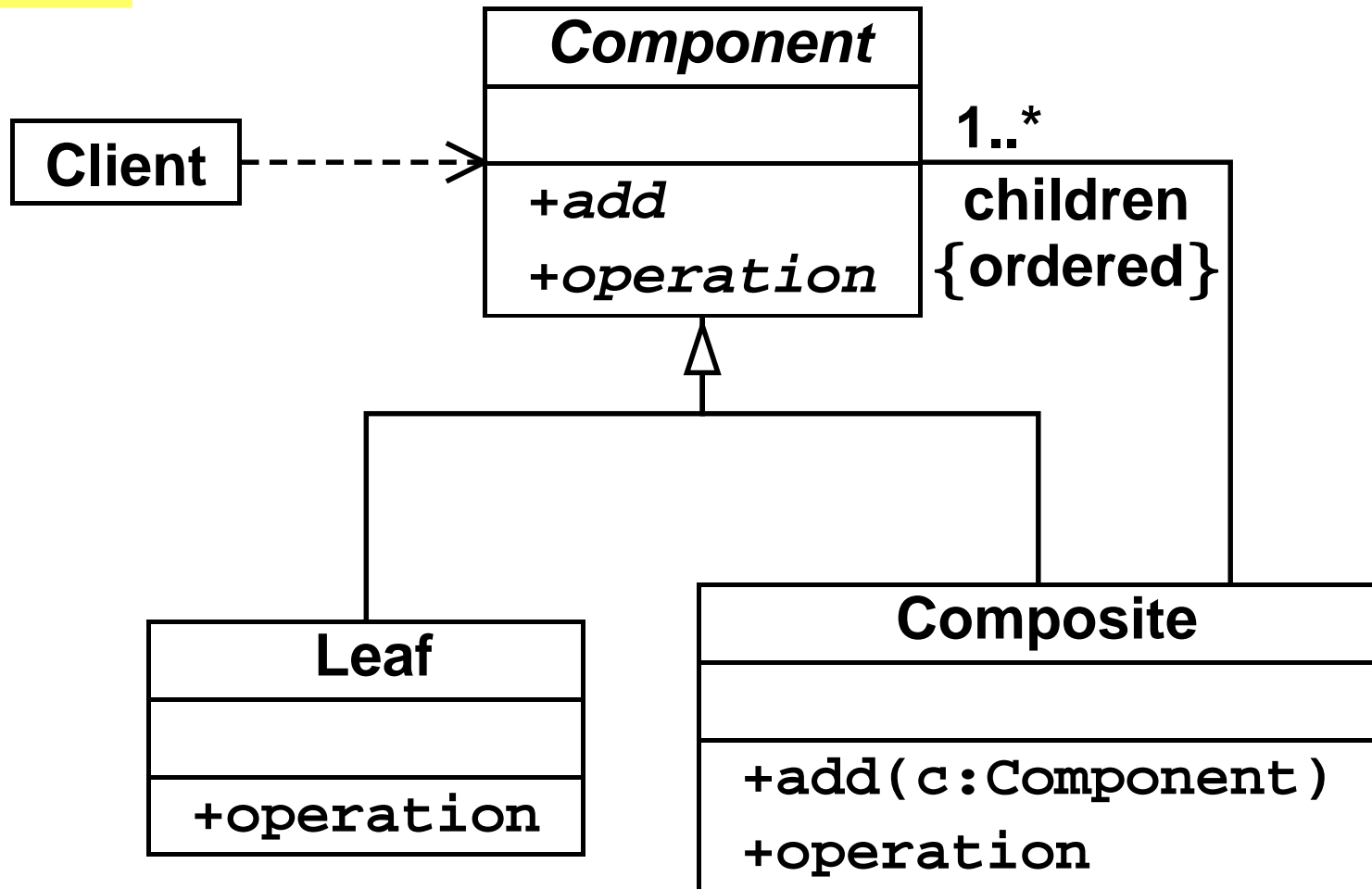
Design with Software Patterns



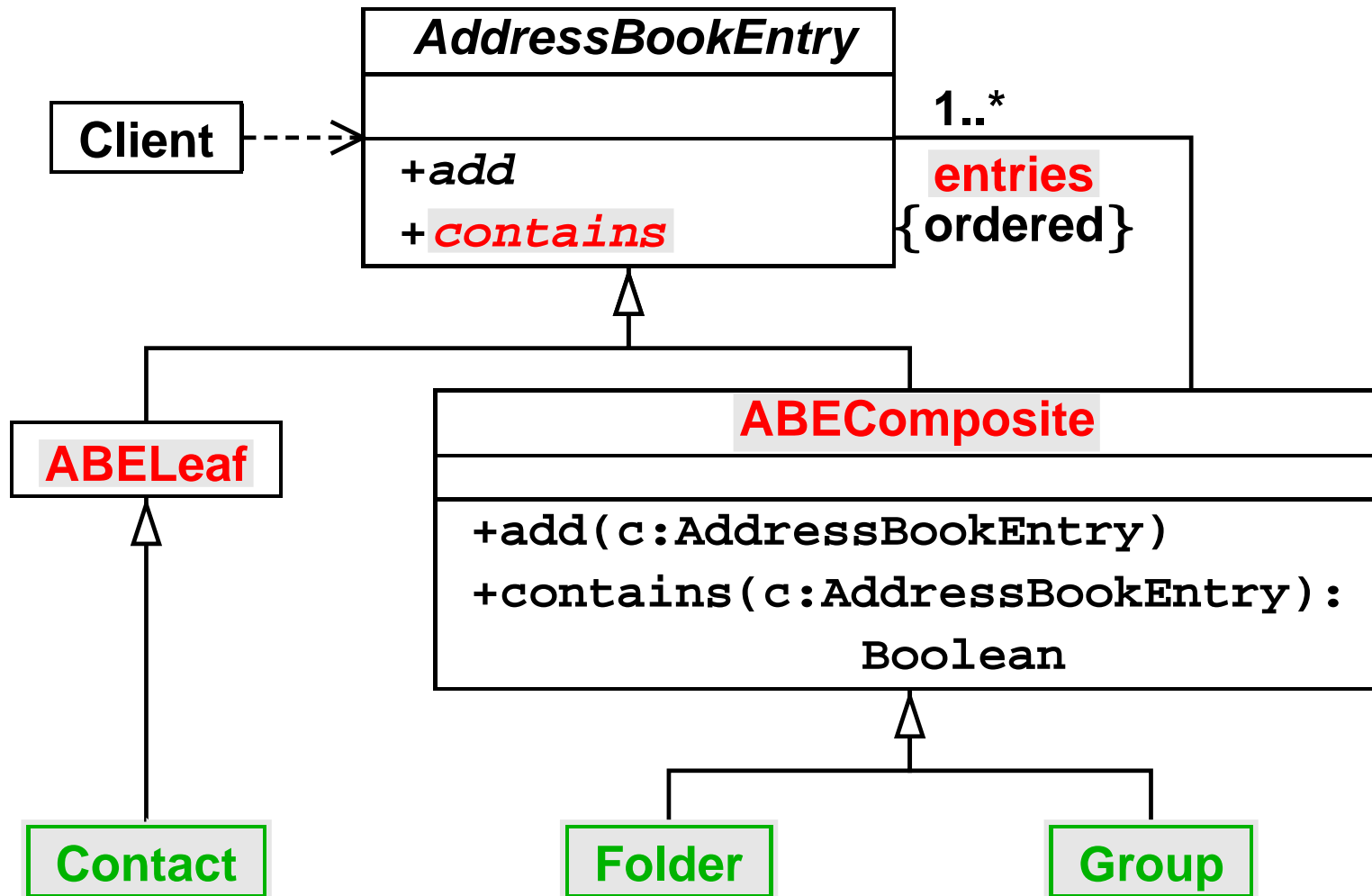
Design Patterns: "Composite"



Structure



Applying “Composite” to AddressBook



Example:

Constraint for “Set Flavor” of “Composite” Pattern:

Each entry occurs exactly once

```
context Composite inv: entriesIsSet
```

```
entries —> size =
```

```
entries —> asSet —> size
```

Adding Constraint Patterns



Example:

Constraint for “Set Flavor” of “Composite” Pattern:

Each entry occurs exactly once

```
context Composite inv: entriesIsSet
```

```
entries —> size =
```

```
entries —> asSet —> size
```

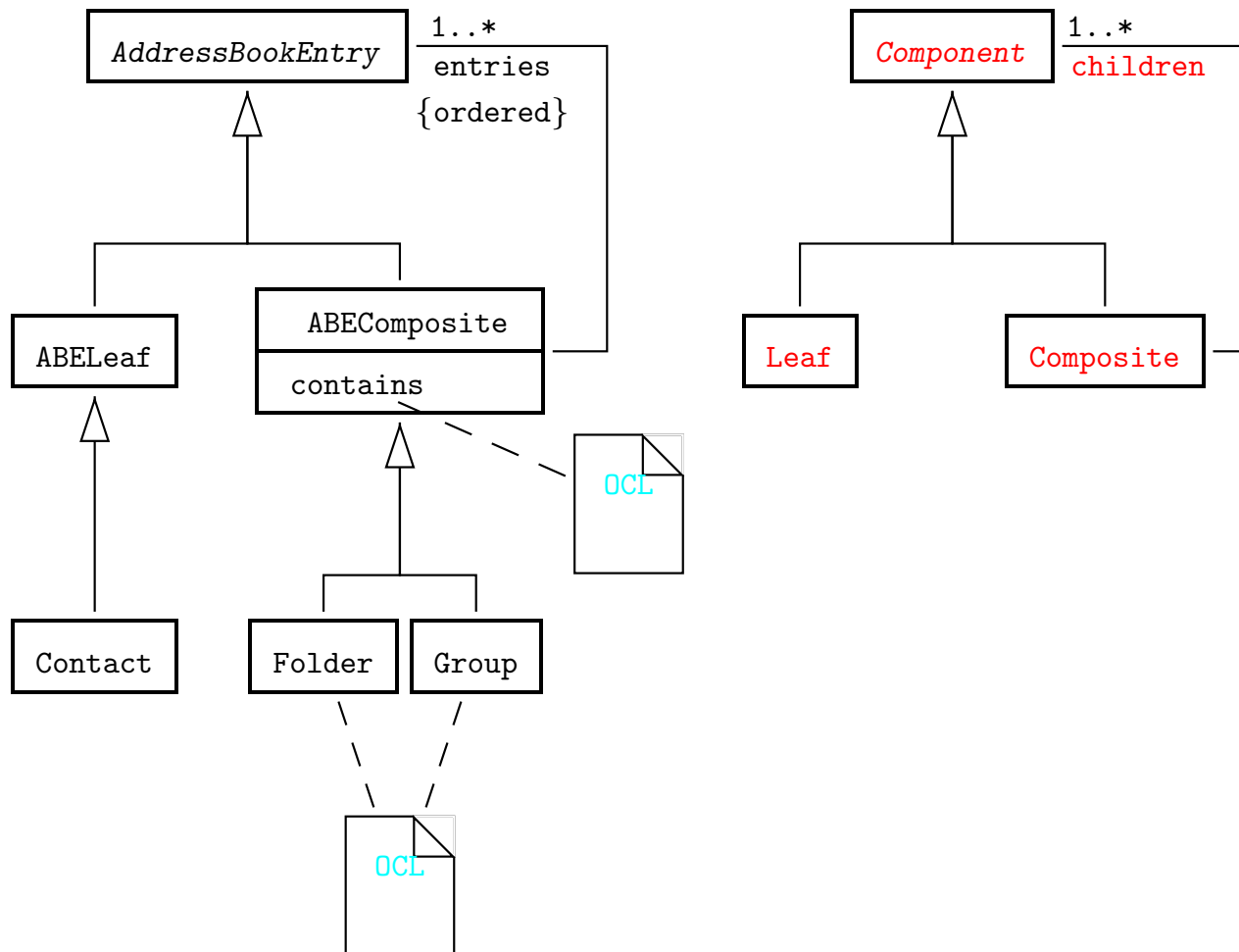
Other Constraints:

- **structure sharing**
- **cyclicity**
- ***groups do not contain folders***

Instantiating Pattern + Constraint



Renaming and Structural Modification



Composite



ABEComposite

```
parameter flavor: {#set, #bag}
context Composite inv:
  if flavor = #set
    then children —> size =
      children —> asSet —> size
    else true
  endif
```

```
context ABEComposite inv:
  if #set = #set
    then entries —> size =
      entries —> asSet —> size
    else true
  endif
```

- state depends on program variables & attributes of existing objects
- evaluation of expressions can have **side effects**:
expressions (program) \neq terms (logic)
- `int`, `short`, arrays, reference types (aliasing)
- exceptions
- initialization of objects

Difference to full JAVA:

- no threads
- no dynamic loading of classes
- not all data types (e.g., `real`, `double`); restricted I/O (no GUI)

Implementation of add() in Folder



```
public boolean add(AddressBookEntry e){
    if (in(e,self.entries)) then {
    } else {
        self.entries = insert(e,self.entries);
    }
}
```

Use OCL's `self` instead of JAVA's `this`

`entries` implemented abstractly over standard list ADT

OCL

context Folder inv: entriesIsSet

self.entries \rightarrow size =

self.entries \rightarrow asSet \rightarrow size

Dynamic Logic

$(\forall f: \text{Folder})$

$(\text{size}(f.\text{entries}) \doteq \text{size}(\text{asSet}(f.\text{entries})))$

- **Assumption:**

Invariant holds true **before** execution of method `add()`

- **Claim:**

Invariant holds true **after** execution of method `add()`

- **Proof:**

Theorem prover applies rules for **symbolic execution** of the Java implementation of `add()`

Proof obligation:

$(\forall f: \text{Folder}) \text{entriesIsSet}(f)$ is an invariant of method `add`

with abbreviation:

$\text{entriesIsSet}(f) := \text{size}(f.\text{entries}) \doteq \text{size}(\text{asSet}(f.\text{entries}))$

Claim to be proven:

$\text{entriesIsSet}(f) \vdash \langle \text{add}(e); \rangle \text{entriesIsSet}(f)$

Symbolic Execution: Method Call



$entriesIsSet(f) \vdash \langle \text{add}(e); \rangle entriesIsSet(f)$

```
public boolean add(AddressBookEntry e) {
    if (in(e, self.entries)) then {} else {
        self.entries = insert(e, self.entries);
    }
}
```

$entriesIsSet(f) \vdash$

\langle `if in(e, self.entries) then {} else { self.entries = insert(e, self.entries); }` $\rangle entriesIsSet(f)$

Symbolic Execution: Conditional



$entriesIsSet(f) \vdash$

$\left\langle \begin{array}{l} \text{if } in(e, self.entries) \text{ then } \{\} \text{ else } \{ \\ \quad self.entries = insert(e, self.entries); \\ \} \end{array} \right\rangle entriesIsSet(f)$

Case distinction as to whether condition is true or not

$entriesIsSet(f), in(e, self.entries) \vdash \langle \{\} \rangle entriesIsSet(f)$

$entriesIsSet(f), \neg in(e, self.entries) \vdash$

$\langle self.entries = insert(e, self.entries); \rangle entriesIsSet(f)$

Symbolic Execution: Assignment


$$entriesIsSet(f), \neg in(e, self.entries) \vdash$$
$$\langle self.entries = insert(e, self.entries); \rangle entriesIsSet(f)$$

Assignments are translated into suitable **state updates**, here:

- o is reference to object of class C
- a is instance attribute of class C
- t is *logical* term

Compute update of value of a in o with t

$$entriesIsSet(f), \neg in(e, self.entries) \vdash$$
$$(entriesIsSet(f)) \{ self.entries \leftarrow insert(e, self.entries) \}$$


$$\text{entriesIsSet}(f), \neg \text{in}(\mathbf{e}, \mathbf{self.entries}) \vdash$$
$$(\text{size}(f.entries) \doteq \text{size}(\text{asSet}(f.entries))) \{ \mathbf{self.entries} \leftarrow \text{insert}(\mathbf{e}, \mathbf{self.entries}) \}$$

f and *self* refer to same object:

$$\text{entriesIsSet}(f), \neg \text{in}(\mathbf{e}, \mathbf{self.entries}), f \doteq \mathbf{self} \vdash$$
$$\text{size}(\text{insert}(\mathbf{e}, \mathbf{self.entries})) \doteq \text{size}(\text{asSet}(\text{insert}(\mathbf{e}, \mathbf{self.entries})))$$

f and *self* refer *not* to same object:

$$\text{entriesIsSet}(f), \neg \text{in}(\mathbf{e}, \mathbf{self.entries}), \neg(f \doteq \mathbf{self}) \vdash \text{entriesIsSet}(f)$$



- **Early payback**
- **Incremental gain for incremental effort**
- **Integrated use**
- **Ease of use**
- **Efficiency**
- **Ease of learning**
- **Error detection oriented**
- **Focused analysis**
- **Evolutionary development**



- Early payback →
- Incremental gain for incremental effort →
- Integrated use →
- Ease of use →
- Efficiency →
- Ease of learning →
- Error detection oriented →
- Focused analysis →
- Evolutionary development →





Methods and tools should provide significant benefits almost as soon as people begin to use them.



Incremental Gain for Incremental Effort



Benefits should increase as developers get more adept or put more effort into writing specifications or using tools.



Methods and tools should work in conjunction with each other and with common programming languages and techniques.

Developers should not have to *buy into* a new methodology completely to begin receiving benefits.

The use of tools for formal methods should be integrated with that of tools for traditional software development, e.g., compilers and simulators.



Tools should be as easy to use as compilers, and their output should be as easy to understand



Tools should make efficient use of a developer's time.

Turnaround time with an interactive tool should be comparable to that of normal compilation. Developers are likely to be more patient, however, with completely automatic tools that perform more extensive analysis.



Notations and tools should provide a starting point for writing formal specifications for developers who would not otherwise write them. The knowledge of formal specifications needed to start realizing benefits should be minimal.





**Methods and tools should be optimized for finding errors,
not for certifying correctness.**

**They should support generating counterexamples as a means
of debugging.**



Methods and tools should be good at analyzing at least one aspect of a system well, e.g., the control flow of a protocol.

They need not be good at analyzing all aspects of a system.





Methods and tools should support evolutionary system development by allowing partial specification and analysis of selected aspects of a system.

