# The Kmelia Multi-Service Component Model

Pascal André, Gilles Ardourel, Christian Attiogbé
LINA CNRS FRE 2729 - University of Nantes
(Pascal.Andre,Gilles.Ardourel,Christian.Attiogbe)@univ-nantes.fr

June, 2006

## General Description

Kmelia is a component model based on services: an elementary Kmelia component encapsulates several services. The service behaviours are captured with specific labelled transition systems that enables optional transitions. Kmelia makes it possible to specify abstract components, to compose them and to check various properties. A Kmelia abstract component is a mathematical model of an open multi-service system that supports synchronous communication with its environment.
A component specification language named Kmelia and a prototype toolbox (COSTO) support the Kmelia model. The toolbox already enables formal analysis via Lotos/CADP and Mec.

## Main Features of Kmelia

### Component Description Style

```
Component C1
  Interface   <Interface descr>          Provided aService ()
  Types       <Type Defs>                 Pre <Predicate>
  Variables   <Var list>                  Post <Predicate>
  Invariant   <Predicate>                 Behaviour
  Initialisation  ...                      init aState
  Services                                 final aState
     ...                                   { state_i --label--> state_j
                                             ... }
                                          end
end // component
```

### Service Description

A *service* $s$ of a component $C$ is defined with an *interface* $I_s$ and a (dynamic) *behaviour* $\mathcal{B}_s$: $\langle I_s, \mathcal{B}_s \rangle$. The interface $I_s$ of a service $s$ is defined by a 5-tuple $\langle \sigma,\ P,\ Q,\ V_s, S_s \rangle$ where $\sigma$ is the service signature (name, arguments, result), $P$ is a precondition, $Q$ is a postcondition, $V_s$ is a set of local declarations and the *service dependency* $S_s$ is a 4-tuple $S_s = \langle sub_s,\ cal_s,\ req_s,\ int_s \rangle$ of disjoint sets where $sub_s$ (resp. $cal_s$, $req_s$, $int_s$) contains the provided services names (resp. the services required from the caller, the services required from any component, the internal services) in the $s$ scope.

The behaviour $\mathcal{B}_s$ of a service $s$ is an *extended labelled transition system* (eLTS) defined by a 6-tuple $\langle S, L, \delta, \Phi, S_0, S_F \rangle$ with $S$ the set of the states of $s$; $L$ is the set of transition labels and $\delta$ is the transition relation ($\delta \in S \times L \rightarrow S$). $S_0$ is the initial state ($S_0 \in S$), $S_F$ is the finite set of final states ($S_F \subseteq S$), $\Phi$ is a state annotation function ($\Phi \in S \rightarrow sub_s$). An eLTS is obtained when we allow nested states and transitions. This provides a means to reduce the LTS size and a flexible description with optional behaviours accesible via the state annotation mechanism.

### Component Description

A component ($C$) is a 8-tuple $\langle \mathcal{W}, Init, \mathcal{A}, \mathcal{N}, I, \mathcal{D}_S, \nu, \mathcal{C}_S \rangle$ with:

- $\mathcal{W} = \langle T, V, V_T, Inv \rangle$ the state space where $T$ is a set of types, $V$ a set of variables, $V_T \subseteq V \times T$ a set of typed variables, and $Inv$ is the state invariant;
- $Init$ the initialisation of the $V_T$ variables;
- $\mathcal{A}$ a finite set of elementary actions;
- $\mathcal{N}$ a finite set of service names;
- $I$ the component interface which is the union of two disjoints finite sets: $I_p$ the set of names of the provided services that are visible in the component environment and $I_r$ the names of required services.
- $\mathcal{D}_S$ is the set of service descriptions which is partitioned into the provided services ($\mathcal{D}_{S_p}$) and the required services ($\mathcal{D}_{S_r}$).
- $\nu : \mathcal{N} \rightarrow \mathcal{D}_S$ is the function that maps service names to service descriptions. Moreover there is a projection of the $I$ partition on its image by $\nu$:
  $n \in I_p \Rightarrow \nu(n) \in \mathcal{D}_{S_p} \wedge n \in I_r \Rightarrow \nu(n) \in \mathcal{D}_{S_r}$
- $\mathcal{C}_S$ is a constraint related to the services of the interface of $\mathcal{C}$ in order to control the usage of the services.

The component relies on the behaviours of its services.

### Assembly, Composition and Formal Analysis

The Kmelia components are composable via the interfaces of the involved services. Interface-compatible and behaviour-compatible services are composed at various levels to form larger components: *assemblies.* Assemblies and services may be encapsulated into a component to form a composition.
Among the possible analysis, behavioural compatibility analysis is performed by considering the correct interaction between the LTS of the involved services.

## An overview of the COSTO toolbox

The figure 1 gives an overview of the main parts of the COSTO toolbox.

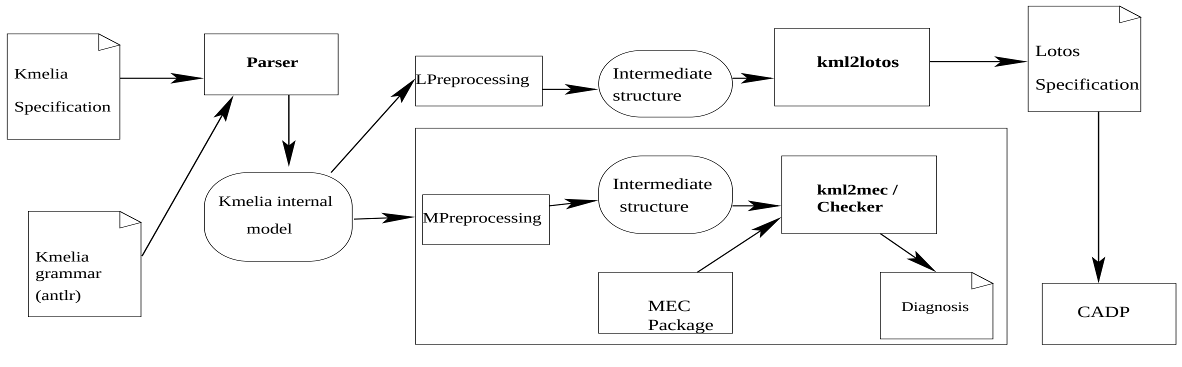

Figure 1: An overview of the COSTO

### Some references

Please see [2, 1] for more details on the Kmelia models.

## References

[1] Pascal André, Gilles Ardourel, and Christian Attiogbé. Coordination and Adaptation for Hierarchical Components and Services. In *Third International ECOOP Workshop on Coordination and Adaptation Techniquesfor Software Entities (WCAT'06)*, 2006. to appear.

[2] Christian Attiogbé, Pascal André, and Gilles Ardourel. Checking Component Composability. In *5th International Symposium on Software Composition, SC'06*, volume 4089 of *LNCS*. Springer, 2006.