A Component-based Run-time Evolution Infrastructure for Resource-constrained Embedded Systems

Juan F. Navas
Jean-Philippe Babau
Jacques Pulou
Complexity management

Increasing number of actors in software development projects, increasing number of required features, capitalization of previously acquired knowledge…
Complexity management

Evolution

Bugs corrections, software maintenance and upgrade, execution-time adaptation to unpredicted operating contexts and/or user preferences...
Complexity management

Evolution

Resource-awareness

Physical size, processing power, memory and energy consumption…
Component-based approaches

- Separation of concerns
- Components reuse
- Industrial adoption
- ...

TODO List:
- Complexity OK
- Evolution
- Resources
• **Dynamic linking of modules** [1,2]
  Granularity of evolution vs. memory occupancy and transmission costs trade-off.

• **Virtual Machine (VM) approach** [3,4]
  Permanent execution of a VM or an interpreter degrades execution performance: memory occupancy, execution times...

• **Evolution-aware design** [5,6,7]
  Components providing evolution related functionalities
research context

answers :: evolution

<table>
<thead>
<tr>
<th></th>
<th>Monolithic Design</th>
<th>Component-Based Design</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>reference</td>
<td>highly flexible</td>
</tr>
<tr>
<td><strong>RTOS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>15303</td>
<td>18246</td>
</tr>
<tr>
<td>Data</td>
<td>124</td>
<td>11916</td>
</tr>
<tr>
<td>Total</td>
<td>15427</td>
<td>30162</td>
</tr>
<tr>
<td><strong>App</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>609</td>
<td>1184</td>
</tr>
<tr>
<td>Data</td>
<td>4</td>
<td>6732</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>16040</td>
<td>30162</td>
</tr>
</tbody>
</table>

**Fig. 7.** Memory Footprint Overheads, (1) for the RTOS level – from Fig. 4, (2) for the application level – from Fig. 6, and finally (3) for the whole system.

**TODO List:**
- Complexity OK
- Evolution OK
- Resources NOK
Aggressive optimization of code generation [8]

Produces low-footprint versions of architectural component-based designs.

However, those are rigid and do not allow run-time modifications of system structure and behavior. Partially-flexible versions may be generated, but what about unpredicted evolution scenarios?
Evolution richness vs. Resources usage trade-off
What if we treat evolution aspects **off-site** (i.e. in a remote machine) ?

Alleviate the workload to be processed by the embedded processor

Meta-data reduction $\rightarrow$ memory occupancy

Richer programming environments

**We need**: 

- A coherent mirror image of running architecture
- To control the whole development process, from Design to Execution phases, and particularly the generation of “glue” code
I. Components
II. Reifications and Mirrors
III. Evolution Infrastructure
IV. Evaluation
Component models define a set of evolution points
- Add/remove components
- Modify bindings directions
- Changing attributes values
Think framework

ADL : sound

component sound {
    attribute int MIN_BEEP_TIME
    provides api.LCC as lcc
    provides bsp.api.sound as sound

    requires uCOS.api.Lock as lock
    requires uCOS.api.Delay as delay
    requires irq.api.nxtirq as irq

    content bsp.lib.sound.sound
}

IDL : bsp.api.sound

package bsp.api;
public interface sound {
    void beep_async(INT32U freq, INT32U ms, INT8U vol);
    void beep(INT32U freq, INT32U ms, INT8U vol);
}

C : bsp.lib.sound.sound.c

// @@ ServerMethod(sound, beep_async) @@
void beep_async(INT32U freq, INT32U ms, INT8U v) {
    if(ms >= MIN_BEEP_TIME) {
        *AT91C_SSC_CM = ((96109714 / 1024) / freq) + 1;
        tone_cycles = (freq * ms) / 2000 - 1;
        volume = v;
        *AT91C_SSC_IER = AT91C_SSC_ENDTX;
        *AT91C_SSC_PTCR = AT91C_PDC_TXTEN;
    }
}
**Think framework**

<table>
<thead>
<tr>
<th>Address</th>
<th>Code/Mnemonic</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>002004bc</td>
<td>.word 0x002004bc</td>
<td>2004bc:</td>
<td>..</td>
</tr>
<tr>
<td>002022e8</td>
<td>.word 0x002022e8</td>
<td>2004c0:</td>
<td>..</td>
</tr>
<tr>
<td>00202268</td>
<td>.word 0x00202268</td>
<td>2004c4:</td>
<td>..</td>
</tr>
<tr>
<td>002004bc</td>
<td>.word 0x002004bc</td>
<td>2004c8:</td>
<td>..</td>
</tr>
<tr>
<td>0020232c</td>
<td>.word 0x0020232c</td>
<td>2004cc:</td>
<td>..</td>
</tr>
<tr>
<td>00202264</td>
<td>.word 0x00202264</td>
<td>2004d0:</td>
<td>..</td>
</tr>
<tr>
<td>002008a4</td>
<td>.word 0x002008a4</td>
<td>2004d4:</td>
<td>..</td>
</tr>
<tr>
<td>002008cc</td>
<td>.word 0x002008cc</td>
<td>2004d8:</td>
<td>..</td>
</tr>
<tr>
<td>00000064</td>
<td>.word 0x00000064</td>
<td>2004dc:</td>
<td>..</td>
</tr>
</tbody>
</table>

**Execution-time**

- **lcc**
- **sound**
- **lock**
- **delay**
- **irq**

**MIN_BEEP_T**

**BC AC**
We call a component **reification** the representation of a component at a given stage of the life-cycle model: reifications are collections of data and behaviors encapsulating certain aspects of a component at a given stage of development process.
Sound component at Execution-time

Mirror component

Embedded reification

reifications & mirrors

mirrors
Mirror components are Fractal components

- They provide evolution interfaces
  - Introspection: subcomponents and Interfaces information
  - Intercession: bindings and attribute values modification

- They require services to read and write into embedded memory regions
  - byte[] read (BinAddress add, int size)
  - void write (BinAddress add, int size, byte[] data)
How mirror components map component-model entities (attributes, bindings...) with embedded memory addresses?

This requires a full control of compilation, deployment and activation stages of components life-cycle.
• Evolution Agent

A task waiting for evolution messages to be parsed and an interface to functional components’ mirrors
• Evolution Agent
• Communication Device

Media-agnostic communication component (e.g. USB)
- Evolution Agent
- Communication Device
- Memory management

Its implementation will depend on system deployment policies
Lego Mindstorms NXT
  • LCD, USB, Bluetooth, multiple sensors, motors…
  • Atmel ARM7, 32-bits, 48MHz processor

Component-based version of µC/OS-II Real-Time OS [6]
evaluation

evolution operations exec. time

<table>
<thead>
<tr>
<th>Message</th>
<th>Low load</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Median</td>
<td>Max.</td>
<td>Median</td>
</tr>
<tr>
<td>1-Bind</td>
<td>3,00</td>
<td>20,00</td>
<td>3,00</td>
</tr>
<tr>
<td>2-Stop sound</td>
<td>25,50</td>
<td>101,33</td>
<td>25,50</td>
</tr>
<tr>
<td>3-Modify</td>
<td>RAM</td>
<td>5,50</td>
<td>22,33</td>
</tr>
<tr>
<td></td>
<td>FLASH</td>
<td>6228,50</td>
<td>6920,00</td>
</tr>
<tr>
<td>4-Restart sound</td>
<td>53,83</td>
<td>171,50</td>
<td>53,83</td>
</tr>
</tbody>
</table>

**Table 3.** Time ($\mu s$) consumed in the embedded device during treatment of messages involved in setAttValue method execution.

**Figure 6.** System tasks execution during three consecutive modifications of attributes values in FLASH.

Time elapsed during Message 3 treatment: parsing of message contents, rewriting of a FLASH page and errors verification, among other operations.
<table>
<thead>
<tr>
<th>Components</th>
<th>Non-Optimized</th>
<th></th>
<th></th>
<th>Optimized</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DATA</td>
<td>TEXT</td>
<td>RAM</td>
<td>TEXT</td>
<td>DATA</td>
<td>TEXT</td>
</tr>
<tr>
<td>EFC</td>
<td>40</td>
<td>0</td>
<td>280</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Flash Memory</td>
<td>280</td>
<td>0</td>
<td>292</td>
<td>0</td>
<td>256</td>
<td>0</td>
</tr>
<tr>
<td>Ram Memory</td>
<td>8</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>Memory Manager</td>
<td>24</td>
<td>352</td>
<td>0</td>
<td>0</td>
<td>292</td>
<td>0</td>
</tr>
<tr>
<td>Evolution Agent</td>
<td>1090</td>
<td>888</td>
<td>0</td>
<td>1070</td>
<td>832</td>
<td>0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>1442</td>
<td>1260</td>
<td>572</td>
<td>1326</td>
<td>1136</td>
<td>528</td>
</tr>
</tbody>
</table>
Table 5. Case study memory footprints (in Bytes) comparison between embedded control interfaces (Ctrl Itfs) and off-line mirrors manipulations.

Figure 7. Size of evolution infrastructures for scenarios (2), (3) and (4) in Table 5. Rectangles areas are directly proportional to memory footprint overheads due to evolution features inclusion.
We vary the location of evolution-related operations treatment to fulfill resource constraints
We keep a minimal evolution infrastructure inside the device

Evolution planning and execution is done through run-time mirror architectures
Mirror components are generated at compile-time

Component-based nature of our solution eases porting to other hardware platforms
Predefined interfaces and reifications’ information exchange policies
questions?

thank you
questions ?


