One Approach to Aspect-Oriented Programming Implementation for the C Programming Language

Evgenij Novikov
PhD student
Institute for System Programming, RAS
joker@ispras.ru

Ekaterinburg, 2011
Aspect-oriented programming (1)

- Source code of a program

```c
...  
lock(res);
...
if (func())
  goto exit_after_error;
...
unlock(res);
...
exit_after_error:
...  
```
Aspect-oriented programming (2)

- Source code of a program

```plaintext
... 
lock(res);
...
if (func())
    goto exit_after_error;
...
unlock(res);
...
exit_after_error:
...
```
### Aspect-oriented programming (3)

- **Source code of a program**
  ```
  lock(res);
  ...
  if (func())
    goto exit_after_error;
  ...
  unlock(res);
  ...
  exit_after_error:
  ...
  ```

- **Aspect**
  ```
  before: call (int lock(lock_t))
  {
    lock_our();
  }
  
  before: call (int unlock(lock_t))
  {
    unlock_our();
  }
  ```

- **Join points**
Aspect-oriented programming (4)

- Source code of a program

```c
... lock(res);
... if (func())
    goto exit_after_error;
... unlock(res);
... exit_after_error:
...```

- Aspect

```c
before: call (int lock(lock_t))
{
    lock_our();
}
before: call (int unlock(lock_t))
{
    unlock_our();
}
```

- Join points
- Advices
- Pointcuts
- Advice bodies
Instrumentation in AOP

- Instrumented source code of a program
- Auxiliary routines

```c
... lock_aux(res);
...
if (func())
    goto exit_after_error;
...
unlock_aux(res);
...
exit_after_error:
...
```

```c
int lock_aux(lock_t a)
{
    lock_our();
    return lock(a);
}
int unlock_aux(lock_t a)
{
    unlock_our();
    return unlock(a);
}
```
AOP implementation

- AOP implementation depends on a programming language

- Different AOP implementations for a given programming language
  - determine their own syntax for AOP constructions
  - use different ways for source code instrumentation
Specific requirements of C

- **Preprocessing**
  - macro expansion and header files including as join points

- **Compilation**
  - a lot of “standard” join points like function calls, variable usage, …
  - specific join points concerned with pointer operations of C

- **Linking**
  - different object files to be linked shouldn’t contain the same defined symbols
Requirements of real application

- Support the standard C with all GNU extensions
- Offer a large set of AOP constructions
- Generate instrumented C source code equivalent to the original one
- Rather easy maintenance
## Related work

<table>
<thead>
<tr>
<th></th>
<th>ACC</th>
<th>InterAspect</th>
<th>SLIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Preprocessing”</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>“Compilation”</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
<tr>
<td>“Linking”</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>C with GNU extensions</td>
<td>±</td>
<td>+</td>
<td>±</td>
</tr>
<tr>
<td>C output</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Maintenance</td>
<td>-</td>
<td>±</td>
<td>+</td>
</tr>
</tbody>
</table>
### Suggested approach (1)

- **Aspect preprocessing (1st stage)**

<table>
<thead>
<tr>
<th>Initial source code</th>
<th>Aspect</th>
<th>Resulting source code</th>
</tr>
</thead>
</table>
| ...
lock(res);
...
unlock(res);
... | before: file (
  $this)
{
#include “aux.h”
int lock_flag = 0;
}
 | #include “aux.h”
int lock_flag = 0;
...
lock(res);
...
unlock(res);
... |
Suggested approach (2)

- Macro weaving (2\textsuperscript{nd} stage)

\begin{itemize}
  \item Source code
  
  \begin{verbatim}
  #define LOCK(t) ...
  #define UNLOCK(t) unlock(t)
  ...
  LOCK(res);
  ...
  UNLOCK(res);
  ...
  \end{verbatim}

  \item Aspect

  \begin{verbatim}
  around: define (LOCK(t))
  {
  lock_our()
  }
  \end{verbatim}

  \item Preprocessed source code

  \begin{verbatim}
  ...
  lock_our();
  ...
  unlock(res);
  ...
  \end{verbatim}
\end{itemize}
Suggested approach (3)

- Advice weaving (3rd stage)

Preprocessed source code

```
... 
lock(res);
...
unlock(res);
... 
```

Aspect

```
before: call ( 
    int lock 
    (lock_t))
{
    lock_our();
}
```

Resulting source code

```
... 
lock(res);
...
unlock(res);
... 
int lock_aux_1 
(loct_t a) { 
    lock_our(); 
    return lock(a); 
}
```
Suggested approach (4)

- Compilation (4th stage)

Preprocessed source code

```c
... lock(res);
... unlock(res);
... int lock_aux_1 (loct_t a) {
    lock_our();
    return lock(a);
}
```

Aspect

```
before: call (int lock (lock_t))
{
    lock_our();
}
```

Resulting source code*

```
... lock_aux_1(res);
... unlock(res);
... int lock_aux_1 (loct_t a) {
    lock_our();
    return lock(a);
}
```

*In fact object code
Suggested approach (5)

- Linking and C code generating (final stages)

### Source code*

```c
...  
lock_aux_1(res);
...
int lock_aux_1(loct_t a) {
  lock_our();
  return lock(a);
}
...
```

*In fact object code*

### Content of this binary file is not shown

### Linked file

```c
...  
lock_aux_2(res);
...
int lock_aux_2 ...
...
void lock_our() {
  check(lock_flag);
  return lock(a);
}
```

### Resulting source code**

```c
...  
lock_aux_1 (res);
...
lock_aux_2 (res);
...
int lock_aux_1 (loct_t a) {
  lock_our();
  return lock(a);
}
int lock_aux_2 ...
...
void lock_our() {
  check(lock_flag);
  return lock(a);
}
```

**In fact it looks quite unlike...**
Drivers of Linux kernel 2.6.31.6 were instrumented by means of a tool implementing the suggested approach:
- a model concerned with mutex lock/unlock rule was used
- almost all 2160 modules were processed successfully

Subsequent static verification showed that the tool behaves rather well:
- more memory is required in comparison with manual instrumentation
- generated constructions are even simpler as original ones

Generated source code is too inconvenient for static verifiers and analysis.
Conclusion

- It was developed the new approach of C AOP implementation that covers specific requirements of both the C programming language and real AOP application.

- The tool implementing the suggested approach successfully works.

- It is required a new way to generate C source code.

- More AOP constructions like C pointer operations should be supported.
Thank you!

Questions?

http://forge.ispras.ru/projects/ldv

joker@ispras.ru
if (strict_strtoul(buf, 0, &val) < 0)
    return -22;

if (val)
    pci_rescan_bus(slot->dev->bus);

fakephp.ko.linked.cbe.c

blast_must_tmp__85 = *(llvm_cbe_buf_addr);
blast_must_tmp__86 = strict_strtoul(blast_must_tmp__85, 0u, (llvm_cbe_val));
if (((signed int)blast_must_tmp__86) < (((signed int)0u)))
    goto llvm_cbe_bb;
else
    goto llvm_cbe_bbl;

llvm_cbe_bb:
    *(llvm_cbe_tmp__73) = 18446744073709551594ull;
    goto llvm_cbe_bb5;

llvm_cbe_bbl:
    blast_must_tmp__87 = *(llvm_cbe_val);
    blast_must_tmp__88 = *(llvm_cbe_slot);
    blast_must_tmp__89 = *((&blast_must_tmp__88->field1));
    if ((blast_must_tmp__87 != 0ull))
        goto llvm_cbe_bb2;
    else
        goto llvm_cbe_bb3;

llvm_cbe_bb2:
    blast_must_tmp__90 = *((&blast_must_tmp__89->field1));
    blast_must_tmp__91 = pci_rescan_bus(blast_must_tmp__90);

llvm_cbe_bb3:
...