An Overview of Component Models: Part 2

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]
  - 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 documntations 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

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 $\psi : I \rightarrow (\Sigma \rightarrow C) \times \{before, after, around\}$
- A tagged instruction : $\bar{i}$
- A matching function: $match : \mathcal{P} \times I \rightarrow bool$
- A weaving relation: $\rightarrow$

CASB: Weaving of a Single Aspect

- Before aspect

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

- After aspect

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

- Around aspect

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

$$(pop_p : C, \Sigma, i : P) \rightarrow (C, \Sigma, P)$$

$$(proceed : C, \Sigma, i : P) \rightarrow (i : push_p\ i : C, \Sigma, P)$$

CASB: Weaving of Several Aspects

- Aspects of the same Kind
  - Before aspects

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

  - After aspects

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

  - Around aspects

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

$$(pop_p\ n : C, \Sigma, x_1 : ... : x_n : P) \rightarrow (C, \Sigma, P)$$

$$(proceed : C, \Sigma, x : P) \rightarrow (x : push_p\ x : C, \Sigma, P)$$

- Aspects of Different Kinds

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

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

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

$$(pop_{\Psi} : C, \Sigma, \psi_i : \Psi) \rightarrow (C, \Sigma, \Psi)$$

- Aspect Instantiation

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

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 :

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

Aborters Category

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

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

Sequential EAOP

- Syntax :

$$A ::= \mu a.A \mid C \triangleright I; A \mid C \triangleright I; a \mid A_1 \Box A_2$$

to be continued...

Concurrent EAOP

Concurrent Aspect Composition in EAOP

- Sequential Functional Composition
- Parallel Conjunctive Composition

Perspectives