Concepts for Describing Composition of Software Artifacts

William Harrison and Harold Ossher
IBM T. J. Watson Research Center

The CME Team
(IBM Hursley Park and IBM Watson)
William Chung, Andrew Clement, Matthew Chapman,
William Harrison, Helen Hawkins, Sian January, Vincent Kruskal, Harold Ossher, Stanley Sutton, Peri Tarr

This research was supported in part by the Defense Advanced Research Projects Agency under grant NBCHC020056
Copyright © IBM Corporation, 2004, 2005
Simple Composition Example

basic

```cpp
class Sys
    int interval;
    void init();

    class RoomSensor
        void report();
        void update(int);

    class AtticSensor
        void report();
        void update(float);

    ... more sensors ...
```

alarm

```cpp
class SensorAddition
    void update(int);
    void update(float);

composed result 1

```cpp
class RoomSensor
    void u_b(int) { /* basic*/}
    void u_a(int) { /* alarm*/
        void update(int i) {
            u_b(i); u_a(i); 
        }

    ... }
```
Simple Composition Example

basic

class Sys
  int interval;
  void init();

class RoomSensor
  void report();
  void update(int);

class AtticSensor
  void report();
  void update(float);
  ...
  more sensors ...

alarm

class SensorAddition
  void update(int);
  void update(float);

composed result 2

class RoomSensor
  alarm a;
  void u_b(int) { /* basic*/}
  void update(int i) {
    u_b(i); a.update(i); }
  ...

Levels of Composition Specification

User Level
- merge basic, alarm as C

CCC Level
- merge order(1, 2) facet:
  - space basic, alarm as C
  - encapsulating(member)
  - exposed
  - exclusively precedence(1)

Assembly Level
- `<type name="Sys" attributes="public"/>
  `<method within="C:Sys" name="init" types="()">
    `<from within="basic:Sys" name="init" types="()"/>
  `<field within="C:Sys" name="interval" type="int">
    `<from within="basic:Sys" name="interval" type="int"/>
  `"/>
Material to be Composed: CIT

Standard interfaces for accessing different kinds of artifacts

- **Entities**
  - Modifiers, Classifiers
  - Attributes

- **Type spaces**
  - Types
    - Fields
    - Methods

- **Container spaces**
  - Containers
    - Elements

Spaces contain unique definitions of names used within them
E.g., Java classpath, collection of UML model files

Copyright © IBM Corporation, 2004, 2005
- Use patterns to define material inside element bodies, treating the matching material as extractable methods

```
class C {
    int x;
    void foo() {
        ...x = 3;
        ...x = y+7;
    }
}
```

```
methoid setX:
   kind = “set”
   field = “x”
```

```
class C {
    int x;
    void foo() {
        ...setX(3);
        ...setX(y+7);
    }
    void setX(int x) {
        this.x = x;
    }
}
```
Methoids

- Open-ended characterizations
- Start with useful language constructs, guided by existing approaches (e.g., AspectJ join points)
  - get/set of specific instance variables
  - method calls, entries and exits
  - synchronization block entries and exits
  - throws and catches of specific exception types
  - downcasting and instanceof
- Can specify arguments, set to local state
  - Perhaps specially-constructed (e.g., thisJoinPoint)
- Various inlining options and, perhaps, restrictions
CCC Weaving Directives

• What elements are to be joined?
  – Correspondence

• How are they to be joined
  – Selection
  – Ordering
  – Structure

• Making assumptions explicit
  – Encapsulation, Opacity

• Resolving multiple weaving directives
  – Exclusivity, Precedence

merge order(1, 2) facet:
  space basic, alarm as C
  encapsulating(member)
  exposed
  exclusively precedence(1)
Identifying Correspondences

• Explicit: queries
  – (class basic:*Sensor, alarm:SensorAddition)
    \{ (RoomSensor, SensorAddition), (AtticSensor, SensorAddition) \}
  – (method basic:*Sensor.update(<type>),
    alarm:SensorAddition.update(<type>))
    \{ (RoomSensor.update(int), SensorAddition.update(int)),
      (AtticSensor.update(float), SensorAddition.update(float)) \}

• Implicit (depending on encapsulation)
  – Like-named types within corresponding spaces
  – Like-named members within corresponding types
Selection

• *Which* inputs in the correspondence should participate in the result:
  – merge
  – override
  – overridemember
  – aroundmethod
  – any
  – unique
  – … (this is an open-ended list)
Ordering

- For *override/around*: which input dominates
- For *merge* of methods: order of execution
  - Generalized as *method combination graphs*

(method basic:*Sensor.update(<type>),
after:: alarm:SensorAddition.update(<type>))
Structure

... more – another open-ended list
Opacity

- Is the type hierarchy structure assumed to be known and taken into account?

\[ a + b \Rightarrow ab \quad \text{ab} \text{ the space is } \text{opaque} \]

\[ a \quad \rightarrow \quad b \]

\[ a + b \Rightarrow ? \quad \text{b} \text{ the space is } \text{exposed} \]
Use to Realize Various Approaches

- AspectJ
- AHEAD
- ...

Copyright © IBM Corporation, 2004, 2005
Driving the Compositor

- Through user-level composition language
- To use directly:
  - “Plainway” language manifests the concepts directly
  - Note: not intended for most developers
- To experiment with a new composition approach:
  - Try compiling down to CCC weaving directives
  - Implement new selections, structures, etc., if needed
  - If CCC proves unsuitable
    - Build directly on the lower-level Concern Assembly interface
  - Saves a lot of detailed implementation work
    - New approach supported for multiple artefacts
Thank you!

http://www.eclipse.org/cme