Szumo: A Compositional Contract Model for Safe Multi-threaded Applications

L. K. Dillon
Department of Computer Science and Engineering
Michigan State University
www.cse.msu.edu/sens/szumo

Collaborators

Faculty:
– Kurt Stirewalt

Postdoc:
– Beata Sarna-Starosta [from Stony Brook]

Students:
– Scott Fleming
– Yi Huang
– Eduardo Diaz

Emeritus members:
– Reimer Behrends [now, postdoc at U. of Oregon]

Research focus

Multi-threaded OO programs:
– Multiple threads manipulate objects in shared memory
– Designer must identify and protect critical sections of code

Problems:
– Easy to fail to protect a critical section
– Easy for “protected” code to lead to deadlock or starvation
– Synchronization primitives interleave “functional” code
– Resulting “bloat” complicates understanding and maintenance

Research addresses how synchronization contracts can ameliorate these problems

Overview of talk

Motivating example:
– Design and extension of multi-threaded web server
– Focus: Deadlocks that arise from feature extension

Solution: Synchronization Units Model (Szumo)
– Declarative synchronization contracts
– Negotiated and enforced at run time.

Result: SzumoEiffel

Result: Web server case study

Conclusions and future work
Motivating example:
  – Design and extension of multi-threaded web server
  – Focus: Deadlocks that arise from feature extension

Solution: Synchronization Units Model (Szumo)
  – Declarative synchronization contracts
  – Negotiated and enforced at run time.

Result: SzumoEiffel
Result: Web server case study

Conclusions and future work
Maintenance extension: Cyclic dependency

Example configuration

Threads and monitors

Entering c1…
Locking/entering content handler

Thread status
- blocked
- ready
- running

Entering f2...

Locking/entering logger

Context switch occurs...

Thread status
- blocked
- ready
- running
Deadlock

AIEEE...Deadlock!

AIEEE...Deadlock!

AIEEE...Deadlock!

AIEEE...Deadlock!

Blocks => context switch

Blocks => context switch

Blocks => context switch

Blocks => context switch

Tries to enter handler; blocks

Tries to enter handler; blocks

Tries to enter handler; blocks

Tries to enter handler; blocks

Arises from incremental locking and holding that easily occurs when using monitors

Arises from incremental locking and holding that easily occurs when using monitors

Arises from incremental locking and holding that easily occurs when using monitors

Arises from incremental locking and holding that easily occurs when using monitors

Could be avoided using a more complex protocol of thread negotiation

Could be avoided using a more complex protocol of thread negotiation

Could be avoided using a more complex protocol of thread negotiation

Could be avoided using a more complex protocol of thread negotiation

Such protocols:

- Require non-local reasoning
- Tightly interleaved with “functional” logic
- Notoriously difficult to design and verify
- Brittle under maintenance

Such protocols:

- Require non-local reasoning
- Tightly interleaved with “functional” logic
- Notoriously difficult to design and verify
- Brittle under maintenance

Such protocols:

- Require non-local reasoning
- Tightly interleaved with “functional” logic
- Notoriously difficult to design and verify
- Brittle under maintenance

Such protocols:

- Require non-local reasoning
- Tightly interleaved with “functional” logic
- Notoriously difficult to design and verify
- Brittle under maintenance
Overview of talk

Motivating example:
- Design and extension of multi-threaded web server
- Focus: Deadlocks that arise from feature extension

Solution: Synchronization Units Model (Szumo)
- Declarative synchronization contracts
- Negotiated and enforced at run time.

Result: SzumoEiffel
Result: Web server case study
Conclusions and future work

Solution idea: Contracts

Formal agreements between suppliers and clients
- Parties have rights and responsibilities
- Useful for documentation/verification
- Can enable optimizations

Zen and the art of sw reliability:
- guarantee more by checking less!

Code programmed w/o checking assumptions is shorter/clearer
- More reliable
- Responsibility for discharging assumptions may be delegated

Synchronization contracts

Key problem: Negotiating for exclusive access to sets of shared resources at “right time”

Key idea:
- Write client code that optimistically assumes exclusive rights to suppliers when they are needed
- Codify client assumptions as synchronization contracts
- Delegate responsibility for discharging assumptions to code generators/run-time systems

Szumo concepts

Synchronization unit: Cohesive group of objects
Synchronization constraint: Specifies when a unit needs exclusive access to one of its suppliers
Synchronization contract: Set of concurrency constraints
Realm: Data space of a thread
- Grows and shrinks over the lifetime of the thread
- Guarantees:
  - $\forall t_1, t_2 \in \text{Thread} \land t_1 \neq t_2 \Rightarrow \text{realm}(t_1) \cap \text{realm}(t_2) = \emptyset$
  - $\forall t \in \text{Thread}, u \in \text{Unit} \land \text{access}(t, u) \Rightarrow u \in \text{realm}(t)$
Synchronization constraint

General form: \( \text{statePred} \implies \text{unitRef} \)
- \( \text{statePred} \) = (optional) boolean expression
- \( \text{unitRef} \) = reference to a (supplier) unit

In client unit \( c \) asserts that:
\( c \) requires exclusive access to \( c \text{.unitRef} \) when \( c \text{.statePred} \) is true

Run-time system negotiates on behalf of threads: migrates units among realms in order to satisfy the synchronization constraints of all units in a thread’s realm
Atomic transition on generating, finalizing := false, true

Threads and realms

r1 transitions to state “generating”

Context switch occurs
r2 attempts transition to "finalizing" and blocks

Thread status
blocked ready running

Web_Dispatcher → Req_Handler_Pool

r1 transitions out of "generating"

Thread status
blocked ready running

Web_Dispatcher → Req_Handler_Pool

r1 attempts transition to "finalizing" and blocks

Thread status
blocked ready running

Web_Dispatcher → Req_Handler_Pool

r2 transitions out of "finalizing"

Thread status
blocked ready running

Web_Dispatcher → Req_Handler_Pool
**Composition of sync constraints**

Horizontal composition:
- Client may specify need for multiple suppliers
- At run-time, these suppliers are acquired atomically

Vertical (transitive) composition:
- Client’s entailment composes with that of the suppliers it entails
- E.g., when “generating,” request handler needs a content generator, which might need other suppliers
- At run-time, when request handler transitions into “generating,” all needed units are acquired atomically

---

**Overview of talk**

Motivating example:
- Design and extension of multi-threaded web server
- Focus: Deadlocks that arise from feature extension

Solution: *Synchronization Units Model* (Szumo)
- Declarative synchronization contracts
- Negotiated and enforced at run time.

Result: **SzumoEiffel**

Result: Web server case study

Conclusions and future work

---

**SzumoEiffel**

Minimal extensions to the Eiffel language:
- New keywords: `synchronization`, `concurrency`, `when`, `shared`
- Small number of library classes

Compiler:
- Catches assignments affecting sync constraints
- Inserts checks for realm violation

Compiler available at: [http://www.cse.msu.edu/sens/szumo](http://www.cse.msu.edu/sens/szumo)

---

**Implementation of realms**

Key ideas:
- Units in a running program form a directed graph
- Realm is a *connected subgraph* of this (global) unit graph
- *Roots* of each realm cannot migrate

Each unit object attributed with:
- pointer to thread that owns it
- count of # of references by other units in the same realm

*Realms can be traversed/maintained using graph-reachability algorithms*
Realm update

Concept: **Realm-affecting operation:**
- User code that modifies `statePred` or `uRef`
- Triggers an update of the realm

Realm updated in two phases:
- **Contraction:** releases unneeded units
- **Completion:** incrementally claims needed units
- Thread blocks if another thread owns a needed unit

**Realm update implements contract re-negotiation**

Deadlock & starvation

Problem:
- Realms of multiple threads update concurrently
- Need to avoid starvation and deadlock (to extent possible)

Two-phase approach:
- Avoid starvation by giving priority to “older” threads
- Detect and recover from avoidable deadlocks using a restart approach

*Can avoid all but “essential” deadlocks*

Overview of talk

Motivating example:
- Design and extension of multi-threaded web server
- Focus: Deadlocks that arise from feature extension

Solution: **Synchronization Units Model (Szumo)**
- Declarative synchronization contracts
- Negotiated and enforced at run time.

Result: SzumoEiffel

**Result: Web server case study**

Conclusions and future work

Case Study in SzumoEiffel

Goal: Validate the efficacy of Szumo to current software engineering practice
- Is Szumo sufficiently expressive to handle designs of real systems?
- Can Szumo support use of modern OO design techniques (e.g. structural modeling notations, design patterns)?
- Is Szumo effective in localizing synchronization concerns?
Approach

Designed and implemented a basic multi-threaded OO web server
- Modeled after Apache
- Exemplar of realistic multi-threaded OO design
- Comprises 51 classes, 31 of which are synch classes
- Written in SzumoEiffel

Tested extensibility/maintainability by adding different types of features

Basic web-server architecture

Extension tasks

Task 1: Authentication
- Added authentication capability
- Wrap legacy libraries to be contract aware

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 1: Add PWD and DB Validators
Making third-party libraries contract-aware

Partition contract-unaware library into “modules”
- Self-contained set of functions and static variables

Create decorator module for each library module
- Exports same interface as library module
- Checks that accesses do not cross realm boundaries
- Import/export arguments/results that could escape

Contract-aware clients invoke operations of an external synchronization unit

Extension 2: Dynamic content handler

Interprets embedded user scripts

Problem:
- User scripts may access web-server resources
- Can interfere with one another and with main system

Solution:
- Implement dynamic content handler as synchronization unit
- Web-server resources are contract-aware components
- Translate user script synchronization requirements into contract with these components
Extension 3: Load balancing

Adapt the number of dynamic handlers to improve QoS

Measuring demand:
- Start stopwatch at beginning of server pipeline
- Stop it at the end of pipeline

Based on demand:
- Maintain factory of interpreters
- Factory creates/shuts down interpreters as needed

Overview of talk

Motivating example:
- Design and extension of multi-threaded web server
- Focus: Deadlocks that arise from feature extension

Solution: Synchronization Units Model (Szumo)
- Declarative synchronization contracts
- Negotiated and enforced at run time.

Result: SzumoEiffel

Result: Web server case study

Conclusions and future work

Goal: End-to-end process

Szumo constraints nicely enhance existing OO modeling methods and frameworks
- Class models naturally represent unit classes and associations (i.e., unit references)
- State models show how a unit moves through its various synchronization states
- Instance diagrams naturally depict unit configurations

Goal: Develop a UML-based process for multi-threaded application design, verification, and evolution
Goal: End-to-end process

Goal: More powerful constraints

Not discussed in this talk:
- Expressing condition synchronization
- Analysis of Szumo design models
- Detecting contract violations

New features:
- Incorporating method pre- and post-condition style contracts
- Supporting intra-thread concurrency
- Better separation of concerns

Thanks!

This work has been supported in part by:
- Department of the Navy, Office of Naval Research under grant N00014-01-1-0744
- NSF grants EIA-0000433, CCR-9901017, CCR-9984727, CDA-9617310, and CCR-9896190
- Defense Advance Research Projects Agency, under the DASADA program