Static Analysis of Aspect Interaction and Composition in Component Models

Abdelhakim Hannousse\textsuperscript{1} \hspace{1em} Rémi Douence\textsuperscript{2} \hspace{1em} Gilles Ardourel\textsuperscript{3}

\textsuperscript{1}ASCOLA Team, Aelos, INRIA, LINA
Ecole des Mines de Nantes, France

\textsuperscript{2}LINA & INRIA,
Nantes, France

\textsuperscript{3}Aelos Team, LINA CNRS UMR6241
Université de Nantes, France

10th International Conference on Generative Programming and Component Engineering, 2011
Outline

1. Motivating Example
   - Airport Internet Access System
   - Extensions and interferences

2. Specification of Components and Aspects
   - Structural part
   - Behavioral part

3. Aspect interaction analysis
   - UPPAAL files generation
   - Composition operators
   - Properties

4. Conclusion
Motivating Example

Airport Internet Access System

Component Provided interface Required interface

*Token
*Timer
:ValidityChecker

:Arbitrator

:FlightTicketManager

:InternetAccessManager

:DHCPServer

:IPDbConnection

:IPAddressManager

:Firewall

:ProxyServer

Legend

Component Provided interface Required interface * Multiple Instances

Airport Internet Access System
Motivating Example

Airport Internet Access System

1- Getting IP address
Motivating Example

Airport Internet Access System

2- Login
Motivating Example

3- Retrieving user info (class, allocated time)
Motivating Example

Airport Internet Access System

Component Provided interface Required interface

- Arbitrator
- User
- Firewall
- ProxyServer
- FlightTicketManager
- InternetAccessManager
- DHCPServer
- IPAddressManager
- IPDbConnection
- Timer
- ValidityChecker
- Token

Legend:
- Component
- Provided interface
- Required interface
- Multiple Instances

4- Allowing access
Motivating Example

Airport Internet Access System

4- Creating a Token for the session
Motivating Example

Airport Internet Access System

**Motivating Example**

4- Setting the Timer for the session
5- the user sends its queries
Motivating Example

Airport Internet Access System

6- Either the user logs out or there is a timeout
7- The timeout reaches the Arbitrator
Motivating Example

1. **Component Provided interface**
   - Arbitrator
   - User
   - FlightTicketManager
   - InternetAccessManager
   - DHCPServer
   - IPAddressManager
   - Firewall
   - ProxyServer
   - Timer
   - ValidityChecker
   - Token
   - TokenCallback
   - TimerCallback
   - DhcpCallback
   - DhcpListener
   - IpMacDb
   - IpDbConnection
   - IpAddressManager
   - DhcpListener

2. **Required interface**
   - IQuery
   -ILogin
   - IAlert
   - IToken
   - ITokenCallback
   - ITimerCallback
   - ITimer
   - IFlyTicketAuth

3. **Multiple Instances**

4. **Legend**
   - Component
   - Provided interface
   - Required interface
   - Multiple Instances

5. **Airport Internet Access System**

6. **Motivating Example**

7. **8** The arbitrator blocks the address and disables it
Extensions of the base program

- add a promotional 10 bonus time
- alert the user 5 minutes before the end of the session
- reduce access when the system is overloaded
- access control for minors
Constraints on the implementation

- base program source code might be unavailable
- extended features might be non modular

Aspect Oriented Programming for component architectures

- VIL declarative pointcut language
- weaving of wrappers intercepting services calls and executing advices
- either skip or proceed the initial call
Interferences

After weaving we have to verify
- system properties we wanted to keep
- new properties introduced by aspects

In some cases, aspects interact badly with each other: interferences.
**Interaction example**

Bonus and Alert are designed independently and both intercept the timeout

**Bonus**
- on first timeout, resets the timer for 10 and skips
- on subsequent timeouts, proceeds

**Alert**
- on setTimeout, reduce the time by 5 and proceeds
- on first timeout, sends an alert and resets the Timer for 5 and skips
- on subsequent timeouts, proceeds
Considering an independent sequence operator which only skips if both skip IndependentSequence(Bonus, Alert) gives

- a Timer set to 55,
- on first timeout, Bonus resets Timer to 10, then skips
- on first timeout, Alert sends and Alert and resets Timer to 5, then skips
Interaction example

Considering an independent sequence operator which only skips if both skip IndependentSequence(Bonus, Alert) gives

- a Timer set to 55,
- on first timeout, Bonus resets Timer to 10, then skips
- on first timeout, Alert sends and Alert and resets Timer to 5, then skips

! the bonus is not correctly applied
Interaction example

Considering an independent sequence operator which only skips if both skip IndependentSequence(Bonus, Alert) gives

- a Timer set to 55,
- on first timeout, Bonus resets Timer to 10, then skips
- on first timeout, Alert sends and Alert and resets Timer to 5, then skips

! the bonus is not correctly applied

IndependentSequence( Alert,Bonus) does not work better.
Considering an independent sequence operator which only skips if both skip
IndependentSequence(Bonus, Alert) gives
- a Timer set to 55,
- on first timeout, Bonus resets Timer to 10, then skips
- on first timeout, Alert sends and Alert and resets Timer to 5, then skips

! the bonus is not correctly applied

IndependentSequence( Alert,Bonus) does not work better. How to detect this situation?
Our Proposal

Specify

- with an ADL that specifies both the structure and the behavior, for components and aspects.
- with compositions operators for aspects

Verify with UPPAAL

- transform the specification to check its properties using Uppaal model checkers
- specify system properties and aspect properties to detect interferences
- use operator templates to solve interactions
The ADL

system airportInternetAccess
    // interfaces
interface ITimer { @sync setTimeout(int) }
interface ITimerCallback { @async timeout() }
    // other interfaces
    // ...

Interfaces - distinguishing synchronous from asynchronous
The ADL

```
system airportInternetAccess
    // interfaces
    // components
primitive Timer (n) {
    provides ITimerCallback iTimerCallback;
    requires ITimer iTimer;
    // behavior
}
primitiveValidityChecker (n) {
    provides ITimer iTimer, IToken iToken;
    requires ITimerCallback iTimerCallback,
            ITokenCallback iTokenCallback;
    // behavior
}
// ...
```

Primitive components with behaviors
The ADL

system airportInternetAccess
   // interfaces
   // components
   //..
composite Token (n) {
   provides ITokenCallback iTokenCallback;
   requires IToken iToken;
   internals Timer, ValidityChecker;
}
   // bindings
client ValidityChecker.iTimer server Timer.iTimer;
   // other bindings
   // ..

Composites and bindings
Aspects and their weaving
Behavior expressed as timed automata

UPPAAL

- designing systems as networks of timed automata
- express properties in CTL and check them
- graphical and textual representation

Automata (templates) parametrized by the number of instances
Behavior in UPPAAL

```
process Timer() {
    clock cl;
    TIME time;
    state l0 { cl <= time }, l1, l2;
    init l0;
    trans
        l0 -> l1 { select t:TIME;
            sync timer_iTimer_setTimeout[t]?
            assign time := t, cl := 0; },
        l1 -> l2 { sync E_timer_iTimer_setTimeout!; },
        l2 -> l0 { sync timer_iTimerCallback_timeout!; };
}
```

Listing 1: UPPAAL-XTA description of the Timer component
Base system in Uppaal

UPPAAL files generation scheme

- **Primitives**: adding parameter if multiple instances
- **Composites**:
  - one process per bound interface,
  - one cycle per service, with emission/reception for asynchronous, plus their respective returns for synchronous composite with multiple instances have parametered processes
- **Bindings**: renaming client channels using server name
- **System**: parallel composition
Weaving Aspects

- Aspects already described in UPPAAL
- Weaving is adding the interception calls which call the advice then either skip or proceed
Composition operators

Several operators
- return skip or proceed to be composable
- implemented in Java for Fractal

In UPPAAL
- specified as templates to be instanciated during transformation
- parametrized for every intercepted service
  - with different treatments on shared or non shared join points
  - with guards depending on skip/proceed or other predicates
- integrated at weaving
## Properties

### System properties in CTL

#### Base system properties

**Live1**  
User(id).Connected → User(id).Disconnected

**Safe1**  
A[] not deadlock

**Safe2**  
A[] ∀(id:ID),∀(ip:IP) (User(id).Connected ∧ currentIp(id)==ip ∧ Firewall.enabled(id)) ⇒ User(id).isConnected

**Safe3**  
A[] ∀(id:ID) User(id).Connected ⇒ User(id).cl<=validity(id)

**Reach1**  
E<> User(0).Connected ∧ (∀(id:ID) id!=0 ⇒ User(id).Connected)
Aspects properties

**Bonus Aspect**

**Safe3bis**  
\[ A[] \ \forall (id:ID) \ \text{User}(id).\text{Connected} \Rightarrow \text{User}(id).cl \leq \text{validity}(id) + \text{BonusTime} \]

**Alert Aspect**

**Live2**  
\[ \text{Use}(id).\text{Connected} \rightarrow \text{User}(id).\text{Disconnected} \land \text{User}(id).\text{isAlerted} \]

**Safe4**  
\[ A[] \ \forall (id:ID) \ \text{User}(id).\text{Alerted} \Rightarrow \text{User}.cl = \text{validity}(id) - \text{AlertTime} \]
9 primitives, 11 interfaces
- Conflicts detected
- Rapid state explosion
Conclusion

Detect interference between aspects in component models
- ADL including behaviour specification and aspect weaving
- transformations to UPPAAL
- aspect and system properties in CTL
- interference detection

Solve interference via operators
- set of composable operators
- implemented in Java for Fractal, but more general
- modelised as UPPAAL templates for checking
Perspectives

- Model transformation for UPPAAL generation
- alternative properties verification (proof)
- adapting to models with rich communications styles (sofa)