Report on
“Specification and Refinement of Dynamic Software Architectures”

Carlos Canal, Ernesto Pimentel, Jose M. Troya
Depto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain
E-mail: {canal, ernesto, troya}@lcc.uma.es
Introduction

- LEDA architecture description language
  - Dynamic architectures
  - $\pi$ calculus-based behavior description
  - Inheritance and parametrization
LEDAn overview

- Dynamic architectures
- Roles
  - Observable behavior description
- Components
  - Composed of other components
  - Attachments
- No connectors
- Specification can be executed and analyzed
\(\pi\)-calculus

- 0 - stop
- \((x)P\) - \(x\) is private channel in \(P\) (hiding)
- \([x=y]P\) - \(P\) if the condition holds, otherwise as 0
- \(\tau.P\) - internal action
- \(x'y.P\) - send \(y\) over \(x\)
- \(x(w).P\) - wait for \(y\) to be sent over \(x\) and then behave as \(P\{y/w\}\)
- \(x'(y).P = (y)x'y\) - widen private \(y\) by sending over \(x\)
- \(P|Q\) - composition
- \(P+Q\) - alternative (local or global)
Components

- Classes and instances
- Contains
  - Interface
  - Composition
  - Attachments
Component interface

Set of role instances

```plaintext
component Sender {
  interface
    writer : Writer;
}

component Receiver {
  interface
    reader : Reader;
}
```
Role specification

role Writer(w,wq) {
    spec is
    \( \tau \cdot (\text{data}) \cdot !\text{w!(data)} \cdot \text{Writer}(w,wq) + \tau \cdot \text{wq!}().0; \)
}

role Reader(w,wq) {
    spec is
    \( w?\cdot (\text{data}) \cdot \text{Reader}(w,wq) + \text{wq?}().0; \)
}
Components and systems not distinguished

component Client {
    interface
    request : Request(request)
    {
        spec is
            (reply)request!(reply).
            reply?(service).
            Request(request);
    }
    composition
    service[] : any;
}

component Server {
    interface
    serve : Serve(request)
    {
        names
            n : Integer := 0;
        spec is
            request?(reply).
            (new service) reply!(service).
            n++.Serve(request);
    }
    composition
    service[] : any;
}
Attachments

- Static
- Reconfigurable
- Multiple
  - 1:m
  - 1:1
Static attachment

```plaintext
component ClientServer {
  interface none;
  composition
    client : Client;
    server : Server;
  attachments
    client.request(r) <> server.serve(r);
}
```
component ReconfigurableClientServer {
    interface none;
    composition
        client : Client;
    attachments
        client.request(r) <>
            if ( server[1].n <= server[2].n )
            then server[1].serve(r)
            else server[2].serve(r);
}
Multiple attachment

**component** ServerPool {
  **interface**
  serve : Pool;
  **composition**
  server[] : Server;
  **attachments**
  server[*].serve(r) >> serve(r);
}

**component** MultipleClientServer {
  **interface** none;
  **composition**
  client[] : Client;
  pool : ServerPool;
  **attachments**
  client[*].request(r) <> pool.serve(r);
}
Extension and refinement

- Role extension
- Architecture instantitation
- Architecture refinement
- Adaptors
Role extension

- Redefinition of parent role
- Extension of parent role

```java
role StatServe(request, statistics) extends Serve {
    adding
    statistics!(n).StatServe(request, statistics);
}
```
Architecture instantiation

- Component extension
- Instantiation using derived subcomponents

derivedComponent : ComponentClass[subcomponent : DerivedSubcomponentClass];
Architecture refinement

component ReceiverClient extends Client {
  interface
    request : RequestSenders(request) extends Request {
      spec is
        (reply)request!(reply).
        (new receiver)reply?(service).
    RequestSenders(request);
  }

  composition
  receiver[] : Receiver;
  service[] : Sender;

  attachments
  receiver[].reader(w,wq) <> service[].writer(w,wq);
}

refinedCS : MultipleClientServer[client: ReceiverClient,
  pool.server[].service[] : Sender];
Adaptors

- Glue between subcomponents
- Modification of interface
- $\pi$-calculus process
Adaptors - Example

component NRServer {
    interface
        serve:NRServe(request,crash){
            spec is
                τ.request?(reply).
                (new service)reply!(service).
                NRServe(request,crash)
                + τ.crash!().0;
        }
}

component FTServerPool extends ServerPool {
    composition
        server[] : NRServer;
        restart : Restart(crash) {
            spec is
                crash?()(new server)Restart(crash);
        }
        attachments
            restart(e),server[*].serve(r,e) >>
            serve(r);
}

ftcs:MultipleClientServer[pool: FTServerPool];
Conclusion

- Dynamic ADL
- No implementation available
- Rather trivial case studies