Overview of lecture series

♦ Background
♦ Introduction to Szumo
♦ Case study
♦ Semantic and implementation details
♦ Related concurrency models
♦ Ongoing and future work
Goal

Validate the efficacy of Szumo to current software engineering practice

- Is Szumo sufficiently expressive to handle designs of complex systems?
- Is Szumo effective in localizing synchronization concerns?
- Can Szumo support use of modern OO design techniques (e.g. structural modeling notations, design patterns)?
Approach

♦ Designed and implemented a multi-threaded OO web server
  – Modeled after Apache
  – Exemplar of realistic multi-threaded OO design

♦ Tested extensibility/maintainability by adding different types of features
  – Authentication
  – Dynamic content generation
  – Load balancing
Overview of lecture series

♦ Background
♦ Introduction to Szumo
♦ Case study
  – A contract-aware web server
  – Extension tasks
♦ Semantic and implementation details
♦ Related concurrency models
♦ Ongoing and future work
Aside on a change in terminology

♦ Yesterday, I used processes and threads as synonyms
♦ Today, I need to distinguish between them
  – **Process**: program running in its own data space (OS process)
  – **Thread**: locus of control within a process (lightweight “process”)
Web server design: Motivating Example

- Authentication
- Content Generation
- Finalization

- Encryption Libraries
- Content Handlers
- Logging & Statistics
Web server design: Motivating Example

- web server main
  - dispatches
  - Authentication
  - Content Generation
  - Finalization
    - Encryption Libraries
    - Content Handlers
    - Logging & Statistics
Web server design: Motivating Example
Web server design: Motivating Example
Web server design: Motivating Example
Web server design: Motivating Example

1. **r1**: request_handler
   - **r1**: request_handler
   - **a1**: authenticator
   - **c1**: content_generator
   - **f1**: finalizer
   - **r2**: request_handler
   - **a2**: authenticator
   - **c2**: content_generator
   - **f2**: finalizer
   - **c**: content_handler
   - **l**: logger
   - **crypt**: crypt

2. **web_dispatcher**: web_dispatcher
   - **req_handler_pool**: req_handler_pool

Laser 2005 - Summer School on Software Engineering
Web server design: Motivating Example

Web server components:
- Web dispatcher
- Request handler pool
- Authenticator
- Content generator
- Finalizer
- Crypt
- Content handler
- Logger
- Request handler

Diagram:
- r1: request_handler
- a1: authenticator
- c1: content_generator
- f1: finalizer
- r2: request_handler
- a2: authenticator
- c2: content_generator
- f2: finalizer
- r2: request_handler
- : logger
- : content_handler
- : crypt
- : req_handler_pool
- : web_dispatcher

Laser 2005 - Summer School on Software Engineering
Web server design: Motivating Example
Web server design: Motivating Example

- `f1: finalizer`
- `c1: content_generator`
- `a1: authenticator`
- `r1: request_handler` (process root)
- `f2: finalizer`
- `c2: content_generator`
- `a2: authenticator`
- `r2: request_handler` (process root)
- `: req_handler_pool`
- `: web_dispatcher` (process root)
- `: logger`
- `: content_handler`
- `: crypt`
Web server design: Motivating Example

- `web_dispatcher`<process root>`
- `req_handler_pool`
- `r1: request_handler`<process root>`
- `a1: authenticator`
- `c1: content_generator`
- `f1: finalizer`
- `a2: authenticator`
- `c2: content_generator`
- `f2: finalizer`
- `r2: request_handler`<process root>`
- `crypt`
- `content_handler`
- `logger`
Web server design: Motivating Example

- f1: finalizer
- c1: content_generator
- a1: authenticator
- logger
- content_handler
- crypt

- f2: finalizer
- c2: content_generator
- a2: authenticator

- r1: request_handler
- r2: request_handler

- : web_dispatcher
- : req_handler_pool
- : crypt
- : content_handler
Web server design: Motivating Example
Basic web-server design
Overview of lecture series

♦ Background
♦ Introduction to Szumo
♦ Case study
  – A contract-aware web server
  – Extension tasks
♦ Semantic and implementation details
♦ Related concurrency models
♦ Ongoing and future work
Extension tasks

♦ Task 1: Authentication
  – Added authentication capability
  – Safely integrated library functions

♦ Task 2: Dynamic content generation
  – Provided for user scripting in the same memory space as the web server

♦ Task 3: Load balancing
  – Modified web server to dynamically adapt the number of content generators as a function of load
Extension tasks

♦ Task 1: Authentication
  – Added authentication capability
  – Safely integrated library functions

♦ Task 2: Dynamic content generation
  – Provided for user scripting in the same memory space as the web server

♦ Task 3: Load balancing
  – Modified web server to dynamically adapt the number of content generators as a function of load
Task 1: Authentication

Purpose: Provide authentication capability

- **PWD_VALIDATOR**: invokes POSIX library’s “crypt” function to validate password
- **DB_VALIDATOR**: uses a third party database engine (which also invokes “crypt”) to authenticate user

Convention: We show existing classes and associations in blue and new ones in black
Problem:
C Libraries need not be thread safe

♦ Example:
crypt(…)
  – May use static buffers to encrypt the key
  – Returns a pointer to a static buffer containing the result string

C Libraries

<table>
<thead>
<tr>
<th>(static vars)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool auth(…</td>
</tr>
<tr>
<td>.. . .crypt(…)...</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>db_rec* query(…)</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>….</td>
</tr>
<tr>
<td>:</td>
</tr>
<tr>
<td>(static vars)</td>
</tr>
<tr>
<td>char * crypt(…) {</td>
</tr>
<tr>
<td>….</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
Problem: C Libraries are not *contract aware*

- Cannot negotiate for synchronization units
- Cannot be negotiated for
- Are not protected against data races

Partial solution: 

*External synchronization units*

- Szumo Eiffel wrapper for a *library module*
- May be instantiated only once
- Other units access library functions only through the singleton wrapper unit
Design of VALIDATOR extension
Processes now correctly negotiate for exclusive access to library modules (via external units)
Processes now correctly negotiate for exclusive access to library modules (via external units)

But …
Remaining Problems

♦ C modules can access one another directly
  – E.g., calls to “crypt” are prevalent in other C libraries
    • SQL implementations, WWW password authentication
  – As these calls bypass realm membership checks, this would make the application vulnerable to illegal accesses

♦ References to module-local data can escape from the module and vice versa
Example: Illegal access vulnerability

If a designer does not know how the database engine implements authentication, she would not know to document the dependence between the external units, DB_ENG and CRYPT.
This would allow an illegal access to crypt’s static buffer through db_engine’s call to auth(…)

C Libraries

(static vars)

bool auth(…
    … crypt(…)
}

(char * crypt(…) {  
    …
})

(char * crypt(…) {
    …
})
Solution: Extend contract-awareness to library modules

♦ Two Levels of Wrapping
  – Top layer: External synchronization unit
    • Translates Eiffel calls and data into C equivalents
    • Provides contract support
  – Middle layer: C *decorator module* containing a wrapper function for each function in the C module
    • Checks for realm membership
    • Copies arguments and return values to prevent pointers from escaping
  – Bottom layer: Original C module

♦ Linker redirects all external calls to library functions to the corresponding wrapper function in the decorator module
Laser 2005 - Summer School on Software Engineering

```
Laser 2005 - Summer School on Software Engineering

C Decorator Modules

bool __wrap_auth(...) {
  . . .__real_auth(...) . . .
}

crypt(…) {
  . . .
}

C Library Modules

bool auth(...) {
  . . .crypt(...) . . .
}

db_rec * query(...) {
  . . .
}
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**C Decorator Modules**

```
C Decorator Modules

(char * __wrap_crypt(…)) {
  . . .__real_crypt(...) . . .
}

db_rec* __wrap_query(…)
  . . .__real_query(...) . . .
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**C Library Modules**

```
C Library Modules

(char * __wrap_crypt(…)) {
  . . .
}
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```

---

**Szumo**

```
Szumo

<< external >>

DB_ENG

auth(…): BOOL
query(…): DB_REC

{ crypt }

crypt

crypt
```
Legal access of crypt through db_eng

C Decorator Modules
C Library Modules
Access to crypt by r1 is prevented
Writing an external synchronization unit

Constructor invokes wrapper’s initialization function

For each C function, a wrapper function:
♦ Converts between Eiffel and C datatypes
♦ Invokes the corresponding library function

eexternal synchronization class CRYPT

. . .
make is
  external
    "header: #include "crypt_decorator.h"
    "body: crypt_initialize((void *)current);"
  end
end

. . .
crypt( key, salt : STRING ) : STRING is
  external
    "header: #include <unistd.h>"
    "body: { char *key = szumo_str2cstr(_key);"
    "body:   char *salt = szumo_str2cstr(_salt);"
    "body:     return "
    "body:      szumo_cstr2str(crypt(key, salt));"
    "body: }"
  end
. . .
end
Writing an external synchronization unit

Additionally:
If library module invokes external library functions
♦ Include a contract with the associated external unit(s)

```
external synchronization class DB_ENG
    
    feature
    crypt: CRYPT
    
    make is
    do
        create crypt.make
    
    concurrency
    crypt
end
```
Writing a C decorator module is semi-automatic

```c
/* Configuration section. */
static synch_unit* ext_unit;

void init_decorator(synch_unit* u)
{
    ext_unit = u;
    Declare Thread Local Vars;
    Other Initialization;
}

/* Wrapper section. */
result_t __wrap_func(t1 arg1, t2 arg2, ...)
{
    result_t result;
    ASSERT IN REALM(ext_unit);
    Deep_Import_arg1(arg1);
    Deep_Import_arg2(arg2);
    ...
    result = __real_func(arg1, arg2, ...);
    Deep_Export_result(result);
    return result;
}
```

Instantiate a decorator module template
Writing a C decorator module is semi-automatic

```c
/* Configuration section. */
static synch_unit* ext_unit;

void init_decorator(synch_unit* u)
{
    ext_unit = u;
    Declare Thread Local Vars;
    Other Initialization;
}

/* Wrapper section. */
result_t __wrap_func(t1 arg1, t2 arg2, ...)
{
    result_t result;
    ASSERT IN REALM(ext_unit);
    Deep_Import_arg1(arg1);
    Deep_Import_arg2(arg2);
    ...
    result = real_func(arg1, arg2, ...);
    Deep_Export_result(result);
    return result;
}

♦ ASSERT_IN_REALM(ext_unit)
    - Raises an exception if the unit referenced by “ext_unit” is not in the realm of the process executing the macro
```
Writing a C decorator module is semi-automatic

Deep_Import_argn(argn)
- Copies argument into a decorator-local variable

/* Wrapper section. */
result_t __wrap_func(t1 arg1, t2 arg2, ...)
{
    result_t result;
    ASSERT IN REALM(ext_unit);
    Deep_Import_arg1(arg1);
    Deep_Import_arg2(arg2);
    ...
    result = real_func(arg1, arg2, ...);
    Deep_Export_result(result);
    return result;
}

/* Configuration section. */
static synch_unit* ext_unit;

void init_decorator(synch_unit* u)
{
    ext_unit = u;
    Declare Thread Local Vars;
    Other Initialization;
}
Writing a C decorator module is semi-automatic

```c
/* Wrapper section. */
result_t __wrap_func(t1 arg1, t2 arg2, ...)
{
    result_t result;
    ASSERT IN REALM(ext_unit);
    Deep_Import_arg1(arg1);
    Deep_Import_arg2(arg2);
    ...
    result = real_func(arg1, arg2, ...);
    Deep_Export_result(result);
    return result;
}

/* Configuration section. */
static synch_unit* ext_unit;

void init_decorator(synch_unit* u)
{
    ext_unit = u;
    Declare Thread Local Vars;
    Other Initialization;
}
```

- **Deep_Export_argn_result** (result)
  - Copies argument into thread-local storage
Writing a C decorator module is semi-automatic

```c
/* Wrapper section. */
result_t __wrap_func(t1 arg1, t2 arg2, ...)
{
    result_t result;
    ASSERT IN REALM(ext_unit);
    Deep_Import_arg1(arg1);
    Deep_Import_arg2(arg2);
    ...
    result = real_func(arg1, arg2, ...);
    Deep_Export_result(result);
    return result;
}

/* Configuration section. */
static synch_unit* ext_unit;

void init_decorator(synch_unit* u)
{
    ext_unit = u;
    Declare Thread Local Vars;
    Other Initialization;
}
```

♦ Declare Thread Local Variables
  – For each function declared in an external unit that returns a reference to the library module’s static data
Task 1: Summary

♦ Extended VALIDATOR with two concrete classes, both of which required third party library modules

♦ Developed a semi-automated method for making library modules contract aware

♦ Did not need to make any changes to existing web-server design
Extension tasks

♦ Task 1: Authentication
  – Added authentication capability
  – Safely integrated library functions

♦ Task 2: Dynamic content generation
  – Provided for user scripting in the same memory space as the web server

♦ Task 3: Load balancing
  – Modified web server to dynamically adapt the number of content generators as a function of load
Task 2: Dynamic content generation

♦ Purpose: Add a scripting facility for generating dynamic content

♦ Problem: Allow scripts to safely access web-server resources
  – E.g., session data, library modules, etc.

♦ Solution:
  – Implement web-server resources and script interpreters as synchronization units
  – Interpreters trigger contracts to negotiate for web-server resources on behalf of a script
A scripting language

♦ User scripts are written in an extension of Lua
  – A light-weight programming language designed for extending applications
  – Powerful meta-mechanisms allow for the definition of new Lua features

♦ The extension provides primitives for
  – Accessing web-server resources
  – Declaring a script’s resource needs
Accessing web-server resources

Extend Lua with *resource proxies*

- Lua objects that are visible to user scripts
- Bound to web-server resources

<table>
<thead>
<tr>
<th>Resource proxy</th>
<th>Web-server resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>mySession</td>
<td>SESSION unit associated with the http request being serviced</td>
</tr>
<tr>
<td>myCrypt</td>
<td>External synchronization unit wrapping the POSIX crypt function and its static variables</td>
</tr>
<tr>
<td>myNameservices</td>
<td>External synchronization unit wrapping name services functions in the DNS library</td>
</tr>
</tbody>
</table>
Declaring a script’s resource needs

A new Lua instruction:

\texttt{acquire(r1, r2, \ldots)}

- Takes a variable number of resource proxies as arguments
- Blocks the script until the designated resources have been acquired
- May be invoked once per script

Example Lua fragment:

\begin{verbatim}
acquire(mySession, myCrypt)
if not mySession.get("auth_flag") then
  if myCrypt.crypt(\ldots) == \ldots then
    \ldots end
end
\end{verbatim}
**SZUMO_LUA_INTERPRETER**

- Encapsulates and provides an OO interface to the LUA interpreter
- Declares unit variables that link to the web resources
- For each such unit variable \( u \), declares
  - A condition variable: \( \text{acquire}_u \)
  - A constraint: \( \text{acquire}_u \Rightarrow u \)
- Implements \( \text{acquire}(r1, r2, \ldots) \)

```plaintext
feature acquire_units (
  use_session,
  use_nameservices,
  use_crypt: BOOLEAN) is
  do
    acquire_session, acquire_nameservices, acquire_crypt :=
      use_session, use_nameservices, use_crypt
  end

SZUMO_LUA_INTERPRETER support function to atomically acquire resource proxies
```
Design of extension to generate content

```
{ acquire_session=> session,
  acquire_crypt => crypt,
  acquire_nameservices => nameservices }
```

```
<<external>>
CRYPT
<<external>>
NAME_SERVICES
```
Task 2: Summary

♦ Extended Lua with primitives for a script to access web-server resources and declare its resource needs

♦ Wrapped Lua interpreters in synchronization units capable of reflecting the resource acquisition needs of scripts

♦ Did not need to make any changes to the existing web-server design
Extension tasks

♦ Task 1: Authentication
  – Added authentication capability
  – Safely integrated library functions

♦ Task 2: Dynamic content generation
  – Provided for user scripting in the same memory space as the web server

♦ Task 3: Load balancing
  – Modified web server to dynamically adapt the number of content generators as a function of load
Task 3: Load balancing

♦ Purpose: Dynamically adjust the number of Szumo Lua interpreters as a function of demand

♦ Problem:
  – Replacement of an existing synchronization unit with one that has different contextual synchronization dependencies
  – Coordination across multiple stages in request handling pipeline
New classes in load balancing extension

♦ LUA_BALANCED_FACTORY (content generation stage)
  – Replaces LUA_INTERPRETER_FACTORY
  – During rebalancing, creates or deletes LUA_INTERPRETER objects to achieve the target number

♦ LOAD_BALANCER (finalization stage)
  – Computes estimate for number of interpreters needed as a function of the arrival rate of requests for dynamic content and the average time to process these requests
  – At regular intervals, acquires LUA_BALANCED_FACTORY and initiates a rebalancing process

♦ TIME_STAMPER (new: initialization stage)
  – Augments a request with a time stamp
Design of load balancing extension

```plaintext
acquire_session => session,
acquire_crypt => crypt,
acquire_nameservices => nameservices
```

```
{ acquire_session=> session,
  acquire_crypt => crypt,
  acquire_nameservices => nameservices }
```

```
0..1
```

```
CRYPT
```

```
<<external>>
```

```
NAME_SERVICES
```

Laser 2005 - Summer School on Software Engineering

Task 3: Summary

♦ Coordinated changes in different stages
  – Added an initialization stage to log timestamps
  – Replaced the original factory for Lua interpreters with a factory that adjusts the number of interpreters to a target number
  – Extended the finalization stage to compute a target number and synchronize with the factory for rebalancing

♦ Except for replacing the factory, did not need to make any changes to the existing web-server design
Summary

Case study:
♦ Designed and implemented OO multi-threaded web server
♦ SubJECTED the design to three extension/maintenance tasks

Corroborated:
♦ Extensions integrate well into the original design
♦ Szumo contracts compliment modern OO design methods
♦ Szumo appears to scale for realistic applications