

The Israeli Workshop on Programming Languages & Development Environments, Haifa, July 1, 2002

Towards Aspect Architectures and Remodularization

Mika Katara*

Department of Computer Science
The Technion

`katara@cs.technion.ac.il`

Shmuel Katz

Department of Computer Science
The Technion

`katz@cs.technion.ac.il`

*On leave from Institute of Software Systems, Tampere University of Technology, Finland

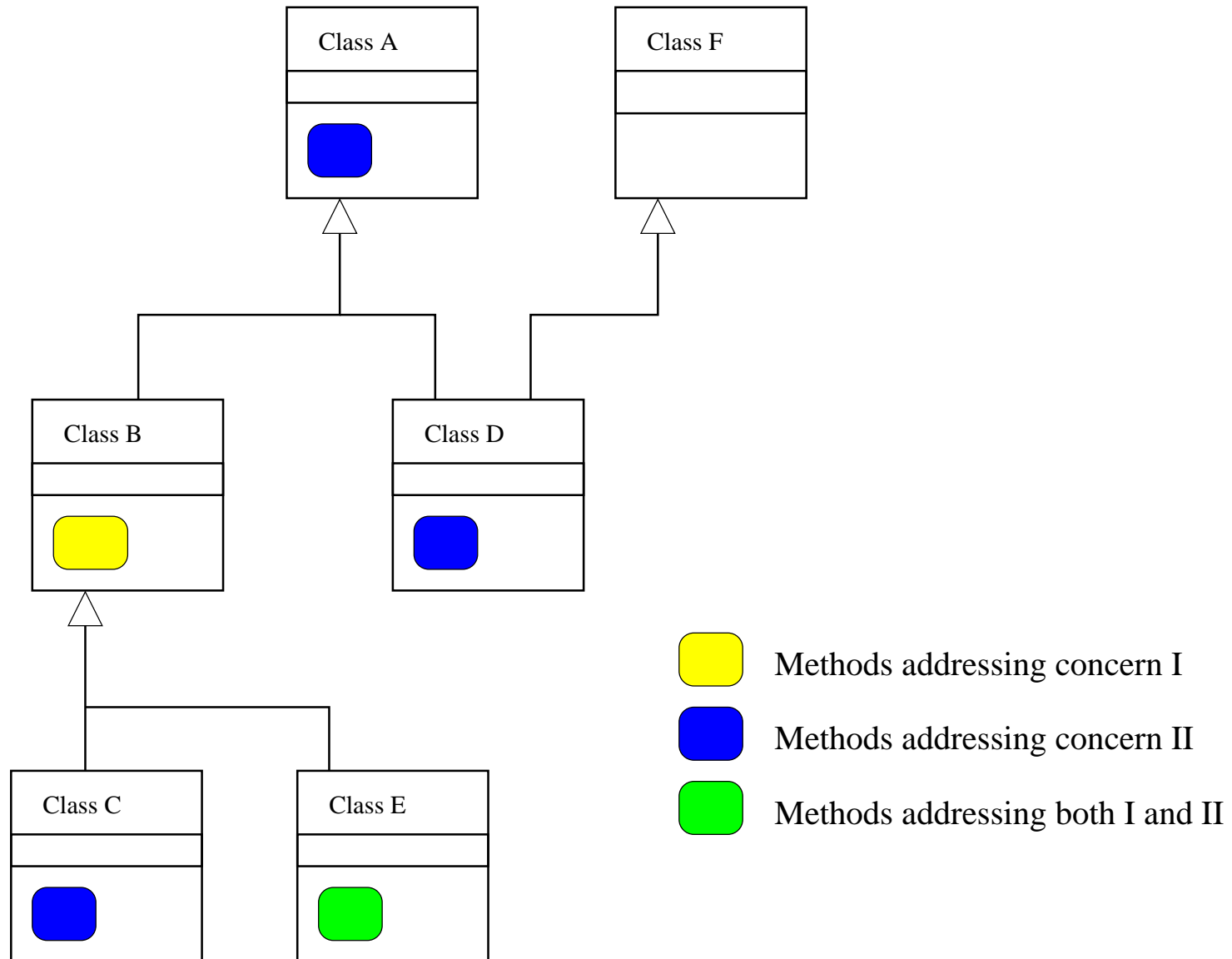
Contents

1. Separation of Concerns
2. Aspect-orientation
3. Aspect architectures
4. Remodularization
5. Conclusions

Separation of Concerns

- Separation of concerns means that a large and complicated system is divided into smaller units so that **concerns** (matters of interest) of importance are localized.
- Examples of such units include classes, components and processes.
- Examples of concerns: Features experienced by users, performance, security, fault-tolerance, treatment of overflow, load balancing, user interface, platform independency etc.
- Concerns that cut across the units, like many emerging from the requirements, are **scattered** around the units and **tangled** with each other inside the units.
- Addition of new features entails **invasive changes** to many units in existing design and code.

Example of scattering and tangling



Aspect-orientation

- Recently many notations and languages have been introduced to alleviate these problems.
- Usually called advanced separation of concerns or aspect-orientation.
- An aspect is a software artifact addressing a cross-cutting concern, which would otherwise be scattered and tangled, in a modular way.

Aspect-orientation

- Recently many notations and languages have been introduced to alleviate these problems.
- Usually called advanced separation of concerns or aspect-orientation.
- An aspect is a software artifact addressing a cross-cutting concern, which would otherwise be scattered and tangled, in a modular way.
- The basic desire I: Each module contains **everything** belonging to the corresponding concern & the module contains **nothing** else.

Aspect-orientation

- Recently many notations and languages have been introduced to alleviate these problems.
- Usually called advanced separation of concerns or aspect-orientation.
- An aspect is a software artifact addressing a cross-cutting concern, which would otherwise be scattered and tangled, in a modular way.
- The basic desire I: Each module contains **everything** belonging to the corresponding concern & the module contains **nothing** else.
- The basic desire II: **Additive** instead of invasive changes to existing artifacts.

AOP & AspectJ

- G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin, *Aspect-Oriented Programming*, *Proc. ECOOP'97*.
- AspectJ is an extension of the Java language and the set of associated open source tools, developed at PARC, supporting encapsulation of cross-cutting concerns.
- Defines a join point model, which enables adding new aspects to an existing Java program without touching the original source. However, recompilation is needed.
- Ready for production use, already used in many companies.
- <http://aspectj.org>

AspectJ example (from AspectJ tutorial)

Standard Java code:

```
class Line {
    private Point p1, p2;

    Point getP1(){
        return p1;
    }
    Point getP2(){
        return p2;
    }

    void setP1(Point p1){
        this.p1 = p1;
    }
    void setP2(Point p2){
        this.p2 = p2;
    }
}
```

AspectJ example (from AspectJ tutorial)

Standard Java code:

```
class Line {
    private Point p1, p2;

    Point getP1(){
        return p1;
    }
    Point getP2(){
        return p2;
    }

    void setP1(Point p1){
        this.p1 = p1;
    }
    void setP2(Point p2){
        this.p2 = p2;
    }
}
```

AspectJ code:

```
aspect DisplayUpdating {

    pointcut move():
        call(void Line.setP1(Point)) ||
        call(void Line.setP2(Point));

    after(): move() {
        Display.update();
    }
}
```

Hyperspaces: Multi-dimensional separation of concerns

- P. Tarr, H. Ossher, W. Harrison, S.M. Sutton, Jr, **N Degrees of Separation: Multi-Dimensional Separation of Concerns**, *Proc. ICSE'99*.
- Point of view: Aspect-orientation is more than just an extension of the object-oriented paradigm.
- Hyperspaces: A notation and language independent model of software artifacts, decomposition, and composition to overcome the above mentioned problems.
- Capture each cross-cutting concern in a separate module called a hyperslice (corresponds to an aspect in AspectJ).
- The artifact described by a slice need not be well-defined, or complete, in the notation used.

- Slices need not respect any module division dominant in the notation, object for instance.
- The slices can be partly overlapping by describing the same units from different perspectives or by using different units to describe the same concepts.
- The slices can be grouped to hypermodules which define how the incorporated slices should be composed to form a well-defined, complete artifacts. Hypermodules produce valid slices, so they can be nested.
- <http://www.research.ibm.com/hyperspace/>

Aspect architectures

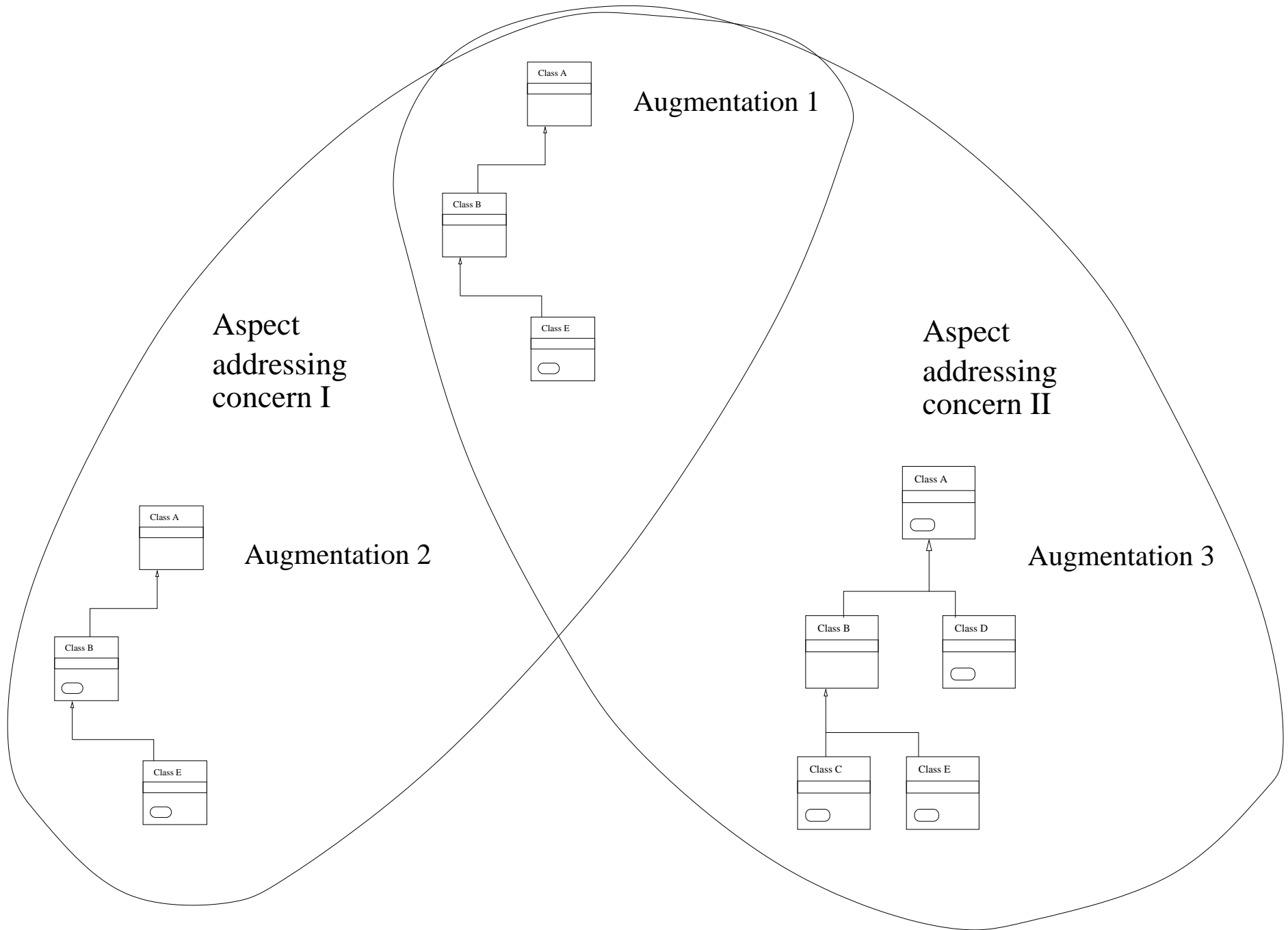
Some observations:

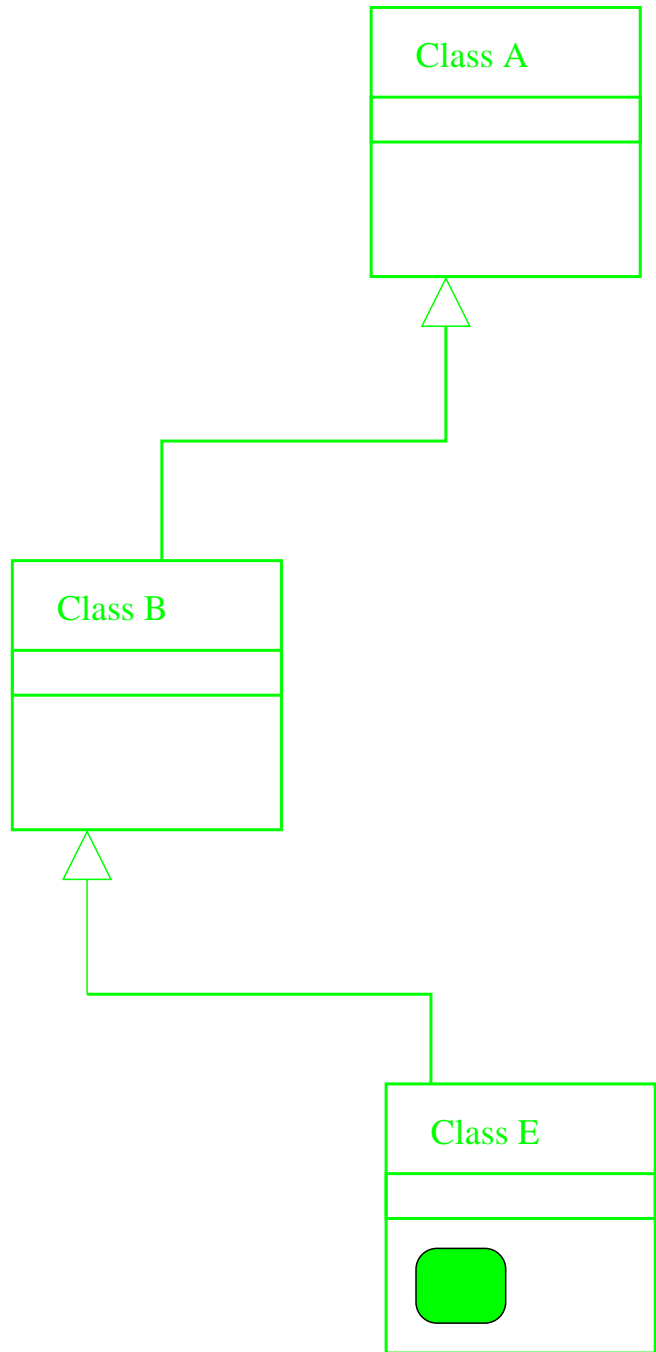
1. Cross-cutting concerns should be modularized already in the design phase.
2. There is no way to divide a system into disjoint modules so that a total separation of concerns would be satisfied, i.e. it is always possible to find a concern that cuts across a given module division.
3. Current aspect-oriented approaches define the overlapping parts of different aspects only implicitly.
 - What are the effects on one aspect caused by the changes in an overlapping aspect?
 - Overlapping parts of the aspects can contain some sub-aspects, interesting in their own right.
 - No support is provided for incremental design of the aspects themselves.

4. Superimposition is a well known set of formal techniques for separation of concerns in distributed systems.
- S. Katz, J. Gil, *Aspects and Superimpositions*, 3rd AOP Workshop in ECOOP'99.
 - A superimposition step maps an existing design into an augmented one.
 - A step can cross-cut any module division of the original design, i.e. aspect-orientation is supported.
 - Classical steps include termination and deadlock detection.
 - Three main types of superimposition, based on what restrictions are placed on manipulating the original design.
 - M. Sihman and S. Katz, *A Calculus of Superimpositions for Distributed Systems*, Proc. 1st International Conference on Aspect-Oriented Software Development, 2002.
 - New steps can be created by composing existing steps using sequential and parallel composition.

Architectures...

- Important concerns rarely cut across **augmentations to existing designs**.
- The augmentations should be lifted to first class entities.
- The augmentations (\approx simple superimposition steps) can be seen as Lego bricks. Each brick **uses** some parts of the original design and **defines** the additional design elements using those parts.
- The augmentations are generic by default.
- Each aspect in an architecture consists of a collection of augmentations which may have been created from more basic augmentations.
- Overlapping parts of different aspects correspond to the augmentations common to them.

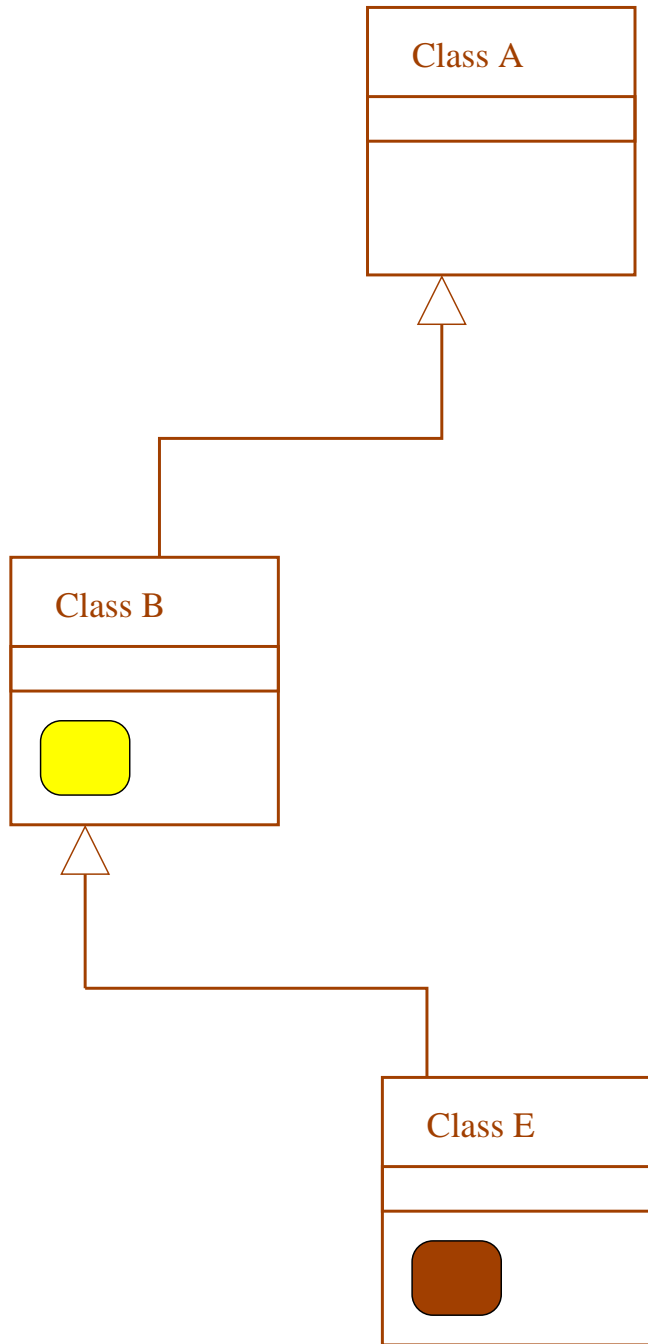






Augmentation 1

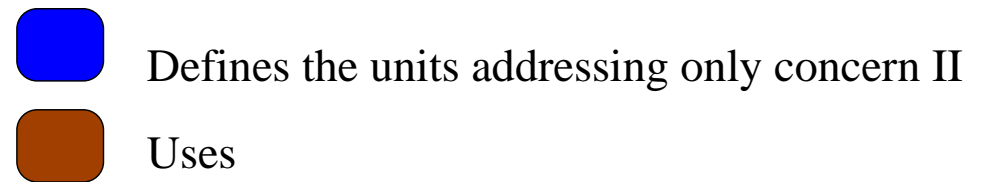
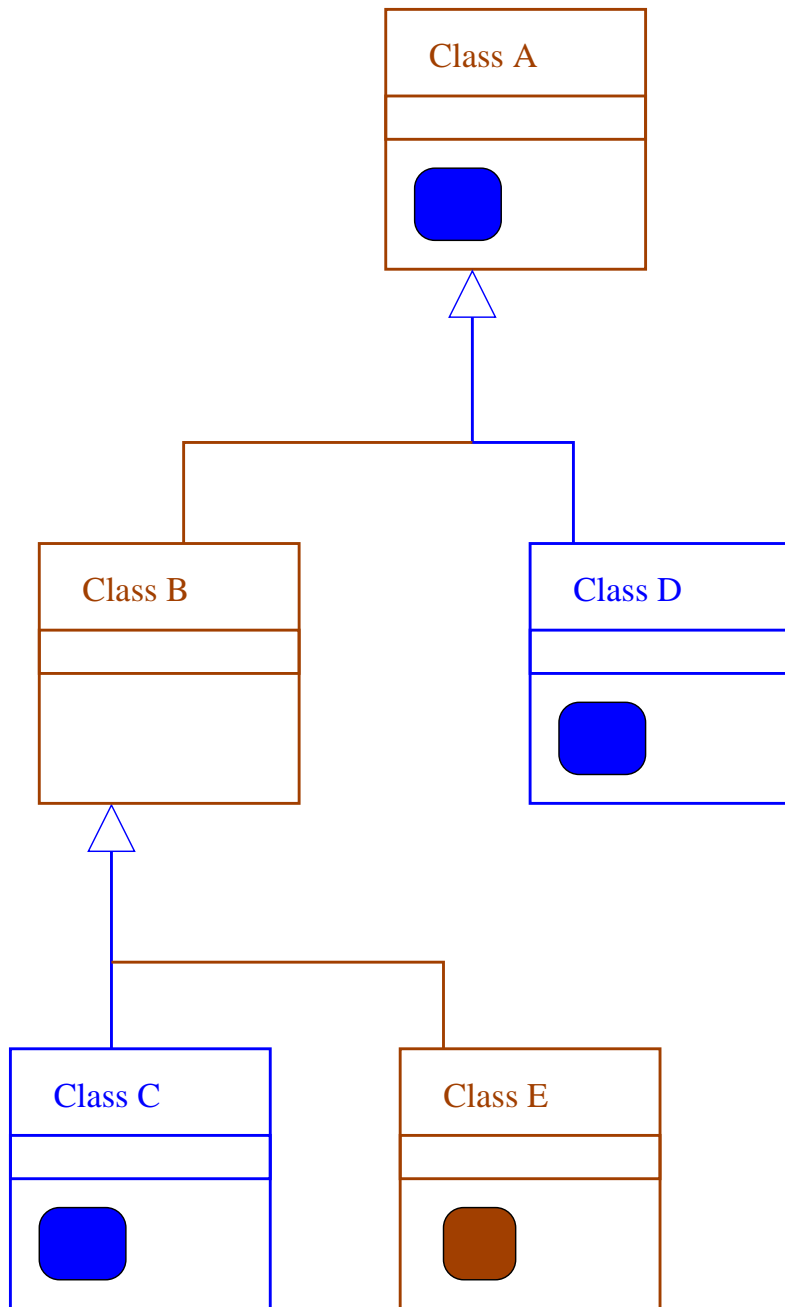
 Units addressing both concerns I and II

Augmentation 2



-  Defines the units addressing only concern I
-  Uses

Augmentation 3



Composing augmentations

- The augmentations can be composed using sequential and parallel composition.

Composing augmentations

- The augmentations can be composed using sequential and parallel composition.
- Sequential: One augmentation is applied before another so that the **uses** part of the latter can be satisfied, perhaps only partly, by the **uses** and **defines** parts of the former.

Composing augmentations

- The augmentations can be composed using sequential and parallel composition.
- Sequential: One augmentation is applied before another so that the **uses** part of the latter can be satisfied, perhaps only partly, by the **uses** and **defines** parts of the former.
- Parallel: The augmentations are applied independently, meaning that the **uses** and **defines** parts of the resulting augmentation are the union of the **uses** and **defines** parts of the components, respectively.

Architectures...

- Aspect architecture makes the relationships between different aspects explicit.
- The architecture also reveals the sub-aspects, which can be good candidates for reuse.
- Like the Hyperspaces model, this model needs to be instantiated for particular artifact development formalism.

How to obtain an aspect architecture?

- Some additional work is needed to create such an aspect architecture.
- Defining the common augmentations of different aspects makes composition of the aspects more automatic.
- Otherwise complex reconciliation is probably needed.
- Investing in the aspect architecture should pay off during the **maintenance phase**.

Remodularization

- The set of relevant concerns is context-sensitive and changes in the course of the software life cycle.
- It should be possible to remodularize software.
- In the architecture a concern matches a collection of augmentations which can cut across any other collection.
- In principle, it is possible to decompose an existing system into augmentations each of which would define exactly one detail.
- These augmentations would then correspond to the Lego bricks that one could compose using sequential and parallel composition to build a complete system.
- Different concerns would be addresses by different collections of the bricks.

Conclusions

- Architectural support needed at the design level to make relationships between different aspects explicit.
- Aspects can be described as collections of augmentations to existing designs.
- Extra work needed to create the augmentations is expected to pay off when maintaining the software.
- An aspect architecture can be remodularized to match changing concerns.
- Current work: Instantiation of the aspect architecture for all diagram types defined by UML.
- Further information:
 - ★ Aspect architectures: `katara@cs.technion.ac.il`
 - ★ Aspect-orientation: <http://aosd.net>