ActiveContext

Erwann Wernli
Outline

• Why dyn. software update is challenging (5’)
• Our approach (5’)
• Current stage of the research (5’)

About safety

• Let’s consider the evolution
About safety

- How can we migrate the state?
- What happens if old running code accesses field html?
- What if an execution mixes old and new code?
Trade-offs

practical

safe

timely
Trade-offs

- Run-time errors
- Dynamic C++
- Smalltalk
- Gemstone
- POLUS
- Boyapati
- JVolve
- Fickle
- HotSwap
- Ginseng
- Fickle\text{MT}

- practical
- timely
- safe

- Delay update
- Impose constraints
Our approach

A context is always version-consistent

```julia
html := aPage html.
newCtx := NewContext ancesor: oldCtx.
newCtx do: [ aPage header ,
            aPage body ].
```
Bi-directional transformation

Shared object has one identity but two states

Solution:
- State transfer
- Safe concurrency
- Reflective extension
1. transformFromAncestor: id
2. | cls html body header |
3. cls := ancestor readClassFor: id.
4. ( cls = Page ) ifTrue: [
5.    html := ancestor readField: 'html' for: id.
6.    html isNil ifFalse: [
7.       body:= html regex: '<body>(.*)</body>'.
8.       header:= html regex: '<header>(.*)</header>'.
9.    ].
10.   self writeClassFor: id value: Page2.
11.   self writeField: 'body' for: id value: body.
12.   self writeField: 'header' for: id value: header.
13.  ]
14. ( cls = AnotherClass ) ifTrue: [
15.   ...
16.  ]
17.  ...
## Design

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Instantiation</th>
<th>Concurrent read/write</th>
<th>Garbage collection</th>
<th>Expected overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eager</td>
<td>Migrate objects</td>
<td>Sync on write</td>
<td>Garbage when not used</td>
<td>High</td>
</tr>
<tr>
<td>Lazy</td>
<td>Does nothing</td>
<td>Sync on dirty read</td>
<td>Force migration prio to GC</td>
<td>Medium</td>
</tr>
</tbody>
</table>
Conclusion

• First-class contexts ensure version consistency

• Safe dynamic software update

• Working implementation

• Future work: enable lazy sync & GC
<table>
<thead>
<tr>
<th>Contextual first-class classes</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>Different versions are the same object</td>
</tr>
<tr>
<td></td>
<td>Nice with inheritance</td>
</tr>
<tr>
<td></td>
<td>Class-side variable synchronized</td>
</tr>
<tr>
<td>no</td>
<td>Different name per context</td>
</tr>
<tr>
<td></td>
<td>Not nice with inheritance</td>
</tr>
<tr>
<td></td>
<td>Class-side variable not synchronized</td>
</tr>
</tbody>
</table>
Validation

• What about daemon threads?

• Can we evolve class hierarchies this way? (interdependent classes, inheritance)

• Is the run-time overhead acceptable? (in space & time)
**Safety:** no type error, no functional inconsistency

**Time:** quickly, or at least, *eventually* install the update

**Practicality:** no extra constraint during development