8
Referências

AGERE SYSTEMS. **Agere network processors**. http://www.agere.com/, 2001. Acesso em: 29 mai. 2004.

AGHA, G. A.; THATI, P. ; ZIAEI, R. **Actors: a Model for Reasoning About Open Distributed Systems**, section 3.1, p. 155–176. Cambridge University Press, New York, EUA, 2001.

ALLEN, R. **A Formal Approach to Software Architecture**. Pittsburgh, Pennsylvania, jan. 1997, 248 p. Tese de doutorado, Carnegie Mellon University, School of Computer Science. Issued as CMU Technical Report CMU-CS-97-144.

ALLEN, R. J.; DOUENCE, R. ; GARLAN, D. Specifying and analysing software architectures. In: INTERNATIONAL CONFERENCE ON FUNDAMENTAL APPROACHES TO SOFTWARE ENGINEERING (FASE98). **Proceedings...** Lisboa, Portugal, mar. 1998.

APPLETON, B. **Sclc and cdiff: Perl scripts for ClearCase**. http://www.cmcrossroads.com/bradapp/clearperl/sclc-cdiff.html, 2003. Acesso em: 9 fev. 2005.

AUJLA, S.; BRUYANT, T. ; SEMMENS, L. Applying formal methods within structured development. **Journal of Selected Areas of Communications**, v. 12, n. 2, p. 258–264, 1994.

AURRECOECHEA, C.; CAMPBELL, A. T. ; HAUW, L. A review of QoS architectures. **ACM Multimedia Systems Journal**, v. 6, n. 3, p. 138–151, nov. 1998.

BASU, A.; HAYDEN, M.; MORRISETT, G. ; VON EICKEN, T. A language-based approach to protocol construction. In: Kamin, S. (Ed.), FIRST ACM SIGPLAN WORKSHOP ON DOMAIN-SPECIFIC LANGUAGES. **Computer Science Reports**. p. 1–15, Urbana, EUA, 1997. University of Illinois.

BATORY, D. S.; GERACI, B. J. Composition validation and subjectivity in GenVoca generators. **IEEE Transactions on Software Engineering**, v. 23, n. 2, p. 67–84, jan. 1997.

BELLISSARD, L.; DE PALMA, N. ; FÉLIOT, D. **The Olan architecture definition language**. Relatório técnico, INRIA Rhône Alpes, Montbonnot Saint-Martain, França, 1998.

BERNDT, H.; GRAUBMANN, P. ; WAKANO, M. Service specification concepts in TINA-C. In: Kugler, H. J.; Mullery, A. ; Niebert, N. (Eds.), SECOND INTERNATIONAL CONFERENCE ON INTELLIGENCE IN SERVICES & NETWORKS (IS&N94). volume 851 de **Lecture Notes in Computer Science**. p. 355–366, Heidelberg, Alemanha, 1994. Springer-Verlag.

BLAIR, L.; BLAIR, G. S. Composition in multiparadigm specification techniques. In: IFIP TC6/WG6.1 THIRD INTERNATIONAL CONFERENCE ON FORMAL METHODS FOR OPEN OBJECT-BASED DISTRIBUTED SYSTEMS (FMOODS). **Proceedings...** p. 401–417, Deventer, Holanda, 1999. Kluwer, B.V.

BLAIR, L.; BLAIR, G. S.; ISSARNY, V.; TUMA, P. ; ZARRAS, A. The role of software architecture in constraining adaptation in component-based middleware platforms. In: INTERNATIONAL CONFERENCE ON MIDDLEWARE. **Proceedings...** 2000.

BLAIR, G. S.; COULSON, G.; ANDERSEN, A.; BLAIR, L.; CLARKE, M.; COSTA, F. M.; DURÁN-LIMÓN, H. A.; FITZPATRICK, T.; JOHNSTON, L.; MOREIRA, R.; PARLAVANTZAS, N. ; SAIKOSKI, K. The design and implementation of OpenORB version 2. **IEEE Distributed Systems Online Journal**, v. 2, n. 6, jun. 2001.

BONACHEA, D.; FISHER, K.; ROGERS, A. ; SMITH, F. Hancock: A language for processing very large-scale data. In: SECOND USENIX CONFERENCE ON DOMAIN-SPECIFIC LANGUAGES. p. 163–176, out. 1999.

BRADEN, R.; ZHANG, L.; BERSON, S.; HERZOG, S. ; JAMIN, S. **RFC2205: Resource ReSerVation Protocol (RSVP) – version 1 functional specification**, 1997.

BRUNETON, E.; COUPAYE, T. ; STEFANI, J. B. Recursive and dynamic software composition with sharing. In: SEVENTH INTERNATIONAL WORKSHOP ON COMPONENT-ORIENTED PROGRAMMING (WCOP02). **Proceedings...** Málaga, Espanha, jun. 2002.

CALDER, M.; KOLBERG, M.; MAGILL, E. H. ; REIFF-MARGANIEC, S. Feature interaction: A critical review and considered forecast. **Computer Networks**, v. 41, n. 1, p. 115–141, jan. 2003.

CAMPBELL, A. T.; DE MEER, H. G.; KOUNAVIS, M. E.; MIKI, K.; VICENTE, J.; VILLELA, D. A. A survey of programmable networks. **ACM SIGCOMM Computer Communications Review**, v. 29, n. 2, p. 7-23, abr. 1999.

CHAN, K.; SELIGSON, J.; DURHAM, D.; GAI, S.; MCCLOGHRIE, K.; HERZOG, S.; REICHMEYER, F.; YAVATKAR, R. ; SMITH, A. **RFC3084: COPS usage for policy provisioning (COPS-PR)**, 2001.

COLCHER, S. **Um metamodelo para aplicações e serviços de comunicação adaptáveis e com qualidade de serviço**. Rio de Janeiro, Brasil, nov. 1999, 125 p. Tese de doutorado, Departamento de Informática - PUC-Rio.

COULSON, G.; BLAIR, G. S.; HUTCHISON, D.; JOOLIA, A.; LEE, K.; UEYAMA, J.; GOMES, A. T. A.; YE, Y. NETKIT: A software component-based approach to programmable networking. **ACM SIGCOMM Computer Communications Review**, v. 33, n. 5, p. 55–66, out. 2003.

COULSON, G.; BLAIR, G. S.; GRACE, P.; JOOLIA, A.; LEE, K. ; UEYAMA, J. OpenCOM v2: a component model for building systems software. In: IASTED SOFTWARE ENGINEERING AND APPLICATIONS. **Proceedings...** Cambridge, Massachusetts, nov. 2004.

DISTRIBUTED MULTIMEDIA RESEARCH GROUP. **DMRG home page**. http://www.comp.lancs.ac.uk/computing/research/mpg/, 2004. Acesso em: 19 fev. 2005.

DA SILVA, S.; FLORISSI, D. ; YEMINI, Y. Composing active services in NetScript. In: DARPA ACTIVE NETWORKS WORKSHOP. **Proceedings...** Tucson, Arizona, mar. 1998.

DASHOFY, E. M.; DER HOEK, A. V. ; TAYLOR, R. N. An infrastructure for the rapid development of XML-based architecture description languages. In: 24TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. **Proceedings...** p. 266–276, Orlando, Florida, 2002. ACM Press.

DE SOUZA, C. T.; CUNHA, P. R. F. A calculus for reconfigurable component-based systems. In: XXIX CONFERENCIA LATINOAMERICANA DE INFORMATICA. **Proceedings...** set. 2003.

DIETRICH, F. **Modelling Object-Oriented Communcation Services with Temporal Logic**. Lausanne, Suíça, 1999, 183 p. Tese de doutorado, Section de Systèmes de Communication, École Polytechnique Fédérale de Lausanne.

DIETRICH, F.; HUBAUX, J. P. Formal methods for communication services: Meeting the industry expectations. **Computer Networks**, v. 38, n. 1, p. 99–120, jan. 2002.

DURHAM, D.; BOYLE, J.; COHEN, R.; HERZOG, S.; RAJAN, R. ; SASTRY, A. **RFC2748: The COPS (common open policy service) protocol**, 2000.

DURÁN-LÍMON, H. A. **A Resource Management Framework for Reflective Multimedia Middleware**. Lancaster, UK, 2001, 233 p. Tese de doutorado, Lancaster University, Computing Department.

FASSINO, J. P.; STEFANI, J. B.; LAWALL, J. L. ; MULLER, G. Think: A software framework for component-based operating system kernels. In: USENIX ANNUAL TECHNICAL CONFERENCE, GENERAL TRACK. **Proceedings...** p. 73–86. USENIX, 2002.

FONTOURA, M. F. M. C. **A Systematic Approach to Framework Development**. Rio de Janeiro, Brasil, jul. 1999, 165 p. Tese de doutorado, Departamento de Informática - PUC-Rio.

FORMAL SYSTEMS (EUROPE) LIMITED. **The FDR2 model-checker**. http://www.fsel.com/, 2003. Acesso em: 4 fev. 2005.

GOMES, A. T. A. **Um framework para provisão de QoS em ambientes genéricos de processamento e comunicação**. Rio de Janeiro, Brasil, mai. 1999, 162 p. Dissertação de mestrado, Departamento de Informática, PUC-Rio.

GOMES, A. T. A.; COLCHER, S. ; SOARES, L. F. G. Modelling QoS provision on adaptable communication environments. In: INTERNATIONAL CONFERENCE ON COMMUNICATIONS (ICC). **Proceedings...** p. 1221–1226, Helsinki, Finland, jun. 2001.

____. Towards a descriptive approach to model adaptable communication environments. In: INTERNATIONAL CONFERENCE ON NETWORKING (ICN). Volume 2094 de **Lecture Notes in Computer Science**. p. 867–876, Heidelberg, Alemanha, 2001. Springer-Verlag.

GOMES, A. T. A.; COULSON, G.; BLAIR, G. S.; SOARES, L. F. G. **A component-based approach to the creation and development of network services in the programmable Internet**. MCC 42/03, PUC-Rio, Rio de Janeiro, Brasil, 2003.

GOMES, A. T. A.; SOARES, L. F. G. Uma abordagem de especificação e imposição de restrições de reconfiguração em redes programáveis. In: XXIII SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES. **Anais...** p. 205–218, Fortaleza, Brasil, mai. 2005. Sociedade Brasileira de Computação.

GRASDIJK, M.; DRETELER, J.; BRAUX, H. ; LE-BAIL, J. L. Modelling services in the portfolio from a service provisioning perspective. In: Kugler, H. J.; Mullery, A. ; Niebert, N. (Eds.), SECOND INTERNATIONAL CONFERENCE ON INTELLIGENCE IN SERVICES & NETWORKS (IS&N94). volume 851 de **Lecture Notes in Computer Science**. p. 133–143, Heidelberg, Alemanha, 1994. Springer-Verlag.

HASELTON, E. F. Service creation environments for intelligent networks. **IEEE Communications Magazine**, v. 30, n. 2, p. 78–81, fev. 1992.

HOARE, C. A. R. **Communicating Sequential Processes**. Prentice-Hall International, New Jersey, EUA, 1985.

IERUSALIMSCHY, R.; FIGUEIREDO, L. H.; CELES, W. **Lua 5.0 reference manual**. MCC 14/03, PUC-Rio, Rio de Janeiro, Brazil, 2003.

INTEL CORPORATION. **IXA network processors**. http://www.intel.com/, 2004. Acesso em: 29 mai. 2004.

INTERNATIONAL BUSINESS MACHINE CORPORATION. **PowerNP network processors**. http://www.ibm.com/, 2004. Acesso em: 29 mai. 2004.

INTERNATIONAL ORGANISATION FOR STANDARDISATION. **ISO/IEC 7498: Open systems interconnection – basic reference model**, 1984.

____. **ISO/IEC 8807: Open systems interconnection – LOTOS – a formal description technique based on the temporal ordering of observational behaviour**, 1989.

____. **ISO/IEC 9074: Open systems interconnection – ESTELLE – a formal description technique based on an extended state transition model**, 1989.

____. **ISO/IEC 10746-1: Reference model of open distributed processing, part 1: Overview**, 1995.

____. **ISO/IEC 14977: Information Technology: Syntactic Metalanguage: Extended BNF**, 1996.

____. **Joint Technical Committee 1**. http://www.jtc1.org/, 2001. Acesso em: 25 jun. 2004.

INTERNATIONAL TELECOMMUNICATIONS UNION. **ITU-T Recommendation Z.100: Specification and description language**, 1992.

____. **ITU-T Recommendation I.312/Q.1201: Principles of intelligent network architecture**, 1992.

____. **ITU-T Recommendation Q.700: Introduction to CCITT signalling system no. 7**, 1993.

ISSARNY, V.; BIDAN, C. Aster: a framework for sound customization of distributed runtime systems. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS. **Proceedings...** Hong Kong, Taiwan, mai. 1996.

JAVACC. **Java Compiler Compiler – JavaCC (tm)**. https://javacc.dev.java.net/, 2003. Acesso em: 2 fev. 2005.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. **IEEE Computer**, v. 36, n. 1, p. 41–50, jan. 2003.

KICZALES, G.; DES RIVIÈRES, J. ; BOBROW, D. G. **The Art of the Metaobject Protocol**. MIT Press, Cambridge, EUA, 1991.

KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C. V.; LOINGTIER, J. M. ; IRWIN, J. Aspect-oriented programing. In: ELEVENTH EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING (ECOOP97). volume 1241 de **Lecture Notes in Computer Science**. p. 220–242, Heidelberg, Alemanha, 1997. Springer-Verlag.

KLARLUND, N.; SCHWARTZBACK, M. I. A domain-specific language for regular sets of strings and trees. **IEEE Transactions on Software Engineering**, v. 25, n. 3, p. 378–386, mai. 1999.

KOLBERG, M.; SINNOTT, R. O.; MAGILL, E. H. Experiences modelling and using formal object-oriented telecommunication service frameworks. **Computer Networks**, v. 31, n. 1, 1999.

KONSTANTINOU, A. **Towards autonomic networks**. New York, EUA, 2003, 202 p. Tese de doutorado, Columbia University, Department of Computer Science.

KOSMAS, N.; TURNER, K. J. Requirements for service creation environments. In: Lovrek, I. (Ed.), INTERNATIONAL WORKSHOP ON APPLIED FORMAL METHODS IN SYSTEM DESIGN. **Proceedings...** p. 133–137, Zagreb, Croácia, jun. 1997.

KUGLER, H. J.; MULLERY, A. P.; NIEBERT, N. (Eds.). Towards a pan-european telecommunication service infrastructure, volume 851 de **Lecture Notes in Computer Science**, Heildelberg, Alemanha, 1994. Springer-Verlag.

LABORATÓRIO TELEMÍDIA. **LindaX: An eXtensible description language for adaptable communication systems**. http://www.telemidia.puc-rio.br/products/lindax/, 2004. Acesso em: 2 fev. 2005.

LADD, D. A.; RAMMING, J. C. Two application languages in software production. In: USENIX VERY HIGH LEVEL LANGUAGES SYMPOSIUM. **Proceedings...** p. 169–178, out. 1994.

LOGEAN, X. **Run-time Monitoring and On-line Testing of Middleware-based Communication Services**. Lausanne, Suíça, 1999. Tese de doutorado, Section de Systèmes de Communication, École Polytechnique Fédérale de Lausanne.

LUCKHAM, D.; VERA, J. An event-based architecture definition language. **IEEE Software**, v. 21, n. 9, p. 717–734, set. 1995.

MAES, P. Concepts and experiments in computational reflection. In: ANNUAL CONFERENCE ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS. **Proceedings...**. p. 147–155, Orlando, Florida, 1987. ACM Press.

MAGEE, J.; KRAMER, J. Dynamic structure in software architectures. In: SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING. **Proceedings...**. p. 3–14, San Francisco, EUA, out. 1996.

MEDVIDOVIC, N.; TAYLOR, R. A classification and comparison framework for software architecture description languages. **IEEE Software**, v. 14, n. 1, p. 70–93, jan. 2000.

MILNER, R. **Calculi for Mobile Processes: the Pi-calculus**. Cambridge University Press, New York, EUA, 1999.

MÖLLER, A.; ÅKERHOLM, M.; FREDRIKSSON, J. ; NOLIN, M. Evaluation of component technologies with respect to industrial requirements. In: 30TH EUROMICRO CONFERENCE. **Proceedings...** p. 56–63, Rennes, França, set. 2004. IEEE Computer Society.

MONROE, R. T.; KOMPANEK, A.; MELTON, R.; GARLAN, D. Architectural styles, design patterns, and objects. **IEEE Software**, v. 14, n. 1, p. 43–52, jan. 1997.

MONROE, R. T. **Capturing software architecture design expertise with Armani**. CMU-CS-98-163, Carnegie Mellon University, Pittsburgh, EUA, 1998.

MORENO, M. F. **Um framework para provisão de QoS em sistemas operacionais**. Rio de Janeiro, Brasil, mai. 2002, 117 p. Dissertação de mestrado, Departamento de Informática, PUC-Rio.

MOREIRA, R. J. S. **FORMAware: Framework of Reflective Components for Managing Architecture Adaptation**. Lancaster, UK, 2003, 213 p. Tese de doutorado, Lancaster University, Computing Department.

MOTA, O. T. J. D. D. L. **Uma arquitetura adaptável para provisão de QoS na Internet**. Rio de Janeiro, Brasil, mai. 2001, 113 p. Dissertação de mestrado, Departamento de Informática, PUC-Rio.

MOY, J. T. **OSPF: Anatomy of an Internet Routing Protocol**. Addison-Wesley Publishing Company, Reading, Massachusetts, 1a. edição, 1998.

MUDHAR, P. A service creation environment for a future intelligent network. In: Kugler, H. J.; Mullery, A. ; Niebert, N. (Eds.), SECOND INTERNATIONAL CONFERENCE ON INTELLIGENCE IN SERVICES & NETWORKS (IS&N94). volume 851 de **Lecture Notes in Computer Science**. p. 333–342, Heidelberg, Alemanha, 1994. Springer-Verlag.

NETWORK PROCESSING FORUM. **History and milestones**. http://www.npforum.org/about/, 2001. Acesso em: 27 set. 2004.

OBJECT MANAGEMENT GROUP. **The Common Object Request Broker: Architecture and specification - revision 2**, 1997.

____. **Unified Modeling Language specification – version 1.5**, 2003.

OBJECTWEB. **The Fractal project**. http://fractal.objectweb.org/, 2004. Acesso em: 11 fev. 2005.

OUSTERHOUT, J. K. **Tcl and the Tk Toolkit**. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

OUTHRED, G.; POTTER, J. A model for component composition with sharing. In: THIRD INTERNATIONAL WORKSHOP ON COMPONENT-ORIENTED PROGRAMMING (WCOP98). **Proceedings...** p. 29–38, Turku, Finlândia, out. 1998. Turku Centre for Computer Science.

PONTEN, L.; HALLSTRAND, J. ; MARQUES, M. M. Building dedicated service creation environments for reuse-based production. In: Kugler, H. J.; Mullery, A. ; Niebert, N. (Eds.), SECOND INTERNATIONAL CONFERENCE ON INTELLIGENCE IN SERVICES & NETWORKS (IS&N94). volume 851 de **Lecture Notes in Computer Science**. p. 169–178, Heidelberg, Alemanha, 1994. Springer-Verlag.

PREE, W. **Design Patterns for Object-Oriented Software Development**. **ACM Press**. Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

RADISYS CORPORATION. **ENP-2611 data sheet: PCI packet processing engines**. http://www.radisys.com/, 2004. Acesso em: 26 jan. 2005.

RATIONAL SOFTWARE CORPORATION. **Unified modeling language: Notation guide**. http://www.rational.com/uml/resources/documentation, 1997. Acesso em: 19 jan. 2005.

REID, A.; FLATT, M.; STOLLER, L.; LEPREAU, J. ; EIDE, E. Knit: Component composition for systems software. In: FOURTH SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION (OSDI2000). **Proceedings...** p. 347–360, San Diego, California, out. 2000.

ROBLES, T.; HUECAS, G.; QUEMADA, J.; VERDEJO, A.; LLANA-DÍAZ, L. F. **Process calculi: E-LOTOS**, section 2.2, p. 77–104. Cambridge University Press, New York, EUA, 2001.

RODRIGUES, M. A. A. **Um framework para provisão de serviço de multicast em ambientes genéricos de comunicação de dados**. Rio de Janeiro, Brasil, mai. 1999. Dissertação de mestrado, Departamento de Informática, PUC-Rio.

ROSA, N. S.; JUSTO, G. R. R.; CUNHA, P. R. F. An approach for reasoning and refining non-functional requirements. **Journal of the Brazilian Computer Society**, v. 10, n. 1, p. 59–77, 2004.

SCHMIDT, D.; SUDA, T. Transport system architecture services with high-performance communication systems. **Journal of Selected Areas of Communications**, v. 11, n. 4, p. 489–506, 1993.

SCHULZE, B.; NANDKUMAR, R. (Eds.). SECOND INTERNATIONAL WORKSHOP ON MIDDLEWARE FOR GRID COMPUTING (MGC2004), **Proceedings...**, Nova York, EUA, 2004. ACM Press.

SILBERSCHATZ, A.; KORTH, H. F. ; SUDARSHAN, S. **Database System Concepts**. McGraw-Hill Book Company, New York, EUA, 4 edição, 2001.

SINNOTT, R. O.; HOGREFE, D. **Finite State Machine Based: SDL**, section 2.1, p. 55–76. Cambridge University Press, New York, EUA, 2001.

SOARES, L. F. G.; LEMOS, G. ; COLCHER, S. **Redes de Computadores: das LANs, MANs e WANs as Redes ATM**. Editora Campus, Rio de Janeiro, Brasil, 2 edição, 1995.

SOARES-NETO, C. S. **Descrição arquitetural de provisão de QoS em ambientes genéricos de processamento e comunicação**. Rio de Janeiro, Brasil, ago. 2003, 115 p. Dissertação de mestrado, Departamento de Informática - PUC-Rio.

SOARES-NETO, C. S.; MORENO, M. F.; GOMES, A. T. A. ; SOARES, L. F. G. Descrição arquitetural da provisão de QoS para suporte a aplicações multimídia.

In: IX SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E WEB. **Anais...** Salvador, Brasil, nov. 2003. Sociedade Brasileira de Computação

SOARES-NETO, C. S.; RODRIGUES, R. F. ; SOARES, L. F. G. Architectural description of qos provisioning for multimedia application support. In: TENTH INTERNATIONAL MULTIMEDIA MODELLING CONFERENCE. **Proceedings...** p. 161–166, Brisbane, Australia, jan. 2004.

SZYPERSKI, C. **Component Software: Beyond Object-oriented Programming**. Addison-Wesley Publishing Company, Reading, Massachusetts, 2a. edição, 2002.

TAYLOR, R. N.; MEDVIDOVIC, N.; ANDERSON, K. M.; WHITEHEAD, E. J.; ROBBINS, J. E.; NIES, K. A.; OREIZY, P.; DUBROW, D. A component- and message-based architectural style for GUI software. **IEEE Transactions on Software Engineering**, v. 22, n. 6, p. 390–406, jun. 1996.

TENNENHOUSE, D. L.; SMITH, J. M.; SINCOSKIE, W. D.; WETHERALL, D. J.; MINDEN, G. J. A survey of active network research. **IEEE Communications Magazine**, v. 35, n. 1, p. 80–86, jan. 1997.

THE REGENTS OF THE UNIVERSITY OF CALIFORNIA. **ArchStudio 3: a software architecture-based development environment**. http://www.isr.uci.edu/projects/archstudio/, 2004. Acesso em: 9 fev. 2004.

UEYAMA, J.; COULSON, G.; BLAIR, G. S.; SCHMID, S.; GOMES, A. T. A.; JOOLIA, A.; LEE, K. A globally-applied component model for programmable networking. In: Wakamiya, N.; Solarski, M.; Sterbenz, J. P. G. (Eds.), FIFTH INTERNATIONAL WORKING CONFERENCE ON ACTIVE NETWORKS (IWAN2003). volume 2982 de **Lecture Notes in Computer Science**. p. 202–214, Heidelberg, Alemanha, 2003. Springer-Verlag.

VAN DEURSEN, A.; KLINT, P. ; VISSER, J. Domain-specific languages: An annotated bibliography. **ACM SIGPLAN Notices**, v. 35, n. 6, p. 26–36, jun. 2000.

WORLD WIDE WEB CONSORTIUM. **XML schema part 0: Primer**. http://www.w3.org/TR/xmlschema-0, 2001. Acesso em: 19 jan. 2005.

WALL, L.; SCHWARTZ, R. L. **Programming perl**. O'Reilly & Associates, Inc., 1991.

WROCLAWSKI, J. **RFC22210: The use of RSVP with IETF integrated services**, 1997.

ZAREMSKI, A. M.; WING, J. M. Specification matching of software components. **ACM Transactions on Software Engineering and Methodology**, v. 6, n. 4, p. 333–369, out. 1997.

ZNATY, S.; HUBAUX, J. P. Telecommunications services engineering: Definitions, architectures and tools. In: Bosch, J.; Mitchell, S. (Eds.), ECOOP 97 WORKSHOPS. volume 1357 de **Lecture Notes in Computer Science**. p. 3–11, Jyvaskyla, Finland, fev. 1998. Springer.

9
Apêndice A

Neste apêndice são apresentados alguns detalhes de implementação das ferramentas do ambiente LindaStudio e da instanciação do *framework* para gerência de adaptações na plataforma OpenCOM.

9.1.
Implementação do LindaStudio

O ambiente LindaStudio consiste em aproximadamente 35.000 linhas de código Java (desconsiderando linhas delimitadoras e comentários[1]), distribuídas por 29 pacotes e 312 arquivos-fonte. Cerca de 40% desse código é dedicado à integração de LindaStudio com o ambiente ArchStudio 3.

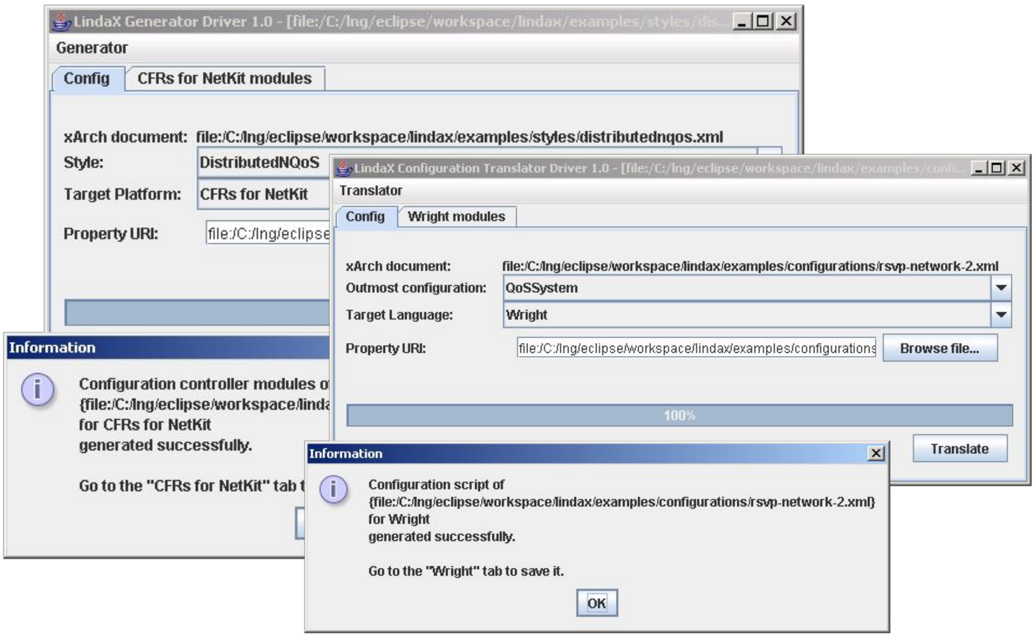
Originalmente, o ambiente ArchStudio 3 oferece uma ADL, chamada xADL (Dashofy et al., 2002), que é definida a partir de um conjunto de esquemas XML derivados do esquema central `xArch`, bem como um conjunto de ferramentas específicas para essa linguagem. Para tornar o ambiente LindaStudio mais "leve"[2], as ferramentas do ambiente ArchStudio específicas para xADL foram removidas.

As telas dos *drivers* das ferramentas `Translator` e `Generator` são apresentadas na Figura 9.1. A Figura 9.2 ilustra uma captura de tela com as outras ferramentas que compõem o ambiente LindaStudio: `FileManager/Invoker`, `ArchEdit`, `CriticGUI` (integrantes do ArchStudio), `StyleEditor` e `ConfEditor` (introduzidas pelo LindaStudio).

Os editores de estilos e configurações (`StyleEditor` e `ConfEditor`) tiveram seus *parsers* da notação sem *tags* introduzida no Capítulo 3 gerados e implementados com a biblioteca JavaCC (2003).

[1] Para obter essa estatística, foi usada a ferramenta `sclc` (Appleton, 2003), com as opções `-delim-ignore` e `-counts ncsl`.

[2] Para sistemas complexos, o estilo C2 pode impor uma sobrecarga considerável de processamento devido ao excesso de mensagens difundidas pelos conectores-barramento sendo recebidas e descartadas por componentes para os quais essas mensagens não são relevantes.



LindaX Generator Driver 1.0 - [file:/C:/lng/eclipse/workspace/lindax/examples/styles/dis...

xArch document: file:/C:/lng/eclipse/workspace/lindax/examples/styles/distributednqos.xml
Style: DistributedNQoS
Target Platform: CFRs for NetKit
Property URI: file:/C:/lng/eclipse

Information
Configuration controller modules o
{file:/C:/lng/eclipse/workspace/linda
for CFRs for NetKit
generated successfully.

Go to the "CFRs for NetKit" tab t

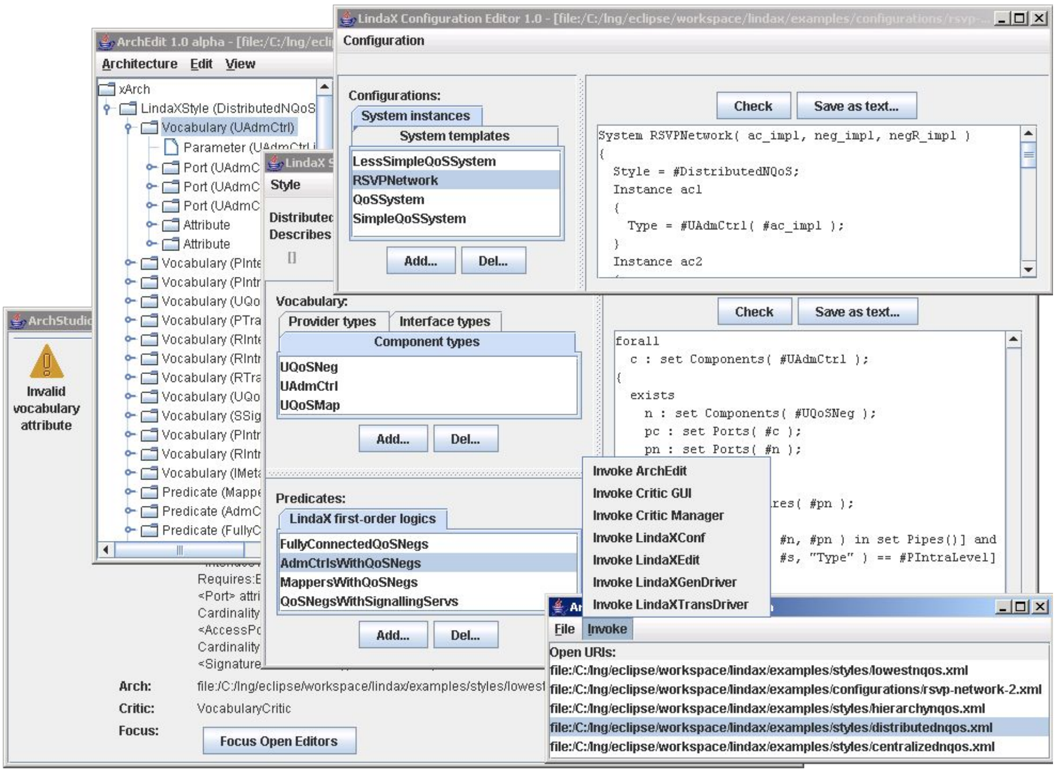
LindaX Configuration Translator Driver 1.0 - [file:/C:/lng/eclipse/workspace/lindax/examples/confi...

xArch document: file:/C:/lng/eclipse/workspace/lindax/examples/configurations/rsvp-network-2.xml
Outmost configuration: QoSSystem
Target Language: Wright
Property URI: file:/C:/lng/eclipse/workspace/lindax/examples/configurations

Information
Configuration script of
{file:/C:/lng/eclipse/workspace/lindax/examples/configurations/rsvp-network-2.xml}
for Wright
generated successfully.

Go to the "Wright" tab to save it.

Figura 9.1. *Drivers* das ferramentas `Translator` e `Generator`.



ArchEdit 1.0 alpha - [file:/C:/lng/ecli
xArch
LindaXStyle (DistributedNQoS
Vocabulary (UAdmCtrl)
Parameter (UAdmCtrl
Port (UAdmC
Port (UAdmC
Port (UAdmC
Attribute
Attribute
Vocabulary (PInte
Vocabulary (PIntr
Vocabulary (UQo
Vocabulary (PTra
Vocabulary (RInte
Vocabulary (RIntr
Vocabulary (RTra
Vocabulary (UQo
Vocabulary (SSig
Vocabulary (PIntr
Vocabulary (RIntr
Vocabulary (IMeta
Predicate (Mappe
Predicate (AdmC
Predicate (FullyC

LindaX Configuration Editor 1.0 - [file:/C:/lng/eclipse/workspace/lindax/examples/configurations/rsvp-...

Configurations:
System instances: LessSimpleQoSSystem, RSVPNetwork, QoSSystem, SimpleQoSSystem

```
System RSVPNetwork( ac_impl, neg_impl, negR_impl )
{
  Style = #DistributedNQoS;
  Instance ac1
  {
    Type = #UAdmCtrl( #ac_impl );
  }
  Instance ac2
```

Style: Distributed
Describes
[]

Vocabulary:
Component types: UQoSNeg, UAdmCtrl, UQoSMap

```
forall
  c : set Components( #UAdmCtrl );
{
  exists
    n : set Components( #UQoSNeg );
    pc : set Ports( #c );
    pn : set Ports( #n );
...res( #pn );
...#n, #pn ) in set Pipes()] and
...#s, "Type" ) == #PIntraLevel]
```

Predicates:
LindaX first-order logics: FullyConnectedQoSNegs, AdmCtrlsWithQoSNegs, MappersWithQoSNegs, QoSNegsWithSignallingServs

ArchStudio
Invalid vocabulary attribute
Requires:E
<Port> attri
Cardinality
<AccessPo
Cardinality
<Signature
Arch: file:/C:/lng/eclipse/workspace/lindax/examples/styles/lowest
Critic: VocabularyCritic
Focus:

Open URIs:
file:/C:/lng/eclipse/workspace/lindax/examples/styles/lowestnqos.xml
file:/C:/lng/eclipse/workspace/lindax/examples/configurations/rsvp-network-2.xml
file:/C:/lng/eclipse/workspace/lindax/examples/styles/hierarchynqos.xml
file:/C:/lng/eclipse/workspace/lindax/examples/styles/distributednqos.xml
file:/C:/lng/eclipse/workspace/lindax/examples/styles/centralizednqos.xml

Figura 9.2. Captura de tela do ambiente LindaStudio.

A seguir são apresentadas as interfaces Java das ferramentas `Generator`, `Translator` e `Expander`.

9.1.1. Interface de Generator

```
public abstract interface IGenerator {
  /* retorna lista de scripts de verificação */
  protected abstract Hashtable generate(
   String url,    // localização do documento LindaX
   String objId,  // ID do estilo
   Hashtable symb // lista de valores para propriedades parametrizadas
  ) throws Exception;
}
```

9.1.2. Interface de Translator

```
public abstract interface ITranslator {
  /* retorna lista de scripts de configuração */
  protected abstract Hashtable translate(
   String url,    // localização do documento LindaX
   String objId,  // ID do sistema ou template
   Hashtable symb // lista de valores para propriedades parametrizadas
  ) throws Exception;
}
```

9.1.3. Interface de Expander

```
public abstract interface IExpander {
  /* retorna estrutura com informação sobre o conjunto de instâncias
     inferidas a partir de uma cláusula “Instance” */
  public abstract InstanceInfo expandInstance(
   ObjRef instRef // ref. a um elemento XML de um documento LindaX
                  //  que representa uma instância
  ) throws Exception;

  /* retorna estrutura com informação sobre o tipo do pipe
     e sobre quais portas dos componentes estão ligadas a ele,
     inferidos a partir de uma cláusula “Pipe” */
  public abstract PipeInfo expandPipe(
   ObjRef pipeRef // ref. a um elemento XML de um documento LindaX
                  //  que representa um pipe
  ) throws Exception;

  /* retorna estrutura com informação sobre o tipo da porta
     externalizada por um componente ou sistema,
     inferido a partir de uma cláusula “Mapping” */
  public abstract PortInfo expandMapping(
   ObjRef mapRef // ref. a um elemento XML de um documento LindaX
                 //  que representa um mapeamento
  ) throws Exception;
}

public class InstanceInfo {
  public CompInfo[] eInfos;
  public PipeInfo[] aInfos;
}

public class CompInfo {
  public String compName;    // nome do componente
  public ObjRef compTypeRef; // ref. a um elemento XML de um
                             //  documento LindaX que representa um
                             //  tipo de componente
}
```

```

public class PipeInfo {
  public PortInfo[] pInfos;
  public ObjRef pipeTypeRef; // ref. a um elemento XML de um
                             //  documento LindaX que representa um
                             //  tipo de pipe
}

public class PortInfo {
  public ObjRef[] instRefs; // refs. a elementos XML de um (ou mais)
                            //  documento(s) LindaX que representam
                            //  o aninhamento de um conjunto de
                            //  instâncias (o último elemento da lista
                            //  é instância de componente primitivo)
  public String instName;   // nome da instância mais
                            //  EXTERNA no aninhamento
  public ObjRef typeRef;    // tipo da instância mais
                            //  INTERNA no aninhamento
  public ObjRef[] portRefs; // refs. a elementos XML de um
                            //  documento LindaX que representam
                            //  as portas externalizadas pela
                            //  instância mais INTERNA no aninhamento
}
```

9.2. Instanciação do *framework* para gerência de adaptações no OpenCOM

O *framework* consiste em aproximadamente 3.000 linhas de código C/C++ (desconsiderando linhas delimitadoras e comentários[3]), distribuídas por 3 pacotes e 19 arquivos-fonte. A distribuição desse código entre os vários componentes do *framework* é apresentada na Tabela 9.1. Essa tabela discrimina quantas linhas de código são dedicadas à lógica e aos *skeletons* em C++ das interfaces OpenCOM de cada um dos componentes. No caso do GAC, é discriminado também o número de linhas de código da API Lua disponibilizada por esse componente aos *scripts* de verificação.

Tabela 9.1. Número de linhas de código de cada componente do *framework*.

Componente	Linhas de código
GAC *Skeleton* *API Lua*	563 *118* *396* (total = 1077)
TC *Skeleton*	1161 *91* (total = 1252)
CG *Skeleton*	460 *38* (total = 498)

[3] Novamente, foi usada a ferramenta `sclc`, com as opções `-delim-ignore` e `-counts ncsl`, no cômputo dessa estatística.

A seguir são apresentadas as IDLs das interfaces OpenCOM disponibilizadas pelos componentes do *framework*.

9.2.1. Interfaces do GAC

```
interface ICFControl {
  void addConstraint( [const] in  string        script,
                      [const] in  string        constraintName,
                              out unsigned long constraintID );

  void removeConstraint( in unsigned long constraintID );

  void enumConstraints(      out unsigned long iCount,
    [array, size_is(iCount)] out unsigned long constraintIDs );

  void getConstraintByID( in  unsigned long constraintID,
                          out string        constraintName );

  void getConstraintByName( [const] in  string        constraintName,
                                    out unsigned long constraintID );

  void removeAllConstraints();

  void replaceConstraintScript( in unsigned long constraintID,
    [const]                     in string        script );
};

interface ICFValidation {
  void validate(             out unsigned long uCount,
    [array, size_is(uCount)] out unsigned long unsatisfConstraints );
};
```

9.2.2. Interface do TC

```
interface ICFTransaction {
  void begin(    out unsigned long transactionID );
  void commit(   in  unsigned long transactionID );
  void rollback( in  unsigned long transactionID );
};
```

9.2.3. Interface do CG

```
interface ICFConfiguration {
  void configure( [const] in string script );
};
```

10
Apêndice B

Este apêndice apresenta a especificação completa dos estilos associados à DSL LindaQoS.

10.1.
Estilo `LowestNQoS`

```
Style LowestNQoS {
  InterfaceType IMetaQoS( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
  }
  InterfaceType IResourceMan( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
  }
  InterfaceType PInterLevel( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
  }

  ComponentType UAdmCtrl( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
    Cardinality = { 1 ..  };

    Port interLevel {
      Cardinality = { 1 ..  };
      Type = #PinterLevel;
      Direction = “in”;
    }
    Port resourceMan {
      Cardinality = 1;
      Level = "meta";
      Type = #IResourceMan;
      Direction = “out”;
    }
    Port metaQoS {
      Cardinality = { 0 .. 1 };
      Level = "meta";
      Type = #IMetaQoS;
      Direction = “in”;
    }
  }
}
```

10.2. Estilo CentralizedNQoS

```
Style CentralizedNQoS {
  InterfaceType IMetaQoS( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
  }
  InterfaceType PIntraLevel( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
  }
  InterfaceType PInterLevel( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
  }
  InterfaceType PTranslate( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
  }

  ComponentType UQoSNeg( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
    Cardinality = 1;

    Port intraLevel {
      Cardinality = { 1 ..  };
      Type = #PIntraLevel;
      Direction = “in”;
    }
    Port interLevel {
      Cardinality = { 1 ..  };
      Type = #PInterLevel;
      Direction = “out”;
    }
    Port translate {
      Cardinality = { 1 ..  };
      Type = #PTranslate;
      Direction = “out”;
    }
    Port metaQoS {
      Cardinality = { 0 .. 1 };
      Level = "meta";
      Type = #IMetaQoS;
      Direction = “in”;
    }
  }

  ComponentType UAdmCtrl( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
    Cardinality = { 1 ..  };

    Port interLevel {
      Cardinality = { 1 ..  };
      Type = #PInterLevel;
      Direction = “in”;
    }
    Port intraLevel {
      Cardinality = 1;
      Type = #PIntraLevel;
      Direction = “out”;
    }
    Port metaQoS {
      Cardinality = { 0 .. 1 };
      Level = "meta";
      Type = #IMetaQoS;
      Direction = “in”;
    }
  }

```

```
ComponentType UQoSMap( impl,behv ) {
  Implementation = #impl;
  Behaviour = #behv;
  Cardinality = { 1 ..  };

  Port translate {
    Cardinality = 1;
    Type = #PTranslate;
    Direction = “in”;
  }
}

PipeType IntraLevelPipe( impl,behv ) {
  Implementation = #impl;
  Behaviour = #behv;
  Cardinality = { 1 .. };

  AccessPoint in {
    Cardinality = 1;
    Type = #PIntraLevel;
    Direction = “in”;
  }
  AccessPoint out {
    Cardinality = 1;
    Type = #PIntraLevel;
    Direction = “out”;
  }
}

PipeType TranslatePipe( impl,behv ) {
  Implementation = #impl;
  Behaviour = #behv;
  Cardinality = { 1 .. };

  AccessPoint in {
    Cardinality = 1;
    Type = #PTranslate;
    Direction = “in”;
  }
  AccessPoint out {
    Cardinality = 1;
    Type = #PTranslate;
    Direction = “out”;
  }
}

// Todo metacomponente de mapeamento
// deve estar ligado a um metacomponente de negociação central.
FolPredicate MappersWithQoSNeg {
  exists n : Components( #UQoSNeg ); {
    forall c : Components( #UQoSMap ); {
      exists pn : Ports( #n );
             pc : Ports( #c ); {
        exists s : Signatures( #pc ); {
          [Pipe( #n,#pn,#c,#pc ) in Pipes()] and
          [PropertyValue( #s,“Type” ) == #PTranslate]
        }
      }
    }
  }
}

// Todo metacomponente de controle de admissão
// deve estar ligado a um metacomponente de negociação central.
FolPredicate AdmCtrlsWithQoSNeg {
  exists n : Components( #UQoSNeg ); {
    forall c : Components( #UAdmCtrl ); {
      exists pc : Ports( #c );
             pn : Ports( #n ); {
        exists s : Signatures( #pn ); {
          [Pipe( #c,#pc,#n,#pn ) in Pipes()] and
          [PropertyValue( #s,“Type”) == #PIntraLevel]
        }
        }
      }
    }
  }
}
```

10.3. Estilo `DistributedNQoS`

```
Style DistributedNQoS {
  InterfaceType IMetaQoS( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
  }
  InterfaceType PIntraLevel( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
  }
  InterfaceType PInterLevel( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
  }
  InterfaceType PIntraNeg( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
  }
  InterfaceType PTranslate( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
  }

  ComponentType UQoSNeg( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
    Cardinality = {2 .. };

    Port intraLevel {
      Cardinality = { 0 ..  };
      Type = #PIntraLevel;
      Direction = “in”;
    }
    Port interLevel {
      Cardinality = { 1 ..  };
      Type = #PInterLevel;
      Direction = “out”;
    }
    Port intraNeg {
      Cardinality = { 1 ..  };
      Signature in {
        Type = #PIntraNeg;
        Direction = “in”;
      }
      Signature out {
        Type = #PIntraNeg;
        Direction = “out”;
      }
    }
    Port translate {
      Cardinality = { 1 ..  };
      Type = #PTranslate;
      Direction = “out”;
    }
    Port metaQoS {
      Cardinality = { 0 .. 1 };
      Level = "meta";
      Type = #IMetaQoS;
      Direction = “in”;
    }
  }
```

```

ComponentType UAdmCtrl( impl,behv ) {
  Implementation = #impl;
  Behaviour = #behv;
  Cardinality = { 1 ..  };

  Port interLevel {
    Cardinality = { 1 ..  };
    Type = #PInterLevel;
    Direction = “in”;
  }
  Port intraLevel {
    Cardinality = 1;
    Type = #PIntraLevel;
    Direction = “out”;
  }
  Port metaQoS {
    Cardinality = { 0 .. 1 };
    Level = "meta";
    Type = #IMetaQoS;
    Direction = “in”;
  }
}

ComponentType UQoSMap( impl,behv ) {
  Implementation = #impl;
  Behaviour = #behv;
  Cardinality = { 1 ..  };

  Port translate {
    Cardinality = 1;
    Type = #PTranslate;
    Direction = “in”;
  }
}

PipeType IntraLevelPipe( impl,behv ) {
  Implementation = #impl;
  Behaviour = #behv;
  Cardinality = { 1 .. };

  AccessPoint in {
    Cardinality = 1;
    Type = #PIntraLevel;
    Direction = “in”;
  }
  AccessPoint out {
    Cardinality = 1;
    Type = #PIntraLevel;
    Direction = “out”;
  }
}

PipeType TranslatePipe( impl,behv ) {
  Implementation = #impl;
  Behaviour = #behv;
  Cardinality = { 1 .. };

  AccessPoint in {
    Cardinality = 1;
    Type = #PTranslate;
    Direction = “in”;
  }
  AccessPoint out {
    Cardinality = 1;
    Type = #PTranslate;
    Direction = “out”;
  }
}

PipeType SignallingPipe( impl,behv ) {
  Implementation = #impl;
  Behaviour = #behv;
```

```
    Cardinality = { 1 .. };

    AccessPoint intraNeg {
      Cardinality = { 2 ..  };
       Signature in {
         Type = #PIntraNeg;
         Direction = “in”;
       }
       Signature out {
         Type = #RIntraNeg;
         Direction = “in”;
       }
    }
  }

  // Todo metacomponente de mapeamento
  // deve estar ligado a um metacomponente de negociação.
  FolPredicate MappersWithQoSNegs {
    forall c : Components( #UQoSMap ); {
      exists n : Components( #UQoSNeg ); {
        exists pn : Ports( #n );
               pc : Ports( #c ); {
          exists s : Signature( #pc ); {
            [Pipe( #n,#pn,#c,#pc ) in Pipes()] and
            [PropertyValue( #s,“Type” ) == #PTranslate]
          }
        }
      }
    }
  }

  // Todo metacomponente de controle de admissão
  // deve estar ligado a um metacomponente de negociação.
  FolPredicate AdmCtrlsWithQoSNegs {
    forall c : Components( #UAdmCtrl ); {
      exists n : Components( #UQoSNeg ); {
        exists pc : Ports( #c );
               pn : Ports( #n ); {
          exists s : Signature( #pn ); {
            [Pipe( #c,#pc,#n,#pn ) in Pipes()] and
            [PropertyValue( #s,“Type” ) == #PIntraLevel]
          }
        }
      }
    }
  }

  // Restrição de grafo conexo, utilizando pipes do tipo
  // SignallingPipe, entre metacomponentes de negociação.
  // SE o grafo de componentes é conexo, haverá sempre um conjunto
  // com todos os componentes na configuração, ordenado do componente
  // com menos associações ao componente com mais associações,
  // que obedece a esse predicado.
  FolPredicate FullyConnectedQoSNegs {
    exists s : SequenceSet( Components( #UQoSNeg ) )
                 where
                   [Cardinality( #s ) > 1] and
                   [Cardinality( #s ) ==
                      Cardinality( Components( #UQoSNeg ) )]; {
      forall i : { 1..Cardinality( #s )-1 }; {
        exists j : { #i+1..Cardinality( #s ) }; {
          exists pni : Ports( At( #s,#i ) );
                 pnj : Ports( At( #s,#j ) ); {
            [Pipe( At( #s,#i ),#pni,At( #s,#j ),#pnj )
               in Pipes( #SignallingPipe )]
          }
        }
      }
    }
  }
}
```

10.4.
Estilo HierarchyNQoS

```
Style HierarchyNQoS {
  PipeType InterLevelPipe( impl,behv ) {
    Implementation = #impl;
    Behaviour = #behv;
    Cardinality = { 1 .. };

    AccessPoint in {
      Cardinality = 1;
      Type = #PInterLevel;
      Direction = “in”;
    }
    AccessPoint out {
      Cardinality = 1;
      Type = #PInterLevel;
      Direction = “out”;
    }
  }
}
```

11
Apêndice C

Este apêndice apresenta a especificação completa dos esquemas-base XML que compõem o núcleo de LindaX, bem como o esquema de extensão para a lógica FOL.

11.1.
Esquema XML `lindaxprop`

```
<xsd:schema
  xmlns="http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace=
           "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"
  elementFormDefault="qualified"
  attributeFormDefault="qualified">

  <xsd:annotation>
    <xsd:documentation>
      LindaX properties - Schema 1.0
    </xsd:documentation>
  </xsd:annotation>

  <!--
    TYPE: Property
    A property is used to specify many things for vocabulary
    elements, such as base- or meta-level indication, type indication,
    etc.
   -->
  <xsd:complexType name="Property">
    <xsd:sequence>
      <xsd:element name= "name"
                   type= "StringValue"/>
      <xsd:element name= "value"
                   type= "ElemExp"/>
    </xsd:sequence>
  </xsd:complexType>

  <!--
    TYPE: StringValue
    A simple string is just a string constant.
   -->
  <xsd:simpleType name="StringValue">
    <xsd:restriction base="xsd:string">
    </xsd:restriction>
  </xsd:simpleType>

  <!--
    TYPE: ElemExp
    Represents a generic element expression, which can be
    simple string, hexBinary, xml-link, set expression or a symbol.
   -->
  <xsd:complexType name="ElemExp">
    <xsd:choice>
      <xsd:element name="stringValue" type="StringValue"/>
      <xsd:element name="integerValue" type="IntegerValue"/>
      <xsd:element name="set" type="Set"/>
      <xsd:element name="externalRef" type="ExternalRef"/>
    </xsd:choice>
  </xsd:complexType>

  <!--
    TYPE: IntegerValue

    A simple integer in LindaX is just a
    representation of an hexBinary.
   -->
  <xsd:simpleType name="IntegerValue">
    <xsd:restriction base="xsd:hexBinary">
    </xsd:restriction>
  </xsd:simpleType>

  <!--
    TYPE: Set

    A Set is a choice among different sets.
   -->
  <xsd:complexType name="Set">
    <xsd:choice>
      <xsd:element name="list" type="List"/>
      <xsd:element name="range" type="Range"/>
    </xsd:choice>
  </xsd:complexType>

  <!--
    TYPE: List

    A List is an enumeration.
   -->
  <xsd:complexType name="List">
    <xsd:sequence>
      <xsd:element name=     "elem"
                   type=     "ElemExp"
                   minOccurs="0"
                   maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>

  <!--
    TYPE: Range

    A Range is a range of numbers.
   -->
  <xsd:complexType name="Range">
    <xsd:sequence>
      <xsd:element name=     "lowElem"
                   type=     "IntegerValue"/>
      <xsd:element name=     "hiElem"
                   type=     "IntegerValue"
                   minOccurs="0"
                   maxOccurs="1" />
    </xsd:sequence>
  </xsd:complexType>

  <!--
    TYPE: ExternalRef
    An ExternalRef is an xml-link with parameters.
   -->
  <xsd:complexType name="ExternalRef">
    <xsd:sequence>
      <xsd:element name=     "externalSymbol"
                   type=     "XMLLink"/>
      <xsd:element name=     "externalParam"
                   type=     "ExternalParam"
                   minOccurs="0"
                   maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
```

```
  <!--
    TYPE: XMLLink

    Encapsulates the parts of the XLink definition
    that are useful in LindaX.
    Imported from xArch definitions.
   -->
  <xsd:complexType name="XMLLink">
    <xsd:attribute ref="xlink:type"/>
    <xsd:attribute ref="xlink:href"/>
  </xsd:complexType>

  <!--
    TYPE: ExternalParam
   -->
  <xsd:complexType name="ExternalParam">
    <xsd:choice>
      <xsd:element name=     "externalSymbol"
                   type=     "XMLLink"/>
      <xsd:element name=     "stringValue"
                   type=     "StringValue"/>
    </xsd:choice>
  </xsd:complexType>

  <!--
    TYPE: Symbol

    A symbol is a string representation of a variable which an
    element can be linked to (eg: externalSymbol in <externalRef>).
   -->
  <xsd:complexType name="Symbol">
    <xsd:attribute name=     "id"
                   type=     "xsd:ID" />
  </xsd:complexType>

  <!--
    TYPE: Parameter

    A parameter is used to specify many things for vocabulary
    elements externally, specially properties.
   -->
  <xsd:complexType name="Parameter">
    <xsd:attribute name=     "id"
                   type=     "xsd:ID" />
  </xsd:complexType>

</xsd:schema>
```

11.2. Esquema `lindaxtyp`

```
<xsd:schema
  xmlns="http://www.telemidia.puc-rio.br/pub/LindaX/lindaxtyp.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:lindaxprop=
           "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"
  targetNamespace=
            "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxtyp.xsd"
  elementFormDefault="qualified"
  attributeFormDefault="qualified">

  <xsd:import namespace=
         "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"
              schemaLocation=
         "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"/>

  <xsd:annotation>
    <xsd:documentation>
      LindaX Type System - Schema 1.0
    </xsd:documentation>
  </xsd:annotation>

  <!--
    TYPE: Vocabulary

    A vocabulary is an specification of a computation or
    communication abstraction.
    The specific format of a vocabulary is unspecified at this level.
   -->
  <xsd:complexType name="Vocabulary">
    <xsd:sequence>
      <xsd:element name=     "property"
                   type=     "lindaxprop:Property"
                   minOccurs="0"
                   maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name=     "id"
                   type=     "xsd:ID" />
  </xsd:complexType>

  <!--
    TYPE: InterfaceType

    The InterfaceType type defines a type of interface.
    At the types level, no semantic information (such as a
    behaviour) is defined in an interface type.
    This can be specified in an extension or understood
    programmatically.
   -->
  <xsd:complexType name="InterfaceType">
    <xsd:complexContent>
      <xsd:extension base="Vocabulary">
        <xsd:sequence>
          <xsd:element name=     "parameter"
                       type=     "lindaxprop:Parameter"
                       minOccurs="0"
                       maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!--
    TYPE: ComponentType

    The ComponentType type defines a type of component.  A type of
    component is identified by its ID and ports.
   -->
  <xsd:complexType name="ComponentType">
    <xsd:complexContent>
      <xsd:extension base="lindaxstyle:Vocabulary">
        <xsd:sequence>
          <xsd:element name=     "parameter"
                       type=     "lindaxprop:Parameter"
                       minOccurs="0"
                       maxOccurs="unbounded" />
          <xsd:element name=     "port"
                       type=     "Port"
                       minOccurs="0"
                       maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!--    TYPE: PipeType

    The PipeType type defines a type of pipe.  A type of
    pipe is identified by its ID and access points.
   -->
  <xsd:complexType name="PipeType">
```

```
  <xsd:complexContent>
    <xsd:extension base="Vocabulary">
      <xsd:sequence>
        <xsd:element name=     "parameter"
                     type=     "lindaxprop:Parameter"
                     minOccurs="0"
                     maxOccurs="unbounded" />
        <xsd:element name=     "accessPoint"
                     type=     "AccessPoint"
                     minOccurs="0"
                     maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!--
  TYPE: Port

  The Port type describes an opaque port for use
  at the structure level.  No semantic information is provided
  at this level.  An opaque port contains an ID and signatures.
  This may be connected to access points via an attachment.
 -->
<xsd:complexType name="Port">
  <xsd:complexContent>
    <xsd:extension base="Vocabulary">
      <xsd:sequence>
        <xsd:element name=     "signature"
                     type=     "Signature"
                     minOccurs="0"
                     maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<!--
  TYPE: AccessPoint

  The AccessPoint type describes an opaque access point for use
  at the structure level.  No semantic information is provided
  at this level.  An opaque access point contains an ID and
  Signatures.  This may be connected to ports via an attachment.
 -->
<xsd:complexType name="AccessPoint">
  <xsd:complexContent>
    <xsd:extension base="Vocabulary">
      <xsd:sequence>
        <xsd:element name=     "signature"
                     type=     "Signature"
                     minOccurs="0"
                     maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!--
  TYPE: Signature

  The Signature type defines one (of many)
  "signatures" that a component or pipe type can possess.
   A signature basically says, "FOO type
  components/providers, when instantiated, should contain an
  instance of BAR type interface.  The 'type' pointer is provided
  as a property.
 -->
<xsd:complexType name="Signature">
  <xsd:complexContent>
    <xsd:extension base="Vocabulary">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

```

</xsd:schema>
```

11.3. Esquema lindaxcnf

```
<xsd:schema
  xmlns="http://www.telemidia.puc-rio.br/pub/LindaX/lindaxcnf.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:lindaxprop=
           "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"
  xmlns:lindaxtyp=
           "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxtyp.xsd"
  targetNamespace=
           "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxcnf.xsd"
  elementFormDefault="qualified"
  attributeFormDefault="qualified">

  <xsd:import namespace=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"
       schemaLocation=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd" />
  <xsd:import namespace=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxtyp.xsd"
       schemaLocation=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxtyp.xsd" />

  <xsd:annotation>
    <xsd:documentation>
      LindaX Configuration Support - Schema 1.0
    </xsd:documentation>
  </xsd:annotation>

  <!--
    ELEMENT: lindaXConfiguration

    The lindaXConfiguration element (of type Configuration)
    is the root element that is the aegis over all other elements
    in the configuration (either run- and design-time) structure.
    This element is important for integration with xArch/ArchStudio.
   -->
  <xsd:element name="lindaXConfiguration" type="Configuration"/>


  <!--
    TYPE: Configuration

    The Configuration type defines a structure for declaring
    configurations (either run- and design-time) based on styles.
   -->
  <xsd:complexType name="Configuration">
    <xsd:complexContent>
      <xsd:extension base="lindaxtyp:Vocabulary">
        <xsd:sequence>
          <xsd:element name=     "parameter"
                       type=     "lindaxprop:Parameter"
                       minOccurs="0"
                       maxOccurs="unbounded" />
          <xsd:element name=     "instance"
                       type=     "Instance"
                       minOccurs="0"
                       maxOccurs="unbounded" />
          <xsd:element name=     "include"
                       type=     "Include"
                       minOccurs="0"
                       maxOccurs="unbounded" />
          <xsd:element name=     "pipe"
                       type=     "Pipe"
                       minOccurs="0"
```

```
                     maxOccurs="unbounded" />
        <xsd:element name=     "mapping"
                     type=     "Mapping"
                     minOccurs="0"
                     maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!--
  TYPE: System

  The System type defines a structure for declaring
  run-time configurations based on styles.
 -->
<xsd:complexType name="System">
  <xsd:complexContent>
    <xsd:extension base="Configuration">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!--
  TYPE: SystemTemplate

  The SystemTemplate type defines a structure for declaring
  design-time configurations based on styles.
 -->
<xsd:complexType name="SystemTemplate">
  <xsd:complexContent>
    <xsd:extension base="Configuration">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!--
  TYPE: Instance
 -->
<xsd:complexType name="Instance">
  <xsd:complexContent>
    <xsd:extension base="lindaxtyp:Vocabulary">
      <xsd:sequence>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!--
  TYPE: Include
 -->
<xsd:complexType name="Include">
  <xsd:sequence>
        <xsd:element name=     "symbol"
                     type=     "lindaxprop:Symbol"
                     minOccurs="1"
                     maxOccurs="1" />
        <xsd:element name=     "externalRef"
                     type=     "lindaxprop:ExternalRef"
                     minOccurs="1"
                     maxOccurs="1" />
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: Pipe
 -->
<xsd:complexType name="Pipe">
  <xsd:complexContent>
    <xsd:extension base="lindaxtyp:Vocabulary">
      <xsd:sequence>
        <xsd:element name=     "peers"
                     type=     "lindaxprop:ExternalRef"
```

```
                       minOccurs="2"
                       maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!--
    TYPE: Mapping
   -->
  <xsd:complexType name="Mapping">
    <xsd:sequence>
          <xsd:element name=     "symbol"
                       type=     "lindaxprop:Symbol"
                       minOccurs="1"
                       maxOccurs="1" />
          <xsd:element name=     "externalRef"
                       type=     "lindaxprop:ExternalRef"
                       minOccurs="1"
                       maxOccurs="1" />
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

11.4. Esquema lindaxres

```
<xsd:schema
  xmlns="http://www.telemidia.puc-rio.br/pub/LindaX/lindaxres.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:lindaxprop=
           "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"
  xmlns:lindaxtyp=
           "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxtyp.xsd"
  targetNamespace=
           "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxres.xsd"
  elementFormDefault="qualified"
  attributeFormDefault="qualified">
  <xsd:import namespace=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"
       schemaLocation=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd" />
  <xsd:import namespace=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxtyp.xsd"
       schemaLocation=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxtyp.xsd" />

  <xsd:annotation>
    <xsd:documentation>
      LindaX Resource View Support - Schema 1.0
    </xsd:documentation>
  </xsd:annotation>

  <!--
    ELEMENT: lindaXResource

    The lindaXResource element (of type Resource)
    is the root element that is the aegis over all other elements
    in an resource view structure.
    This element is important for integration with xArch/ArchStudio.
  -->
  <xsd:element name="lindaXResource" type="Resource"/>


  <!--
    TYPE: Resource

    The Resource type defines a structure for declaring
    resource views.
   -->
  <xsd:complexType name="Resource">
    <xsd:complexContent>
      <xsd:extension base="lindaxtyp:Vocabulary">
        <xsd:sequence>
          <xsd:element name=     "parameter"
                       type=     "lindaxprop:Parameter"
                       minOccurs="0"
                       maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!--
    TYPE: Task

    The Task type defines a structure for declaring
    computational resource containers.
   -->
  <xsd:complexType name="Task">
    <xsd:complexContent>
      <xsd:extension base="Resource">
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!--
    TYPE: Provider

    The Provider type defines a structure for declaring
    communication resource containers.
   -->
  <xsd:complexType name="Provider">
    <xsd:complexContent>
      <xsd:extension base="Resource">
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

11.5. Esquema `lindaxsty`

```
<xsd:schema
  xmlns="http://www.telemidia.puc-rio.br/pub/LindaX/lindaxsty.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:lindaxprop=
           "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"
  xmlns:lindaxtyp=
           "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxtyp.xsd"
  targetNamespace=
           "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxsty.xsd"
  elementFormDefault="qualified"
  attributeFormDefault="qualified">

  <xsd:import namespace=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"
       schemaLocation=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd" />
  <xsd:import namespace=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxtyp.xsd"
       schemaLocation=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxtyp.xsd" />

  <xsd:annotation>
    <xsd:documentation>
      LindaX styles - Schema 1.0
    </xsd:documentation>
  </xsd:annotation>


  <!--
    ELEMENT: lindaXStyle

    The lindaXStyle element (of type Style) is the root element
    that is the aegis over all other elements in the style structure.
    This element is important for integration with xArch/ArchStudio.
   -->
  <xsd:element name="lindaXStyle" type="Style"/>

  <!--
    TYPE: Style

    The Style type describes the vocabulary and predicates that make
    part of an architectural style in LindaX.
    Styles at this level have no semantic information associated
    with them.  However, they give the designer and associated
    tools the ability to understand that an assembly of
    computation and communcation elements that conform to a
    particular set of predicates and vocabulary definitions.
    No specifications of what c&c elements actually do are provided
    at this level beyond the opaque descriptions associated with
    the vocabulary and predicates.
   -->
  <xsd:complexType name="Style">
    <xsd:sequence>
      <xsd:element name=     "superstyle"
                   type=     "lindaxprop:XMLLink"
                   minOccurs="0"
                   maxOccurs="1" />
      <xsd:element name=     "vocabulary"
                   type=     "lindaxtyp:Vocabulary"
                   minOccurs="0"
                   maxOccurs="unbounded" />
      <xsd:element name=     "predicate"
                   type=     "Predicate"
                   minOccurs="0"
                   maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name=     "id"
                   type=     "xsd:ID" />
  </xsd:complexType>

  <!--
    TYPE: Predicate

    A Predicate is an specification that must be satisfied
    by any assembly of components subject to the enclosing style.
    The specific format of a predicate is unspecified at this level.
   -->
  <xsd:complexType name="Predicate">
    <xsd:sequence>
      <xsd:element name=     "property"
                   type=     "lindaxprop:Property"
                   minOccurs="0"
                   maxOccurs="unbounded" />
      <xsd:element name=     "parameter"
                   type=     "lindaxprop:Parameter"
                   minOccurs="0"
                   maxOccurs="unbounded" />
   </xsd:sequence>
    <xsd:attribute name=     "id"
                   type=     "xsd:ID" />
  </xsd:complexType>

</xsd:schema>
```

11.6. Esquema **`lindaxfolpredicate`**

```
<xsd:schema
  xmlns=
   "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxfolpredicate.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:lindaxprop=
           "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"
  xmlns:lindaxsty=
            "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxsty.xsd"
  targetNamespace=
   "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxfolpredicate.xsd"
  elementFormDefault="qualified"
  attributeFormDefault="qualified">

  <xsd:import namespace=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd"
              schemaLocation=
        "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxprop.xsd" />
  <xsd:import namespace=
         "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxsty.xsd"
              schemaLocation=
         "http://www.telemidia.puc-rio.br/pub/LindaX/lindaxsty.xsd" />

  <xsd:annotation>
    <xsd:documentation>
      First-order predicates in LindaX styles - Schema 1.0
    </xsd:documentation>
  </xsd:annotation>


  <!--
    TYPE: FolPredicate

    A FolPredicate is an specification in first-order logic that
    must be satisfied by any assembly of components subject to
    the enclosing style.
   -->
  <xsd:complexType name="FolPredicate">
    <xsd:complexContent>
      <xsd:extension base="lindaxsty:Predicate">
        <xsd:sequence>
          <xsd:element name=     "logicalExp"
                       type=     "FolLogicalExp"
                       minOccurs="1"
                       maxOccurs="1" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!--
    TYPE: FolLogicalExp

    Type that specifies a logical expression in FOL.
   -->
  <xsd:complexType name="FolLogicalExp">
    <xsd:choice>
      <xsd:element name=     "forAll"
                   type=     "FolForAll"
                   minOccurs="1"
                   maxOccurs="1"/>
      <xsd:element name=     "exists"
                   type=     "FolExists"
                   minOccurs="1"
                   maxOccurs="1"/>
      <xsd:element name=     "boolExp"
                   type=     "FolBooleanExp"
                   minOccurs="1"
                   maxOccurs="1"/>
    </xsd:choice>
```

```
</xsd:complexType>


<!--
  TYPE: FolForAll

  Type that specifies a universal operator in FOL.
 -->
<xsd:complexType name="FolForAll">
  <xsd:sequence>
    <xsd:element name="decl"
                 type="FolFormalParam"
                 minOccurs="1"
                 maxOccurs="unbounded" />
    <xsd:element name="logicalExp"
                 type="FolLogicalExp"
                 minOccurs="1"
                 maxOccurs="1" />
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolExists

  Type that specifies an existencial operator in FOL.
 -->
<xsd:complexType name="FolExists">
  <xsd:sequence>
    <xsd:element name="decl"
                 type="FolFormalParam"
                 minOccurs="1"
                 maxOccurs="unbounded" />
    <xsd:element name="logicalExp"
                 type="FolLogicalExp"
                 minOccurs="1"
                 maxOccurs="1" />
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolBooleanExp

  Type that specifies a boolean expression in FOL.
 -->
<xsd:complexType name="FolBooleanExp">
  <xsd:choice>
    <xsd:element name="and" type="FolAnd"/>
    <xsd:element name="or" type="FolOr"/>
    <xsd:element name="not" type="FolNot"/>
    <xsd:element name="cmpExp" type="FolCmpExp"/>
    <xsd:element name="boolParenExp" type="FolBoolParenExp"/>
  </xsd:choice>
</xsd:complexType>

<!--
  TYPE: FolFormalParam

  Type that specifies a formal parameter in FOL.
 -->
<xsd:complexType name="FolFormalParam">
  <xsd:sequence>
    <xsd:element name=     "symbol"
                 type=     "lindaxprop:Symbol"
                 minOccurs="1"
                 maxOccurs="1"/>
    <xsd:element name=     "set"
                 type=     "FolSetExp"
                 minOccurs="1"
                 maxOccurs="1"/>
    <xsd:element name="condition"
                 type="FolBooleanExp"
                 minOccurs="0"
                 maxOccurs="1" />
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolSetExp

  Type that specifies a set expression in FOL.
 -->
<xsd:complexType name="FolSetExp">
  <xsd:choice>
    <xsd:element name="union" type="FolUnion"/>
    <xsd:element name="intersection" type="FolIntersection"/>
    <xsd:element name="minus" type="FolMinus"/>
    <xsd:element name="simpleSetValue" type="FolSimpleSetValue"/>
    <xsd:element name="setParenExp" type="FolSetParenExp"/>
    <xsd:element name="function" type="FolFunction"/>
    <xsd:element name="externalSymbol" type="lindaxprop:XMLLink"/>
  </xsd:choice>
</xsd:complexType>

<!--
  TYPE: FolUnion
 -->
<xsd:complexType name="FolUnion">
  <xsd:sequence>
    <xsd:element name="setExp1" type="FolSetExp"/>
    <xsd:element name="setExp2" type="FolSetExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolIntersection
 -->
<xsd:complexType name="FolIntersection">
  <xsd:sequence>
    <xsd:element name="setExp1" type="FolSetExp"/>
    <xsd:element name="setExp2" type="FolSetExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolMinus
 -->
<xsd:complexType name="FolMinus">
  <xsd:sequence>
    <xsd:element name="setExp1" type="FolSetExp"/>
    <xsd:element name="setExp2" type="FolSetExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolSetParenExp

  Type that specifies a set expression in parenthesis in FOL.
 -->
<xsd:complexType name="FolSetParenExp">
  <xsd:sequence>
    <xsd:element name="setExp"
                 type="FolSetExp"
                 minOccurs="1"
                 maxOccurs="1" />
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolSimpleSetValue

  A FolSet is a choice among different sets known in FOL.
 -->
<xsd:complexType name="FolSimpleSetValue">
  <xsd:choice>
    <xsd:element name="list" type="FolList"/>
    <xsd:element name="range" type="FolRange"/>
  </xsd:choice>
</xsd:complexType>

<!--
  TYPE: FolList

  A FolList is an enumeration.
 -->
<xsd:complexType name="FolList">
  <xsd:sequence>
    <xsd:element name=     "elem"
                 type=     "FolElemExp"
                 minOccurs="0"
                 maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolRange

  A FolRange is a range of numbers.
 -->
<xsd:complexType name="FolRange">
  <xsd:sequence>
    <xsd:element name=     "lowElem"
                 type=     "FolIntegerExp"
                 minOccurs="0"
                 maxOccurs="1" />
    <xsd:element name=     "hiElem"
                 type=     "FolIntegerExp"
                 minOccurs="0"
                 maxOccurs="1" />

  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolFunction

  A FolFunction is a function that can be evaluated in the FOL.
 -->
<xsd:complexType name="FolFunction">
  <xsd:sequence>
    <xsd:element name=     "name"
                 type=     "lindaxprop:StringValue"
                 minOccurs="1"
                 maxOccurs="1" />
    <xsd:element name=     "param"
                 type=     "FolElemExp"
                 minOccurs="0"
                 maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolAnd

  Represents a logical AND expression with its both sides
  (subexpressions), using polish notation
 -->
<xsd:complexType name="FolAnd">
  <xsd:sequence>
    <xsd:element name="booleanExp1" type="FolBooleanExp"/>
    <xsd:element name="booleanExp2" type="FolBooleanExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolOr

  Represents a logical OR expression with its both sides
  (subexpressions), using polish notation
 -->
```

```
<xsd:complexType name="FolOr">
  <xsd:sequence>
    <xsd:element name="booleanExp1" type="FolBooleanExp"/>
    <xsd:element name="booleanExp2" type="FolBooleanExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolNot

  Represents the unary logical NOT operation
 -->
<xsd:complexType name="FolNot">
  <xsd:sequence>
    <xsd:element name="booleanExp" type="FolBooleanExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolBoolParenExp

  Type that specifies a boolean expression in parenthesis in FOL.
 -->
<xsd:complexType name="FolBoolParenExp">
  <xsd:sequence>
    <xsd:element name="boolExp"
                 type="FolBooleanExp"
                 minOccurs="1"
                 maxOccurs="1" />
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolCmpExp

  Represents a comparison operation evaluating to TRUE or FALSE.
 -->
<xsd:complexType name="FolCmpExp">
  <xsd:choice>
    <xsd:element name="equals"
                 type="FolEquals"/>
    <xsd:element name="notEquals"
                 type="FolNotEquals"/>
    <xsd:element name="greaterThanOrEquals"
                 type="FolGreaterThanOrEquals"/>
    <xsd:element name="lessThanOrEquals"
                 type="FolLessThanOrEquals"/>
    <xsd:element name="greaterThan"
                 type="FolGreaterThan"/>
    <xsd:element name="lessThan"
                 type="FolLessThan"/>
    <xsd:element name="inSet"
                 type="FolInSet"/>
  </xsd:choice>
</xsd:complexType>

<!--
  TYPE: FolEquals

  Represents a comparison operation evaluating to TRUE or FALSE.
 -->
<xsd:complexType name="FolEquals">
  <xsd:sequence>
    <xsd:element name="elemExp1" type="FolElemExp"/>
    <xsd:element name="elemExp2" type="FolElemExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolNotEquals

  Represents a comparison operation evaluating to TRUE or FALSE.
 -->
```

```
<xsd:complexType name="FolNotEquals">
  <xsd:sequence>
    <xsd:element name="elemExp1" type="FolElemExp"/>
    <xsd:element name="elemExp2" type="FolElemExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolGreaterThanOrEquals

  Represents a comparison operation evaluating to TRUE or FALSE.
 -->
<xsd:complexType name="FolGreaterThanOrEquals">
  <xsd:sequence>
    <xsd:element name="elemExp1" type="FolElemExp"/>
    <xsd:element name="elemExp2" type="FolElemExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolLessThanOrEquals

  Represents a comparison operation evaluating to TRUE or FALSE.
 -->
<xsd:complexType name="FolLessThanOrEquals">
  <xsd:sequence>
    <xsd:element name="elemExp1" type="FolElemExp"/>
    <xsd:element name="elemExp2" type="FolElemExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolGreaterThan

  Represents a comparison operation evaluating to TRUE or FALSE.
 -->
<xsd:complexType name="FolGreaterThan">
  <xsd:sequence>
    <xsd:element name="elemExp1" type="FolElemExp"/>
    <xsd:element name="elemExp2" type="FolElemExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolLessThan

  Represents a comparison operation evaluating to TRUE or FALSE.
 -->
<xsd:complexType name="FolLessThan">
  <xsd:sequence>
    <xsd:element name="elemExp1" type="FolElemExp"/>
    <xsd:element name="elemExp2" type="FolElemExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolInSet

  Represents a comparison operation evaluating to TRUE or FALSE.
 -->
<xsd:complexType name="FolInSet">
  <xsd:sequence>
    <xsd:element name="elemExp" type="FolElemExp"/>
    <xsd:element name="setExp" type="FolSetExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolElemExp

  Represents a generic element expression, which can be
  simple string, integer expression, set expression,
  function expression or a symbol.
 -->
<xsd:complexType name="FolElemExp">
  <xsd:choice>
    <xsd:element name="simpleStr"
                 type="lindaxprop:StringValue"/>
    <xsd:element name="externalSymbol"
                 type="lindaxprop:XMLLink"/>
    <xsd:element name="function"
                 type="FolFunction"/>
    <xsd:element name="set"
                 type="FolSetExp"/>
    <xsd:element name="intExp"
                 type="FolIntegerExp"/>
  </xsd:choice>
</xsd:complexType>

<!--
  TYPE: FolIntegerExp

  Type that specifies an integer expression in FOL.
 -->
<xsd:complexType name="FolIntegerExp">
  <xsd:choice>
    <xsd:element name="simpleInt"
                 type="lindaxprop:IntegerValue"/>
    <xsd:element name="externalSymbol"
                 type="lindaxprop:XMLLink"/>
    <xsd:element name="function"
                 type="FolFunction"/>
    <xsd:element name="add"
                 type="FolAdd"/>
    <xsd:element name="sub"
                 type="FolSub"/>
    <xsd:element name="mult"
                 type="FolMult"/>
    <xsd:element name="div"
                 type="FolDiv"/>
    <xsd:element name="intParenExp"
                 type="FolIntParenExp"/>
  </xsd:choice>
</xsd:complexType>

<!--
  TYPE: FolAdd

  Represents an integer sum operation.
 -->
<xsd:complexType name="FolAdd">
  <xsd:sequence>
    <xsd:element name="intExp1" type="FolIntegerExp"/>
    <xsd:element name="intExp2" type="FolIntegerExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolSub

  Represents an integer subtraction operation.
 -->
<xsd:complexType name="FolSub">
  <xsd:sequence>
    <xsd:element name="intExp1" type="FolIntegerExp"/>
    <xsd:element name="intExp2" type="FolIntegerExp"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  TYPE: FolMult

  Represents an integer multiplication operation.
 -->
<xsd:complexType name="FolMult">
  <xsd:sequence>
```

```
      <xsd:element name="intExp1" type="FolIntegerExp"/>
      <xsd:element name="intExp2" type="FolIntegerExp"/>
    </xsd:sequence>
  </xsd:complexType>

  <!--
    TYPE: FolDiv

    Represents an integer division operation.
   -->
  <xsd:complexType name="FolDiv">
    <xsd:sequence>
      <xsd:element name="intExp1" type="FolIntegerExp"/>
      <xsd:element name="intExp2" type="FolIntegerExp"/>
    </xsd:sequence>
  </xsd:complexType>

  <!--
    TYPE: FolIntParenExp

    Represents an integer parenthesised expression.
   -->
  <xsd:complexType name="FolIntParenExp">
    <xsd:sequence>
      <xsd:element name="intExp" type="FolIntegerExp"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```