HIGH-LEVEL LANGUAGE SUPPORT FOR THE CONTROL OF RECONFIGURATIONS IN COMPONENT BASED ARCHITECTURES

Frederico ALVARES\textsuperscript{1}, Eric RUTTEN\textsuperscript{1} and Lionel SEINTURIER\textsuperscript{2}

Inria Grenoble - Ctrl-a Team\textsuperscript{1} / Lille - Spirals\textsuperscript{2}

October 17, 2014
Context: Component-based Software Engineering

• Modularity and reuse capabilities of Software Components
• Reconfigurability and Adaptability
  ◦ Reflection (introspection and intercession) capabilities
    → add, remove, (un)bind components; set attributes at runtime

Architectures are capable to adapt to new requirements and execution conditions at runtime
Context: Component-based Software Engineering

- Modularity and reuse capabilities of Software Components

Architectures are capable to adapt to new requirements and execution conditions at runtime.
Context: Component-based Software Engineering

- Modularity and reuse capabilities of Software Components
- Reconfigurability and Adaptability
  - Reflection (introspection and intercession) capabilities
    → add, remove, (un)bind components; set attributes at runtime
Context: Component-based Software Engineering

- Modularity and reuse capabilities of Software Components
- Reconfigurability and Adaptability
  - Reflection (introspection and intercession) capabilities
    - add, remove, (un)bind components; set attributes at runtime

Architectures are capable to adapt to new requirements and execution conditions at runtime
Dynamic Reconfiguration

Back End

- Logger
- Request Analyser
- Request Handler
Dynamic Reconfiguration

Back End

Cache Handler
Request Analyser
Request Handler

Logger
How can configs and reconfigs be described?

- ADLs to describe initial configurations

```xml
<definition>
  <component name="backend">
    <interface name="s" role="server"/>
    <component name="RequestAnalyzer">
      <interface name="s" role="server"/>
      <interface name="c" role="client"/>
    </component>
  </component>
</component>
...
</definition>
```

- DSLs (e.g., FPath/FScript) to describe reconfig actions

```plaintext
stop($requestAnalyzer);
add($backend,$logger);
bind($requestAnalyzer/service::c,$logger/service::s);
start($requestAnalyzer);
```
When things get way too complex...

- Exhaustive
- Infeasible
- Error-prone

There is a need for:

1. High-level and specific language support for:
   - Configurations, Reconfigurations and Properties
2. Mechanisms to control reconfigurations while enforcing properties
When things get way too complex...

- Exhaustive
- Infeasible
- Error-prone
When things get way too complex...

- Exhaustive
- Infeasible
- Error-prone

There is a need for:

1. High-level and specific language support for:
   - Configurations, Reconfigurations and Properties
When things get way too complex...

- Exhaustive
- Infeasible
- Error-prone

There is a need for:

1. High-level and specific language support for:
   - Configurations, Reconfigurations and Properties

2. Mechanisms to control reconfigurations while enforcing properties
We propose...

- A Domain-Specific Language for Self-adaptive Component-based Applications
- Extension of current ADLs with constructs for describing
  - Configurations
  - Reconfigurations behaviour
  - Properties to be enforced
- It can be translated into a Reactive Model and benefit from
  - Verification and control tools
  - Generation of executable code
Outline

1. Example Application: Znn.com

2. Introduction to Ctrl-F

3. Controlling Reconfigurations
   3.1 Heptagon/BZR
   3.2 Translating Ctrl-F into BZR

4. Final Remarks
1

EXAMPLE APPLICATION: ZNN.COM
Znn.com

- Self-adaptive platform developed at CMU
Znn.com

- Self-adaptive platform developed at CMU
- Based on news website like cnn.com or rockymountainnews.com
Znn.com

- Self-adaptive platform developed at CMU
- Based on news website like cnn.com or rockymountainnews.com
- Adjustable pool of web servers to cope with the varying demand
Znn.com

- Self-adaptive platform developed at CMU
- Based on news website like cnn.com or rockymountainnews.com
- Adjustable pool of web servers to cope with the varying demand
- Modes (or content fidelity) to cope with workload spikes
  - Text, image and HD image
Znn.com

- Self-adaptive platform developed at CMU
- Based on news website like cnn.com or rockymountainnews.com
- Adjustable pool of web servers to cope with the varying demand
- Modes (or content fidelity) to cope with workload spikes
  - Text, image and HD image
- Objective: performance, cost and content fidelity
Znn.com - Extension

- Many content service providers sharing the same infrastructure

![Diagram showing Znn.com instances and content providers]

**Soccer Content Provider**

**Politics Content Provider**

Znn.com Provider

\[ m_1, m_2, m_3, \ldots, m_k \]
Znn.com - Extension

- Many content service providers sharing the same infrastructure
- Several component instances with different Service Level Agreements
Znn.com - Extension

- Many content service providers sharing the same infrastructure
- Several component instances with different Service Level Agreements
  - Different priorities
Znn.com - Extension

- Many content service providers sharing the same infrastructure
- Several component instances with different Service Level Agreements
  - Different priorities
  - More interesting from the self-adaptive and control perspectives
2

INTRODUCTION TO CTRL-F
Introduction to Ctrl-F

Declarative Part

- Resources
- Events
- Composites, components
- Configurations: assemblies and modes
- Properties

Imperative Part

- Behavioural language
Ctrl-F Syntax

Example in Znn.com:

```
1 eventType oload
2 eventType uoload
3
4 resource pool {
5    capacity 10
6 }
7 ...
```
Example in Znn.com:

```
1 eventType oload
2 eventType uload
3 resource pool {
4   capacity 10
5 }
6 ...
7 ...
```
Ctrl-F Syntax

Example in Znn.com:

```
1   eventType oload
2   eventType uoload
3
4   resource pool {
5       capacity 10
6   }
7   ...
```
component Znn {
 server interface si1
 ... 
 sink sink1 oload
 sink sink2 uload
 attribute overallQuality
 component LoadBalancer {...}
 component AppServer {...}
 configuration config1 {...}
 ... 
 behaviour {...}
 property {...}
}
1 component Znn {
  2
  3    server interface sil
  4    ... 
  5
  6    sink sink1 oload
  7    sink sink2 uload
  8
  9      attribute overallQuality
 10
 11    component LoadBalancer {...}
 12    component AppServer {...}
 13
 14    configuration config1 {...}
 15    ...
 16    behaviour {...}
 17    property {...}
 18 }

High-level Language Support for the Control of Reconfigurations in Component Based Architectures
component Znn {
  server interface si1
  ...
  sink sink1 unload
  sink sink2 unload
  attribute overallQuality
  component LoadBalancer {...}
  component AppServer {...}
  configuration config1 {...}
  ...
  behaviour {...}
  property {...}
}
Composite

Interface:
- component
- Interface
- client
- server

Event Sink/Source:
- sink
- source

Attribute:
- attribute

```java
1  component Znn {
2  
3    server interface si1
4    ...
5  
6    sink sink1 oload
7    sink sink2 uoload
8  
9      attribute overallQuality
10  
11    component LoadBalancer {...}
12    component AppServer {...}
13  
14    configuration config1 {...}
15    ...
16    behaviour {...}
17    property {...}
18  }
```
1 component Znn {
2    server interface si1
3    ...
4
5    sink sink1 uload
6    sink sink2 uload
7
8    attribute overallQuality
9
10   component LoadBalancer {...}
11
12   component AppServer {...}
13
14   configuration config1 {...}
15   ...
16   behaviour {...}
17   property {...}
18 }

High-level Language Support for the Control of Reconfigurations in Component Based Architectures 14 / 38
Composite

Interface:

Event Sink/Source:

Attribute:

Component:

```
1 component Znn {
2    server interface si1
3    ...
4
5    sink sink1 oload
6    sink sink2 uload
7
8    attribute overallQuality
9
10   component LoadBalancer {...}
11   component AppServer {...}
12
13   configuration config1 {...}
14   ...
15   behaviour {...}
16   property {...}
17
18 }
```
Atomic Components

```java
1 component AppServer {
2    server interface si
3    sink sink1 oload
4    sink sink2 uload
5    attribute fidelity
6
7    configuration text {
8        delivers fidelity as {0.25}
9        consumes 0.2 of pool
10    }
11    configuration img-1d {
12        delivers fidelity as {0.5}
13        consumes 0.6 of pool
14    }
15    configuration img-hd {...}
16    ...
17    behaviour {...}
18    property {...}
19 }
```
Atomic Components

```
component AppServer {
    server interface si
    sink sink1 oload
    sink sink2 uload
    attribute fidelity

collection text {
    delivers fidelity as {0.25}
    consumes 0.2 of pool
}
collection img-ld {
    delivers fidelity as {0.5}
    consumes 0.6 of pool
}
collection img-hd {...}

... behaviour {...}
property {...}
}
```
component Znn {
  ...
  initial configuration config1 {
    lb:LoadBalancer
    server1:AppServer
    bind lb.cil to server1.si
    bind lb.si to si
    delivers overallQuality as {server1.fidelity}
  }
  ...
}

Assembly (1/2)

```
1 component Znn {
2     ...
3     initial configuration config1 {
4         lb: LoadBalancer
5         server1: AppServer
6         bind lb.c11 to server1.si
7         bind lb.si to si
8         delivers overallQuality as
9             {server1.fidelity}
10     }
11     ...
12 }
```
```
1 component Znn {
2   ...
3   initial configuration config1 {
4     lb: LoadBalancer
5     server1: AppServer
6     bind lb.cil to server1.si
7     bind lb.si to si
8     delivers overallQuality as
9       {server1.fidelity}
10   }
11   ...
12 }
```
component Znn {
    ...
    initial configuration config1 {
        lb: LoadBalancer
        server1: AppServer
        bind lb.ci1 to server1.si
        bind lb.si to si
        delivers overallQuality as {server1.fidelity}
    }
    ...
}
component Znn {
  ...
  initial configuration config1 {...}
  configuration config2 extends config1 {
    server2: AppServer
    bind lb.ci2 to server2.si
    delivers overallQuality as
    {
      (server1.fidelity + server2.fidelity)/2
    }
  }
  configuration config3 {...}
  behaviour {...}
  property {...}
  ...
}

component Znn {
  ...
  initial configuration config1 {...}
  configuration config2 extends config1 {
    server2: AppServer
    bind lb.ci2 to server2.si
    delivers overallQuality as
    { (server1.fidelity + server2.fidelity)/2 }
  }
  configuration config3 {...}
  behaviour {...}
  property {...}
  ...
}
Main Composite in Znn.com

```plaintext
component Application {
    component Znn {
        server interface si
        component LoadBalancer {...}
        component AppServer {...}
        configuration config1 {...}
        ...
    }
    initial configuration main {
        znn-soccer:Znn
        znn-politics:Znn
    }
    property {...}
}
```
Behaviour

- High-level imperative language
- Order and conditions under which configurations take place
- Defined with the help of composable behaviours
- Atomic behaviour: configuration
  - It never ends unless a behavior explicitly states so
- Reconfigurations are implicitly treated
  - When a configuration is terminated ...
  - ... and the next one(s) is started
Behavioural Language

\( B \text{ when } e_1 \text{ do } B_1, \ldots, \text{ end do } B_n \text{ end} \)

Executes \( B \) until \( e_i \) then execute \( B_i \)

\( \text{case } c_1 \text{ then } B_1, \ldots, \text{ cn then } B_n \text{ end} \)

Executes \( B_i \) if \( c_i \) holds

\( B_1 \mid B_2 \)

Executes either \( B_1 \) or \( B_2 \)

\( B_1 \mid | B_2 \)

Executes \( B_1 \) and \( B_2 \) in parallel

\( \text{every } e \text{ do } B \text{ done} \)

\( \text{do } B \text{ every } e \)

Executes \( B \) at every occurrence of \( e \)
Behaviours in Znn.com

```plaintext
compont AppServer {
  ...
  behaviour {
    do
    text | img-ld | img-hd
    every (oload or uload)
  }
}

compont Znn {
  ...
  behaviour {
    do
    conf1 when oload do
    do
      conf2 when oload do (conf3 when uload do emit-e2 end),
      uload do emit-e1
      end
every e2
    end
every e1
  }
}
```
Properties

- Implicit
  - Resource constraints (capacity)
  - e.g., never go to a state where the overall resource consumption exceeds 10 machines

- Explicit
  - Explicit constraints on components/composites’ attributes
  - Temporal constraints on configurations
    - precedes/succeeds, between, during
  - Optimization
Properties

Example of temporal constraints

```java
component AppServer {
    property {
        img-ld succeeds text
    }
    property {
        img-ld succeeds img-hd
    }
}
```

Example of constraints on attributes

```java
component Application {
    property {
        znn-politics.overallQuality >= 0.5
    }
    property {
        maximize znn-soccer.overallQuality
    }
}
```
3

CONTROLLING RECONFIGURATIONS
1

HEPTAGON/BZR
Synchronous Reactive Languages based on FSA

- Basic formalism for Discrete Control Theory

idle

wait

active

\( \text{act}=false \)

\( \text{act}=true \)

\( r \land \neg c \)

\( r \land c \)

\( s \)

\( c \)

\( \neg s \)

\( \text{delayable}(r,c,e) = \text{act},s \)

node two tasks \((r_1,c_1,e_1,r_2,c_2,e_2:\text{bool})\):

\((\text{act}_1,\text{act}_2,s_1,s_2:\text{bool})\)

let \((\text{act}_1,s_1) = \text{delayable}(r_1,c_1,e_1)\);

\((\text{act}_2,s_2) = \text{delayable}(r_2,c_2,e_2)\)
Synchronous Reactive Languages based on FSA

- Basic formalism for Discrete Control Theory
- Support for spec. languages, verification and code generation
Synchronous Reactive Languages based on FSA

- Basic formalism for Discrete Control Theory
- Support for spec. languages, verification and code generation
- General execution scheme: at each reaction a step is performed
  - taking input flows, computing transitions, updating states, triggering actions, emitting output flows
Synchronous Reactive Languages based on FSA

- Basic formalism for Discrete Control Theory
- Support for spec. languages, verification and code generation
- General execution scheme: at each reaction a step is performed
  - taking input flows, computing transitions, updating states, triggering actions, emitting output flows

\[ \text{delayable}(r,c,e) = \text{act}, s \]

- Idle
- Wait
- Active

- \( r \land \neg c \) → Act=false
- \( e \) → Act=false
- \( r \land c/s \) → Act=true
- \( c/s \) → Act=false

High-level Language Support for the Control of Reconfigurations in Component Based Architectures
Synchronous Reactive Languages based on FSA

- Basic formalism for Discrete Control Theory
- Support for spec. languages, verification and code generation
- General execution scheme: at each reaction a step is performed
  - taking input flows, computing transitions, updating states, triggering actions, emitting output flows

```
delayable(r,c,e)=act,s
```

```
node twotasks(r1,r2,c1,c2,e1,e2:bool):
  (act1,act2,s1,s2:bool)
  let
    (act1,s1)=delayable(r1,c1,e1);
    (act2,s2)=delayable(r2,c2,e2)
  tel
```
Discrete controller synthesis: principle

<table>
<thead>
<tr>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enforcing</strong> a temporal property ( \Phi ) on a system (on which ( \Phi ) does not a priori hold)</td>
</tr>
</tbody>
</table>
Discrete controller synthesis: principle

Goal

**Enforcing** a temporal property $\Phi$ on a system (on which $\Phi$ does not a priori hold)

Principle (on implicit equational representation)

- **State**: memory
- **Trans**: transition function
- **Out**: output function

![Diagram of the controller synthesis process](image)
**Discrete controller synthesis: principle**

**Goal**

*Enforcing a temporal property* \( \Phi \) *on a system (on which* \( \Phi \) *does not a priori hold)*

**Principle (on implicit equational representation)**

- **State** memory
- **Trans** transition function
- **Out** output function

- Partition of inputs into controllable \( (Y^c) \) and uncontrollable \( (Y^u) \) inputs
Discrete controller synthesis: principle

**Goal**

**Enforcing** a temporal property $\Phi$ on a system (on which $\Phi$ does not a priori hold)

**Principle (on implicit equational representation)**

- $State$ memory
- $Trans$ transition function
- $Out$ output function

- Partition of inputs into controllable ($Y^c$) and uncontrollable ($Y^u$) inputs

- Computation of a controller such as the controlled system satisfies $\Phi$
### BZR: contracts and DCS

```plaintext
node twotasks(r1,r2,e1,e2:bool):(a1,a2,s1,s2:bool)
enforce not(a1 and a2)
with (c1,c2)
let
  (a1,s1)=delayable(r1,c1,e1);
  (a2,s2)=delayable(r2,c2,e2)
tel
```

- Built on top of nodes in heptagon (M. Pouzet e.a.)
**BZR: contracts and DCS**

```plaintext
node twotasks(r1,r2,e1,e2:bool):(a1,a2,s1,s2:bool)
enforce not(a1 and a2)
with (c1,c2)
let
    (a1,s1)=delayable(r1,c1,e1);
    (a2,s2)=delayable(r2,c2,e2)
tel
```

- Built on top of nodes in heptagon (M. Pouzet e.a.)
- To each contract, associate controllable additional variables
BZR: contracts and DCS

```plaintext
node twotasks(r1,r2,e1,e2:bool):(a1,a2,s1,s2:bool)
enforce not(a1 and a2)
with (c1,c2)
let
    (a1,s1)=delayable(r1,c1,e1);
    (a2,s2)=delayable(r2,c2,e2)
tel
```

- Built on top of nodes in heptagon (M. Pouzet e.a.)
- To each contract, associate controllable additional variables
- Compute a local controller (Sigali, H. Marchand)
2

TRANSLATING CTRL-F INTO BZR
Ctrl-F Compilation Chain

Verification/Control

Synchronous Program

Discrete Control Synthesis

High-level Language Support for the Control of Reconfigurations in Component Based Architectures
Ctrl-F Compilation Chain

Ctrl-F Autonomic Manager
Modeling Components/Composites

\[ req_c = \bigvee_{req_{conf} \in RConf^c} req_{conf} \]
where \( RConf^c \) is the set request signals for configs component \( c \) belongs to.
Modeling Behaviours

\[ B \]

\[ e_1 / \text{req}_b'_{1}, \text{end}_b \]

\[ B'_{1} \]

\[ e_2 \land \neg e_1 / \text{req}_b'_{2}, \text{end}_b \]

\[ B'_{2} \]

\[ e_n \land \neg (\lor_{i=1}^{n-1} e_i) / \text{req}_b'_{n}, \text{end}_b \]

\[ B'_{n} \]

\[ B \text{ when } e_1 \text{ do } B'_{1}, \text{ e}_2 \text{ do } B'_{2}, \ldots, e_n \text{ when } B'_{n} \text{ end} \]
Modeling Behaviours

\[\begin{align*}
& e_1 / req\_b_1', \ end\_b \\
& e_2 \land \neg e_1 / req\_b_2', \ end\_b \\
& e_n \land \neg (\lor_{i=1}^{n-1} e_i) / req\_b_n', \ end\_b
\end{align*}\]

\[B \text{ when } e_1 \text{ do } B_1', \ e_2 \text{ do } B_2', \ldots, e_n \text{ when } B_n' \text{ end}\]

\[\begin{align*}
& W \xrightarrow{s/ \text{req}\_b} B \\
& s/ \text{done}
\end{align*}\]

High-level Language Support for the Control of Reconfigurations in Component Based Architectures
Modeling Behaviours

\[ B \text{ when } e_1 \text{ do } B_1', \text{ end }_b \]
\[ e_2 \wedge \neg e_1 / \text{ req }_b' \]
\[ e_n \wedge \neg (\bigvee_{i=1}^{n-1} e_i) / \text{ req }_b', \text{ end }_b \]

\[ B \text{ when } e_1 \text{ do } B_1', \text{ end }_b \]
\[ e_2 \wedge \neg e_1 / \text{ req }_b' \]
\[ e_n \wedge \neg (\bigvee_{i=1}^{n-1} e_i) / \text{ req }_b', \text{ end }_b \]

\[ B \text{ when } e_1 \text{ do } B_1', \text{ end }_b \]
\[ e_2 \wedge \neg e_1 / \text{ req }_b' \]
\[ e_n \wedge \neg (\bigvee_{i=1}^{n-1} e_i) / \text{ req }_b', \text{ end }_b \]

\[ \text{ case } c_1 \text{ then } B_1'; c_2 \text{ then } B_2'; \ldots; c_n \text{ then } B_n' \]
\[ B_1' \mid B_2' \mid \cdots \mid B_n' \]

High-level Language Support for the Control of Reconfigurations in Component Based Architectures
Modeling Behaviours

\[B \text{ when } e_1 \text{ do } B'_1, \text{ end } B \]
\[e_2 \land \neg e_1 / \text{ req } b'_2, \text{ end } b \]
\[B'_2 \]
\[e_n \land \neg (\lor_{i=1}^{n-1} e_i) / \text{ req } b'_n, \text{ end } b \]
\[B'_n \]

\[c_1 / \text{ req } b'_1, \text{ end } b \]
\[c_2 \land \neg c_1 / \text{ req } b'_2, \text{ end } b \]
\[c_n \land \neg (\lor_{i=1}^{n-1} c_i) / \text{ req } b'_n, \text{ end } b \]
\[B'_1 \text{ | } B'_2 \text{ | } \cdots \text{ | } B'_n \]

Case \(c_1\) then \(B'_1\); \(c_2\) then \(B'_2\); \ldots; \(c_n\) then \(B'_n\)

\[
\begin{align*}
B'_1 \| B'_2 \| \cdots \| B'_n
\end{align*}
\]
Modeling Properties

- Constraints on attributes: direct translation into Heptagon/BZR equations/contracts
Modeling Properties

- Constraints on attributes: direct translation into Heptagon/BZR equations/contracts
- Temporal Constraints: observers

\[ \text{img-ld between (text, img-hd)} \]

- \( s_{ld}, s_t \): stop signals for img-ld and text
- \( a_{ld}, a_{hd} \): start signals for img-ld and img-hd
Modeling Properties

- Constraints on attributes: direct translation into Heptagon/BZR equations/contracts
- Temporal Constraints: observers

\[
\begin{align*}
\text{img-ld between (text,img-hd)} \\
\text{s}_{ld}, s_{t} &: \text{stop signals for img-ld and text} \\
\text{a}_{ld}, a_{hd} &: \text{start signals for img-ld and img-hd} \\
\text{Synthesis objective :} \\
&\quad \circ \text{invariance of the state set deprived of } E
\end{align*}
\]
4

FINAL REMARKS
Summary

• DSL for describing reconfigurations in Component-based Architectures

• Extension of classic ADLs with high-level constructs to describe
  ◦ Configurations (modes and assemblies)
  ◦ Reconfiguration with a set of composable behavioural operators
  ◦ Properties to be enforced all over the execution

• Translation into synchronous reactive models for verification and control purposes
Current and Future Work

- Translation from Ctrl-F to Heptagon/BZR
Current and Future Work

- Translation from Ctrl-F to Heptagon/BZR
- Modularity (of controllers)
  - Modular compilation and distributed controllers
- Mechanisms for the coordination of controllers
- Import existing architecture descriptions
  - Instead of rewriting the whole architecture in Ctrl-F
  - Focus on the configurations, reconfigurations and properties
- More use cases
Current and Future Work

- Translation from Ctrl-F to Heptagon/BZR
- Modularity (of controllers)
  - Modular compilation and distributed controllers
- Mechanisms for the coordination of controllers
Current and Future Work

- Translation from Ctrl-F to Heptagon/BZR
- Modularity (of controllers)
  - Modular compilation and distributed controllers
- Mechanisms for the coordination of controllers
- Import existing architecture descriptions
  - Instead of rewriting the whole architecture in Ctrl-F
  - Focus on the configurations, reconfigurations and properties
Current and Future Work

- Translation from Ctrl-F to Heptagon/BZR
- Modularity (of controllers)
  - Modular compilation and distributed controllers
- Mechanisms for the coordination of controllers
- Import existing architecture descriptions
  - Instead of rewriting the whole architecture in Ctrl-F
  - Focus on the configurations, reconfigurations and properties
- More use cases
Thank you

Frederico ALVARES\textsuperscript{1}, Eric RUTTEN\textsuperscript{1} and Lionel SEINTURIER\textsuperscript{2}

Inria Grenoble - Ctrl-a Team\textsuperscript{1} / Lille - Spirals\textsuperscript{2}