De-constructing and Re-constructing Aspect-Orientation

Bill Harrison
Department of Computer Science
Trinity College, Dublin

Presentation at FOAL 2008. For more comprehensive material see
William Harrison, "De-constructing and Re-constructing Aspect-Orientation",
Proceedings of Seventh Annual Workshop on Foundations of Aspect

This work is supported by a grant from Science Foundation, Ireland

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
AOSD Community: Themes

Treatment of Concerns as Independent Artifacts
Patterned Identification of Publishable Events
Identification of Intent / “Higher-Order” State
Concern Mining / Extraction
AOSD Community: Themes

Treatment of Concerns as Independent Artifacts
- Each Containing State, Behaviour, and Flow for Classes
- Join Points – Cooperative Method Call / Events - Creation, Call, Response
- Dispatch / Routing / Orchestration of Joined Methods
- Design / Code

Patterned Identification of Publishable Events

Identification of Intent / “Higher-Order” State

Concern Mining / Extraction

=> Cooperative Method Call
- Events / Flow of Events
- Intention
- Generalized Dispatch

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
AOSD Community: Themes

Treatment of Concerns as Independent Artifacts
• Each Containing State and Behaviour for Classes
• Join Points – Cooperative Method Call / Events - Creation, Call, Response
• Dispatch / Routing / Orchestration of Joined Methods
• Design / Code

Patterned Identification of Publishable Events
• Expected Joinpoints : Method Calls
• Injected Joinpoints : Pointcuts – Obliviousness / Asymmetry

Identification of Intent / “Higher-Order” State
Concern Mining / Extraction
AOSD Community: Themes

Treatment of Concerns as Independent Artifacts
- Each Containing State and Behaviour for Classes
- Join Points – Cooperative Method Call / Events - Creation, Call, Response
- Dispatch / Routing / Orchestration of Joined Methods
- Design / Code

Patterned Identification of Publishable Events
- Expected Joinpoints : Method Calls
- Injected Joinpoints : Pointcuts – Obliviousness / Asymmetry
- Joinpoint Shadows, Methoids
- Query Formulations, Exported Pointcuts – Expected Joinpoints

Identification of Intent / “Higher-Order” State
Concern Mining / Extraction

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
AOSD Community: Themes

Treatment of Concerns as Independent Artifacts
- Each Containing State and Behaviour for Classes
- Join Points – Cooperative Method Call / Events - Creation, Call, Response
- Dispatch / Routing / Orchestration of Joined Methods
- Design / Code

Patterned Identification of Publishable Events
- Method Calls: Expected Joinpoints
- Pointcuts: Injected Joinpoints – Obliviousness / Asymmetry
- Query Formulations, Exported Pointcuts – Expected Joinpoints
- Joinpoint Shadows, Methoids

Identification of Intent / “Higher-Order” State
- Cflow
- Complex-event processing

Concern Mining / Extraction

=> Events / Concurrency
- Events / Flow of Events

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
AOSD Community: Themes

Treatment of Concerns as Independent Artifacts
- Each Containing State and Behaviour for Classes
- Join Points – Cooperative Method Call / Events - Creation, Call, Response
- Dispatch / Routing / Orchestration of Joined Methods
- Design / Code

Patterned Identification of Publishable Events
- Method Calls: Expected Joinpoints
- Pointcuts: Injected Joinpoints – Obliviousness / Asymmetry
- Query Formulations, Exported Pointcuts – Expected Joinpoints
- Joinpoint Shadows, Methoids, Generalized Artifacts

Identification of Intent / “Higher-Order” State
- Cflow
- Complex-event processing

Concern Mining / Extraction
- Program Slicing
- Extraction – Complex region methoids/shadows
- Design / Code / Generalized Artifacts

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
AOSD Community: Growth & Extension Issues

Dispatching Flexibility
- Not to the target
- Controlled by external rules
- Outside the Language
- Methods & Events

Other Communities
- Grid Computing
- Ubiquitous / Mobile / Autonomic Computing
- Service-Oriented Computing
- Complex-Event Processing
- Multi-core chip suppliers

Continuum
- Base
  - is the flow of events
  - can include flow constraints / model

Aspects
- export and support cooperative method calls and events – self or other responses
- provide plug-in Services model

Intentions and Glossaries
- adding Malleability to service plug-ins

Events & Concurrency
- extend Services model
- extend Malleability characteristics

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
Environment

Communities Needing Flexible Routing
• Grid Computing
• Ubiquitous / Mobile / Autonomic Computing
• Service-Oriented Computing
• Complex-Event Processing
• Multi-core realisations

Service-Oriented: Real Black Boxes

Servicing implementations (classes) not chosen until execution time
• All selection criteria must be manifest (explicit at use-time), not latent (examined at development-time)
• Precise functions performed
• Side-effects
• Dependencies

Emphasis on Concurrency
• Variable Latency - local / remote / delayed
• Physical limits on sequentially over-constrained behaviour
• Simplified expression of concurrent behavior
“Base” Skeletal Flow of Events

Base
• is just the flow of events
  • can include flow constraints / model
    • “must occur before”
    • “must be seen before”
    • “must be followed by”
    • etc.
  • events derived from *community* of aspects “plugged-in”
    • directly (as cooperative method calls)
    • as exported pointcuts
    • or derived as part of a system architecture

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
“Base” Skeletal Flow of Events

Base
• is just the flow of events
  • can include flow constraints / model
    • “must occur before”
    • “must be seen before”
    • “must be followed by”
    • etc.
  • events derived from community of aspects “plugged-in”
    • directly (as cooperative method calls)
    • as exported pointcuts
    • or derived as part of a system architecture

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
Aspect /Service Integration

Aspects / Services
- exports pointcuts to support cooperative behaviour (method calls and events)
  - co-operators can be self or other responses
  - synchronous cooperative call or asynchronous (independent) events
- provide plug-in Services model
  - "first-class" linguistic element
  - stores "persistent" state for objects
  - provides reference decapsulation for methods to access object state
  - bounds definition of ambiguity
  - plugs into base (community)
    - dynamically
    - if consistent with base event model
    - when needed

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
Annotation for Intention / Malleability

Annotation
• supports run-time “service-finding” rather than development-time selection
  • must say much more about required, expected, and supplied behaviour
• adds greater flexibility to matching services with clients
  • multiple requirements satisfied by composite services
• documents what functions methods do or must do
  • replaces / supplements method name <reduces point of rigidly>
  • allows satisfaction by service composition
  • documents real intention of method use <pointcut annotation>

Glossary
• meets minimum requirements
• can be subject to direct matching
• glossaries are simplest of knowledge organizations, supertype of ontologies, etc.

Convenience
• applied in method definitions, not use – binds function to local name

```
"interface First {void one(int y) does(HardWork);}  
((First) thing).one(6);"
```

Areas of application
• method name / function definition
• pointcut definition
• parameter matching / ordering
```
"void one(Object x for control, int y for size) does(HardWork);"
```
Events and Concurrency

Stage 1 – Attachment of aspect behaviour in advices as events

• “event() p(int a) {...};”
  • advice produces “fork without join”

Stage 2 – Explicit sending of events

• “send methodName(this, “hello”);”
Events and Concurrency

Stage 1 – Attachment of aspect behaviour in advices as events
- “event() p(int a) {...};”
- advice produces “fork without join”

Stage 2 – Explicit sending of events
- “send methodName(this, “hello”);”

Stage 3 – Events with future commitments
- “void one(Object x, int y) sends two(Object x, real z)”
- commits that if “one” is called, eventually “two” will be sent with same “x”
  - could be sent by “one”
  - could be sent by some event sent by “one”
  - assured by static type-checking
- needs the “services” model
  - hold the implementations for “two”
  - deal with failures

encourage use with convenience:
- “send one(this, 6) expect two(MyClass this, int a) {...};”
- avoids scattering and preserves local logic continuity

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
Events and Concurrency

Stage 1 – Attachment of aspect behaviour in advices as events

- “event() p(int a) {...};”
- advice produces “fork without join”

Stage 2 – Explicit sending of events

- “send methodName(this, “hello”);”

Stage 3 – Events with future commitments

- “void one(Object x, int y) sends two(Object x, real z)”
- “send one(this, 6) expect two(MyClass this, int a) {...};”

Stage 4 – Errors in realising expectations

- failure to meet or assure future expectation results in failure message
- same name and arguments
- caught by catch blocks
- “event two(Object x, real z) {...} catch(Error e) {... x ...}”
Extended Malleability – Service-Orientation

Removed Some Barriers to Smooth Integration
  • naming
    • use function annotation
  • method bundling
    • use service composition
  • method grouping
    • structural typing for interfaces
  • parameter order
    • glossary (call-by-keyword)

Remaining Barriers:

*Clients Know what Interfaces a Class Supports*

*Clients Know Where Implementations Are Located*
  • dynamic expectations
    • local knowledge of class capabilities
  • static expectations
    • floating responsibility for assurance

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
Extended Malleability – Service-Orientation

Removed Some Barriers to Smooth Integration
  • naming
    • use function annotation
  • method bundling
    • use service composition
  • method grouping
    • structural typing
  • parameter order
    • glossary (call-by-keyword)

Remaining Barriers:
  Clients Know what Interfaces a Class Supports
  Clients Know Where Implementations Are Located
  • dynamic expectations
    • local knowledge of class capabilities

```java
void meth( Store{put(Store,Item}) store1, Store store2);

Store{put(Store,Item), boolean inStock(Item,Store}) more;

more = ({boolean inStock(Item,Store)}) store1;
more = store2;

boolean t = item.inStock(store2);
```

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
Extended Malleability – Service-Orientation

Unusual Characteristics

- Class (Store) and supported Interface \{put(Store,Item)\} can be asserted by client

```java
void meth( Store{put(Store,Item)} store1, Store store2);

Store{put(Store,Item), boolean inStock(Item,Store)} more;

more = ({boolean inStock(Item,Store)}) store1;
more = store2;

boolean t = item.inStock(store2);
```

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
Extended Malleability – Service-Orientation

Unusual Characteristics

• Class (Store) and supported Interface \{\text{put}(\text{Store}, \text{Item})\} can be asserted by client
• New facts \{\text{boolean inStock}(\text{Item}, \text{Store})\} about classes (Store) “proven” by successful downcasts (flow-dependent)

```java
void meth( Store{put(Store,Item)} store1, Store store2);

Store{put(Store,Item), boolean inStock(Item,Store)} more;

more = ({boolean inStock(Item,Store)}) store1;
more = store2;

boolean t = item.inStock(store2);
```

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
Extended Malleability – Service-Orientation

Unusual Characteristics

• Class (Store) and supported Interface \{\text{put(Store,Item)}\} can be asserted by client
• New facts \{\text{boolean inStock(Item,Store)}\} about classes (Store) “proven” by successful downcasts (flow-dependent)
• Facts about a class are true for all references to that class

```java
void meth( Store{put(Store,Item)} store1, Store store2);

Store{put(Store,Item), boolean inStock(Item,Store)} more;

more = ({boolean inStock(Item,Store)}) store1;
more = store2;

boolean t = item.inStock(store2);
```

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
Extended Malleability – Service-Orientation

Unusual Characteristics

• Class (Store) and supported Interface \{\text{put}(\text{Store}, \text{Item})\} can be asserted by client
• New facts \{\text{boolean inStock}(\text{Item}, \text{Store})\} about classes (Store) “proven” by successful downcasts (flow-dependent)
• Facts about a class are true for all references to that class
• Facts may include knowledge about other classes

```java
void meth( Store{put(Store,Item)} store1, Store store2);

Store{put(Store,Item), boolean inStock(Item,Store)} more;
more = ({boolean inStock(Item,Store)}) store1;
more = store2;

boolean t = item.inStock(store2);
```
Extended Malleability – Service-Orientation

Unusual Characteristics

• Class (Store) and supported Interface {put(Store, Item)} can be asserted by client
• New facts {boolean inStock(Item, Store)} about classes (Store) “proven” by successful downcasts (flow-dependent)
• Facts about a class are true for all references to that class
• Facts may include knowledge about other classes

• Facts can be transferred from one variable declaration to another (in the right flow circumstances)
• Classes are names without implied characteristics, but arranged in a type hierarchy
• Interfaces are structurally typed, their names are irrelevant

```java
void meth( Store{put(Store, Item)} store1, Store store2);

Store{put(Store, Item), boolean inStock(Item, Store)} more;

more = ({boolean inStock(Item, Store)}) store1;
more = store2;

boolean t = item.inStock(store2);
```

(c) Copyright 2008 William Harrison, Trinity College, Dublin.
Extended Malleability – Service-Orientation

moved Some Barriers to Smooth Integration

naming
• use function annotation
function bundling in methods
• use service composition
method grouping
• structural typing
parameter order
• glossary (call-by-keyword)

aining Barriers:
ents Know what Interfaces a Class Supports
nts Know Where Implementations Are Located
dynamic expectations
• local knowledge of class capabilities
Extended Malleability – Service-Orientation

Over Some Barriers to Smooth Integration

- use function annotation
- use service composition

Grouping

- structural typing
- parameter order
- glossary (call-by-keyword)

Removal Barriers:

- Events Know what Interfaces a Class Supports
- Events Know Where Implementations Are Located

- dynamic expectations
  - local knowledge of class capabilities
- static expectations
  - floating responsibility for assurance

```java
int meth( Store{put(Store,Item}) store1, Store store2);

put(Store,Item), boolean inStock(Item,Store}) more;

e = ({boolean inStock(Item,Store}) store1;
e = store2;
```
AOSD Community: Themes

Treatment of Concerns as Independent Artifacts
- Patterned Identification of Publishable Events
- Identification of Intent / “Higher-Order” State
- Concern Mining / Extraction

AOSD Community: Growth & Extension Issues

- Generative Method Call
  - Events / Flow of Events
  - Intention
- Generalized Dispatch
  - Exports + Supports
- / Concurrency
  - Events / Flow of Events

Other Communities
- Grid
- Ubiquitous / Mobile / Autonomic
- Service-Oriented
- Complex-Event Processing
- Multi-core chips

Dispatching Flexibility
- Not to the target
- Controlled by external rules
- Outside the Language
- Methods & Events

Topics
- Base as Event Flow
- Aspects with Exported Pointcuts
- Aspects as Service Providers
- Intention & Annotation
- Malleability
Thank you!