Model Driven Context Aware Reactive Applications

Tony Clark (with Dean Kramer and Samia Oussena, UWL)

School of Engineering and Information Sciences, Middlesex University

October 5, 2011
Context aware applications are: intelligent applications that can monitor the user’s context and, in case of changes in this context, consequently adapt their behaviour in order to satisfy the user’s current needs or anticipate the user’s intentions.


Reactive applications must deal with events that are received from their environment and react accordingly.

The Rise of CARA
The Rise of CARA
The Rise of CARA
Model Driven Development

- Use models as the primary system artifact.
- Focus on key aspects of systems.
- Use standards to support interoperability and tools.
- Use meta-modelling to construct domain-specific abstractions.
- Analyse, verify and validate the models.
- Generate (parts of) the system from models.
Knowledge Connext and KTP

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
Knowledge Connext and KTP

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
MDD should be available for CARA applications - right?
MDD should be available for CARA applications - right?
Some attempts:
MDD should be available for CARA applications - right?

Some attempts:
- Complex or tool specific.
MDD should be available for CARA applications - right?

Some attempts:
- Complex or tool specific.
- Incomplete, lacks execution.
Problem and Proposal

- MDD should be available for CARA applications - right?
- Some attempts:
  - Complex or tool specific.
  - Incomplete, lacks execution.
- Can we design a simple, universal MDD solution?
CARA Overview

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
Buddy

- Mobile phones have address lists.

Buddy

- Mobile phones have address lists.
- A phone is always in contact with its network provider.

Buddy

- Mobile phones have address lists.
- A phone is always in contact with its network provider.
- Each phone has a unique address, e.g. tony@widget.org.

Buddy

- Mobile phones have address lists.
- A phone is always in contact with its network provider.
- Each phone has a unique address, e.g. tony@widget.org.
- Each phone maintains a database of contacts.

Buddy

- Mobile phones have address lists.
- A phone is always in contact with its network provider.
- Each phone has a unique address, e.g. tony@widget.org.
- Each phone maintains a database of contacts.
- Users want to know about contacts in their database that are co-located.

Buddy

- Mobile phones have address lists.
- A phone is always in contact with its network provider.
- Each phone has a unique address, e.g. tony@widget.org.
- Each phone maintains a database of contacts.
- Users want to know about contacts in their database that are co-located.
- When one phone moves into the vicinity of the other then both phones are told of the availability of the other in terms of the contact address.

Buddy

- Mobile phones have address lists.
- A phone is always in contact with its network provider.
- Each phone has a unique address, e.g. tony@widget.org.
- Each phone maintains a database of contacts.
- Users want to know about contacts in their database that are co-located.
- When one phone moves into the vicinity of the other then both phones are told of the availability of the other in terms of the contact address.
- If the address is in the user’s database then the phone flashes the contact.

Tony’s Phone
Tony Knows Sally

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
Context Aware

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
Domain Analysis

**GUI Trees**  GUI elements are organized in trees.

**Events**  Elements generate events and listeners handle them.

**Platform**  Each platform provides a different collection of event generators.

**Object-Orientiation**  Implementation technologies are mostly OO.

**Transitional Behaviour**  Application execute by performing state transitions.

**Data Persistence**  Applications manage simple databases.

**Static Typing**  Generally lacked by most implementation technologies for platforms.
<<external>> Application components supplied by platform.

<<widget>> User defined components, tree-shaped.

Containership Associations, black-diamond.

Root Root containers define application states.

<<event>> Generated by the platform, context events.

<<handler>> Process events.

Machines State transitions.
Single Button Model

context Push inv:
  label = 'PUSHME'

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
**Toggle Button Model**

**context** Main1 **inv:**

\[\text{display}.\text{label} = 'PUSHME'\]

**context** Main2 **inv:**

\[\text{display}.\text{label} = 'PUSHED'\]
Widget Calculus: Key Features

Take a standard $\lambda$-calculus and add:

- **widgets** tree-structured records that generate and handle events.
- **externals** each platform provides some built-in widgets.
- **commands** change the state of the world.
- **methods** each external provides some built-in commands.
- **cycle** eval, command, wait

CARA models translate onto the widget-calculus. The calculus provides the action-language for the models.
### A Button (1)

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0&lt;-push(0)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
main = 
  widget self (screen(50,50,50,50,push)) {
    move(x,y) = do { return self }
  };
push = 
  widget self (button('PUSHME')) {
    push(i) = do { return self }
  }
```
A Button (2)

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td>0&lt;-&gt;push(0)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
<widget self (screen(50,50,50,50, widget self (button('PUSHME')) { push(i) = do { return self } }) { move(x,y) = do { return self } }

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
```
A Button (3)

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>wait</th>
</tr>
</thead>
</table>
| 0<-push(0) | 0 1 | widget(3) self (screen(2,50,50,50,50,50,50),
           | 2 3 | widget(1) self (button(0,'PUSHME')) {
           |     | push(i) = do { return self }
           |     | } {
           |     | move(x,y) = do { return self }
<pre><code>       |     | } |
</code></pre>
<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>eval</th>
</tr>
</thead>
</table>
|        | 0 1 | \texttt{widget}(3) self (screen(2,50,50,50,50,50,}\texttt{widget}(1) self (button(0,’PUSHME’)) {
|       | 2 3 | push(i) = \texttt{do} \{ \texttt{return} self \}

move(x,y) = \texttt{do} \{ \texttt{return} self \} \}

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
## A Button (5)

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1</td>
<td>widget(3) self (screen(2, 50, 50, 50, 50, 50,</td>
</tr>
<tr>
<td></td>
<td>2 3</td>
<td>&lt;do {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>return widget(1) self (button(0, 'PUSHME'))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>push(i) = do { return self }</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>move(x, y) = do { return self }</td>
</tr>
</tbody>
</table>

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
## A Button (6)

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>wait</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>widget(3) self (screen(2,50,50,50,50,50,</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>widget(1) self (button(0,'PUSHME')) {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>push(i) = do { return self }</td>
</tr>
<tr>
<td></td>
<td></td>
<td>)} {</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>move(x,y) = do { return self }</td>
</tr>
</tbody>
</table>
|        | 3   | }

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
### Toggle (1)

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>eval</th>
</tr>
</thead>
</table>
| 0<-push(0) |         | main = 
                  |                | widget  self (screen(50,50,50,50,push)) {
                  |                | move(x,y) = do { return self } 
                  |                | }; 
                  |                | push = 
                  |                | widget (button(‘PUSHME’)) {
                  |                | push(i) = do { return pushed } 
                  |                | }; 
                  |                | pushed = 
                  |                | widget (button(‘PUSHED’)) {
                  |                | push(i) = do { return push } 
                  |                | }
### Toggle (2)

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>command</th>
</tr>
</thead>
</table>
| 0<-push(0) |    | `<widget self (screen(50,50,50,50,50,50)) {
                push(i) = do {
                    return widget (button('PUSHME')) {
                        push(i) = do {
                            return widget (button('PUSHED')) {
                                push(i) = do {
                                    return push |
                                }
                            }
                        }
                    }
                }
            } |
## Toggle (3)

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>0&lt;-push(0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1&lt;-push(1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

```javascript
widget(5) self (screen(4,50,50,50,50,50,
  widget(3) (button(0, 'PUSHME')) {
    push(i) = do {
      return widget(2) (button(1, 'PUSHED')) {
        push(i) = do { return push }
      }
    }
  }
})

move(x,y) = do { return self }
```

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
Toggle (4)

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;-push(1)</td>
<td>0</td>
<td>widget(5) self (screen(4,50,50,50,50,50, \n widget(3) (button(0,'PUSHME')) { \n push(i) = do { \n     return widget(2) (button(1,'PUSHED')) { \n         push(i) = do { \n             return push \n         } \n     } \n } \n move(x,y) = do { \n     return self \n } \n}}</td>
</tr>
</tbody>
</table>
| 1          | 1   | \n| 2          | 2   | \n| 3          | 3   | \n| 4          | 4   | \n| 5          | 5   | \n
Tony Clark (with Dean Kramer and Samia Oussena, UWL)
Tony Clark (with Dean Kramer and Samia Oussena, UWL)
**Toggle (6)**

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;-push(1)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
widget(5) self (screen(4,50,50,50,50,50,
  widget(2) (button(1,'PUSHED')) {
    push(i) = do { return push }
  })) {
  move(x,y) = do { return self }
}
```

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
### Toggle (7)

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
widget(5) self (screen(4,50,50,50,50,50,
<widget(2) (button(1,'PUSHED'))) {
    push(i) = do { return push }
}.push(1)>)) {
    move(x,y) = do { return self }
```
### Toggle (8)

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>widget(5) self (screen(4,50,50,50,50,50, <code>&lt;do { return push }&gt;</code>)) {</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>move(x,y) = do { return self }</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Toggle (9)

<table>
<thead>
<tr>
<th>events</th>
<th>ids</th>
<th>wait</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
widget(5) self (screen(4,50,50,50,50,50,
widget(3) (button(0,'PUSHME')) { 
    push(i) = do { 
        return widget(2) (button(1,'PUSHED')) { 
            push(i) = do { return push } 
        } 
    } ) 
move(x,y) = do { return self }
```
Commands can:

- Create widgets.
- Create, access or update variables.
- Read or write files and databases.
- Access local context, e.g. orientation, battery charge, light levels.
- Connect to servers.
- Write to servers.
- Send emails.
Contents

1 CARA Applications
   • Definitions
   • Research
   • Case Study

2 Modelling
   • Diagrams
   • Widget Calculus

3 Case Study Model

4 Other Issues
   • Types
   • Implementation
Add New Contacts

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
Contents

1 CARA Applications
   - Definitions
   - Research
   - Case Study

2 Modelling
   - Diagrams
   - Widget Calculus

3 Case Study Model

4 Other Issues
   - Types
   - Implementation
Widget Types

Standard functional types:

- **Atoms** int, str, bool
- **Functions** $\alpha \rightarrow \beta$
- **Lists** $[\alpha]$
- **Definitions** `type name = \alpha`

Extra types:

- **Widget** `Widget(\alpha){x : \beta;...}
- **Commands** `<\alpha>
- **Events** *
Simple Button (with types)

external widget.Screen;
external widget.Button;

type Push = rec B.Widget(Button) { push:(int)-><B> }

type Main = rec M.Widget(Screen[<Push>]){ move:(int,int)-><M> }

rec val main:<Main> =
  fold[<Main>]
  widget self:Main (screen[<Push>](50,50,50,50,push)) {
    move(x:int,y:int):<Main> = do { return self }
  };

val push:<Push> =
  fold[<Push>]
  widget self:Push (button('PUSHME')) {
    push(i:int):<Push> = do { return self }
  }

Tony Clark (with Dean Kramer and Samia Oussena, UWL)
Java

- Java parser for language.
- Java type-checker for Widget.
- Java interpreter for Widget.
- Swing-based external library.
- Phone 'platform' in Swing.
Next Steps

- Implement the profile using standard UML tools.
- Link CARA models and Widget calculus via model transformations.
- More external libraries (Swing, HTML, Android).
- Simplify types.