Referências Bibliográficas

[BBG+02] BAK, L.; BRACHA, G.; GRARUP, S.; GRIESEMER, R.; GRISWOLD, D. ; HOLZLE, U.. **Mixins in Strongtalk.** In: PROCEEDINGS OF THE "INHERITANCE WORKSHOP"AT ECOOP'02, 2002. 4.1.1

[BD01] BETTINI, L.; NICOLA, R. D.. **Translating strong mobility into weak mobility.** Lecture Notes in Computer Science, 2240:182–197, 2001. 2.1, 2.3.1

[BHKP+04] BOUCHENAK, S.; HAGIMONT, D.; KRAKOWIAK, S.; PALMA, N. D. ; BOYER, F.. **Experiences implementing efficient Java thread serialization, mobility and persistence.** Software: Practice & Experience, 34(4):355–393, 2004. 2.1, 2.2.3

[BSS94] VON BANK, D. G.; SHUB, C. M. ; SEBESTA, R. W.. **A unified model of pointwise equivalence of procedural computations.** ACM Trans. Program. Lang. Syst., 16(6):1842–1874, 1994. 2.1

[BWD96] BRUGGEMAN, C.; WADDELL, O. ; DYBVIG, R. K.. **Representing control in the presence of one-shot continuations.** In: PLDI '96: PROCEEDINGS OF THE ACM SIGPLAN 1996 CONFERENCE ON PROGRAMMING LANGUAGE DESIGN AND IMPLEMENTATION, p. 99–107, New York, NY, USA, 1996. ACM. 4.2.8

[CFH+05] CLARK, C.; FRASER, K.; HAND, S.; HANSEN, J. G.; JUL, E.; LIMPACH, C.; PRATT, I. ; WARFIELD, A.. **Live migration of virtual machines.** In: NSDI'05: PROCEEDINGS OF THE 2ND CONFERENCE ON SYMPOSIUM ON NETWORKED SYSTEMS DESIGN & IMPLEMENTATION, p. 273–286, Berkeley, CA, USA, 2005. USENIX Association. 2.2

[CG98] CARDELLI, L.; GORDON, A. D.. **Mobile ambients.** In: FOUNDATIONS OF SOFTWARE SCIENCE AND COMPUTATION STRUCTURES: FIRST INTERNATIONAL CONFERENCE, FOSSACS '98. Springer-Verlag, Berlin Germany, 1998. 2.1

[CHK94] CHESS, D.; HARRISON, C. ; KERSHENBAUM, A.. **Mobile agents: are they a good idea?** Technical Report RC 19887, IBM Research Division, T.J. Watson Research Center, 1994. 1

[CJK95] CEJTIN, H.; JAGANNATHAN, S. ; KELSEY, R.. **Higher-order distributed objects.** ACM Transactions on Programming Languages and Systems (TOPLAS), 17(5):704–739, 1995. 2.1

[CS02] CHANCHIO, K.; SUN, X.-H.. **Data collection and restoration for heterogeneous process migration.** Software: Practice & Experience, 32(9):845–871, 2002. 2.1, 2.2.3

[Cardelli95] CARDELLI, L.. **A language with distributed scope.** In: PROCEEDINGS OF THE 22ND ACM SIGPLAN-SIGACT SYMPOSIUM ON PRINCIPLES OF PROGRAMMING LANGUAGES (POPL '95), p. 286–297, New York, NY, USA, 1995. ACM. 2.1

[Cardelli97] CARDELLI, L.. **Mobile Computation.** In: Jan Vitek; Christian Tschudin, editors, MOBILE OBJECT SYSTEMS: TOWARDS THE PROGRAMMABLE INTERNET, volumen 1222, p. 3–6. Springer-Verlag: Heidelberg, Germany, 1997. 2.1

[Cardelli99] CARDELLI, L.. **Abstractions for Mobile Computation.** In: SECURE INTERNET PROGRAMMING, p. 51–94, 1999. 2.1

[DM95] DEMERS, F.-N.; MALENFANT, J.. **Reflection in logic, functional and object-oriented programming: a short comparative study.** In IJCAI '95 Workshop on Reflection and Metalevel Architectures and their Applications in AI. 3.2

[DO99] DOUGLIS, F.; OUSTERHOUT, J.. **Transparent process migration: design alternatives and the sprite implementation.** p. 57–86, 1999. 1.1

[Eskicioglu90] ESKICIOGLU, M. R.. **Design issues of process migration facilities in distributed systems.** In: IEEE COMPUTER SOCIETY TECHNICAL COMMITTE ON OPERATING SYSTEMS AND APPLICATION ENVIRONMENTS NEWSLETTER, volumen 4, p. 3–13, 1990. 1

[FCG00] FERRARI, A.; CHAPIN, S. ; GRIMSHAW, A.. **Heterogeneous process state capture and recovery through Process Introspection.** Cluster Computing, 3(2):63–73, 2000. 2.1, 2.3.1

[FF06] FELLEISEN, M.; FLATT, M.. **Programming languages and lambda calculi.** 3.4

[FPV98] FUGGETTA, A.; PICCO, G. P. ; VIGNA, G.. **Understanding Code Mobility.** IEEE Transactions on Software Engineering, 24(5):342–361, 1998. 1, 2.1

[FW84] FRIEDMAN, D. P.; WAND, M.. **Reification: Reflection without metaphysics.** In: LFP '84: PROCEEDINGS OF THE 1984 ACM SYMPOSIUM ON

LISP AND FUNCTIONAL PROGRAMMING, p. 348–355, New York, NY, USA, 1984. ACM. 3.2

[Funfrocken98] FÜNFROCKEN, S.. **Transparent migration of Java-based mobile agents.** In: PROCEEDINGS OF THE SECOND INTERNATIONAL WORKSHOP ON MOBILE AGENTS (MA '98), p. 26–37. Springer-Verlag, 1999. 2.1, 2.2.3

[HWW93] HSIEH, W. C.; WANG, P. ; WEIHL, W. E.. **Computation migration: enhancing locality for distributed-memory parallel systems.** In: PROCEEDINGS OF THE FOURTH ACM SIGPLAN SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING (PPOPP '93), p. 239–248, New York, NY, USA, 1993. ACM. 2.1, 4.2.6, 6

[Henderson80] HENDERSON, P.. **Functional Programming.** Prentice Hall PTR, Upper Saddle River, NJ, USA, 1980. 3.4

[IFC05] IERUSALIMSCHY, R.; DE FIGUEIREDO, L. H. ; FILHO, W. C.. **The Implementation of Lua 5.0.** Journal of Universal Computer Science, 11(7):1159–1176, 2005. (document), 4.2.10, 5.2

[IKKW02] ILLMANN, T.; KRUEGER, T.; KARGL, F. ; WEBER, M.. **Transparent migration of mobile agents using the Java Platform Debugger Architecture.** In: PROCEEDINGS OF THE 5TH INTERNATIONAL CONFERENCE ON MOBILE AGENTS (MA '01), p. 198–212. Springer-Verlag, 2002. 2.2.3

[Ierusalimschy06] IERUSALIMSCHY, R.. **Programming in Lua, Second Edition.** Lua.Org, 2006. 1.1, 5.4

[JC04] JIANG, H.; CHAUDHARY, V.. **Process/thread migration and checkpointing in heterogeneous distributed systems.** In: PROCEEDINGS OF THE 37TH ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (HICSS'04) - TRACK 9, p. 10pp. IEEE, 2004. 2.1

[KahLua] KARLSSON, K.. **KahLua, a Lua Virtual Machine for Java.** http://code.google.com/p/kahlua/, 2008. 5.3.1

[Kennedy04] KENNEDY, A. J.. **Functional pearl: Pickler combinators.** Journal of Functional Programming, 14(6):727–739, 2004. 1.1

[LL87] LU, C.; LIU, J. W. S.. **Correctness criteria for process migration.** Technical Report NASA-CR-182939, University of Illinois, 1987. 2.1

[LO98] LANGE, D. B.; OSHIMA, M.. **Programming and deploying Java(TM) mobile agents with Aglets(TM).** Addison-Wesley Professional, 1 edition, 1998. 2.1

[LTBL97] LITZKOW, M.; TANNENBAUM, T.; BASNEY, J. ; LIVNY, M.. **Checkpoint and migration of UNIX processes in the Condor distributed processing system.** Technical Report UW-CS-TR-1346, University of Wisconsin - Madison Computer Sciences Department, April 1997. 1.1

[Landin64] LANDIN, P.. **The mechanical evaluation of expressions.** Computer Journal, 6(4):308–320, 1964. 3.4

[MDPW+00] MILOJICIC, D.; DOUGLIS, F.; PAINDAVEINE, Y.; WHEELER, R. ; ZHOU, S.. **Process migration.** ACM Computing Surveys, 32(3):241–299, 2000. 1, 2.1

[MI05] MASCARENHAS, F.; IERUSALIMSCHY, R.. **Running Lua Scripts on the CLR through Bytecode Translation.** Journal of Universal Computer Science, 11(7):1275–1290, jul 2005. 5.3.1

[MI09] MOURA, A. L.; IERUSALIMSCHY, R.. **Revisiting coroutines.** ACM Transactions on Programming Languages and Systems, 2009. Aceito para publicação. 4.2.7

[MJD96] J. MALENFANT, M. J.; DEMERS, F.-N.. **A tutorial on behavioral reflection and its implementation.** In: G., K., editor, PROCEEDINGS OF THE REFLECTION '96 CONFERENCE, p. 1–20, 1996. 3.2

[MP96] MILOJICIC, D. S.; PAINDAVEINE, Y.. **Process vs. task migration.** In: PROCEEDINGS OF THE 29TH HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (HICSS'96), volumen 1, p. 636, Washington, DC, USA, 1996. IEEE. 2.1

[MRS08] MILANÉS, A.; RODRIGUEZ, N. ; SCHULZE, B.. **State of the art in heterogeneous strong migration of computations.** Concurrency and Computation: Practice and Experience, 20(13):1485–1508, 2008. 1, 2, 2.1, 2.3, 6.3

[Maes87] MAES, P.. **Concepts and experiments in computational reflection.** In: OOPSLA '87: CONFERENCE PROCEEDINGS ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS, p. 147–155, New York, NY, USA, 1987. ACM. 3.2, 4.1.1

[Maia08] MAIA, R.. **LOOP: Lua Object-Oriented Programming.** http://loop.luaforge.net/, 2008. Last visited on 20/10/2008. 4.2.7, A

[Man06] MAN, K.. **A No-Frills Introduction to Lua 5.1 VM Instructions. Version 0.1.** http://luaforge.net/docman/view.php/83/98/ ANoFrillsIntroToLua51VMInstructions.pdf, 2006. Last visited on 06/01/2008. 4.2.10

[McAffer95] MCAFFER, J.. **Meta-level architecture support for distributed objects.** In: IWOOOS '95: PROCEEDINGS OF THE 4TH INTERNATIONAL WORKSHOP ON OBJECT-ORIENTATION IN OPERATING SYSTEMS, p. 232. IEEE, 1995. 2.2.1

[Miller06] MILLER, M. S.. **Robust composition: towards a unified approach to access control and concurrency control.** PhD thesis, Johns Hopkins University, Baltimore, MD, USA, 2006. 3.1, 6

[Moura04] MOURA, A. L.. **Revisitando co-rotinas.** PhD thesis, PUC-Rio, 2004. 4.2.7

[Nuttall94] NUTTALL, M.. **Survey of systems providing process or object migration.** Technical Report 94/10, Imperial College Research Report, 1994. 1

[PMOY06] PESCHANSKI, F.; MASUYAMA, T.; OYAMA, Y. ; YONEZAWA, A.. **MobileScope: A programming language with objective mobility.** Available at http://www-desir.lip6.fr/~ pesch/data/pesch-ijwmc-2006-final.ps, 2006. 2.1

[RDT08] RÖTHLISBERGER, D.; DENKER, M. ; Í. TANTER. **Unanticipated partial behavioral reflection: Adapting applications at runtime.** Computer Languages, Systems and Structures, 34(2-3):46–65, 2008. 3.2, 6

[RFC4506] EISLER, M.. **XDR: External data representation standard**, 2006. 2.2.1

[SH98] SMITH, P.; HUTCHINSON, N. C.. **Heterogeneous process migration: the Tui system.** Software: Practice & Experience, 28(6):611–639, 1998. 2.1, 2.2.1, 2.2.3

[SJ95] STEENSGAARD, B.; JUL, E.. **Object and native code thread mobility among heterogeneous computers.** In: PROCEEDINGS OF THE 15TH ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, p. 68–78, 1995. 2.1

[SLW+07] SEWELL, P.; LEIFER, J. J.; WANSBROUGH, K.; NARDELLI, F. Z.; ALLEN-WILLIAMS, M.; HABOUZIT, P. ; VAFEIADIS, V.. **Acute: High-level programming language design for distributed computation.** J. Funct. Program., 17(4-5):547–612, 2007. 1.1, 6

[SMY99] SEKIGUCHI, T.; MASUHARA, H. ; YONEZAWA, A.. **A simple extension of Java language for controllable transparent migration and its portable implementation.** In: PROCEEDINGS OF THE THIRD INTERNATIONAL CONFERENCE ON COORDINATION LANGUAGES AND MODELS (COORDINATION '99), p. 211–226. Springer-Verlag, 1999. 2.1, 2.2.1, 2.3.1, 2.3.1

[SRI08] A. SKYRME ; RODRIGUEZ, N. ; IERUSALIMSCHY, R.. **Exploring Lua for Concurrent Programming.** In: ANAIS DO XII SIMPÓSIO BRASILEIRO DE LINGUAGENS DE PROGRAMAÇÃO (SBLP), 2008. 4.2.9

[Shub90] SHUB, C. M.. **Native code process-originated migration in a heterogeneous environment.** In: PROCEEDINGS OF THE 1990 ACM ANNUAL CONFERENCE ON COOPERATION (CSC '90), p. 266–270. ACM, 1990. 2.1

[Smith84] SMITH, B. C.. **Reflection and semantics in LISP.** In: PRINCIPLES OF PROGRAMMING LANGUAGES (POPL84), January 1984. 3.2

[Smith88] SMITH, J. M.. **A survey of process migration mechanisms.** ACM SIGOPS Operating Systems Review, 22(3):28–40, 1988. 1

[Smith97] SMITH, P. W.. **The Possibilities and Limitations of Heterogeneous Process Migration.** PhD thesis, Department of Computer Science, The University of British Columbia, October 1997. 2.1, 2.2.3

[Sunshine2005] SUNSHINE-HILL, B.. **Pluto: Heavy-duty persistence for lua.** http://luaforge.net/projects/pluto/, 2005. last visited on 05/09/2008. 2.2.1, 4.1

[TH91] THEIMER, M.; HAYES, B.. **Heterogeneous process migration by recompilation.** In: 11TH INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, p. 18–25, 1991. 2.1

[TKS06] TACK, G.; KORNSTAEDT, L. ; SMOLKA, G.. **Generic pickling and minimization.** Electronic Notes in Theoretical Computer Science, 148(2):79–103, 2006. 2.2.1, 3.1, 6

[TNCC03] TANTER, É.; NOYÉ, J.; CAROMEL, D. ; COINTE, P.. **Partial behavioral reflection: spatial and temporal selection of reification.** In: OOPSLA '03: PROCEEDINGS OF THE 18TH ANNUAL ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMING, SYSTEMS, LANGUAGES, AND APPLICATIONS, p. 27–46, New York, NY, USA, 2003. ACM. 6

[TRVC+00] TRUYEN, E.; ROBBEN, B.; VANHAUTE, B.; CONINX, T.; JOOSEN, W. ; VERBAETEN, P.. **Portable support for transparent thread migration in Java.** In: PROCEEDINGS OF THE JOINT SYMPOSIUM ON AGENT SYSTEMS AND APPLICATIONS / MOBILE AGENTS (ASA/MA), p. 29–43, 2000. 2.1, 2.2.3

[TVP02] TANTER, E.; VERNAILLEN, M. ; PIQUER, J.. **Towards Transparent Adaptation of Migration Policies.** In: 8TH ECOOP WORKSHOP ON MOBILE OBJECT SYSTEMS (EWMOS 2002), Málaga, Spain, June 2002. 3.2

[WHB01] WANG, X.; HALLSTROM, J. ; BAUMGARTNER, G.. **Reliability through strong mobility.** In: PROCEEDINGS OF THE 7TH ECOOP WORKSHOP ON MOBILE OBJECT SYSTEMS: DEVELOPMENT OF ROBUST AND HIGH CONFIDENCE AGENT APPLICATIONS (MOS '01), p. 1–13, 2001. 2.1

[ZG95] ZHOU, H.; GEIST, A.. **'Receiver Makes Right' data conversion in PVM.** In: PROCEEDINGS OF THE 14TH INTERNATIONAL CONFERENCE ON COMPUTERS AND COMMUNICATIONS, p. 458–464, 1995. 2.2.1

A
Biblioteca de serialização

Este arquivo é uma extensão à biblioteca de serialização do LOOP [Maia08]. LOOP (Lua Object-Oriented Programming) é um conjunto de pacotes para a implementação de diferentes modelos de programação orientada a objeto em Lua. Devido às limitações da versão corrente de Lua, não é possível manter o compartilhamento de upvalues na restauração, nem capturar co-rotinas. Estas limitações podem ser resolvidas através do uso dos mecanismos de reificação e instalação de LuaNua. A seguir se lista o arquivo que extende a biblioteca de serialização do LOOP para permitir o acesso a estes mecanismos.

```
local _G = _G
local getmetatable = getmetatable
local setmetatable = setmetatable
local getfenv = getfenv
local setfenv = setfenv
local package = package
local assert = assert
local select = select
local pairs = pairs
local pcall = pcall
local ipairs = ipairs
local loadstring = loadstring
local rawget = rawget
local rawset = rawset
local require = require
local tostring = tostring
local tonumber = tonumber
local error = error
local type = type

local debug = debug
local string = require "string"
local table = require "loop.table"
local oo = require "loop.simple"
local coroutine = require "coroutine"
local Serializer = require "loop.serial.Serializer"
```

```
local print = print
module "loop.serial.LuaNuaSerializer"

oo.class(_M, Serializer)

__mode = "k"

function value(self, id, type, ...)
local value = self[id]
if not value then
if type == "proto" then
value = debug.install(..., "proto")
elseif type == "upval" then
value = debug.install(0, "upval")
elseif type == "function" then
value = debug.install(..., "function")
elseif type == "thread" then
value = debug.newthread()
else
return Serializer.value(self, id, type, ...)
end
self[id] = value
else
return Serializer.value(self, id, type, ...)
end
return value
end

--------------------------------------------------------------------------

function serialthread(self, thread, id)
self[thread] = self.namespace..":value(..id..)"
self:write(self.namespace,":setup(")
self:write(self.namespace,":value(",id,",'thread'),")
        local i = 0
        local ci = {}
        repeat
        ci[i] = debug.content(thread, i)
        i = i + 1
until (ci[i-1]==nil)
self:serialize(ci)
```

```
self:write(",")
self:serialize(coroutine.status(thread))
self:write(",")
self:serialize(debug.getopenupvals(thread))
self:write(")")
end

function serialupvalue(self, upvalue, id)
self[upvalue] = self.namespace..":value("..id..")"

-- serialize contents
self:write(self.namespace,":value(",id,",'upval')")
end

-- Recebe um proto
function serialproto(self, proto, id)
self[proto] = self.namespace..":value("..id..")"

-- serialize contents
self:write(self.namespace,":value(",id,",'proto',")
local content = debug.content(proto)
self:serialize(content)
self:write(")")
end

function serialfunction(self, func, id)
self[func] = self.namespace..":value("..id..")"
local content = debug.content(func)

if content.isC==1 then --Alarm, it should never happen
error ("C functions cannot be serialized")
else
self:write(self.namespace,":setup(")

-- serialize contents
self:write(self.namespace,":value(",id,",'function',")
self:serialize(content)
self:write(")")


-- serialize environment
local env
```

```
if self.getfenv then
env = self.getfenv(func)
if env == self.globals then env = nil end
end
self:write(",")
self:serialize(env)

local nups = debug.getinfo(func,"u").nups
self:write(",",nups)
        self:write(",",id) --> Debug

-- serialize upvalues contents
for _, upval in ipairs(content.upvals or {}) do
self:write(",")
self:serialize(debug.content(upval))
end

self:write(")")
end
end

_M["upval"]    = serialupvalue
_M["proto"]    = serialproto
_M["function"] = serialfunction
_M["thread"] = serialthread
```

Na biblioteca Serializer original somente foi modificada a função setup, mostrada a seguir:

```
function setup(self, value, ...)
local type = type(value)
if type == "function" then
if self.setfenv then self.setfenv(value, ... or self.globals) end
local nups = select(2, ...)
local setupvalue = self.setupvalue
if setupvalue then
for i=1, nups do
setupvalue(value, i, select(3+i, ...) or nil)
end
end

elseif type == "thread" then
local ci = select(1, ...)
```

```
for i = #ci,0,-1 do
value = debug.install(ci[i], value, 0)
end

local status = select(2, ...)
local openupvals = select(3, ...)
if openupvals then debug.openupvals(openupvals,value) end
debug.setstatus(value, status)
else
local loader = getmetatable(value)
if loader then loader = loader.__load end
if loader then loader(value, ...) end
end
return value
end
```