An Overview of Aspect and Component Models: Part 3

Hakim Hannousse

ECOLE DES MINES DE NANTES
DEPARTMENT OF COMPUTER SCIENCE

Outline

- Component Models
  - Kmelia Component Model [5 Papers]
  - Fractal Component Model [1 Paper + 1 Technical Report]
  - Kmelia and Fractal Models: Comparison
- Aspect Oriented Programming Issues
  - Formal Semantics for Aspects : CASB [1 Technical Report]
  - Aspect Classification [1 paper]
  - Aspect Interaction [2 paper]
  - Formalizing Concurrent Aspects [1 paper]
- Perspectives

Kmelia is a Formal Component Model [Attiog06]

$$Component \stackrel{def}{=} < \mathcal{W},\ Init,\ \mathcal{A},\ \mathcal{N},\ \mathcal{I},\ \mathcal{D}_s,\ v,\ \mathcal{C}_s >$$

- $\mathcal{W} \stackrel{def}{=} < T,\ V, V_T,\ Inv >$
- $\mathcal{D}_s \stackrel{def}{=} < I_s,\ \mathcal{B}_s >$
  - $I_s \stackrel{def}{=} < \sigma,\ P,\ Q,\ V_s,\ S_s >$
    - $S_s \stackrel{def}{=} < sub_s,\ cal_s,\ req_s,\ int_s >$
  - $\mathcal{B}_s \stackrel{def}{=} < S,\ L,\ \delta,\ \phi,\ S_0,\ S_F >$

$$Assembly \stackrel{def}{=} < \mathcal{C},\ links,\ subs >$$

A component composition is defined as a well-formed assembly which is encapsulated within a component.

Kmelia is a Hierarchical Component Model [Pascal06]

- Services in Kmelia are not simple operations
- Kmelia introduces the concept of Assemblies
- Kmelia proposes three hierarchy levels :
  - Links Hierarchy
  - Services interfaces Hierarchy
  - Component Composition is an encapsulation of an assembly

Kmelia defines Components Protocols [Pascal07a]

- A protocol is a pre-ordering of services calls that should be respected during the system execution.
- A protocol has a behavior
- A protocol in Kmelia is a specific service defined using vertical structuring operators
  - Sate annotation << >>
  - Transition annotation [[ ]]
- Protocol inconsistency detection can be made using pre/post conditions.

Kmelia introduces HBIDL to describe components and services [Pascal07b]

- HBIDL extends IDL by the specification of the behavior of services with their architectures
- HBIDL has many advantages:
  - provides detailed documentations of complex interaction services
  - supports compatibility levels
  - serves as an intermediate between CBSE and SBSE
- HBIDL has some adaptation problems such as:
  - Parameters vs Messages mismatch
  - Hierarchichal mismatch

Kmelia has a Formal Anlyser Toolbox: COSTO [Pascal07c]

- COSTO is a toolbox that supports the design and analysis of Kmelia's abstract component model
- COSTO is an eclipse plugin
- COSTO toolbox includes:
    - COSTO core module
    - Verification module
    - LOTOS Module
    - MEC Module
    - Export Module
- COSTO takle state explosion problem

Fractal Component Model (1) [Bruneton04, Bruneton06]

A Fractal Component is an entity that has two parts:

- Membrane
- Content

Fractal Model supports three kind of Components:

- Basic Components
- Primitive Components
- Composite Components

Fractal Supports two kinds of Components Binding:

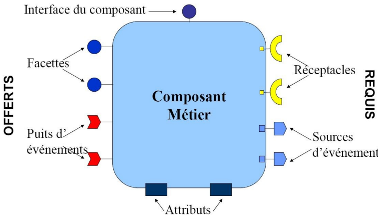
- Primitive binding
- Composite Binding

Fractal Component Model (2) [Bruneton04, Bruneton06]

Fractal Component Model has the following main features:

- Fractal is a hierarchical model
- Fractal supports sharing components
- Fractal is a reflective model
- Fractal has an implementation model named Julia

CORBA Component Model (CCM) [OMG'04, Marvie'06]



CCM = $\sum$ Component Types + $\sum$ Component Homes + $\sum$ Links

- Component Type = Name + Attributes + [Ports]
  - Attributes are used for configuration purpose
  - Ports = Facets $\cup$ Receptacles $\cup$ Event Sources $\cup$ Event Sinks
  - An inheritance relationship is defined between component types
- Component Types are two kinds : Basic and Extended
- Component Home is a meta-type that manages component instances

CCM: Facets [OMG'04, Marvie'06]

- A Facet defines a role that can be performed upon a component
- Each Facet has its own reference
- Several copies of a Facet reference may exist at a time.
- Facets can be specified only in a static way
- A client can navigate between Facets

CCM: Receptacles [OMG'04, Marvie'06]

- A receptacle allows a component type to accept a reference
- A receptacles maybe simples or multiples
- A cookie is created for each connection in the case of multiple receptacles
- Receptacles can be used for reconfiguration.

CCM: Events [OMG'04, Marvie'06]

- Events are used for asynchronous communications
- Event sources are two kinds: Emitters and Publishers
- An Event Sink may receive events from various sources at the same time

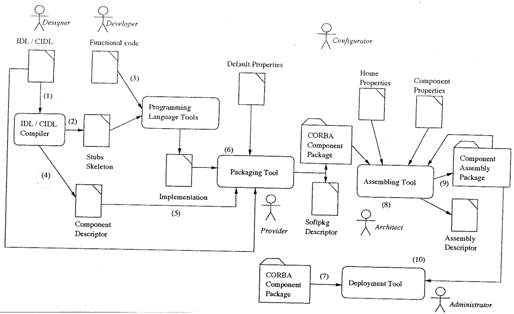
CCM: Component Homme [OMG'04, Marvie'06]

- A Component Home is a component manager that provides instantiation of component types at runtime
- Many home types may manage the same component type only with different instances
- Component Homes are two kinds: Keyless and with primary key Homes
- Component Homes are not components $\Rightarrow$ non hierarchical model

CCM: Components Configuration [OMG'04, Marvie'06]

- A Component configuration is implemented using configuration objects
- A Component home has Factory operations for component instances

CCM: Global Software Production Process [OMG'04, Marvie'06]



Kmelia and Fractal Component Models: A Comparison

- Kmelia is Service Based Model $\neq$ Fractal is a Component Based Model
- Kmelia follows monadic semantics $\neq$ Fractal follows demi-polyadic semantics
- Three hierarchy levels are allowed in Kmelia $\neq$ One hierarchy level for Fractal
- No sharing Components for Kmelia $\neq$ Sharing Components is allowed with Fractal
- reconfiguration is limited in Kmelia $\neq$ reconfiguration is more developed in Fractal

Formal Semantics for Aspects

CASB: Semantic Elements

- CASB introduces the concept of configurations $(C\ ,\ \Sigma)$
- A program C is of the form $C\ ::=\ i\ :\ C \mid \varepsilon$
- Semantic is described in term of binary relation $\rightarrow_b$
- A single reduction : $(i : C, \Sigma) \rightarrow_b (C', \Sigma')$
- An aspect is a function $\psi : I \rightarrow (\Sigma \rightarrow C) \times \{\mathit{before}, \mathit{after}, \mathit{around}\}$
- A tagged instruction : $\bar{i}$
- A matching function: $\mathit{match} : \mathcal{P} \times I \rightarrow \mathit{bool}$
- A weaving relation: $\rightarrow$

CASB: Weaving of a Single Aspect

- Before aspect

$$\frac{\psi(i) = (\phi, \mathit{before})}{(i : C, \Sigma) \rightarrow (\mathit{test}\ \phi : \bar{i} : C, \Sigma)}$$

- After aspect

$$\frac{\psi(i) = (\phi, \mathit{after})}{(i : C, \Sigma) \rightarrow (\bar{i} : \mathit{test}\ \phi : C, \Sigma)}$$

- Around aspect

$$\frac{\psi(i) = (\phi, \mathit{around})}{(i : C, \Sigma, P) \rightarrow (\mathit{test}\ \phi : \mathit{pop}_p : C, \Sigma, \bar{i} : P)}$$

$$(\mathit{pop}_p : C, \Sigma, i : P) \rightarrow (C, \Sigma, P)$$

$$(\mathit{proceed} : C, \Sigma, i : P) \rightarrow (i : \mathit{push}_p\ i : C, \Sigma, P)$$

CASB: Weaving of Several Aspects

- Aspects of the same Kind
  - Before aspects

$$\frac{\psi(i) = ((\phi_1 ... \phi_n), \mathit{before})}{(i : C, \Sigma) \rightarrow (\mathit{test}\ \phi_1 : ... : \mathit{test}\ \phi_n : \bar{i} : C, \Sigma)}$$

  - After aspects

$$\frac{\psi(i) = ((\phi_1 ... \phi_n), \mathit{after})}{(i : C, \Sigma) \rightarrow (\bar{i} : \mathit{test}\ \phi_1 : ... : \mathit{test}\ \phi_n : C, \Sigma)}$$

  - Around aspects

$$\frac{\psi(i) = ((\phi_1 ... \phi_n), \mathit{around})}{(i : C, \Sigma, P) \rightarrow (\mathit{test}\ \phi_1 : \mathit{pop}_p\ n : C, \Sigma, \mathit{test}\ \phi_2 : ... : \mathit{test}\ \phi_n : \bar{i} : C, \Sigma)}$$

$$(\mathit{pop}_p\ n : C, \Sigma, x_1 : ... : x_n : P) \rightarrow (C, \Sigma, P)$$

$$(\mathit{proceed} : C, \Sigma, x : P) \rightarrow (x : \mathit{push}_p\ x : C, \Sigma, P)$$

- Aspects of Different Kinds

$$\frac{\psi(i)=((\phi_1,t_1)...(\phi_n,t_n)) \quad \gamma((\phi_1,t_1)...(\phi_n,t_n))=((\phi'_1,\mathit{around})...(\phi'_n,\mathit{around}))}{(i{:}C,\Sigma,P)\rightarrow(\mathit{test}\ \phi'_1{:}\mathit{pop}_p\ n{:}C,\Sigma,\mathit{test}\ \phi'_2{:}...{:}\mathit{test}\ \phi'_n{:}\bar{i}{:}P)}$$

CASB: Pointcuts

$$\mathcal{P} ::= T_i \mid \mathcal{P}_1 \wedge \mathcal{P}_2 \mid \mathcal{P}_1 \vee \mathcal{P}_2 \mid \neg\mathcal{P}$$

$$\begin{array}{lcl}
match(T_i, i) & = & \text{true if } \exists\sigma : \sigma(T_i) = i \\
 & = & \text{false otherwise} \\
match(P_1 \wedge P_2, i) & = & match(P_1, i) \wedge match(P_2, i) \\
match(P_1 \vee P_2, i) & = & match(P_1, i) \vee match(P_2, i) \\
match(\neg P, i) & = & \neg match(P, i)
\end{array}$$

CASB: Exception Handling

- Exception Syntax :

$$S \ ::= \ try\ S_1\ catch\ ex\ S_2 \mid throw\ ex \mid \ldots$$

- Semantics :

$$(try\ S_1\ catch\ ex\ S_2 : C, \Sigma, E) \rightarrow_b (S_1 : pop_e : C, \Sigma, (ex, S_2 : C) : E)$$

$$(throw\ ex : C, \Sigma, (ex_0, C_0) : \ldots : (ex_k, C_k) : (ex, C') : E) \rightarrow_b (C', \Sigma, E)$$

$$(pop_e : C, \Sigma, X : E) \rightarrow_b (C, \Sigma, E)$$

CASB: Advanced Aspect Features

- Aspect Deployment

$$(\mathit{deploy}\ \mathit{id}\ S : C, \Sigma, \Psi) \rightarrow (S : \mathit{pop}_{\Psi} : C, \Sigma, \psi_{\mathit{id}} : \Psi)$$

$$(\mathit{pop}_{\Psi} : C, \Sigma, \psi_i : \Psi) \rightarrow (C, \Sigma, \Psi)$$

- Aspect Instantiation

$$\frac{\mathit{update}(\Psi, i, \Sigma) = (\Psi', \Sigma') \quad (\circ\Psi')(i) = (\phi, \mathit{before})}{(i, C, \Sigma, \Psi) \rightarrow (\mathit{test}\ \phi : \bar{i} : C, \Sigma', \Psi')}$$

Aspects Classification

- Organize aspects into categories sharing some properties
- $--$ $>$ Specify the preserved properties by the aspects of each category
- $--$ $>$ Optimization of the verification time

Observers Category

- Definition :

$$\forall (C,\Sigma).\Sigma^{\psi} \in \mathcal{A}_a \Leftrightarrow proj_b(\alpha) = proj_b(\tilde{\alpha}) \wedge preserve_b(\tilde{\alpha})$$

- Preserved Properties :

$$\begin{array}{lll}
\varphi^o & ::= & sp \mid \neg sp \mid \varphi_1^o \wedge \varphi_2^o \mid \varphi_1^o \vee \varphi_2^o \\
 & & \mid \varphi_1^o \cup \varphi_2^o \mid \varphi_1^o W \varphi_2^o \mid true \cup \varphi'^o \\
\varphi'^o & ::= & ep \mid \neg ep \mid sp \mid \neg sp \mid \varphi_1'^o \wedge \varphi_2'^o \mid \varphi_1'^o \vee \varphi_2'^o \\
 & & \mid \varphi_1'^o \cup \varphi_2'^o \mid \varphi_1'^o W \varphi_2'^o \mid true \cup \varphi'^o
\end{array}$$

Aborters Category

- Definition : $\forall(C,\Sigma).\Sigma^{\psi} \in \mathcal{A}_o \Leftrightarrow (\mathit{proj}_b(\alpha) = \mathit{proj}_b(\tilde{\alpha}) \vee \exists(i \geq 0), \exists(j \geq i).\mathit{proj}_b(\alpha_{\rightarrow i}) = \mathit{proj}_b(\tilde{\alpha}_{\rightarrow j}) \wedge \forall(k > j).\tilde{\alpha}_k = (\epsilon, \_)) \wedge \mathit{preserve}_b(\tilde{\alpha})$
- Preserved Properties :

$$\begin{array}{lcl} \varphi^a & ::= & sp \mid \neg sp \mid \varphi_1^a \wedge \varphi_2^a \mid \varphi_1^a \vee \varphi_2^a \mid \varphi_1^a W \varphi_2^a \mid \mathit{true} \cup \varphi'^a \\ \varphi'^a & ::= & \neg ep \mid \varphi_1'^a \wedge \varphi_2'^a \mid \varphi_1'^a \vee \varphi_2'^a \mid \mathit{true} \cup \varphi'^o \end{array}$$

Aspect Interactions: Detection and Resolution (EAOP)[Rémi'02]

- An aspect in EAOP is :

$$\begin{aligned} A \quad ::= \quad & \mu a.A \\ & \mid C \triangleright I;\ A \\ & \mid C \triangleright I;\ a \\ & \mid A_1 \,\Box\, A_2 \end{aligned}$$

- Crosscuts and Inserts :

$$C \ ::= \ T \mid C_1 \wedge C_2 \mid C_1 \vee C_2 \mid \neg C$$

- A term T is :

$$T \ ::= \ f\ T_1 \ ...\ T_n \mid x$$

- Aspect Composition :

$$(\mu a.C_1 \triangleright I_1;\ a) \,||\, (\mu a.C_2 \triangleright I_2;\ a)$$

Aspect Weaving (1)

$$
\begin{array}{lcll}
\textbf{sel}\, j\, (\mu a.A) & = & \textbf{sel}\, j\, A & \\
\textbf{sel}\, j\, (C \triangleright I;\ A) & = & \phi & \text{if C j=fail} \\
 & = & \{C. \triangleright I\} & \text{otherwise} \\
\textbf{sel}\, j\, (A_1 \,\Box\, A_2) & = & \textbf{sel}\, j\, A_1 & \text{if } \textbf{sel}\, j\, A_1 \neq \phi \\
 & = & \textbf{sel}\, j\, A_2 & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
\textbf{next}\, j\, (\mu a.A) & = & \textbf{next}\, j\, A[\mu a.A/a] & \\
\textbf{next}\, j\, (C \triangleright I;\ A) & = & C \triangleright I;\ A & \text{if C j=fail} \\
 & = & A & \text{otherwise} \\
\textbf{next}\, j\, (A_1 \,\Box\, A_2) & = & \textbf{next}\, j\, A_1 & \text{if } \textbf{sel}\, j\, A_1 \neq \phi \\
 & = & \textbf{next}\, j\, A_2 & \text{if } \textbf{sel}\, j\, A_2 \neq \phi \\
 & = & (A_1 \,\Box\, A_2) & \text{otherwise}
\end{array}
$$

Aspect Weaving (2)

- The Monitor:

$$[j,\ P,\ \sigma]^{\phi} \mapsto \sigma[end]$$

$$\frac{\mathcal{S} = \{C \triangleright I\} \cup \mathcal{S}' \quad C\,j = \psi \quad (\downarrow, \psi I, \sigma) \xrightarrow{*} (\uparrow, \psi I, \sigma')}{[j,\ P,\ \sigma]^{\mathcal{S}} \mapsto [j,\ P,\ \sigma']^{\mathcal{S}'}}$$

- Woven Execution:

$$\frac{[j,\ P,\ \sigma]^{\textbf{sel}\ j\ A} \overset{*}{\mapsto}\ \sigma_a \quad (j,\ P,\ \sigma_a) \rightarrow\ (j',\ P,\ \sigma')}{(A,\ j,\ P,\ \sigma) \Rightarrow\ (\textbf{next}\,j\,A,\ j',\ P,\ \sigma')}$$

Aspect Strong Independence

- Laws for Aspects :

$$
\begin{array}{lll}
[(un)fold] & \mu a.A = A[\mu a.A/a] & \\
[assoc] & (A_1 \Box A_2) \Box A_3 \;=\; A_1 \Box (A_2 \Box A_3) & \\
[commut] & (C_1 \triangleright I_1;\; A_1) \Box (C_2 \triangleright I_2;\; A_2) \;=\; (C_2 \triangleright I_2;\; A_2) \Box (C_1 \triangleright I_1;\; A_1) & \text{if } C_1 \wedge C_2 = fail \\
[elim_1] & C \triangleright I \;=\; false \triangleright I & \text{if } C = fail \\
[elim_2] & (false \triangleright I;\; A_1) \Box A_2 \;=\; A_2 & \\
[elim_3] & false \triangleright I;\; C_1 \triangleright I_1;\; A \;=\; false \triangleright I;\; A & \\
[priority] & (C_1 \triangleright I_1;\; A_1) \Box (C_2 \triangleright I_2;\; A_2) \;=\; (C_1 \triangleright I_1;\; A_1) \Box (C_2 \wedge \neg C_1 \triangleright I_2;\; A_2) & \\
[propag] & let\ A \;=\; (C_1 \triangleright I_1;\; A_1) \Box \ldots. \Box (C_n \triangleright I_n;\; A_n) & \\
 & and\ A' \;=\; (C'_1 \triangleright I'_1;\; A'_1) \Box \ldots. \Box (C'_m \triangleright I'_m;\; A'_m) & \\
 & then\ A \parallel A' \;=\; \Box_{i=1..n}^{j=1..m} C_i \wedge C'_j \triangleright (I_i \bowtie I'_j);\; (A_i \parallel A'_j) & \\
 & \qquad\qquad \Box_{i=1..n} C_i \triangleright I_i;\; (A_i \parallel A') & \\
 & \qquad\qquad \Box_{j=1..m} C'_j \triangleright I'_j;\; (A \parallel A'_j) &
\end{array}
$$

Aspect Independence w.r.t a Program

$$
\begin{array}{lcl}
I_w(A,j) & = & \textit{if } \mathbf{sel}\, j\, A = \phi \ \textit{ then } \Box_{j' \in (\mathbf{step}_p\ j)} I_w(A, j') \\
& & \textit{else if } \ \mathbf{sel}\, j\, A = \{C \rhd I\} \ \textit{ then } C \rhd I;\ \Box_{j' \in (\mathbf{step}_p\ j)} I_w(\mathbf{next}\, j\, A, j') \\
& & \textit{else if } \ \mathbf{sel}\, j\, A = \{C \rhd I,\ C' \rhd I'\} \ \textit{ then } C \wedge C' \rhd (I \bowtie I');\ \Box_{j' \in (\mathbf{step}_p\ j)} I_w(\mathbf{next}\, j\, A, j')
\end{array}
$$

Conflict Resolution

- Using parallel operators such us $\|_{seq}$ and $\|_{fst}$
- Defining scopes for Aspects: **scope** *id Idset A*

Extended version for the Framework [Rémi'04]

- Inter-crosscut Variables :

$$C \ ::= \ v \doteq T \mid C_1 \wedge C_2 \mid C_1 \vee C_2 \mid \neg C$$

$$C \ = \ C[(\bullet \ \doteq \ T/\hat{T})]$$

$$C \ = \ C[z/\bar{x}] \wedge x \ \doteq \ z$$

$$\begin{array}{lcll} \textbf{next}\, j\ (C \triangleright I;\ A) & = & C \triangleright I;\ A & \text{if C j=fail} \\ & = & \psi A & \text{otherwise} \end{array}$$

$$\begin{array}{lcl} \psi(\mu a.A) & = & \mu a.\psi A \\ \psi(C \triangleright I;\ A) & = & \psi' C \triangleright \psi' I;\ \psi' A \\ \psi(A_1 \square A_2) & = & (\psi A_1 \square \psi A_2) \end{array}$$

Composition Operators

- Sequential Composition Operator : $A_1 - C \rightarrow A_2$
- Adaptors $A_1 \parallel_O A_2$:

$$
\begin{array}{lcl}
O & ::= & \mu a.O \\
 & & \mid C \triangleright F;\ O \\
 & & \mid C \triangleright F;\ a \\
 & & \mid O_1 \Box O_2 \\
F & ::= & (U \oplus B) \\
U & ::= & id \mid skip \\
B & ::= & \bowtie \ \mid seq \mid fst \mid snd \mid skip
\end{array}
$$

Aspect Requirements

- Requirements are aspects
- CFA is expressed by:

$$
\begin{array}{lcl}
J & ::= & f\ J_1 \ldots J_n \mid ? \\
S & ::= & \mu s.S \\
 & & \mid (J_1 \rightarrow S_1)[] \ldots [](J_n \rightarrow S_n) \\
 & & \mid s
\end{array}
$$

Sequential EAOP [Rémi'06]

- Aspects are translated from EAOP syntax to the FSP
- Events are introduced to synchronize advices together or advices with the base program
- Weaving is modeled as parallel composition of the FSP describing the base program and the Aspect.

Concurrent EAOP

- Each program is viewed as a parallel composition of several FSPs automatons describing its threads
- Aspects are viewed as independent processes that run in parallel and synchronized with the base program
- A hiding operator is introduced to implement the concurrency.

Concurrent Aspect Composition in EAOP

- Sequential Functional Composition: $Fun(aspect_1,\ aspect_2)$
- Parallel Conjunctive Composition: : $parAnd(aspect_1,\ aspect_2)$

Perspectives