Complex Event Processing with Event Modules

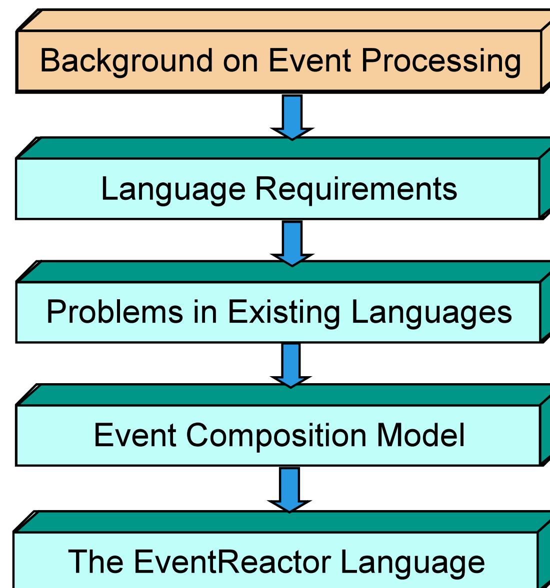
SOMAYEH MALAKUTI

SOFTWARE TECHNOLOGY GROUP

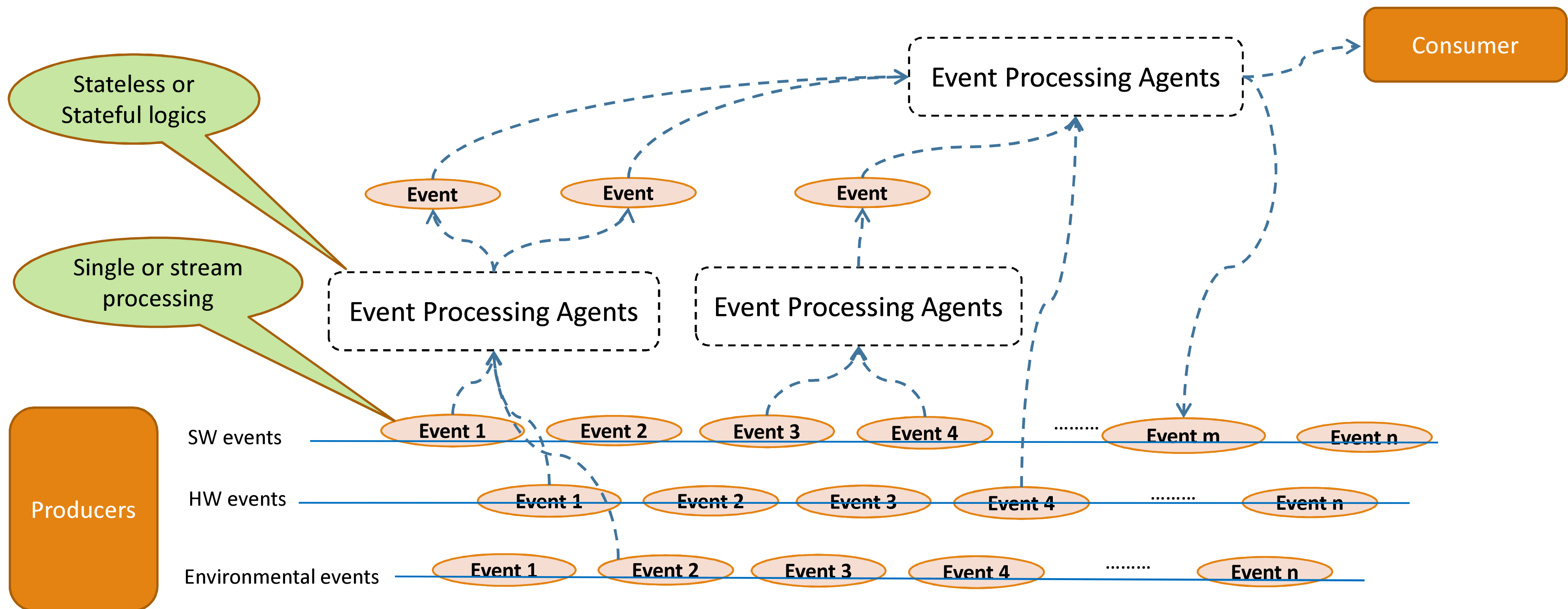
TECHNICAL UNIVERSITY OF DRESDEN, GERMANY

28.10.2013

Outline



Background: Event Processing

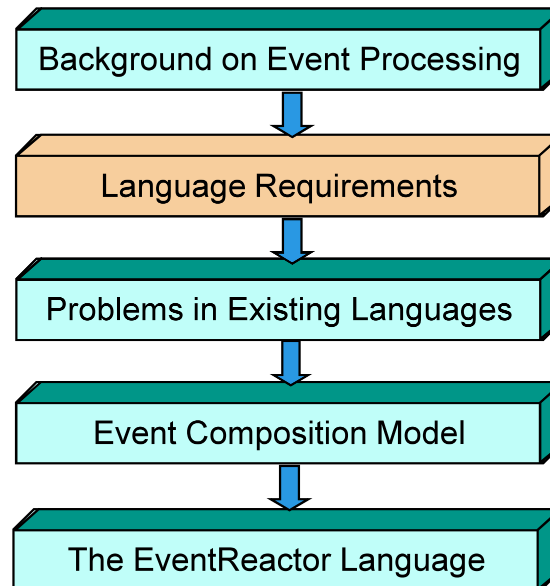


Background: Event Processing (cont.)

- There are various kinds of applications whose base functionality ***is extended*** with certain kind of ***event processing***:
  - Runtime verification techniques check the events that occur in software against the formally specified properties of the software, and detect the failures.
  - Self-adaptive software systems monitor environmental changes, analyze them, and adapt themselves accordingly.
  - Traffic monitoring software systems receive traffic flow information from the sensors that are embedded in roads, and reason about traffic flow in the roads.

- We face the following challenges:
  - Modular definition of event processing logics.
  - Composition of event processing logics with base modules.

Outline

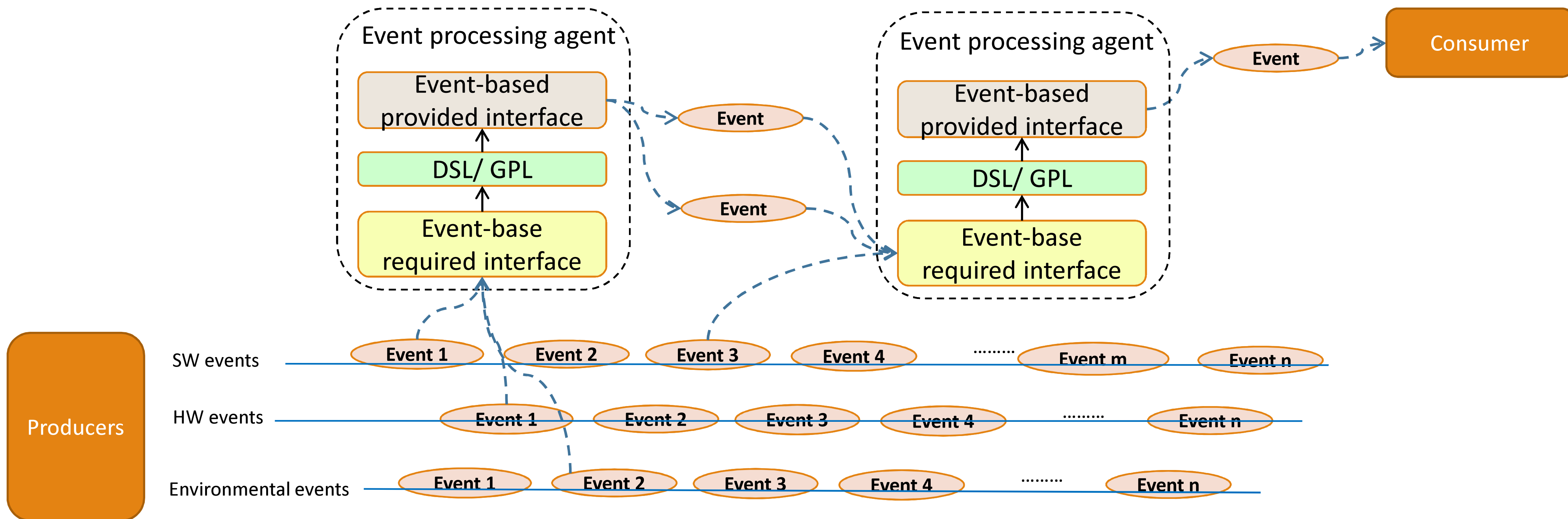


Language Requirements

- **Event representation:** Events are the core abstractions in event processing applications, which may be provided by different kinds of producers.
- A language must provide suitable means to
  - Define the events of interest
  - Detect their occurrence
  - Select them from event streams
  - Provide them to event processing agents and event consumers
- If a language falls short in these matters, programmers may be obliged to provide workaround code in the implementations, which may increase the complexity of the programs.

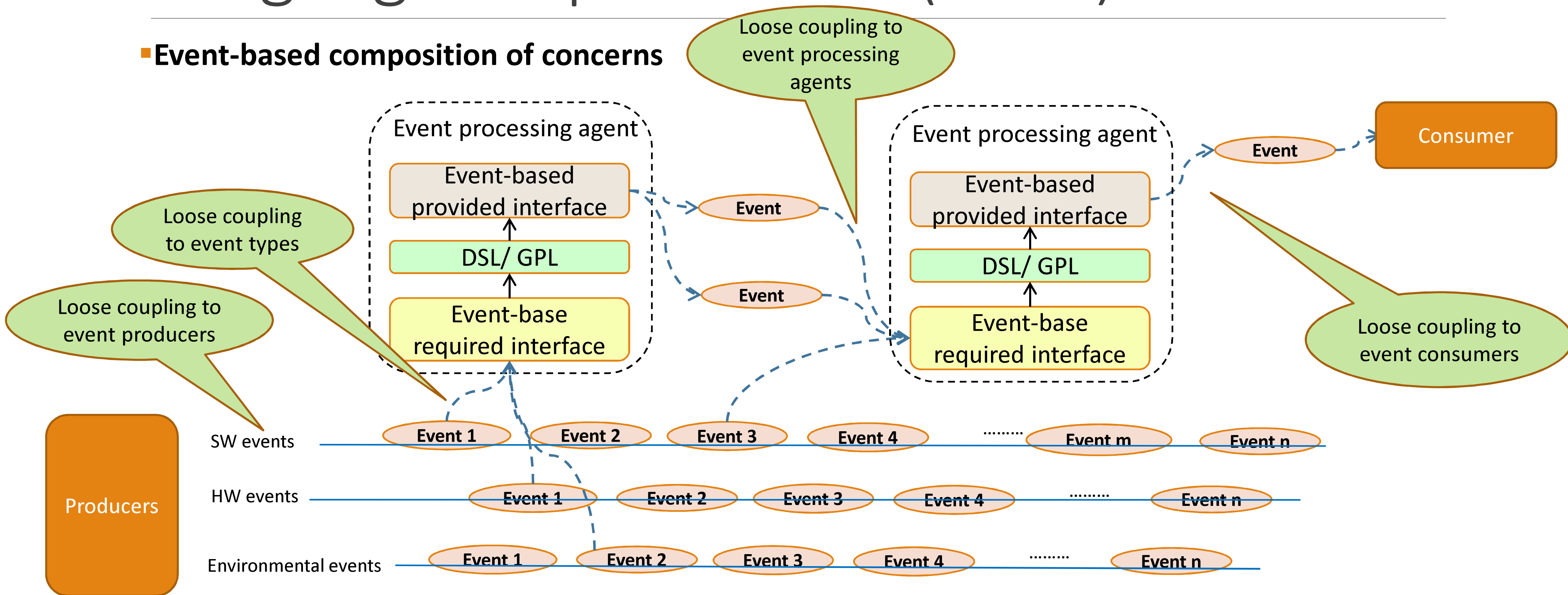
Language Requirements (cont.)

- **Event-based modularization of concerns**

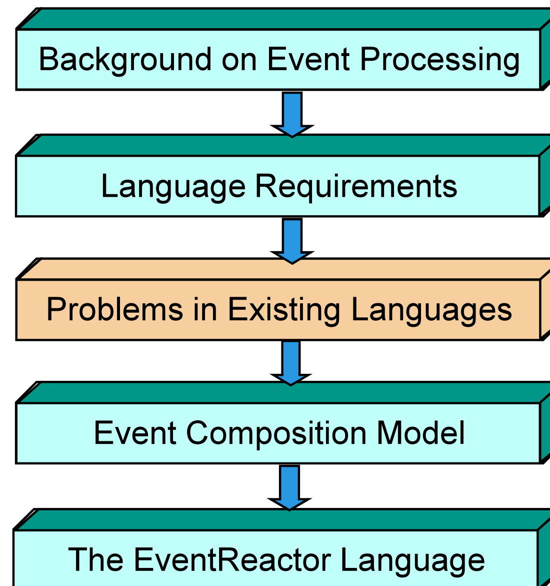


Language Requirements (cont.)

- **Event-based composition of concerns**



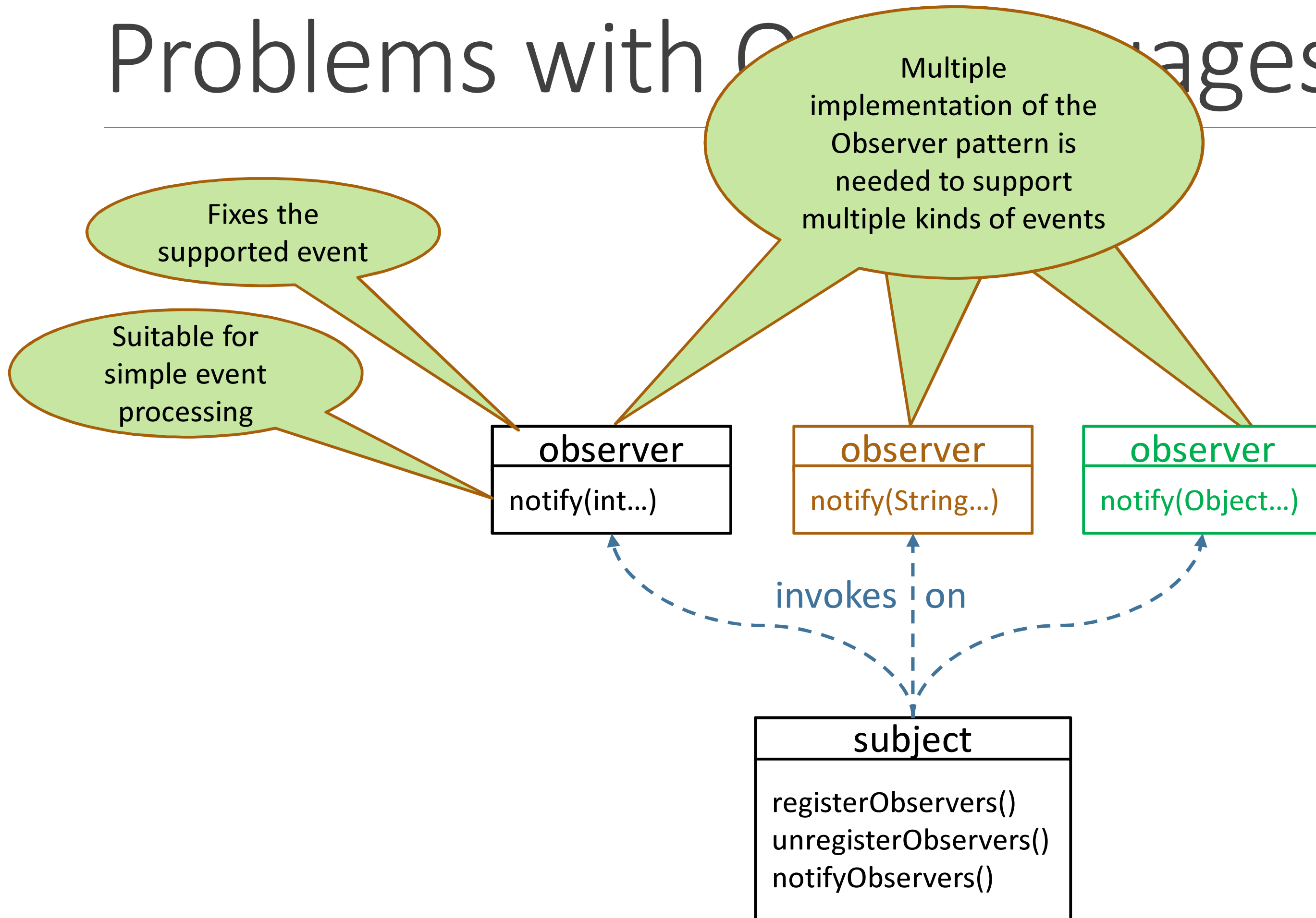
Outline



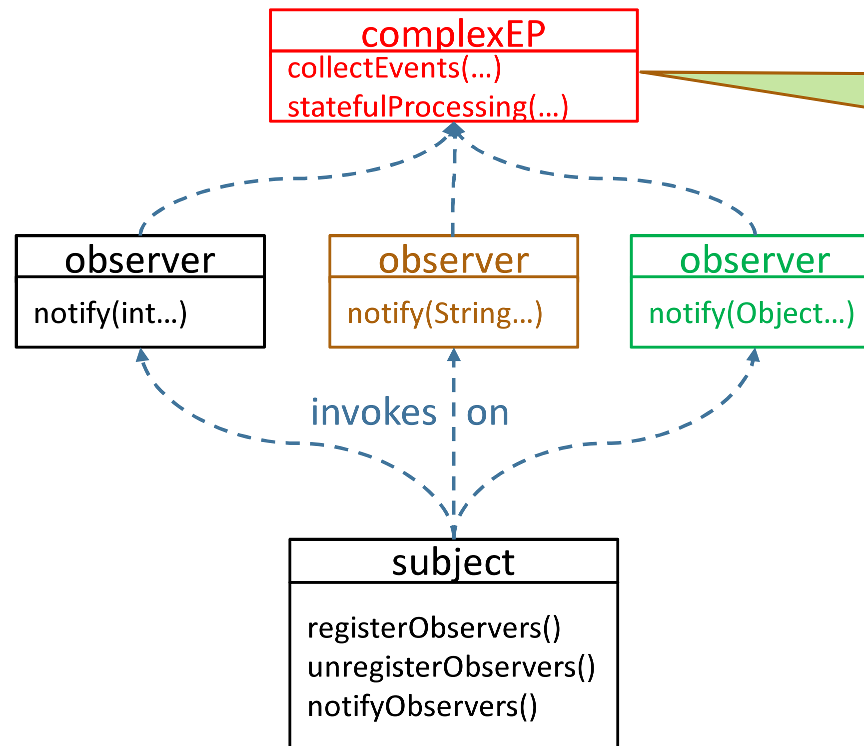
Problems with OO Languages

- In object-oriented (OO) languages, ***objects*** are means to modularize the concerns of interest.
- Objects communicate with each other via ***message passing*** (e.g. method invocation, events).
- Techniques such as ***polymorphism*** along with various ***design patterns*** can be adopted to achieve loose coupling in the implementations.

Problems with O… …ages (cont.)

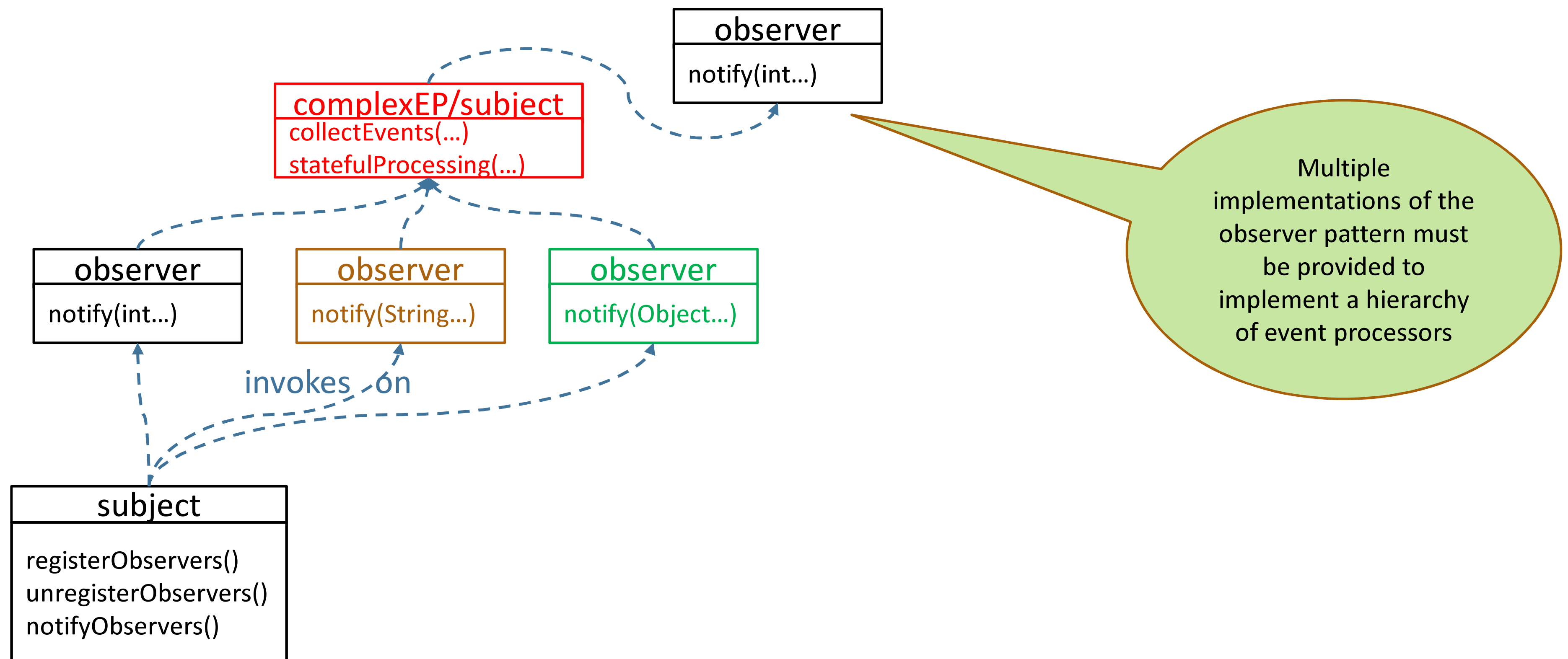


Problems with OO Languages (cont.)

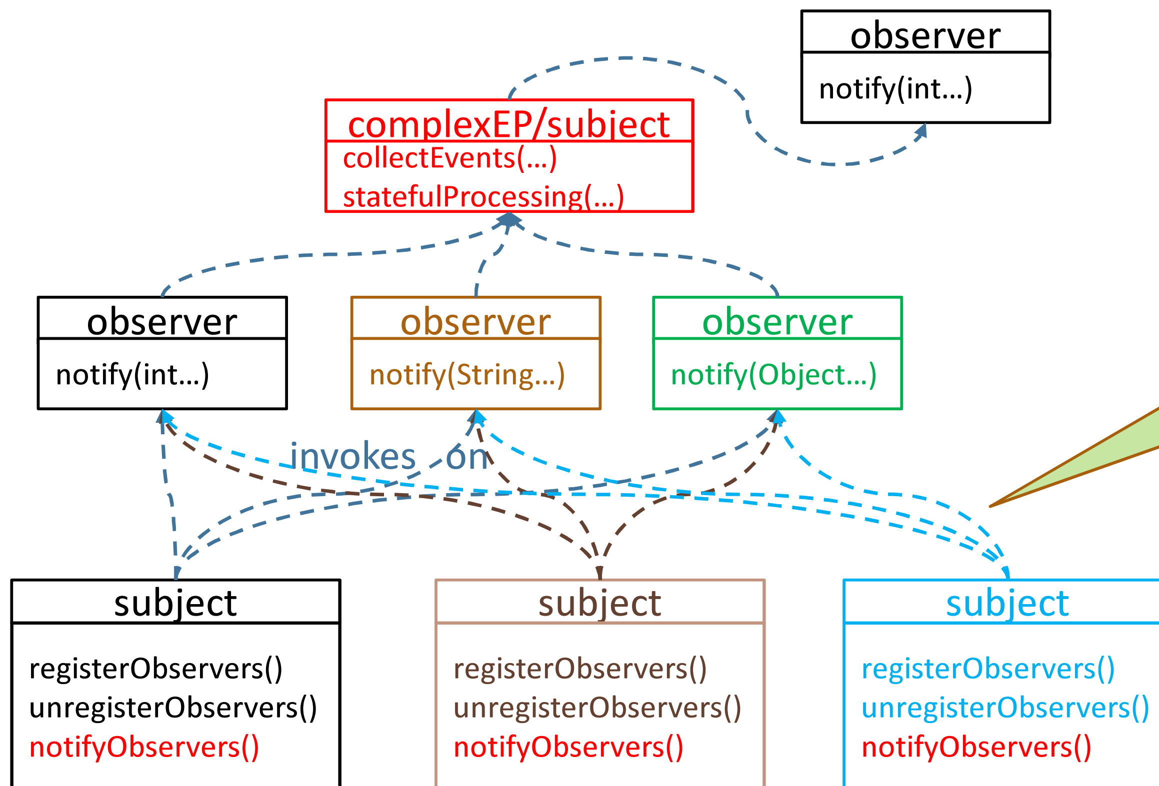


To implement event stream processing or complex event processing, extra code must be provided to collect events from observers

Problems with OO Languages (cont.)



Problems with OO Languages (cont.)



If events must be collected from multiple sources, the invocations to observers scatters across multiple subjects.

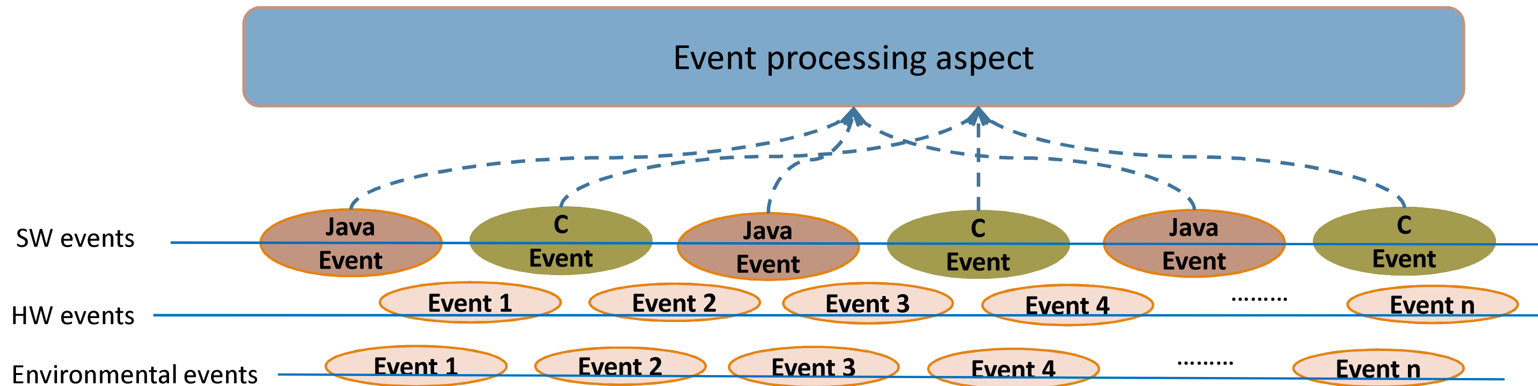
Problems with AO Languages (cont.)

- Due to the ***crosscu*** ... consider adopting aspect-oriented (A(
- In AO languages:
  - ***Join points*** are me
  - ***Pointcut designat***
  - ***Advice*** code is a m
  - In many AO langu. ... designators and advice code.

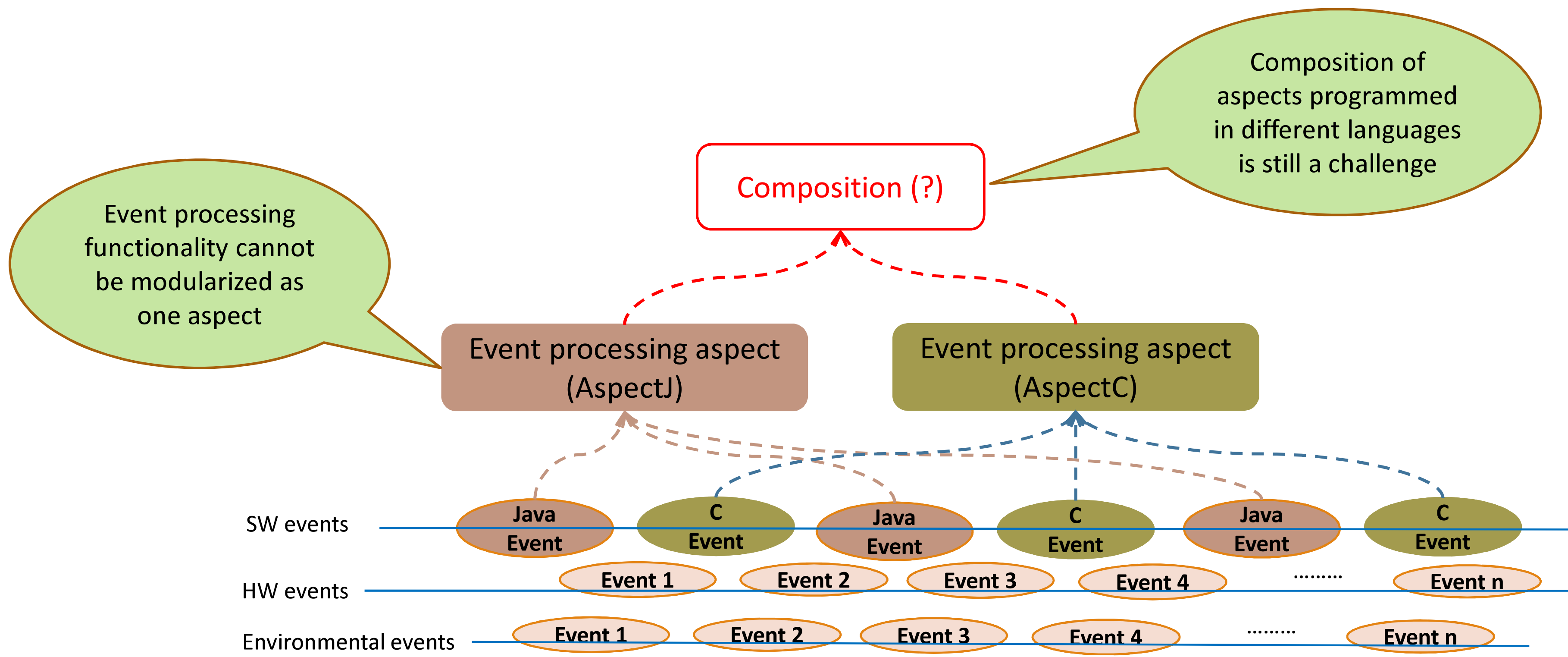
```
public aspect monitoring {
  boolean isOpen;

  pointcut readFile() : call (* File.read());
  pointcut openFile() : call (* File.open());

 before () : openFile() {isOpen = true;}
 before () : readFile() {
    if (isOpen == false)
    throw new MyFileException("Error");
 }
}
```

Problems with AO Languages (cont.)

- **Event representation:** the set of supported events is defined by the join point model of the adopted AO language.
  - Some AO languages such as AspectJ and Compose* support a fixed join point model.
  - If desired events are not defined in the join point model, workaround mappings must be provided. ; this may increase the

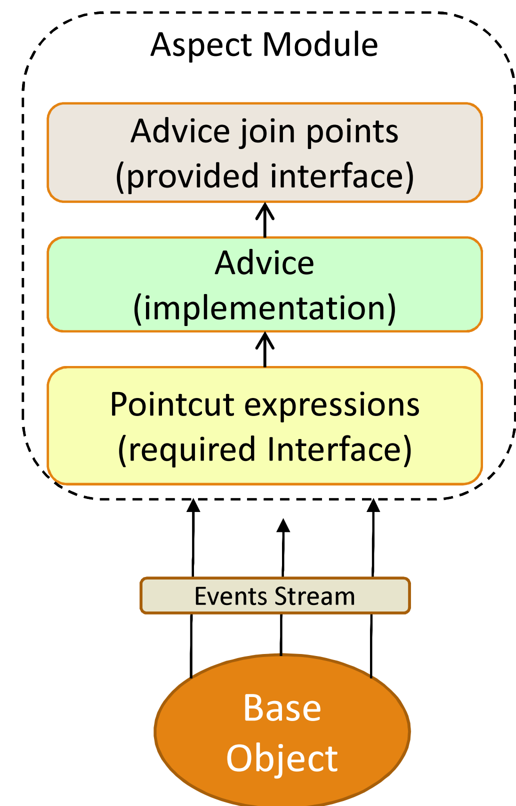


Problems with AO Languages (cont.)



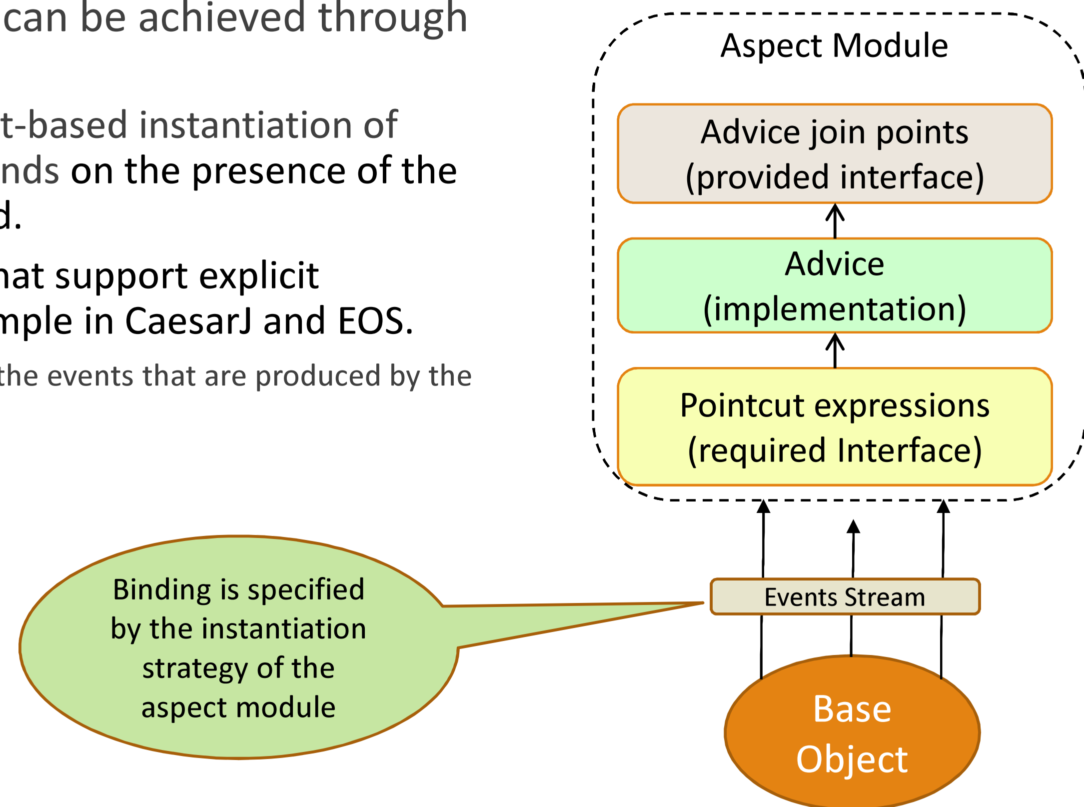
Problems with AO Languages (cont.)

- **Event-based modularization of concerns:**
  - Limited expression power of pointcut designators is a known problem.
  - There is a limited number of AO DSLs; they fall short in defining event processing logics.
  - AO languages have limited support to select the join points that are activated within aspects.



Problems with AO Languages (cont.)

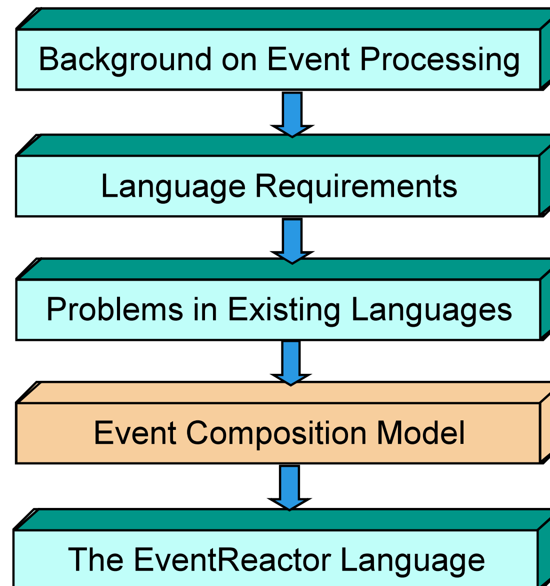
- **Event-based composition:** such a composition can be achieved through join points and pointcut designators.
  - In AspectJ-like languages, which support pointcut-based instantiation of aspects, the presence of an aspect instance depends on the presence of the base object to which the aspect instance is bound.
  - Such a coupling does not exist in the languages that support explicit construction and deployment of aspects; for example in CaesarJ and EOS.
    - In these languages, however, an aspect is limited to process the events that are produced by the objects on which it is deployed.



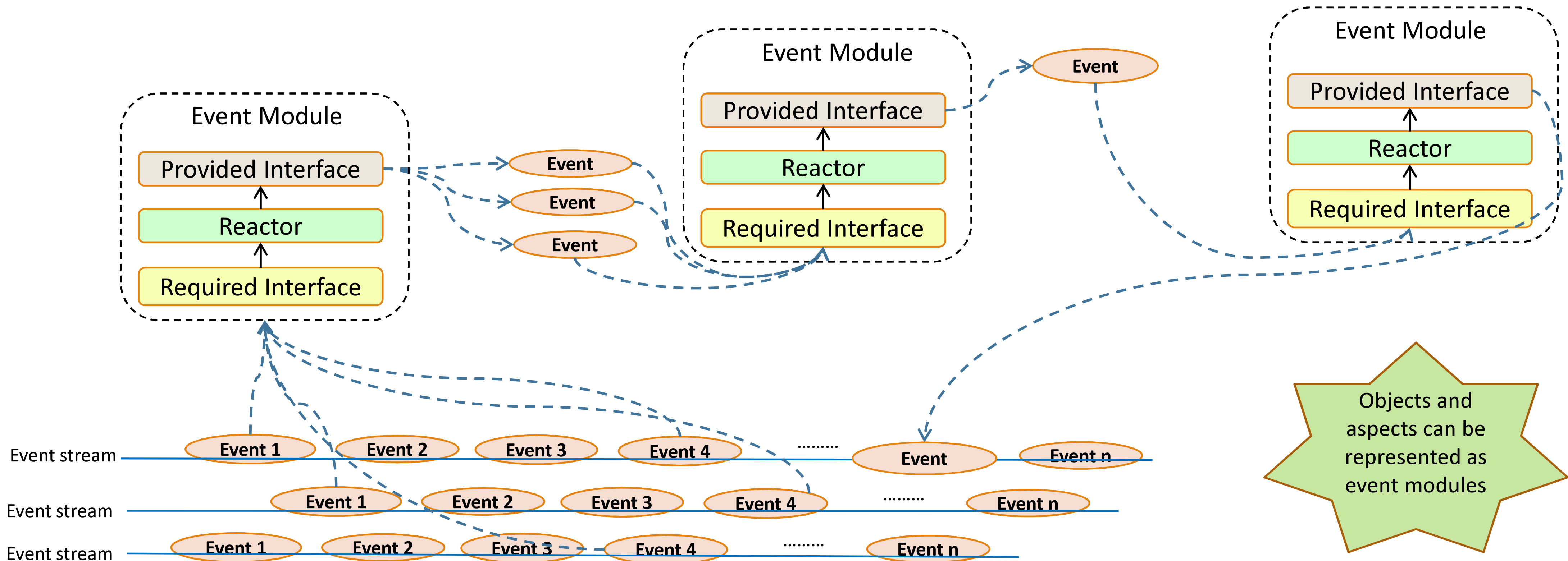
Dedicated Languages

- Several different dedicated languages are introduced for event stream processing, examples are Esper and EPL of Oracle.
  - They have a dedicated focus on the event processing logics, with no support for modularization and composition of concerns.
- There are numerous DSLs introduced in the literature, 30+ only for the domain of RV.
  - The advanced RV DSLs adopt an AO language (such as AspectJ) as their base languages. Hence, they suffer from the same limitations as the AO languages.
- There are many languages and language extensions with a dedicated support for event processing:
  - Event-delegate mechanism of C#, Ptolemy, EventJava, EventCJ, …

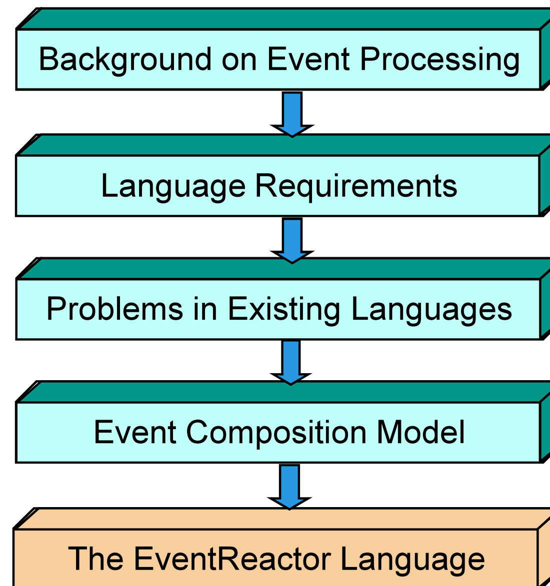
Outline



Event Composition Model



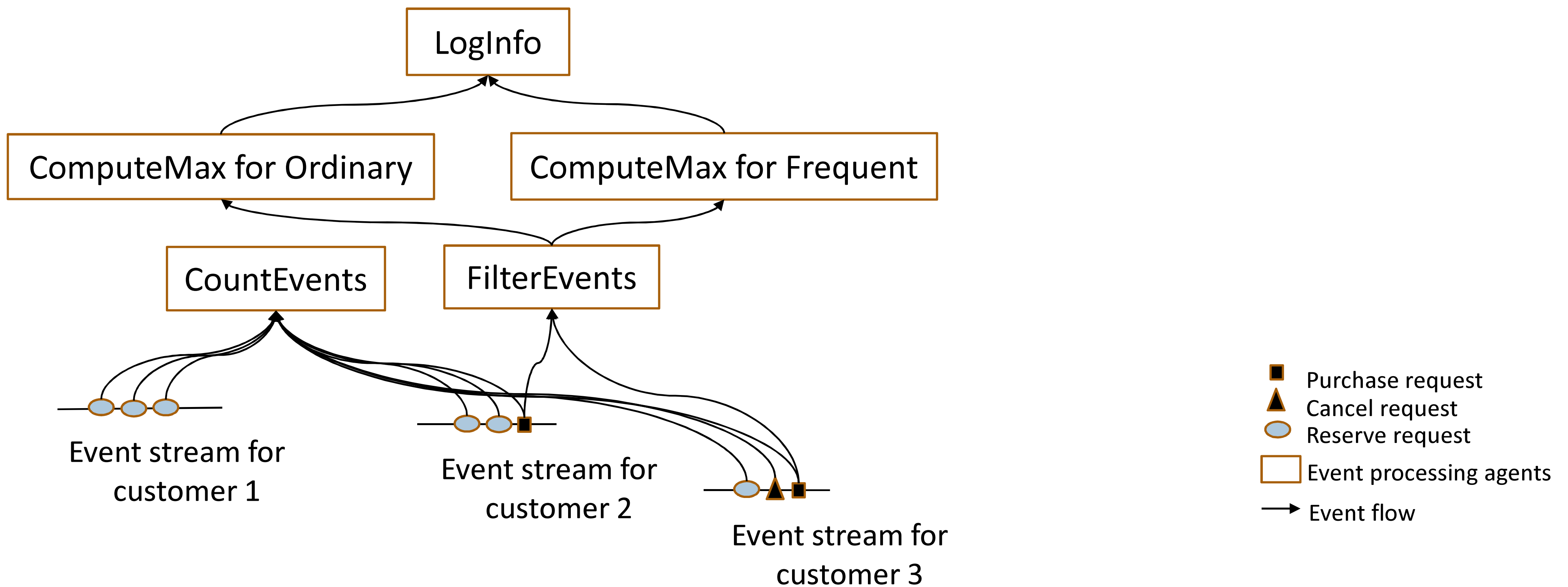
Outline



The EventReactor Language

- The EventReactor language implements the concepts introduced by Event Composition Model.
  - It offers dedicated languages to define event types and events.
  - It offers APIs to publish events from Java and non-Java programs.
  - It makes use of the Prolog language to select primitive events of interest based on event attributes.
  - It offers constructs to define event modules.
  - It offers dedicated operators to compose event modules.

Illustrative Example



Specification of Event Types

Application-specific event types can be defined.

An event type is a data structure, defining a set of attributes.

There can be inheritance relation among event types

```
eventtype Purchase extends EventType{
dynamiccontext:
    long customerID;
    long productID;
}
eventtype FrequentPurchase extends Purchase{
dynamiccontext:
    Purchase inner;
    long frequency;
}
eventtype OrdinaryPurchase extends Purchase{...}
eventtype MaxPurchase extends Purchase{...}
```

Publishing Events

```
Purchase event = new Purchase ();
event.dynamiccontext.customerID = 1;
event.dynamiccontext.productID = 10;
EventReactor.publish(event);
```

To publish an event from a Java program, it must be instantiated, and its dynamic attributes must be initialized.

Dedicated API for publishing an event

Specification of Event Modules (EventReactor 1.0)

```
eventpackage example{
 selectors
      p_event = {E | hasEventType (E, 'Purchase')};
      r_event = {E | hasEventType (E, 'Reserve')};
      c_event = {E | hasEventType (E, 'Cancel')};
 eventmodules
      CountEvents := {p_event, r_event, c_event} <- Counter -> {};
      FilterEvents := {p_event} <- Filtering -> {OrdinaryPurchase o_event,  FrequentPurchase f_event};
      ...
 constraints
      precede (CountEvents, FilterEvents);
}
```

Prolog is used to query events.

DSLs are used to provide the functionality of event modules

Specification of Event Modules (EventReactor 1.1)

```
eventmodule CountEvents{
        requires{ Purchase p_event; Reserve r_event; Cancel c_event;}
        provides{}
        reactor{
                if (Shopping.computeElapsedTime()< 20){
                        if (p_event) p_counter++;
                        else if (r_event) r_counter++;
                                else if (c_event) c_counter++;
                }
                else{
                         Shopping.log(p_counter, r_counter, c_counter);
                         Shopping.reset(p_counter, r_counter, c_counter);
                }
        }
        variables{ long p_counter, r_counter, c_counter;}
}
```

Event modules have event-based required and provided interfaces, which refer to event types.

The functionality to process required events and to publish events can be expressed in Java.

Specification of Event Modules (EventReactor 1.1)

```
eventmodule FilterEvents{
    requires{ Purchase p_event;}
    provides{ OrdinaryPurchase o_event; FrequentPurchase f_event;}
    reactor{
        frequency = Shopping.getPurchaseFrequency(p_event, 10);
        if (frequency > 30){
        f_event.inner = p_event;
        f_event.frequency = frequency;
        publish f_event;}
        else{
        o_event.inner = p_event; publish o_event;}
    }
    variables{ long frequency;}
}
```

Events may be processed by more than one event module.

Specification of Event Modules (EventReactor 1.1)

```
eventmodule ComputeMax{
        requires{ Purchase p_event;}
        provides{ MaxPurchase mp_event;}
        reactor{
                if (Shopping.computeElapsedTime()< 20){
                maxpurchase = Shopping.max(p_event.amount, maxpurchase);}
                else{ mp_event.max = maxpurchase; publish mp_event;
                }
        variables{ long maxpurchase;}
}
eventmodule LogInfo{
        requires{ MaxPurchase event;}
        provides{}
        reactor{ Shopping.log(event.max);}
}
```

Specification of Compositions (EventReactor 1.1)

```
composition {
  CountEvents ce;
  FilterEvents fe;
  ComputeMax cpmaxOrdinary;
  ComputeMax cpmaxFrequent;
  LogInfo li;

  bind (fe.o_event, cpmaxOrdinary.p_event);
  bind (fe.f_event, cpmaxFrequent.p_event);
  bind (cpmaxOrdinary.mp_event, li.event);
  bind (cpmaxFrequent.mp_event, li.event);

  precede (ce, fe);
}
```

To utilize event modules, they must be instantiated.

Multiple instances of an event module can be defined.

Explicit binding of event modules to each other is supported. Implicit binding based on event types is also supported.

Conclusions

- Event-based composition, in principle, can help to achieve loose coupling among modules.
  - However, to achieve an effective event-based composition, we require event-based modularization.
- Event Composition Model can be regarded as a base model for developing AO and/or event-processing languages:
  - Unlike current AO languages, EventReactor is open-ended with new (domain-specific) event types and events, as well as DSLs to express the functionality of event modules.
  - These facilitate representing domain-specific concerns in their DSL, without the need for designing an AO DSL from scratch.
  - Composition of event modules with each other is a means to compose the concerns that are implemented in different DSLs
- In the context of the HAEC (Highly Adaptive Energy-efficient Computing) project:
  - EventReactor is being applied to self-energy-adaptive software systems.
  - Event modules are adopted to model the architecture of self-energy-adaptive software systems.

References

Evolution of Composition Filters to Event Composition

Somayeh Malakuti and Mehmet Aksit
Software Engineering group, University of Twente, 7500 AE Enschede, the Netherlands
{s.malakuti,m.aksit}@ewi.utwente.nl

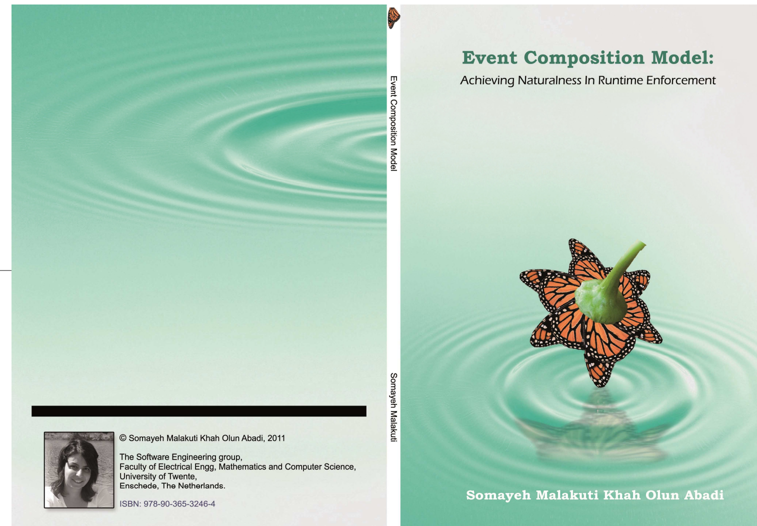
ABSTRACT

Various different aspect-oriented (AO) languages are introduced in the literature, and naturally are evolved due to the research activities and the experiences gained in applying them to various domains. Achieving *modularity*, *composability* and *abstractness* in the implementation of crosscutting concerns are typical requirements that these languages aim to fulfill; … fers per langu… what are the … perspective of … and/or new la… the limitations. This paper elaborates on the limitations of the current AO languages by means of runtime enforce-

1. INTRODUCTION

Various different aspect-oriented (AO) languages are introduced in the literature [7, 1, 10, 3, 12, 2], and naturally are evolved due to the research activities and the experiences gained in applying … ing *modularity*, *composa*… plementation of crosscut… …hat AO language… …e languages offe… …so that the mod… …rators to compos… …am. Some AO l… aspects at the higher lev… rating unnecessary imple…

@ ACM SAC 2012



Event-Based Modularization of Reactive Systems

Somayeh Malakuti and Mehmet Aksit

Software Engineering Group
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente, PO Box 217, 7500 AE Enschede, The Netherlands
{s.malakuti, m.aksit}@ewi.utwente.nl

Abstract. There is a large number of complex software systems that have reactive behavior. As for any other software system, reactive systems are subject to evolution demands. This paper defines a set requirements that must be fulfilled so that reuse of reactive software systems can be increased. Detailed analysis of a set of representative languages reveals that these requirements are not completely … and as such reuse of reactive sys… …the Event Composition Model and …e, which fulfill the requirements. …EventReactor language in creating reusable reactive systems is illustrated.

To appear in COB '13

Event Modules

Modularizing Domain-Specific Crosscutting RV Concerns

Somayeh Malakuti[1] and Mehmet Aksit[2]

[1] Software Technology group, Technical University of Dresden, Germany
somayeh.malakuti@tu-dresden.de
[2] Software Engineering group, University of Twente, the Netherlands
m.aksit@utwente.nl

Abstract. Runtime verification (RV) facilitates detecting the failures of software during its execution. Due to the complexity of RV techniques, there is an increasing interest in achieving abstractness, modularity and compose-ability in their implementations by means of dedicated linguistic mechanisms. This paper defines a design space to evaluate the existing …ues, and identifies …ages with respect …w languages, this paper advocates the need for a language composition framework for the

To appear in TAOSD '13