CosmOpen: Dynamic reverse engineering on a budget

François Taïani¹, Marc-Olivier Killijian², Jean-Charles Fabre²
¹Lancaster University, UK
²LAAS-CNRS, France
(f.taiani@lancaster.ac.uk)
The best and safest method of philosophizing seems to be, first to inquire diligently into the properties of things, and to establish those properties by experiences and then to proceed more slowly to hypotheses for the explanation of them.

Isaac Newton
Lancaster Middleware Group

- Facilitate development of **new distributed systems**
  - Abstractions?
  - Mechanisms?
  - Exotic environments: MANETs, WSN, routers, …
Lancaster Middleware Group

- Facilitate development of **new distributed systems**
  - Abstractions?
  - Mechanisms?
  - Exotic environments: MANETs, WSN, routers, …

- Flip-side: study of **existing middleware**
  - Emergent structures?
  - Development practices?
  - How to ease refactoring / dissemination?
Example: one CORBA request

ORBacus Request Processing

one request,
2065 individual invocations,
over 50 C-functions and 140 C++ classes.

How to analyse this?
Outline

- **Why** reverse engineering middleware?
- Our approach: software “interferometry”
- Does it work? A case study
Outline

- Why reverse engineering middleware?
- Our approach: software “interferometry”
- Does it work? A case study
Why?

- Original goal
  Control non-determinism in industry grade middleware
  Observation of mutex activities
  Link to request life cycle

Original diagram:

- OS
- Intergiciel(s)
- Middleware services
  - (RPC, pub/sub, ...)
- Syscalls, syslibs
  - (synchronisation, memory management...)
Original goal
Control non-determinism in industry grade middleware
Observation of mutex activities
Link to request life cycle

Complex & layered system

Hard to observe and analyse

P1: Large behavioural data-set
P2: Cross-layer entangling
P1: Large behavioural data-sets

- Example: **Globus**

- **Huge piece of** software (3.9.x)
  - 123,839 lines in Java (without reused libraries)
  - (1,908,810 lines in C/C++, including reused libraries)

- **Many libraries** layered
  - XML, WSDL (Descr. Lang), WSRF (Resource Fwork)
  - Axis (SOAP), Xerces (XML Parsing), com.ibm.wsdl

- **Java: exhaustive tracing** (outside the JVM libs)
  - client: 1,544,734 local method call (sic)
  - server: 6,466,652 local method calls (sic) [+time out]
P1: Large behavioural data-sets

(globus client, 1 creation, 4 requests, 1 destruction)

Projection w.r.t.
stack depth
package (structure)

Calls

Stack Depth

AT&T
P1: Large behavioural data-sets

Far too large for manual analysis

Exhaustive tracing: high observation costs

Intractable interferences

(globus client, 1 creation, 4 requests, 1 destruction)
P2: Cross-layer entangling

```c
int main () {
    pthread_t threadN1, threadN2 ;
    pthread_create(&threadN1, NULL, dummy1, NULL) ;
    pthread_create(&threadN2, NULL, dummy2, NULL) ;
    pthread_join ( threadN1, NULL) ;
    pthread_join ( threadN2, NULL) ;
}
```
P2: Cross-layer entangling

```c
int main () {
    pthread_t threadN1, threadN2;
    pthread_create(&threadN1, NULL, dummy1, NULL);
    pthread_create(&threadN2, NULL, dummy2, NULL);
    pthread_join ( threadN1, NULL );
    pthread_join ( threadN2, NULL );
}
```

observation by tracing
P2: Cross-layer entangling

Can we reconstruct main’s behaviour?
int main () {
    pthread_t threadN1, threadN2;
    pthread_create(&threadN1, NULL, dummy1, NULL);
    pthread_create(&threadN2, NULL, dummy2, NULL);
    pthread_join(threadN1, NULL);
    pthread_join(threadN2, NULL);
};
P2: Cross-layer entangling

Much more complex than original code

Program behaviour not apparent

Reason: mixes several abstraction levels

int main()
{
  pthread_t threadN1, threadN2;
  pthread_create(&threadN1, NULL, dummy1, NULL);
  pthread_create(&threadN2, NULL, dummy2, NULL);
  pthread_join(threadN1, NULL);
  pthread_join(threadN2, NULL);
}
Outline

- Why reverse engineering middleware?
  - Our approach: software “interferrometry”
- Does it work? A case study
Approach: “interferometry”

- **Limit** observation costs: *more for less*
  - capture stack traces (more)
  - limit observation points to component boundaries (less)

- **Filter** out unneeded data: “interferometry”
  - graph manipulation script language
  - reusable filters

‘cheap’
but post-processing needed

no turbulence
but very costly
More for less

```c
#0 0x405c8f20 in send () from /lib/libpthread.so.0
#1 0x4015804a in omni::tcpConnection::Send ()
#2 0x4011364c in omni::giopStream::sendChunk ()
[..]
#8 0x400bdb0a in omniAsyncWorker::run ()
#9 0x405abbdf in omni_thread_wrapper ()
#10 0x405c2bf0 in pthread_start_thread ()
#11 0x405c2c6f in pthread_start_thread_event ()
```
Reconstructing a call tree

- Problem: stack traces are ambiguous

- Choice: “smallest” compatible call tree
  a variant of smallest prefix tree (with timestamps)
  can be characterised formally

2 identical traces
Graph manipulation filters

Provides **variables** to store intermediary results

A number of graph **operators**

- **selection**
  
  put ::pthread_create* G CREATE

- **recursive extension**
  
  forward CREATE G

- **boolean algebra**
  
  remove ::pthread_create* CREATE

- **temporal operations**
  
  fuse ::pthread_create* ::pthread_start_thread* G
Revisiting pthread

fuse ::pthread_create* ::pthread_start_thread* G 23
Revisiting pthread

‘Reconstructing’ data-flow from temporal succession
Revisiting pthread

fuse ::pthread_create* ::pthread_start_thread* G
put ::pthread_create* G CREATE
forwN 1 CREATE G
remove ::pthread_create* CREATE
remove ::pthread_start_thread* CREATE
forward CREATE G
exclude CREATE G
absPatern ::pthread_create* G
absPatern ::pthread_start_thread* G
Revisiting pthread

fuse ::pthread_create* ::pthread_start_thread* G
put ::pthread_create* G CREATE
forwN 1 CREATE G
remove ::pthread_create* CREATE
remove ::pthread_start_thread* CREATE
forward CREATE G
exclude CREATE G
absPatern ::pthread_create* G
absPatern ::pthread_start_thread* G

Generic script: applies to any pthread code
Case study

- 3 C/C++ industry-grade CORBA products
  ORBacus, omniORB, TAO

- Set up
  ~ 60 breakpoints (locks, memory, threading, callbacks)
  one ping-pong request
  1GHz Pentium III, Linux kernel 2.4, gdb 5.1-1
Case study

- 3 C/C++ industry-grade CORBA products
  ORBacus, omniORB, TAO

- Set up
  ~ 60 breakpoints (locks, memory, threading, callbacks)
  one ping-pong request
  1GHz Pentium III, Linux kernel 2.4, gdb 5.1-1

- Observation **overheads** (Orbacus)
  non-instrumented: < 1s
  fully instrumented: 1h 2m 11s
  lock tracing disabled during initialisation: 4m 53s
Outline

- Why reverse engineering middleware?
- Our approach: software “interferometry”
- Does it work? A case study
Stack capture speedup

<table>
<thead>
<tr>
<th>ORB</th>
<th>threads</th>
<th>traces</th>
<th>frames</th>
<th>invocations</th>
<th>invocations/traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORBACUS 4.1</td>
<td>8</td>
<td>658</td>
<td>9178</td>
<td>2066</td>
<td>3.13</td>
</tr>
<tr>
<td>omniORB 4</td>
<td>7</td>
<td>1828</td>
<td>16807</td>
<td>3088</td>
<td>1.68</td>
</tr>
<tr>
<td>TAO 1.2.1</td>
<td>6</td>
<td>512</td>
<td>11260</td>
<td>1352</td>
<td>2.64</td>
</tr>
</tbody>
</table>

breakpoint activation = cost
reconstructed tree = added value
ratio = speedup

(with lock tracing disabled during initialisation)
Orbacus complete graph

2065 individual invocations, over 50 C-functions and 140 C++ classes.
Result

26 individual invocations, 4 C functions, and 23 C++ classes.
Deterministic replication
link OS level ↔ app level
use reflective approach
instrument MW to realise meta-model
ref: [DSN’03] and [DSN’05]
Conclusion

- **Pragmatic** approach to dynamic reverse engineering addresses complex layered software minimises interferences by using stack traces reconstruction and graph manipulation for analysis

- **Caveat**
  reverse engineering remains a *human* activity constructed graphs provide a roadmap / guide but must be supported by other activities

- **Future**
  better analysis of help provided (*user studies*)
  better formalisation of *consistency* in graph manipulation
(Some) references

- François Taïani, Marc-Olivier Killijian, Jean-Charles Fabre, CosmOpen: Dynamic reverse-engineering on a budget. Software: Practice and Experience, to appear [contact me for most recent version]


- Jerding DF, Stasko JT, and Ball T. Visualizing interactions in program executions. Proc. of the 19th Int. Conf. on Software Engineering (ICSE’97), ACM Press, 1997; 360-370


(Some) references (cont)


- http://ftaiani.ouvaton.org/7-software/
Thank you

Questions?