Alchemy:
Transmuting Raw Code into Reusable Components

Jay Lepreau  (PI)  Alastair Reid
Matthew Flatt  (co-PI)  Leigh Stoller
Wilson Hsieh  
Jeanette Wing  CMU  Eric Eide
Mike Hibler

University of Utah

http://www.cs.utah.edu/flux/alchemy/
Alchemy

Component language and tools for low-level systems software

Not a specific architecture or framework

Example: The OSKit [SOSP97]
Characteristics of Low-Level Systems

- Lack of clear layers among components

Diagram:

- Threads
- Virtual Memory
Characteristics of Low-Level Systems

- Lack of clear layers among components
Characteristics of Low-Level Systems

- Lack of clear layers among components
Characteristics of Low-Level Systems

- Lack of clear layers among components

- Local changes shift global constraints

OneThread

single-threaded
Characteristics of Low-Level Systems

- Lack of clear layers among components

- Local changes shift global constraints

multi-threaded
Alchemy Outline

• **Language**
  ◦ define components
  ◦ link components

• **Types and constraints**
  ◦ verify linking
  ◦ infer adaptors

• **Performance**
  ◦ zero componentization overhead

• **Applications**
  ◦ The OSKit
  ◦ Linux, ...
Start with **unit** model of components [PLDI98]

\[
\begin{array}{|c|}
\hline
\text{import}_1 \ldots \text{import}_n \\
\hline
\text{definition}_1 \\
\vdots \\
\text{definition}_m \\
\hline
\text{export}_1 \ldots \text{export}_k \\
\hline
\end{array}
\]

*atomic unit shape*
Start with \textbf{unit} model of components [PLDI98]

\begin{tabular}{|l|}
\hline
\textbf{lock_t} \hspace{1em} \textbf{void} \hspace{1em} \textbf{lock(}\textbf{lock_t}\textbf{)} \hspace{1em} ... \\
\hline
\textbf{void*} \hspace{1em} \textbf{mmap(}\textbf{long }\textbf{s}\textbf{)} \hspace{1em} \{ \hspace{1em} ... \hspace{1em} \} \\
\hspace{1em} ... \\
\hline
\textbf{void*} \hspace{1em} \textbf{mmap(}\textbf{long}\textbf{)} \hspace{1em} ... \\
\hline
\end{tabular}

\textit{atomic unit example}
Language

Start with **unit** model of components [PLDI98]

```
import₁ ... importₙ

...    ...
                       ...

import
...    ...

...    ...

export

export₁ ... exportₖ
```

*compound unit shape*
Language

Start with **unit** model of components [PLDI98]

```
import \_1 ... import \_n

import
  ...
  ...
  ...

export
  ...
  ...
  export

export\_1 ... export\_k
```

*compound unit shape*
Start with **unit** model of components [PLDI98]

```
import₁ ... importₙ

import  
...
...

export

export₁ ... exportₖ
```

compound unit shape
Language

Start with **unit** model of components [PLDI98]

<table>
<thead>
<tr>
<th>Thread</th>
<th>VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>void* mmap(long)</td>
<td>void lock(lock_t)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>void lock(lock_t)</td>
<td>void* mmap(long)</td>
</tr>
</tbody>
</table>

**compound unit example**
• well-defined interfaces,
• hierarchical,
• mutual dependencies,
• multiple instances, ...
### Language

<table>
<thead>
<tr>
<th>Thread</th>
<th>VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>void* mmap(long)</td>
<td>void lock(lock_t)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>void lock(lock_t)</td>
<td>void* mmap(long)</td>
</tr>
</tbody>
</table>

... and **static**!
Usability

Unit language for C:

- Must be easy for system hackers to use
- Must provide a mechanism for deriving configurations
Alchemy Outline

- **Language**
  - define components
  - link components

- **Types and constraints**
  - verify linking
  - infer adaptors

- **Performance**
  - zero componentization overhead

- **Applications**
  - The OSKit
  - Linux, ...
Types and Constraints

Unit model provides basic type checking:

\[
\begin{array}{c}
\ldots \\
\ldots \\
\text{lock}_t \text{ lock} (\text{lock}_t) \ldots \\
\text{lock}_t \text{ lock} (\text{lock}_t) \ldots \\
\ldots \\
\ldots 
\end{array}
\]
Types and Constraints

For systems software, also need to ensure that

• Multi-threaded code doesn’t call single-threaded code
• Only one virtual memory manager is active
• Components are initialized in the correct order
• Components obey resource constraints
• and more

These are extra constraints, outside "normal" types
Constraint Checking

Reject mismatches:

\[
\text{single } f(...) \\
\text{multi } f(...)
\]
Constraint Checking

Reject mismatches:

...  
...  
**single f(...)**  

X  

**multi f(...)**  
...  
...
Constraint Propagation

Some components propagate constraints:

\[
\begin{array}{|c|}
\hline
\ldots \\
\hline
\ldots \\
\hline
\text{single } f(\ldots) \\
\hline
\end{array}
\]

\[
\begin{array}{|c|}
\hline
\alpha f(\ldots) \\
\hline
g() \{ \ldots f() \ldots \} \\
\hline
\alpha g(\ldots) \\
\hline
\end{array}
\]
Inferring Adaptor Components

Automate mismatch repairs:

\[
\begin{array}{c}
\ldots \\
\ldots \\
\text{single } f(\ldots) \\
\text{multi } f(\ldots) \\
\ldots \\
\ldots \\
\end{array}
\]
Automate mismatch repairs:

- `single f(...)`
- `single f(...) lock(lock_t) ...`
- `multi f(...)`
- `multi f(...)`
- `...`
Ordering Component Initializations

Auto-schedule component initialization:

```
Threads ———> Locks

Virtual Memory

⇒

<table>
<thead>
<tr>
<th>import₁ ... importₙ</th>
<th>init-dep₁ ... init-depₚ</th>
</tr>
</thead>
<tbody>
<tr>
<td>definition₁</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>definitionₘ</td>
<td></td>
</tr>
<tr>
<td>export₁ ... exportₖ</td>
<td></td>
</tr>
</tbody>
</table>
```
• Components used with GC must be GC-friendly

• Boundary between GC and `malloc/free` components may need adaptors
Constraint Challenges

Summary of constraint challenges:

- Extensible
- Automatable checking
- Automatable adaptation
Alchemy Outline

- **Language**
  - define components
  - link components

- **Types and constraints**
  - verify linking
  - infer adaptors

- **Performance**
  - zero componentization overhead

- **Applications**
  - The OSKit
  - Linux, ...
Performance

Performance goals:

+ Make aggressive componentization practical

- Speed up existing code
Linking and Optimization

... 

... 

lock(lock_t) { } 
unlock(lock_t) { } 

lock unlock 

lock unlock 

f() { 
    lock(l); 
    do work 
    unlock(l); 
} 

... 

...
Linking and Optimization

...  

...  
lock(lock_t) { }  
unlock(lock_t) { }  
lock unlock  
lock unlock  
f() {  
  lock(l);  
do work  
  unlock(l);  
}

...  

...  

lock(lock_t) { }  
unlock(lock_t) { }  
f() {  
  lock(l);  
do work  
  unlock(l);  
}

...  

...
Linking and Optimization

```c
lock(lock_t) { }
unlock(lock_t) { }
f() {
    lock(l);
    do work
    unlock(l);
}

...
### Linking and Optimization

```c
lock(lock_t) { }
unlock(lock_t) { }
f() {
    lock(l);
    do work
    do work
    unlock(l);
}
```

= 

```c
f() {
    do work
}
```

...
Alchemy Outline

- **Language**
  - define components
  - link components

- **Types and constraints**
  - verify linking
  - infer adaptors

- **Performance**
  - zero componentization overhead

- **Applications**
  - The OSKit
  - Linux, ...
The OSKit

The OSKit: a set of components for systems software

• Extensive use of legacy code (e.g., Linux device drivers)

• 500 components when transmuted (estimate)

• Component sizes vary
  ○ Large: TCP/IP from FreeBSD, 18,000 lines
  ○ Small: Serial console, 200 lines
  ○ Smaller: Simple adaptors
The OSKit

The OSKit: a set of components for systems software

- **ld works, but not well**
  - Especially problematic for a new user

- **COM works, but not well**
  - Also problematic for a new user
  - Late discovery of errors
  - Reference counting difficult to maintain

⇒ Alchemy
The Transmuted OSKit

Transmuted OSKit = substrate for further PCES research

- Infrastructure for systems builders
- Bridge for more powerful analyses on systems code
Linux

- Linux
- Current OSKit
- Expanded OSKit
  - Regular Linux
  - Linux Lite
  - Linux RT
Alchemy

Component language and tools for low-level systems:

- Static linking makes analysis and optimization tractable
- Constraints for global consistency checking
- Result: robust components, bridge to further analyses