An R7RS Compatible Module System for Termite Scheme

ELS’20

Frédéric Hamel
Marc Feeley
Termite Scheme

• Built on top of Gambit Scheme

• Designed to simplify programming distributed systems composed of a network of communicating nodes

• Uses the Actor model: each node executes one or more threads reacting to messages received in their mailbox
Heterogeneous Systems

• A common situation is using nodes with different characteristics (instruction set, peripherals, type and version of OS, etc)

• To allow code to run on any type of node the code is either interpreted or compiled to a portable bytecode or compiled to machine code for each type of node (the best in terms of runtime performance)

• How to send messages that contain code (procedures) in a heterogeneous system that compiles to machine code?

ARM/Linux

CODE

network

x86/Windows
Gambit Features

• Gambit compiles to **fast portable C code** (machine/OS agnostic)

• Messages transferred between nodes are encoded by Gambit using a **machine independent** sequence of bytes

\[
\#(1\ 2\ 3) \xrightarrow{\#x23\ #x51\ #x52\ #x53} \#(1\ 2\ 3)
\]

• The serialization format supports **procedures/closures, continuations, sharing** and **cycles**

• This simplifies programming:
  • Remote Procedure Call (RPC)  => send a procedure/closure
  • Task migration  => send a continuation
  • Hot code update  => send a proc./closure/cont. of code not previously known by destination node
(import (termite))

(node-init ":7000") ;; on port 7000

(define server ;; pong service thread
    (spawn
        (lambda ()
            (let loop ()
                (recv
                    ((from tag 'PING)
                        (! from (list tag 'PONG))))
                (loop)))))

(publish-service 'pong-server server)

(thread-join! server) ;; wait for end
server

(import (termite))

(node-init ":7000") ;; on port 7000

(define server ;; pong service thread
(spawn
  (lambda ()
    (let loop ()
      (recv
       ((from tag 'PING)
        (! from (list tag 'PONG))))
      (loop)))))

(publish-service 'pong-server server)

(thread-join! server) ;; wait for end

Termite Scheme cheat sheet

(get next message from mailbox and pattern match)

(reqv
  (pattern action)
  ...
  )

(! dest msg) send msg to dest

(!? dest msg) send (self tag msg) to dest and receive (tag result)
**Hot Code Update Example**

```scheme
(define server ;; pong service thread
  (spawn
    (lambda ()
      (let loop ()
        (recv
          ((from tag 'PING)
            (! from (list tag 'PONG))))
        (loop)))))
(publish-service 'pong-server server)
(thread-join! server) ;; wait for end
```

---

**Termite Scheme cheat sheet**

- ```recv (pattern action) ...``` get next message from mailbox and pattern match
- ```(! dest msg)``` send msg to dest
- ```(!? dest msg)``` send (self tag msg) to dest and receive (tag result)
(import (termite))

(node-init ":7000") ;; on port 7000

(define server ;; pong service thread
  (spawn
    (lambda ()
      (let loop ()
        (recv
          ((from tag 'PING)
            (! from (list tag 'PONG))))
        (loop)))))

(publish-service 'pong-server server)

(thread-join! server) ;; wait for end

(import (termite))

(node-init) ;; on fresh port

(define server
  (remote-service 'pong-server ":7000"))

(println (!? server 'PING))
Add Support for Hot Code Update

server

(import (termite))

(node-init ":7000") ;; on port 7000

(define server ;; pong service thread
  (spawn
    (lambda ()
      (let loop ()

        (recv

          (from tag 'PING)
          (! from (list tag 'PONG))))

          (from tag ('UPDATE k))
          (! from (list tag 'ACK))
          (k #t))

          (loop)))))

(publish-service 'pong-server server)

(thread-join! server) ;; wait for end

handling of the UPDATE message that replaces the behaviour of the server with a new continuation k contained in the message
(import (termite))

(node-init ":7000") ;; on port 7000

(define server ;; pong service thread
    (spawn
        (lambda ()
            (let loop ()
                (recv
                    ((from tag 'PING)
                        (! from (list tag 'PONG)))
                    ((from tag ('UPDATE k))
                        (! from (list tag 'ACK))
                        (k #t))
                    ((from tag ('MIGRATE dest))
                        (call/cc
                            (lambda (k)
                                (!? dest (list 'UPDATE k))
                                (! from (list tag 'ACK)))))))
                (loop))))))

(publish-service 'pong-server server)

(thread-join! server) ;; wait for end

(define new-server
    (spawn
        (lambda ()
            (let loop ()
                (recv
                    ((from tag 'PING)
                        (! from (list tag 'HELLO)))
                    ((from tag ('UPDATE k))
                        (! from (list tag 'ACK))
                        (k #t))
                    ((from tag ('MIGRATE dest))
                        (call/cc
                            (lambda (k)
                                (!? dest (list 'UPDATE k))
                                (! from (list tag 'ACK)))))))
                (loop))))))

(!? new-server (list 'MIGRATE server))

(println (!? server 'PING)) ;; HELLO
(import (termite))

(node-init ":7000") ;; on port 7000

(define server ;; pong service thread
  (spawn
    (lambda ()
      (let loop ()
        (recv
          ((from tag 'PING)
            (! from (list tag 'PONG)))
          ((from tag ('UPDATE k))
            (! from (list tag 'ACK))
            (k #t))
          ((from tag ('MIGRATE dest))
            (call/cc
              (lambda (k)
                (!? dest (list 'UPDATE k))
                (! from (list tag 'ACK)))))))
        (loop)))))

(publish-service 'pong-server service)

(thread-join! server) ;; wait for end

(code for new behaviour of pong service)

(define new-server
  (spawn
    (lambda ()
      (let loop ()
        (recv
          ((from tag 'PING)
            (! from (list tag 'HELLO)))
          ((from tag ('UPDATE k))
            (! from (list tag 'ACK))
            (k #t))
          ((from tag ('MIGRATE dest))
            (call/cc
              (lambda (k)
                (!? dest (list 'UPDATE k))
                (! from (list tag 'ACK))))))
        (loop)))))

(!? new-server (list 'MIGRATE server))

(println (!? server 'PING)) ;; HELLO
import (termite)

(node-init :7000) ;; on port 7000
define server ;; pong service thread
(spawn
 (lambda ()
  (let loop ()
    (recv
     ((from tag 'PING)
      (! from (list tag 'PONG)))
    ((from tag ('UPDATE k))
     (! from (list tag 'ACK))
     (k #t)))
    ((from tag ('MIGRATE dest))
     (call/cc
      (lambda (k)
        (!? dest (list 'UPDATE k))
        (! from (list tag 'ACK))))))
  (loop)))))
(publish-service 'pong-server server)
(thread-join! server) ;; wait for end

server handling of the UPDATE message that replaces the behaviour of the server with a new continuation k contained in the message...

(define new-server
 (spawn
  (lambda ()
   (let loop ()
    (recv
     ((from tag 'PING)
      (! from (list tag 'HELLO)))
    ((from tag ('UPDATE k))
     (! from (list tag 'ACK))
     (k #t))
    ((from tag ('MIGRATE dest))
     (call/cc
      (lambda (k)
        (!? dest (list 'UPDATE k))
        (! from (list tag 'ACK))))))
  (loop)))))
(!? new-server (list 'MIGRATE server))
(println (!? server 'PING)) ;; HELLO
server

(import (termite))

(define server
    ;; pong service thread
    (spawn
        (lambda ()
            (let loop ()
                (recv
                    (from tag 'PING)
                    (! from (list tag 'PONG)))
                (from tag ('UPDATE k))
                (! from (list tag 'ACK))
                (k #t))
            (loop)))))

(publish-service 'pong-server server)

(thread-join! server) ;; wait for end

server handling of the UPDATE message that replaces the behaviour of the server with a new continuation k contained in the message...

(define new-server
    (spawn
        (lambda ()
            (let loop ()
                (recv
                    (from tag 'PING)
                    (! from (list tag 'HELLO)))
                (from tag ('UPDATE k))
                (! from (list tag 'ACK))
                (k #t))
            (from tag ('MIGRATE dest))
            (call/cc
                (lambda (k)
                    (!? dest (list 'UPDATE k))
                    (! from (list tag 'ACK))))
            (loop)))))

(!? new-server (list 'MIGRATE server))

(updater)

...
Issues with Compiled code

- The original implementation of Termite Scheme allows unrestricted serialization/deserialization of *interpreted code*

- Compiled code can only be deserialized when the receiving node contains the *same compiled code* (by identifying each control point symbolically, e.g. *control point #5 in procedure foobar*)

- This restriction
  - limits changing the code base during execution
  - hinders the use of fast compiled code in RPC
  - precludes the use of hot code update of a compiled program

- Our work brings a solution to this issue in the form of a *R7RS compatible module system that installs and compiles code on demand*
Our Solution

• A module’s source code is **hosted on a VCS server** accessible on the network, such as **github** or **gitlab**

• A hosted module’s name identifies its **location** and **version**:

  
  (github.com/fred hello @2.0)

  or equivalently

  github.com/fred/hello@2.0

• The hosted module’s name is **embedded in the name of procedures defined in the module** (in the namespace) allowing the deserialization process to locate, fetch and compile the module’s source code if it is not yet installed:

  github.com/fred/hello@2.0#hi

  namespace prefix of module

  name in module
Module Syntax

Standard in R7RS

Extensions (mostly for build options)

(define-library name

(export <export spec>...)
(import <import set>...)
(begin <command or definition>...)
(include <filename>...)
(include-ci <filename>...)
(include-library-declarations <filename>...)
(cond-expand <cond expand features>...)

(namespace <namespace>)
(cc-options <options>...)
(ld-options <options>...)
(ld-options-prelude <options>...)
(pkg-config <options>...)
(pkg-config-path <path>...)

name does not mention the version because it is implicitly stored in the VCS
Sample 2 Module Program

```lisp
(define-library (github.com/fred hello)
  (export hi)
  (import (only (scheme base) define)
    (github.com/fred hello @1.0))
  (begin
    (define (main)
      (hi "lion")
      (hi "tiger")))
)

(cats.sld version 2.0)

(define-library (github.com/zoo cats)
  (import (only (scheme base) define)
    (github.com/fred hello @1.0))
  (begin
    (define (main)
      (hi "lion")
      (hi "tiger")))
)

(hello.sld version 1.0)

(define-library (github.com/fred hello)
  (export hi)
  (import (only (scheme base) define)
    (rename (scheme write) (display show)))
  (begin
    (define (hi str)
      (show "hello ")
      (show str)
      (show \n)))
)
Implementation

• The module system is implemented as an **expansion** to the following Gambit preexisting forms:

```plaintext
(##declare (block))  ; assume block compilation (no set! in other modules to local variables)

(##namespace ("ns"))       ; add ns# prefix to all free identifiers
(##namespace ("ns#" id1 id2 ...))  ; add ns# prefix only to id1, id2, ...
(##namespace ("ns#" (id1 id2) ...))  ; rename id1 to id2, ...

(##supply-module name)       ; declare name of module to be name
(##demand-module name)       ; register dependency on module name
```

• Dependencies registered with `##demand-module` are handled by the **module loader** that has been extended to download and compile dependent hosted modules not currently installed.
Expansion of cats.sld

(define-library (gitlab.com/zoo cats)

(import (only (scheme base) define)
         (github.com/fred hello @1.0))

(begin
  (define (main)
    (hi "lion")
    (hi "tiger")))

(declare (block))

(supply-module gitlab.com/zoo/cats@2.0)
(demand-module github.com/fred/hello@1.0)

(namespace ("gitlab.com/zoo/cats@2.0")
            ("define")
            ("github.com/fred/hello@1.0" hi))

(define (main) ;; defines gitlab.com/zoo/cats@2.0#main
  (hi "lion") ;; calls github.com/fred/hello@1.0#hi
  (hi "tiger") ;; same
Other Features

- Convenient other features not essential to Termite:
  - **Whitelist** for allowing automatic installation from safe sites
  - **Manual module management** tool integrated to interpreter
  - **Optional version**: useful for development phase
  - **Module aliases** can be defined (and are lexically scoped):

```scheme
(define-module-alias (gitlab.com/zoo cats)
  (gitlab.com/zoo cats @2.0))

(define-module-alias (fh)
  (github.com/fred hello))

(import (gitlab.com/zoo cats)) ;; forces use of version 2.0

(import (fh @1.0)) ;; import (github.com/fred hello @1.0)
```
Evaluation

- Goal: determine the performance gain achieved by the now possible compilation of the modules

- Used 3 standard Scheme benchmarks of various source code sizes and execution time when interpreted, modified to be executed through a RPC

  - **4K** (“Puzzle” program, ~4 Kbytes, ~0.1 sec)
  - **40K** (“Scheme” program, ~40 Kbytes, ~1 sec)
  - **400K** (“Compiler” program, ~400 Kbytes, ~10 secs)

- Used 3 machines, with different OS (linux/macOS), processors (x86/ARM), and performance (Raspberry pi and desktop):
  - $M_{ARM/Linux}$ (slowest) / $M_{x86/macOS}$ / $M_{x86/Linux}$ (fastest)
Evaluation

- Used 3 execution scenarios:
  - **INTERPRETED**: no compilation of module (original Termite)
  - **STEADY-STATE**: compilation + module previously installed
  - **FIRST-INSTALL**: compilation + module not previously installed

- Timing (ms) for $M_{\text{ARM/Linux}}$ doing RPC to $M_{\text{x86/Linux}}$

<table>
<thead>
<tr>
<th></th>
<th>Time (ms)</th>
<th>4K</th>
<th>40K</th>
<th>400K</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTERPRETED</strong></td>
<td>Total for RPC</td>
<td>179.2 ± 1.6</td>
<td>1002.7 ± 8.1</td>
<td>10801.1 ± 11.0</td>
</tr>
<tr>
<td></td>
<td>On destination</td>
<td>132.6 ± 0.7</td>
<td>954.6 ± 0.0</td>
<td>10390.5 ± 2.6</td>
</tr>
<tr>
<td><strong>STEADY-STATE</strong></td>
<td>Total for RPC</td>
<td>26.9 ± 1.2</td>
<td>60.4 ± 0.6</td>
<td>492.5 ± 0.7</td>
</tr>
<tr>
<td></td>
<td>On destination</td>
<td>2.2 ± 0.0</td>
<td>35.2 ± 0.0</td>
<td>462.8 ± 0.3</td>
</tr>
<tr>
<td><strong>FIRST-INSTALL</strong></td>
<td>Total for RPC</td>
<td>1159.8</td>
<td>2502.0</td>
<td>153272.6</td>
</tr>
<tr>
<td></td>
<td>On destination</td>
<td>2.2</td>
<td>37.1</td>
<td>464.1</td>
</tr>
</tbody>
</table>
Evaluation

- Used 3 execution scenarios:
  - **INTERPRETED**: no compilation of module (original Termite)
  - **STEADY-STATE**: compilation + module previously installed
  - **FIRST-INSTALL**: compilation + module not previously installed

- Timing (ms) for $M_{ARM/Linux}$ doing RPC to $M_{x86/Linux}$

<table>
<thead>
<tr>
<th>Scenario</th>
<th>4K</th>
<th>40K</th>
<th>400K</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTERPRETED</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total for RPC</td>
<td>179.2 ± 1.6</td>
<td>1002.7 ± 8.1</td>
<td>10801.1 ± 11.0</td>
</tr>
<tr>
<td>On destination</td>
<td>132.6 ± 0.7</td>
<td>954.6 ± 0.0</td>
<td>10390.5 ± 2.6</td>
</tr>
<tr>
<td><strong>STEADY-STATE</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total for RPC</td>
<td>26.9 ± 1.2</td>
<td>60.4 ± 0.6</td>
<td>492.5 ± 0.7</td>
</tr>
<tr>
<td>On destination</td>
<td>2.2 ± 0.0</td>
<td>35.2 ± 0.0</td>
<td>462.8 ± 0.3</td>
</tr>
<tr>
<td><strong>FIRST-INSTALL</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total for RPC</td>
<td>1159.8</td>
<td>2502.0</td>
<td>153272.6</td>
</tr>
<tr>
<td>On destination</td>
<td>2.2</td>
<td>37.1</td>
<td>464.1</td>
</tr>
</tbody>
</table>
Related Work

- **Go**: VCS hosted modules with versions, no dynamic install
- **QuickLisp**: need to register modules, not tied to deserial.
- **Erlang**: hot code update, only manual install of modules
- **Nix**: similar idea of keeping multiple versions of modules
- **R6RS Scheme**: has versions but not implicit from VCS
- Other module systems for Gambit include: **Black Hole** / **JazzScheme** / **Gerbil** / **SchemeSpheres**

None of these offers transparent deserialization of compiled procedures and continuations needed for hot code update.