Referências Bibliográficas

[1] AKSIT, M.. **On the Design of the Object-Oriented Language Sina**. PhD thesis, University of Twente, 1989.

[2] AKSIT, M.; BERGMANS, L. ; VURAL, S.. **An object-oriented language-database integration model: The composition-filters approach**. In: Madsen, O. L., editor, PROC. 7TH EUROPEAN CONF. OBJECT-ORIENTED PROGRAMMING, p. 372–395. Springer-Verlag Lecture Notes in Computer Science, 1992.

[3] AKSIT, M.; WAKITA, K.; BOSCH, J.; BERGMANS, L. ; YONEZAWA, A.. **Abstracting Object Interactions Using Composition Filters**. In: Guerraoui, R.; Nierstrasz, O. ; Riveill, M., editors, WORKSHOP ON OBJECT-BASED DISTRIBUTED PROGRAMMING AT ECOOP'93, p. 152–184. Springer-Verlag, 1994.

[4] AKSIT, M.; BOSCH, J.; V.D. STERREN, W. ; BERGMANS, L.. **Real-time specification inheritance anomalies and real-time filters**. In: Tokoro, M.; Pareschi, R., editors, PROC. 8TH EUROPEAN CONF. OBJECT-ORIENTED PROGRAMMING, p. 386–407. Springer Verlag LNCS 821, July 1994.

[5] AKSIT, M.; BERGMANS, L.. **Solving the evolution problems using Composition Filters**. Advanced Software Composition: Obstacles and Approaches (ECOOP 2001 Tutorial), 2001. Available at http://trese.cs.utwente.nl.

[6] ALDAWUD, O.; ELRAD, T. ; BADER, A.. **UML Profile for Aspect-Oriented Software Development**. In: WORKSHOP ON ASPECT-ORIENTED MODELING WITH UML (AOSD-2003), March 2003.

[7] ALWIS, B.; GUDMUNDSON, S.; SMOLYN, G. ; KICZALES, G.. **Coding Issues in AspectJ**. In: WORKSHOP ON ADVANCED SEPARATION OF CONCERNS AT THE CONFERENCE ON OBJECT-

ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS (OOPSLA), 2000.

[8] **Aspect-Oriented Software Development Web site**. http://aosd.net/.

[9] **AOM Website**. http://lglwww.epfl.ch/workshops/aosd2003/links.html.

[10] ASPECTJ TEAM. **The AspectJ Programming Guide**, 2003. http://eclipse.org/aspectj.

[11] **AspectJ Project**, 2003. http://www.eclipse.org/aspectj/.

[12] **AspectJ users list**, 2003. (users@aspectj.org).

[13] BARDOU, D.. **Roles, subjects and aspects: How do they relate?** In Bardou98 [147].

[14] BERGMANS, L.. **Composing Concurrent Objects**. PhD thesis, University of Twente, 1994.

[15] BERGMANS, L.; AKSIT, M.. **Composing Crosscutting Concerns Using Composition Filters**. Communications of the ACM, 44(10):51–57, October 2001.

[16] BOOCH, G.. **Object-Oriented Design with Applications**. Benjamin-Cummings, 1991.

[17] BOOCH, G.; RUMBAUGH, J. ; JACOBSON, I.. **The Unified Modeling Language User Guide**. Addison-Wesley, 1999.

[18] BOOCH, G.. **Through the Looking Glass**, July 2001. Software Development On-line.

[19] BORGIDA, A.; MYLOPOULOS, J. ; WONG, H. K. T.. **Generalization/Specialization as a Basis for Software Specification**. In: Brodie, M. L.; Mylopoulos, J. ; Schmidt, J. W., editors, ON CONCEPTUAL MODELLING: PERSPECTIVES FROM ARTIFICIAL INTELLIGENCE, DATABASES, AND PROGRAMMING LANGUAGES, p. 87–114. Springer, New York, 1984.

[20] BRADSHAW, J.. **An Introduction to Software Agents**. American Association for Artificial Intelligence/MIT Press, 1997. J. Bradshaw (ed.).

[21] CZARNECKI, K.; EISENECKER, U. ; STEYAERT, P.. **Beyond objects: Generative programming**. In CES97a [146].

[22] CHAVEZ, C.; GARCIA, A. ; LUCENA, C.. **Some Insights on the Use of AspectJ and Hyper/J**. In: WORKSHOP ON AOP AND SOC AT LANCASTER UNIVERSITY, UK, August 2001.

[23] CHAVEZ, C.; GARCIA, A. ; LUCENA, C.. **From AOP to MDSoC: An Experience Report**. Technical Report PUC-RioInf.MCC29/01, Pontifical Catholic University of Rio de Janeiro, 2001. Technical Report PUC-RioInf.MCC29/01.

[24] CHAVEZ, C.; LUCENA, C.. **Design-level Support for Aspect-oriented Software Development**. In: OOPSLA'2001 COMPANION, DOCTORAL SYMPOSIUM (OOPSLA 2001), October 2001.

[25] CHAVEZ, C.; LUCENA, C.. **Design Support for Aspect-oriented Software Development**. In: WORKSHOP ON ADVANCED SEPARATION OF CONCERNS IN OBJECT-ORIENTED SYSTEMS (OOPSLA 2001), October 2001.

[26] CHAVEZ, C.; LUCENA, C.. **A Metamodel for Aspect-Oriented Modeling**. In: WORKSHOP ON ASPECT-ORIENTED MODELING WITH THE UML, 1ST INT'L CONF. ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT (AOSD 2002), April 2002.

[27] CHAVEZ, C.; LUCENA, C.. **A Theory of Aspects for Aspect-Oriented Software Development**. In: PROCEEDINGS OF THE XVII BRAZILIAN SIMPOSIUM ON SOFTWARE ENGINEERING, p. 130–145, October 2003.

[28] CHEN, P.. **The Entity Relationship Model - Towards a Unified View of Data**. ACM Transactions on Database Systems, 1(1):9–36, March 1976.

[29] CLARKE, S.; WALKER, R. J.. **Mapping composition patterns to AspectJ and Hyper/J**. In: WORKSHOP ON ADVANCED SEPARATION OF CONCERNS IN SOFTWARE ENGINEERING (ICSE 2001), May 2001.

[30] CLARKE, S.; WALKER, R. J.. **Composition patterns: an approach to designing reusable aspects**. In: PROCEEDINGS OF

THE 23RD INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, p. 5–14. IEEE Computer Society, 2001.

[31] CLARKE, S.; WALKER, R.. **Towards a standard design language for AOSD**. In ClW02 [139], p. 113–119.

[32] CLARKE, S.. **Composition of Object-Oriented Software Design Models**. PhD thesis, Dublin City University, January 2001.

[33] COADY, Y.; KICZALES, G.; FEELEY, M. ; SMOLYN, G.. **Using AspectC to Improve the Modularity of Path-specific Customization in Operating System Code**. In: JOINT EUROPEAN SOFTWARE ENGINEERING CONFERENCE (ESEC) AND 9TH ACM SIGSOFT INT. SYMP. ON THE FOUNDATIONS OF SOFTWARE ENGINEERING (FSE-9), September 2001.

[34] DAHL, O.; NYGAARD, K.. **Simula - a language for programming and description of discrete event systems**. Technical report, Oslo 3, Norway, Norwegian Computing Center, Forskningveien 1B, September 1967.

[35] DAHCHOUR, M.; PIROTTE, A. ; ZIMÁNYI, E.. **A Generic Role Model for Dynamic Objects**. In: PROCEEDINGS OF THE 14TH INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, p. 643–658. Springer-Verlag, 2002.

[36] DIJKSTRA, E.. **A Discipline of Programming**. Prentice-Hall, 1976.

[37] DIJKSTRA, E. W.. **Go To statement considered harmful**. Comm. ACM, 11(3):147–148, 1968. Letter to the Editor.

[38] **DJ Library Web Page**. http://www.ccs.neu.edu/research/demeter/DJ.

[39] EDER, J.; KAPPEL, G. ; SCHREFL, M.. **Coupling and Cohesion in Object-oriented Systems**, 1992.

[40] ELRAD, T.; FILMAN, R. ; BADER, A.. **Aspect-Oriented Programming**. Communications of the ACM, 44(10):29–32, October 2001.

[41] EVANS, A.; MASKERI, G.; SAMMUT, P. ; WILLANS, J. S.. **Building Families of Languages for Model-Driven System Development**, October 2003. Workshop in Software Model Engineering,

The Sixth International Conference on the Unified Modeling Language, UML 2003, San Francisco, California, USA.

[42] FILMAN, R.; FRIEDMAN, D.. **Aspect-Oriented Programming is Quantification and Obliviousness**. In FF00 [153].

[43] FONTOURA, M.; PREE, W. ; RUMPE, B.. **The UML profile for framework architectures**. Addison-Wesley, 2001.

[44] FAUCONNIER, G.; TURNER, M.. **The Way We Think: Conceptual Blending and The Minds Hidden Complexities**. Basic Books, 2002.

[45] FILMAN, R.. **What Is Aspect-Oriented Programming, Revisited**. In Fil01 [150].

[46] GAMMA, E.; HELM, R.; JOHNSON, R. ; VLISSIDES, J.. **Design Patterns - Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1996.

[47] GARCIA, A.; LUCENA, C. ; COWAN, D.. **Agents in Object-Oriented Software Engineering**. Technical Report CS-2001-07, Computer Science Department, University of Waterloo, May 2001.

[48] GARCIA, A.; LUCENA, C.. **An Aspect-Based Object-Oriented Model for Multi-Agent Systems**. In: ADVANCED SEPARATION OF CONCERNS IN SOFTWARE ENGINEERING AT ICSE'2001, TORONTO, May 2001.

[49] GARCIA, A.; CHAVEZ, C.; SILVA, O.; SILVA, V. ; LUCENA, C.. **Promoting Advanced Separation of Concerns in Intra-Agent and Inter-Agent Software Engineering**. In: WORKSHOP ON ADVANCED SEPARATION OF CONCERNS IN OBJECT-ORIENTED SYSTEMS (OOPSLA 2001), October 2001.

[50] GARCIA, A.; CORTES, M. ; LUCENA, C.. **A Web Environment for the Development and Maintenance of E-Commerce Portals based on a Groupware Approach**. In: Information Resources Management Association Int'l Conference (IRMA 2001), p. 722–724, 2001.

[51] GARCIA, A.; SILVA, V.; CHAVEZ, C. ; LUCENA, C.. **Engineering Multi-Agent Systems with Patterns and Aspects**. Journal of the Brazilian Computer Society, 8(1):57–72, 2002.

[52] GARCIA, A.; SANTANNA, C.; CHAVEZ, C.; SILVA, V.; LUCENA, C. ; VON STAA, A.. **Agents and Objects: An Empirical Study on the Design and Implementation of Multi-Agent Systems**, May 2003. Proc. of the 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2003) at ICSE 2003, Portland, USA.

[53] GARCIA, A.; LUCENA, C. ; COWAN, D.. **Agents in Object-Oriented Software Engineering**, 2004. Software: Practice & Experience, Elsevier (To appear).

[54] GARCIA, A. F.. **From Objects to Agents: An Aspect-Oriented Approach**. PhD thesis, Pontificial Catholic Universisty of Rio de Janeiro, Brazil, April 2004. (submitted).

[55] GUDMUNDSON, S.; KICZALES, G.. **Addressing Practical Software Development Issues in AspectJ with a Pointcut Interface**. In: ADVANCED SEPARATION OF CONCERNS AT ECOOP'2001, 2001.

[56] HANENBERG, S.; UNLAND, R.. **Using and Reusing Aspects in AspectJ**. In: WORKSHOP ON ADVANCED SEPARATION OF CONCERNS AT OOPSLA'01, 2001.

[57] HANNEMANN, J.; KICZALES, G.. **Design pattern implementation in Java and AspectJ**. In: Norris, C.; Fenwick, J. J. B., editors, PROCEEDINGS OF THE 17TH ACM CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS (OOPSLA-02), volumen 37, 11 de **ACM SIGPLAN Notices**, p. 161–173, New York, Nov. 4–8 2002. ACM Press.

[58] HANENBERG, S.; SCHMIDMEIER, A.. **Idioms for building software frameworks in AspectJ**. In: THE SECOND AOSD WORKSHOP ON ASPECTS, COMPONENTS, AND PATTERNS FOR INFRASTRUCTURE SOFTWARE (ACP4IS), 2003.

[59] HARRISON, W.; OSSHER, H.. **Subject-Oriented Programming (A Critique of Pure Objects)**. In: 7TH CONF. ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES AND APPLICATIONS (OOPSLA93), p. 411–428, 1993.

[60] HAYES, B.. **The Post-OOP Paradigm**. American Scientist, 91(2):106–110, March-April 2003.

[61] HERRERO, J.; SÁNCHEZ, F.; LUCIO, F. ; TORO, M.. **Introducing Separation of Aspects at Design Time**. In HerSL+00 [149].

[62] HO, W. M.; JÉZÉQUEL, J.-M.; PENNANEAC'H, F. ; PLOUZEAU, N.. **A toolkit for weaving aspect oriented UML designs**. In HoJez02 [139], p. 99–105.

[63] HURSCH, W. L.; LOPES, C. V.. **Separation of Concerns**, 1995.

[64] **Hyper/J Web Page, http://www.research.ibm.com/ hyperspace/HyperJ/HyperJ.htm**, 2001.

[65] JACOBSON, I.; CHRISTERSON, M.; JONSSON, P. ; OVERGAARD, G.. **Object-Oriented Software Engineering: A Use Case Driven Approach**. Addison-Wesley, 1 edition, 1992.

[66] KANG, K.; COHEN, S.; HESS, J.; NOVAK, W. ; PETERSON, A.. **Feature-oriented domain analysis (FODA) feasibility study**. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[67] KANDE, M. M.; KIENZLE, J. ; STROHMEIER, A.. **From AOP to UML - A Bottom-Up Approach**. In: ASPECT-ORIENTED MODELING WITH UML WORKSHOP, 1ST INTL CONF. ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, April 2002.

[68] KAPLAN, M.; OSSHER, H.; HARRISON, W. ; KRUSKAL, V.. **Subject-Oriented Design and the Watson Subject Compiler**. In: Subjectivity Workshop at OOPSLA'96, 1996. (http://www.research.ibm.com/sop/).

[69] KERSTEN, M. A.; MURPHY, G. C.. **Atlas: A Case Study in Building a Web-based Learning Environment using Aspect-oriented Programming**. In: 14TH CONF. ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS (OOPSLA'99), p. 340–352. ACM, 1999.

[70] KICZALES, G.; DES RIVIERES, J. ; BOBROW, D.. **The Art of the Metaobject Protocol**. MIT Press, Cambridge (MA), USA, 1991.

[71] KICZALES, G.. **Aspect-Oriented Programming**. ACM Computing Surveys, 28(4es):154, 1996.

[72] KICZALES, G.. **Beyond the black box: open implementation**. IEEE Software, 13(1), Jan 1996.

[73] KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C.; LOINGTIER, J. ; IRWIN, J.. **Aspect-Oriented Programming**. In: Aksit, M.; Matsuoka, S., editors, 11TH EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, volumen 1241 de **LNCS**, p. 220–242. Springer-Verlag, 1997.

[74] KICZALES, G.; HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J. ; GRISWOLD, W. G.. **An Overview of AspectJ**. In: Knudsen, J., editor, 15TH EUROPEAN CONF. ON OBJECT-ORIENTED PROGRAMMING, volumen 2072 de **LNCS**, p. 327–353. Springer-Verlag, 2001.

[75] KICZALES, G.; HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J. ; GRISWOLD, W.. **Getting started with AspectJ**. Communications of the ACM, 44(10):59–65, October 2001.

[76] KICZALES, G.; HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J. ; GRISWOLD, W.. **An Overview of AspectJ**. In: EUROPEAN CONF. ON OBJECT-ORIENTED PROGRAMMING, p. 18–22, Budapest, Hungary, June 2001.

[77] KULESZA, U.; GARCIA, A. ; LUCENA, C.. **Generating Aspect-Oriented Agent Architectures**. In: WORKSHOP ON EARLY ASPECTS: Aspect-ORIENTED REQUIREMENTS ENGINEERING AND ARCHITECTURE DESIGN, 3RD INTL CONF. ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, 2004.

[78] LIEBERHERR, K.; HOLLAND, I.. **Assuring Good Style for Object-Oriented Programs**. IEEE Software, p. 38–48, September 1989.

[79] LIEBERHERR, K.; PATT-SHAMIR, B.. **Traversals of Object Structures: Specification and Efficient Implementation**. Technical Report NU-CCS-97-15, College of Computer Science, Northeastern University, Boston, MA, July 1997.

[80] LAI, A.; MURPHY, G. C. ; WALKER, R. J.. **Separating Concerns with Hyper/J: An Experience Report**. In: INT'L WORKSHOP ON MULTI-DIMENSIONAL SEPARATION OF CONCERNS IN SOFTWARE ENGINEERING AT ICSE'00, 2000.

[81] LAMPING, J.. **The Role of Base in Aspect-oriented Programming**. In: INT'L WORKSHOP ON ASPECT-ORIENTED PROGRAMMING AT ECOOP'99, 1999.

[82] LIEBERHERR, K.. **Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns**. PWS Publishing Company, Boston, 1996. ISBN 0-534-94602-X.

[83] LIEBERHERR, K.; LORENZ, D. ; MEZINI, M.. **Programming with Aspectual Components**. Technical Report NU-CCS-99-01, College of Computer Science, Northeastern University, Boston, MA, March 1999.

[84] LIEBERHERR, K.; ORLEANS, D. ; OVLINGER, J.. **Aspect-Oriented Programming with Adaptive Methods**. Communications of the ACM, 44(10):39–41, October 2001.

[85] LISKOV, B.. **Data abstraction and hierarchy**. SIGPLAN Notices, 23(5), May 1988.

[86] LOPES, C. V.; LIEBERHERR, K.. **Abstracting Process-to-Function Relations in Concurrent Object-Oriented Applications**, volumen 821 de **Lecture Notes in Computer Science**, p. 81–99. Spring-Verlag, 1994.

[87] LOPES, C. V.. **Adaptive Parameter Passing**. In: ISOTAS, p. 118–136, 1996.

[88] LOPES, C.; KICZALES, G.. **D: A Language Framework for Distributed Programming**. Technical Report SPL-97-010, Palo Alto Research Center, 1997.

[89] LOPES, C. V.. **D: A Language Framework for Distributed Programming**. PhD thesis, College of Computer Science, Northeastern University, 1998.

[90] LOPES, C. V.. **Aspect-oriented programming: An historical perspective (what's in a name?)**. Technical Report UCI-ISR-

02-5, Institute for Software Research, University of California, Irvine, 2003.

[91] **Omg model driven architecture**, 2003. http://www.omg.org/mda.

[92] **Meta-Object Facility (MOF), version 1.4**, 2002. http://www.omg.org.

[93] MAES, P.. **Concepts and experiments in computational reflection**. In: PROC. OF THE 2ND ANNUAL CONF. ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS (OOPSLA'87), p. 147–155. ACM Press, October 1987.

[94] MARTIN, R.. **Design Principles and Design Patterns**, 2000. http://www.objectmentor.com.

[95] MASUHARA, H.; KICZALES, G.. **Modeling Crosscutting in Aspect-Oriented Mechanisms**. In: PROCEEDINGS OF THE ECOOP 2003, p. 2–28, 2003.

[96] MEHNER, K.; RASHID, A.. **Towards a Standard Interface for Runtime Inspection in AOP Environments**. In: Chu-Carrol, M.; Murphy, G., editors, WORKSHOP ON TOOLS FOR ASPECT-ORIENTED SOFTWARE DEVELOPMENT AT OOPSLA'02, 2002.

[97] MEHNER, K.; RASHID, A.. **GEMA: A Generic Model for AOP (Extended Abstract)**. In: BELGIAN AND DUTCH WORKSHOP ON ASPECT-ORIENTED PROGRAMMING, 2003.

[98] MENDHEKAR, A.; KICZALES, G. ; LAMPING, J.. **RG: A Case-Study for Aspect-Oriented Programming**. Technical Report SPL-97-009, Palo Alto Research Center, 1997.

[99] **Merriam-webster online**, February 2004. http://www.webster.com/.

[100] MEYER, B.. **Object-oriented Software Construction**. Prentice Hall, 1988.

[101] MURPHY, G. C.; WALKER, R. J. ; BANIASSAD, E. L.. **Evaluating Emerging Software Development Technologies: Lessons Learned from Assessing Aspect-oriented Programming**. IEEE Trans. on Software Engineering, 25(4):438–455, 1999.

[102] MURPHY, G. C.; LAI, A.; WALKER, R. J. ; ROBILLARD, M. P.. **Separating Features in Source Code: An Exploratory Study**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), p. 275–284, May 2001.

[103] NAGY, I.; AKSIT, M. ; BERGMANS, L.. **Composition Graphs: a Foundation for Reasoning about Aspect-Oriented Composition**. In: WORKSHOP ON FOUNDATIONS OF ASPECT-ORIENTED LANGUAGES (FOAL) AT AOSD'2003, 2003.

[104] NORDBERG III, M.. **Aspect-Oriented Dependency Inversion**. In: Proc. OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems (2001), 2001.

[105] OSTERMANN, K.; KNIESEL, G.. **Independent Extensibility—An Open Challange for AspectJ and Hyper/J**. In: INT'L WORK. ON ASPECTS AND DIMENSIONAL COMPUTING AT ECOOP'00, 2000.

[106] Object Management Group – Agent Platform Special Interest Group. *Agent Technology – Green Paper*. Version 1.0, September 2000.

[107] OSSHER, H.; TARR, P.. **Operation-Level Composition: A Case in (Join) Point**. In: INT'L WORKSHOP ON ASPECT-ORIENTED PROGRAMMING AT ECOOP'98, 1998.

[108] OKAMURA, H.; ISHIKAWA, Y.. **Object location control using meta-level programming**. Lecture Notes in Computer Science, 821:299–320, 1994.

[109] OPDAHL, A. L.; SINDRE, G.. **Facet Modeling: An Approach to Flexible and Integrated Conceptual Modeling**. Information Systems, 22(5):291–323, 1997.

[110] ORLEANS, D.; LIEBERHERR, K.. **DJ: Dynamic Adaptive Programming in Java**. Technical Report NU-CCS-2001-02, College of Computer Science, Northeastern University, Boston, MA, March 2001.

[111] OSSHER, H.; TARR, P.. **Multi-dimensional Separation of Concerns in Hyperspace**. In: INT'L WORKSHOP ON ASPECT-ORIENTED PROGRAMMING AT ECOOP'99, 1999.

[112] OSSHER, H.; TARR, P.. **Multi-dimensional separation of concerns using hyperspaces**. Technical Report 21452, IBM, Apr 1999.

[113] OSSHER, H.; TARR, P.. **Using Multi-dimensional Separation of Concerns to (Re)Shape Evolving Software**. Communications of the ACM, 44(10):43–50, October 2001.

[114] PARNAS, D.. **On the Criteria To Be Used in Decomposing Systems into Modules**. Communications of the ACM, 15(12):1053–1058, December 1972.

[115] RUMBAUGH, J. R.; (AND WILLIAM LORENSEN, M. R. B.; EDDY, F. ; PREMERLANI, W.. **Object-Oriented Modeling and Design**. Prentice Hall, 1 edition, 1990.

[116] RUMBAUGH, J.; JACOBSON, I. ; BOOCH, G.. **The Unified Modeling Language Reference Manual**. Addison-Wesley, 1999.

[117] RYMAN, A.. **The Theory-Model Paradigm in Software Design**. Technical Report TR74.048, IBM Tech. Report, IBM Toronto, Ont., October 1989.

[118] SANTANNA, C.; GARCIA, A.; CHAVEZ, C.; VON STAA, A. ; LUCENA, C.. **On the reuse and maintenance of aspect-oriented software: An evaluation framework**. In: PROCEEDINGS OF THE XVII BRAZILIAN SIMPOSIUM ON SOFTWARE ENGINEERING, p. 19–34, October 2003.

[119] SHAW, M.; GARLAN, D.. **Software Architecture: Perspective on an Emerging Discipline**. Prentice Hall, 1996.

[120] SILVA, V.; GARCIA, A.; BRANDAO, A.; CHAVEZ, C.; LUCENA, C. ; ALENCAR, P.. **Taming Agents and Objects in Software Engineering**, p. 1–25. Software Engineering for Large-Scale Multi-Agent Systems – LNCS 2603. Springer-Verlag, April 2003. A. Garcia, C. Lucena, F. Zambonelli, A. Omicini, J. Castro (Eds).

[121] SKIPPER, M.. **The Watson Subject Compiler & AspectJ (A Critique of Practical Objects)**. In: WORKSHOP ON MULTI-DIMENSIONAL SEPARATION OF CONCERNS (OOPSLA 1999), November 1999.

[122] SMITH, B. C.. **Reflection and semantics in lisp**. In: PROCEEDINGS OF THE 11TH ACM SIGACT-SIGPLAN SYMPOSIUM ON

PRINCIPLES OF PROGRAMMING LANGUAGES, p. 23–35. ACM Press, 1984.

[123] SNYDER, A.. **Encapsulation and Inheritance in Object-oriented Programming Languages**, 1986.

[124] SOARES, S.; LAUREANO, E. ; BORBA, P.. **Implementing distribution and persistence aspects with AspectJ**. In: PROCEEDINGS OF THE 17TH ACM CONFERENCE ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES, AND APPLICATIONS, OOPSLA"02. ACM Press, November 2002. ACM SIGPLAN Notices.

[125] SOARES, S.; BORBA, P.. **PaDA: A pattern for distribution aspects**. In: SECOND LATIN AMERICAN CONFERENCE ON PATTERN LANGUAGES OF PROGRAMMING — SUGARLOAF-PLOP, Aug. 2002.

[126] **SoC and MAS Research Group Website**. http://www.teccomm.les.inf.puc-rio.br/SoCAgents.

[127] STEIN, D.. **An Aspect-Oriented Design Model Based on AspectJ and UML**. Master's thesis, University of Duisburg-Essen, January 2002.

[128] STEIN, D.; HANENBERG, S. ; UNLAND, R.. **A uml-based aspect-oriented design notation for aspectj**. In: PROCEEDINGS OF THE 1ST INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, p. 106–112. ACM Press, 2002.

[129] SUZUKI, J.; YAMAMOTO, Y.. **Extending UML with Aspects: Aspect Support in the Design Phase**. In: 3RD ASPECT-ORIENTED PROGRAMMING (AOP) WORKSHOP AT EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING (ECOOP), 1999.

[130] TAIVALSAARI, A.. **On the Notion of Inheritance**. ACM Computing Surveys, 28(3):438–479, September 1996.

[131] TARR, P.; OSSHER, H.. **Hyper/J User and Installation Manual**. http://www.alphaworks.ibm.com/tech/hyperj.

[132] TARR, P.; OSSHER, H.; HARRISON, W. ; JR., S. S.. **N Degrees of Separation: Multi-Dimensional Separation of Concerns**. In: 21st Int'l Conf. on Software Engineering (ICSE'99), p. 107–119, May 1999.

[133] **Technology Review 10 (Ten Emerging Technologies That Will Change the World)**. Technology Review, 104(1):97–113, January/February 2001.

[134] **OMG Unified Modeling Language Specification, Version 1.4**, September 2001. http://www.omg.org/uml/.

[135] **Unified Modeling Language (UML) Specification: Infrastructure Version 2.0**, December 2003. http://www.omg.org/uml/.

[136] WALKER, R. J.; BANIASSAD, E. L. ; MURPHY, G. C.. **An Initial Assessment of Aspect-Oriented Programming**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), p. 120–130, May 1999.

[137] WEGNER, P.. **Dimensions of Object-based Language Design**. In: 2ND CONF. ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES, AND APPLICATIONS (OOPSLA'87), p. 168–182, October 1987.

[138] WEGNER, P.. **The Paradigm Shift from Algorithms to Interaction**. DRAFT, October 1996.

[139] **Proc. 1st Int'l Conf. on Aspect-Oriented Software Development (AOSD-2002)**. ACM Press, April 2002.

[140] **Workshop on Identifying, Separating and Verifying Concerns in the Design (AOSD-2002)**, Mar. 2002.

[141] **Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design (AOSD-2002)**, Mar. 2002.

[142] **Workshop on Aspect-Oriented Modeling with UML (AOSD-2002)**, Mar. 2002.

[143] **Proc. 2nd Int'l Conf. on Aspect-Oriented Software Development (AOSD-2003)**. ACM Press, March 2003.

[144] **Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design (AOSD-2003)**, Mar. 2003.

[145] **Workshop on Aspect-Oriented Modeling with UML (AOSD-2003)**, March 2003.

[146] **Workshop on Aspect Oriented Programming (ECOOP 1997)**, June 1997.

[147] **Workshop on Aspect Oriented Programming (ECOOP 1998)**, June 1998.

[148] **Int'l Workshop on Aspect-Oriented Programming (ECOOP 1999)**, June 1999.

[149] **Workshop on Aspects and Dimensions of Concerns (ECOOP 2000)**, June 2000.

[150] **Workshop on Advanced Separation of Concerns (ECOOP 2001)**, June 2001.

[151] **Int'l Workshop on Aspect Oriented Programming (ICSE 1998)**, Apr. 1998.

[152] **Workshop on Advanced Separation of Concerns in Software Engineering (ICSE 2001)**, May 2001.

[153] **Workshop on Advanced Separation of Concerns (OOPSLA 2000)**, Oct. 2000.

[154] **Workshop on Advanced Separation of Concerns in Object-Oriented Systems (OOPSLA 2001)**, October 2001.

A
Transformações

A.1
De aSideML para o metamodelo aSide

Nomes de metaclasses estão escritos com a primeira letra de cada palavra em maiúscula.

– Aspecto

Um símbolo de aspecto é mapeado em um elemento `Aspect` (6.2). O conteúdo do compartimento de nome é mapeado no nome do aspecto. O compartimento de atributos é mapeado em uma lista de `Attributes` locais ao elemento Aspect, e o compartimento de operações mapeia em uma lista de `Operations` locais ao elemento Aspect. Cada interface transversal é mapeada em um elemento `CrosscuttingInterface` (B.1.3.6).

A adição de uma caixa tracejada de template a um símbolo de aspecto faz com quem ocorra a adição dos nomes de parâmetros em cada lista como `ModelElement`s no contexto do `Namespace` do elemento `CrosscuttingInterface` correspondente.

– Interface transversal

O símbolo de interface transversal é mapeado em um elemento `CrosscuttingInterface` (B.1.3.6) que pertence ao `Namespace` de um aspecto. O conteúdo do compartimento de nome é mapeado no nome da interface transversal. Os outros compartimentos são mapeados em uma lista de elementos do tipo `CrosscuttingFeatures` (B.1.3.8).

A adição de uma caixa tracejada de template a um símbolo de aspecto faz com quem ocorra a adição dos nomes de parâmetros em cada lista como `ModelElement`s no contexto do `Namespace` do elemento `CrosscuttingInterface` correspondente.

– Característica transversal

Um entrada textual (identificador) no compartimento `Uses` é mapeada em um elemento `BehavioralFeature`. Um entrada textual (identificador) no compartimento `Additions` é mapeada em um elemento `StructuralCrosscuttingFeature` (B.1.3.8) ou em um elemento `BehavioralCrosscuttingFeature` (B.1.3.9). Entradas textuais em outros compartimentos são mapeadas em `BehavioralCrosscuttingFeature`s (B.1.3.9).

Todas as entradas textuais são mapeadas em elementos ligados a um `CrosscuttingInterface` (B.1.3.6) que corresponde ao símbolo de interface transversal.

– Crosscutting

O símbolo de crosscutting é mapeado em um relacionamento de *crosscutting* , denotado pela metaclasse `Crosscutting` (B.1.3.3). A lista de bindings é mapeada a uma lista de `TemplateMatch` (B.1.3.4).

O símbolo de crosscutting no estilo um-para-vários com `n` destinos é mapeado para `n` elementos `Crosscutting`.

– Order

O símbolo do relacionamento Order é mapeado em uma `Dependency` (de UML) com estereótipo `precede`.

– Requirement

O símbolo do relacionamento é mapeado em um elemento `Usage` (de UML) com estereótipo `require`.

– Diagrama de aspectos

Cada símbolo de aspecto é mapeado em um elemento `Aspect` (B.1.3.5).

– Diagrama de classes estendido

Cada símbolo de aspecto é mapeado em um elemento `Aspect` (B.1.3.5). O símbolo de *crosscutting* é mapeado em um relacionamento de `Crosscutting` (5.2.4.1). O mapeamento de outros elementos segue as regras de UML.

– Instância de aspecto

O símbolo de instância de aspecto é mapeado em um elemento `AspectInstance` (B.3.3.1) do `Aspect` obtido a partir do valor de `aspectName`.

– Colaboração aspectual

Uma colaboração aspectual é mapeada em um elemento `AspectualCollaboration`. O mapeamento de papéis (roles) e interações segue as regras de UML para `Collaboration` ([134], pp. 3-118).

– Interação aspectual

Uma interação aspectual é mapeada em um elemento `AspectualInteraction` (B.3.3.3).

– Diagrama de colaboração aspectual

Um diagrama de colaboração aspectual é mapeado em um elemento `AspectualCollaboration`.

– Diagrama de interação aspectual

Um diagrama de interação aspectual é mapeado em um elemento `AspectualInteraction` e um elemento `AspectualCollaboration`.

A.2 De `aSideML` para AspectJ

AspectJ	`aSideML`
Aspecto	Um aspecto parametrizado de `aSideML` é mapeado para um aspecto abstrato de AspectJ e aspectos concretos podem ser definidos usando os argumentos fornecidos para parâmetros de template através de relacionamentos de *crosscutting*.
Introduction	Operações de *crosscutting* definidas no compartimento Additions
Advice	Operações de *crosscutting* definidas nos compartimentos Refinements e Redefinitions
Pointcut	Os parâmetros de template de operação podem ser definidos e referenciados dentro de especificações de interação, denotando que são pontos de combinação para o comportamento *crosscutting*. Esses templates podem ser substituídos por operações várias vezes e são, portanto, equivalentes a pointcuts.
Precedence	Relacionamentos de precedência (`precede`)
Declare parents	Relacionamentos entre interfaces transversais, interfaces e classes

Tabela A.1: Mapeamento entre `aSideML` e AspectJ

AspectJ e `aSideML` suportam a teoria de aspectos definida no Capítulo 3 e, desse modo, as duas linguagens compartilham um framework conceitual que facilita o mapeamento entre seus elementos (veja a Tabela 3.3). A Tabela A.1 mostra alguns detalhes do mapeamento preliminar

entre elementos das duas linguagens. Esse mapeamento precisa ser elaborado, inclusive para permitir a engenharia reversa de programas AspectJ para modelos representados em `aSideML`.

O processo de transformação e as regras implícitas contidas na Tabela A.1 também merecem investigação mais detalhada, no sentido de otimizá-los, por exemplo, para minimizar o número de aspectos gerados, para fornecer meios de abstrair expressões de *wildcards* que denotam elementos base afetados por aspectos e organizá-los em pointcuts, etc.

A.3 De `aSideML` para Hyper/J

A transformação de um modelo `aSideML` para um modelo Hyper/J requer um mapeamento entre seus elementos. Nesse contexto, vale recordar que, a rigor, não caracterizamos Hyper/J como orientado a aspectos. Porém, se relaxarmos alguns parâmetros da equação de orientação a aspectos proposta na Seção 3.2, de modo a tratar Hyper/J como uma abordagem orientada a aspectos, observamos que um aspecto em `aSideML` pode ser mapeado em um hyperslice e pelo menos um arquivo de hypermodule. Nesse caso, a especificação das operações de *crosscutting* aparece no hyperslice e a especificação de pontos de combinação aparece em hypermodules.

Em relação ao relacionamentos de *crosscutting* , a partir de experimentos realizados [23], observamos que podemos fazer a transformação de um relacionamento de *crosscutting* (e seus argumentos) expresso em `aSideML` para Hyper/J usando pelo menos duas alternativas distintas.

Na primeira alternativa – baseada na estratégia de composição *mergeByName* – o nome de um método afetado por uma operação de *crosscutting* é usado para nomear o corpo dessa operação. O nome do aspecto é usado para nomear um novo hyperslice ou pacote Java, e o nome da classe afetada pelo aspecto é usado para nomear uma nova classe criada naquele pacote. Esta alternativa é mais simples de implementar e gera arquivos hypermodule menores.

Na segunda alternativa – baseada na estratégia de composição *equate and mergeByName* – o nome da operação de *crosscutting* é usado para nomear o corpo dessa mesma operação, que se torna um método comum. O novo método gerado deve ser explicitamente *equated* com os métodos base que ele afeta. O nome do aspecto é usado para nomear um novo hyperslice ou pacote Java, e o nome de cada interface transversal é usado para nomear uma nova classe criada naquele pacote. As novas classes são *equated* com

as classes base que elas afetam. Esta alternativa requer que as unidades correspondentes sejam explicitamente declaradas no arquivo hypermodule, tornando os arquivos gerados maiores.

Em [22, 23], nós propusemos um mapeamento do modelo bi-dimensional de POA para um modelo multi-dimensional, resultando em uma série de regras de transformação para gerar uma implementação em Hyper/J a partir de uma implementação em AspectJ. A transformação propriamente dita foi ilustrada através do SMA Portalware. Um dos propósitos desse trabalho de mapeamento foi o de conhecer melhor cada abordagem (AspectJ e Hyper/J), relacionar seus conceitos e avaliar seus pontos fortes e fracos. Ele também ajudou a validar o modelo de aspectos independente de linguagem proposto nesta tese e permitiu-nos explorar as possibilidades de mapeamento entre eles.

B
Especificação do Metamodelo `aSide`

Este Apêndice descreve o metamodelo `aSide`, detalhando informações apresentadas no Capítulo 6.

Organização do Capítulo

As Seções B.1 a B.6 descrevem os quatro submodelos que agregam os principais elemento do modelo `aSide`. Cada um desses submodelos é descrito por um metamodelo, e os novos conceitos são apresentados como metaclasses.

A apresentação segue um padrão adaptado a partir da especificação do metamodelo de UML, v1.4 [134]. Ela se inicia com um ou mais diagramas que descrevem a sintaxe abstrata das construções de linguagem (i.e. as metaclasses e seus relacionamentos). Para cada metaclasse, as seguintes informações são fornecidas:

- **Descrição**: Definição informal da metaclasse que especifica a nova construção de linguagem.
- **Associações**: Uma lista de extremidades opostas das associações conectadas à metaclasse.
- **Semântica**: Uma descrição da semântica em linguagem natural de como usar os elementos de modelagem [134].

As regras de modelos bem formados não constam deste Apêndice.

B.1 O Pacote Aspect Core

B.1.1 Visão Geral

O pacote *Aspects.Foundations.Core*, ou simplesmente o pacote *Aspect Core*, define as construções abstratas e concretas básicas do metamodelo necessárias para a criação de modelos de aspectos. As construções abstratas definidas em Aspect Core incluem BaseElement, CrosscuttingElement e CrosscuttingFeature. As construções concretas especificadas em Aspect Core incluem Aspect, Crosscutting e CrosscuttingInterface.

B.1.2 Sintaxe Abstrata

A sintaxe abstrata para o pacote Aspect Core é expressa em notação gráfica nas figuras a seguir.

A Figura B.1 mostra os elementos de modelagem que incorporam o backbone estrutural do metamodelo `aSide`.

A Figura B.2 mostra os elementos de modelagem que definem aspectos.

A Figura B.3 mostra os elementos de modelagem que definem relacionamentos.

B.1.3 Descrições de Classe

Esta Seção contém a descrição das classes definidas no pacote Aspect Core.

B.1.3.1 BaseElement

Um elemento base é um elemento nomeado que pode participar de um relacionamento de *crosscutting* com um elemento *crosscutting* e, portanto, pode ser melhorado por um elemento *crosscutting* em pontos de combinação bem definidos. Ele pode participar de vários relacionamentos de *crosscutting*. `BaseElement` é uma metaclasse abstrata que especifica elementos base.

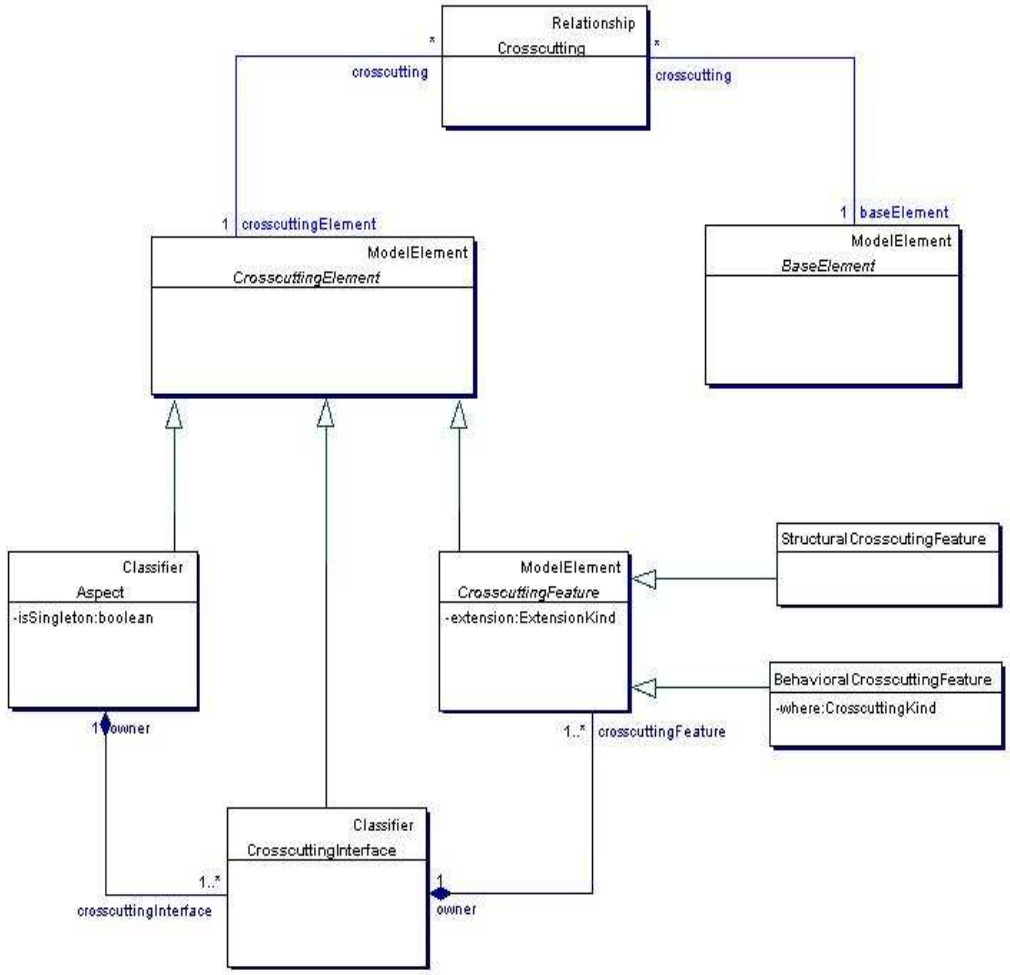

Figura B.1: Aspect Core: Backbone

Associações

o crosscutting Uma coleção de relacionamentos Crosscutting de um ou mais CrosscuttingElements que melhoram o BaseElement. o joinPoint (associação derivada) Um conjunto de JoinPoints referenciados no contexto de um relacionamento Crosscutting.

– *crosscutting*

Uma coleção de relacionamentos `Crosscutting` de um ou mais `CrosscuttingElements` que estendem `BaseElement`.

– *joinPoint* (associação derivada)

Um conjunto de `JoinPoints` que são referenciados por um relacionamento de `Crosscutting`.

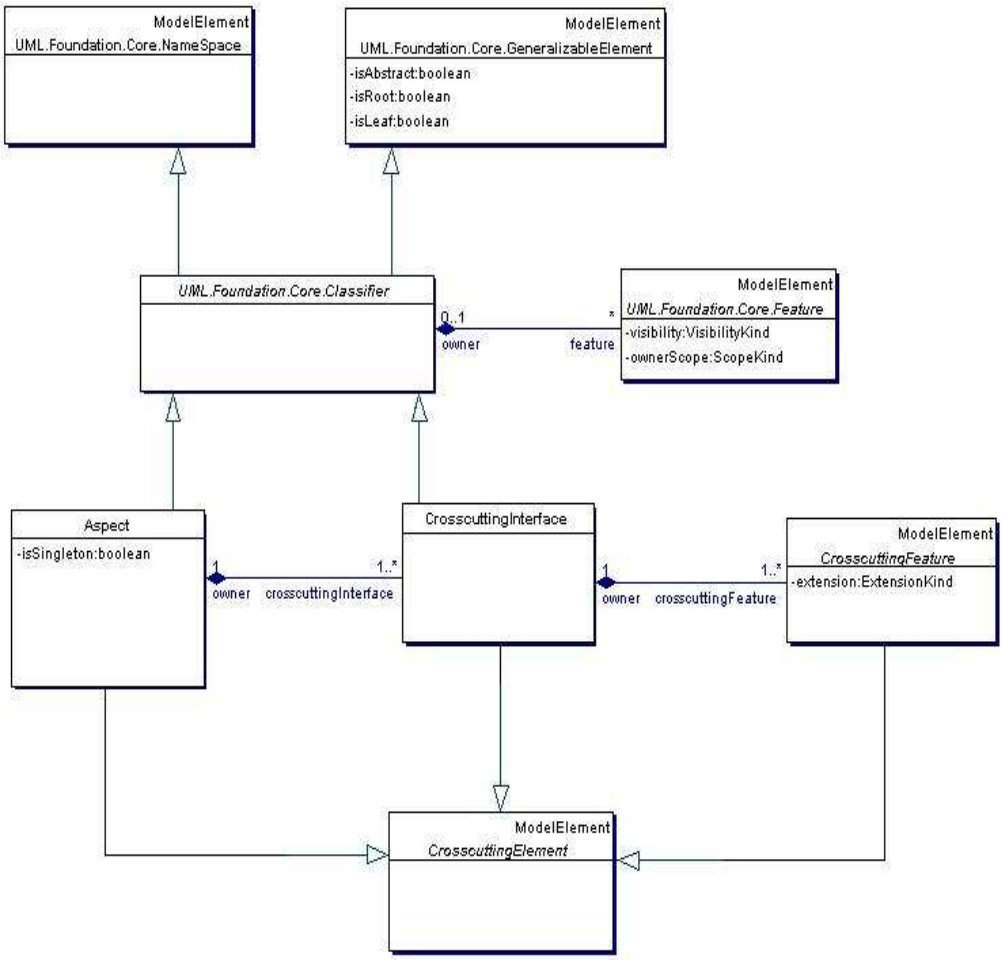

Figura B.2: Aspect Core: Aspectos

Semântica

`BaseElement` representa the general state or ability of being enhanced by a `CrosscuttingElement` in the context of a crosscutting relationship. The detailed semantics of enhancement varies for each specialization of `BaseElement`.

Um `BaseElement` representa o estado geral ou a capacidade de ser melhorado por um `CrosscuttingElement` no contexto de um relacionamento de *crosscutting*. A semântica detalhada da melhoria varia para cada especialização de `BaseElement`.

`BaseElement` é uma metaclasse abstrata que ajuda a definição do relacionamento de *Crosscutting*.

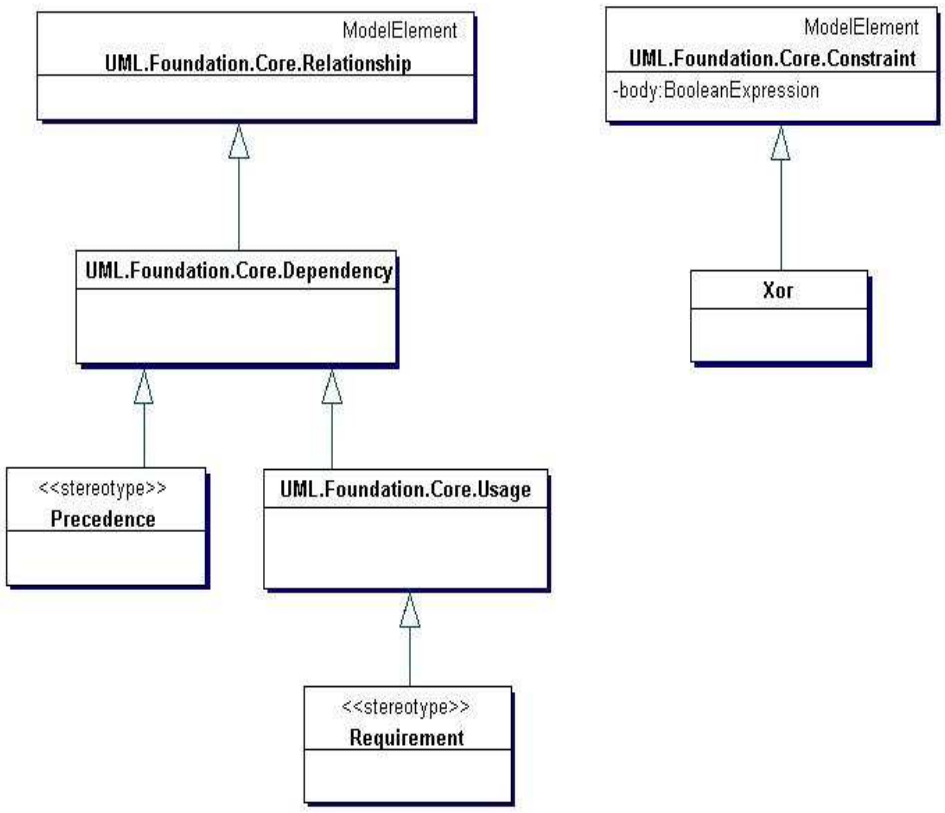

Figura B.3: Aspect Core: Dependências

B.1.3.2 CrosscuttingElement

Um elemento crosscutting é um elemento nomeado que pode participar de um relacionamento de *crosscutting* com um elemento base e, portanto, pode melhorá-lo por meio de *crosscutting*.

Um elemento *crosscutting* pode participar de vários relacionamentos de *crosscutting*.

`CrosscuttingElement` é uma metaclasse abstrata que especifica elementos *crosscutting*. Um `CrosscuttingElement` com pelo menos uma associação `TemplateParameter` é um elemento *crosscutting* parametrizado.

Associações

– *crosscutting*

Uma coleção de relacionamentos `Crosscutting` de um ou mais `BaseElements` que o `CrosscuttingElement` melhora.

– *baseElement*(associação derivada)

Um conjunto de `BaseElements` melhorado por `CrosscuttingElement`.

– *templateParameter* (herdado de ModelElement)

O conjunto completo dos `TemplateParameters` formais para o `CrosscuttingElement`.

Semântica

Um `CrosscuttingElement` representa capacidade geral de melhorar um elemento base no contexto de um relacionamento de *crosscutting*. A semântica detalhada de melhoria varia para cada especialização de `CrosscuttingElement`.

B.1.3.3 Crosscutting

Crosscutting é um relacionamento de um elemento *crosscutting* com um elemento base que especifica que o elemento *crosscutting* melhora a estrutura ou comportamento do elemento base em locais bem definidos.

Quando o elemento *crosscutting* é parametrizado, o relacionamento de *crosscutting* possui um conjunto de correspondências de parâmetros de template. Cada correspondência associa um parâmetro formal declarado no elemento *crosscutting* para parâmetros reais que indicam os locais nos elementos base em que outras estruturas e comportamentos serão combinados (Figure B.4).

Pode ser aplicada a restrição {`xor`} a um conjunto de relacionamentos *crosscutting* que compartilham uma conexão a um elemento base, especificando que nesse conjunto, apenas um aspecto afetará esse elemento base.

`Crosscutting` é uma metaclasse concreta que especifica relacionamentos de *crosscutting* .

Associações

– *baseElement*

O `BaseElement` associado a esse `Crosscutting`.

– `crosscuttingElement`

O `CrosscuttingElement` associado a esse Crosscutting.

– *match*

Os `TemplateMatches` associados com este `Crosscutting`.

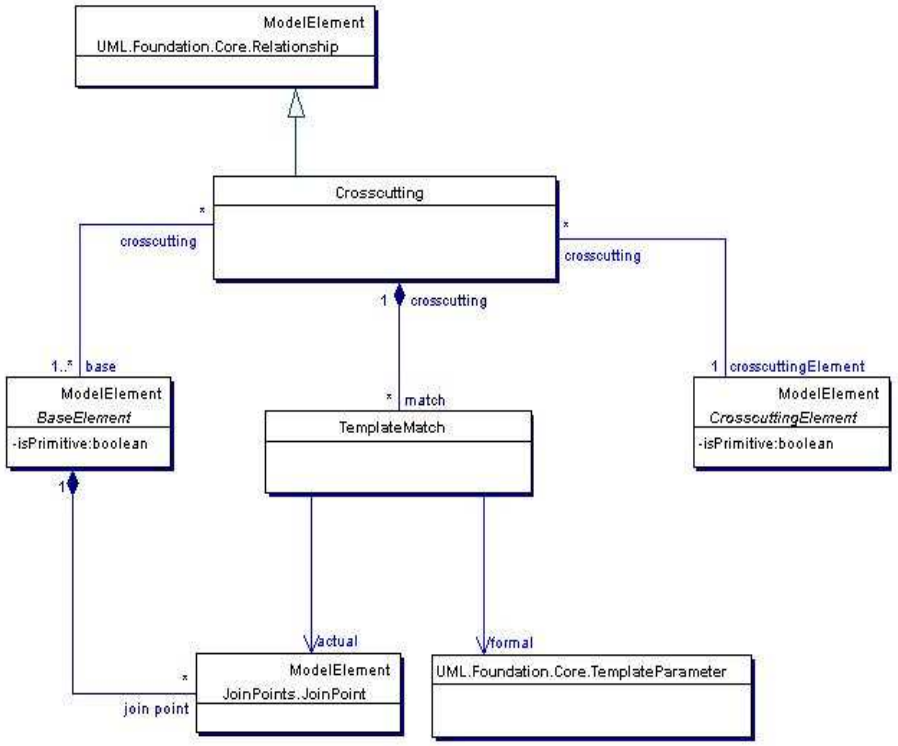

Figura B.4: Crosscutting.

– *strategy*

O WeavingStrategy associado a esse `Crosscutting`.

Semântica

Quando um relacionamento de *crosscutting* relaciona um elemento crosscutting a um elemento base, cada instância do elemento base pode ser melhorada por uma instância do elemento crosscutting. *Crosscutting estático* oferece melhorias à estrutura de elementos base. *Crosscutting dinâmico* oferece melhorias ao comportamento de elementos base. A semântica detalhada de melhoria varia para cada especialização de `BaseElement` e `CrosscuttingElement`.

O relacionamento de *crosscutting* denota um novo mecanismo de extensão para a combinação de descrições incrementais da estrutura e do comportamento localizados em elementos *crosscutting* em elementos base. O relacionamento de *crosscutting* deve denotar a combinação de propriedades *crosscutting* adicionais de objetos reais, enquanto o relacionamento de generalização comum e o mecanismo de herança lidam com propriedades essenciais e estáticas.

Para elementos *crosscutting* parametrizados, o relacionamento de *crosscutting* também realiza uma associação que define os elementos base e as operações que substituem os parâmetros de template associados aos elementos *crosscutting*.

O *mecanismo de combinação* realiza a composição de elementos relacionados pelo relacionamento de *crosscutting* de acordo com uma *estratégia de combinação*.

B.1.3.4 TemplateMatch

Uma correspondência (ou casamento) de template relaciona o(s) parâmetro(s) real(is) a um parâmetro de template formal como parte de um relacionamento de *crosscutting*.

`TemplateMatch` é uma metaclasse que associa um ou mais parâmetros reais a um parâmetro de template formal dentro do contexto de um relacionamento de *crosscutting*.

Associações

- *actual*

  O conjunto de elementos que são parâmetros reais dessa correspondência.

- *formal*

  O TemplateParameter formal associado a essa correspondência.

- *crosscutting*

  O Crosscutting que possui essa correspondência.

B.1.3.5 Aspect

Um aspecto é uma descrição de um conjunto de características, organizadas em interfaces transversais, que melhoram a estrutura e o comportamento de classes por meio de *crosscutting* de formas sistêmicas. Um aspecto deve ter no mínimo uma interface transversal.

Um *aspecto parametrizado* se abstrai em relação à identidade das classes que irão afetar, declarando parâmetros formais a fim de manter os nomes reais de classes e métodos.

Um aspecto não pode ser explicitamente instanciado. Sua instanciação depende da instanciação das classes que melhora, relacionadas pelo relacionamento de *crosscutting*. Os pontos de combinação são definidos por meio da associação das características dos elementos base aos parâmetros dos aspectos.

`Aspect` é uma metaclasse concreta que especifica aspectos. É uma especialização de `CrosscuttingElement` e `Classifier`.

Atributos

– *isSingleton*

Especifica se o Aspecto possui apenas uma `AspectInstance`. Se verdadeiro, então um Aspecto possui apenas uma instância de aspecto, disponível para cada objeto base que afeta. Como o aspecto, é informalmente chamada de um aspecto único. Se falso, então há uma instância de aspecto para cada objeto base.

Associações

– *crosscuttingInterface*

O conjunto de `CrosscuttingInterfaces`.

– *crosscuttingFeature* (associação derivada)

O conjunto de `CrosscuttingFeatures`.

– *feature*

(herdado de `Classifier`) O conjunto de `Features` locais.

– *ownedElement*

(herdado de `Namespace`) O conjunto de elementos internos definidos dentro do aspecto.

Semântica

Como um `CrosscuttingElement`, um `Aspect` pode participar de relacionamentos Crosscutting com um ou mais `BaseElement`s.

Como um `Classifier`, um `Aspect` pode declarar um conjunto de Features locais, como Attributes, Methods e Operations. Attributes locais são usados a fim de representar o estado do Aspect e Operations atuam sobre eesses Attributes. Um Aspect possui um nome, que é único no Namespace que o contém.

Como Classifier é um filho de GeneralizableElement e Namespace, Aspect também é um filho de GeneralizableElement e Namespace. Como um `GeneralizableElement`, um Aspect pode herdar características locais e CrosscuttingInterfaces e também introduzir novas características locais e CrosscuttingInterfaces.

Como um `Namespace`, um Aspect pode declarar outros Classifiers aninhados nesse escopo. As regras de visibilidade para Classifiers aninhados também são verdadeiras para Aspects.

Como um `CrosscuttingElement` e um `Classifier`, um Aspect define um conjunto de CrosscuttingFeatures que serão combinados a `BaseElement`s em JoinPoints bem definidos (locais específicos nos elementos base em que essa combinação ocorrerá) por meio de *crosscutting*.

As especificações de `JoinPoints` e `CrosscuttingFeatures` são organizadas em `CrosscuttingInterfaces`. Uma `CrosscuttingInterface` especifica um conjunto de pontos de combinação e descreve as melhorias fornecidas pelo Aspect nesses pontos: a introdução de novas `Features`, o refinamento ou redefinição de `BehavioralFeatures` existentes etc. Para cada `CrosscuttingFeature` comportamental, o Aspect oferece `Aspectual Interactions` a fim de descrever as interações entre instâncias base e instâncias de aspecto.

Um Aspect pode ter várias `CrosscuttingInterfaces`. Um Aspect que define duas ou mais `CrosscuttingInterfaces` oferecem suporte a *crosscutting* horizontal (ou heterogêneo) em um conjunto de `BaseElement`s. Isso significa que o Aspect melhora dois ou mais `BaseElement`s de formas diferentes. Como as Características Crosscutting definidas em uma `CrosscuttingInterface` podem usar Características Crosscutting definidas em outra `CrosscuttingInterface` (definida no mesmo Aspecto), a especificação de um Aspecto parece uma `Colaboração`. Entretanto, diferente de uma `Collaboration`, os Aspectos podem definir características locais e também podem melhorar os objetos que estão em conformidade com suas

`CrosscuttingInterfaces`. Os aspectos podem adicionar (introduzir), refinar (before, after, around) e redefinir (around) características.

Um `Aspect` com pelo menos uma associação `TemplateParameter` é um aspecto parametrizado (por definição).

Um aspecto parametrizado é um aspecto com um ou mais parâmetros formais livres. A semântica dos (modelos de) elementos parametrizados de UML é adotada com algumas variações. Em UML, o conteúdo dos modelos não está sujeito às regras de boa formação dos modelos.

Os parâmetros formais podem ser classificadores ou operações. O nome da interface transversal é um parâmetro, bem como os nomes de algumas características comportamentais. O nome de uma operação requerida é um parâmetro que deve ser associado a uma operação de um elemento base. O relacionamento de *crosscutting* associa os parâmetros a valores. Esses valores são nomes definidos em algum elemento base. Um parâmetro de modelo possui escopo somente dentro de sua interface transversal.

Os aspectos podem ser organizados em *hierarquias de tipo/subtipo*, nas quais os subaspectos herdam as propriedades de seus superaspectos. As propriedades herdáveis são características locais e interfaces transversais. Os subaspectos podem estender seus superaspectos ao fornecer outras interfaces transversais ou especializando características locais.

Rationale

Os aspectos parametrizados são usados para a modelagem de aspectos sistêmicos. Os aspectos comuns (*plain*) são usados para a modelagem de elementos *crosscutting* que afetam apenas um elemento base.

Um aspecto parametrizado também representa uma construção de projeto que pode ser usada repetidamente em diferentes projetos. Os participantes no Aspecto, incluindo os Classificadores e os Relacionamentos, podem ser parâmetros definidos em uma `CrosscuttingInterface` do Aspect genérico.

B.1.3.6 CrosscuttingInterface

Uma interface transversal é um conjunto nomeado de características transversais e operações requeridas que organiza muitos comportamentos e estruturas *crosscutting* coesos definidos dentro de aspectos.

Um aspecto pode ter uma ou mais interfaces transversais. Cada interface transversal oferece uma descrição parcial do comportamento e da estrutura *crosscutting* do aspecto.

Uma interface transversal consiste em um conjunto de características transversais e características requeridas. Uma característica transversal declara uma característica estrutural ou comportamental que melhora elementos base. As características necessárias especificam serviços que um aspecto precisa usar a fim de realizar sua função.

Uma interface transversal abstrai da identidade de elementos base. A abstração é suportada através de parametrização. O nome da interface transversal é um parâmetro formal que denota o nome do elemento base, e os nomes das características transversais comportamentais denotam características comportamentais desse elemento base. O pronome `base` pode ser usado no corpo de uma característica transversal comportamental a fim de representar uma instância de um elemento base.

A interface transversal pode afetar diferentes elementos base. Para cada elemento base, ela especifica uma visão enriquecida, na qual as melhorias estruturais enriquecem a interface modular dos elementos base, e as melhorias comportamentais enriquecem algumas de suas interações. Uma interface transversal também especifica uma visão restrita do elemento base, uma vez que enfatiza apenas os atributos e as operações que serão usados ou melhorados pelo aspecto.

Uma interface transversal especifica:

- um conjunto de atributos e operações que o aspecto adiciona a uma classe base;
- um conjunto de assinaturas de métodos de classes que devem ser usados pelo aspecto;
- um conjunto de operações de aspecto que devem refinar métodos de classes.

`CrosscuttingInterface` é uma metaclasse concreta.

Associações

- *owner*

  O `Aspect` que possui a `CrosscuttingInterface`.

- *crosscuttingFeature*

  Uma lista ordenada de `CrosscuttingFeatures` que pertencem a `CrosscuttingInterface`.

– *feature* (herdado `Classifier`)

Uma lista ordenada de operações requeridas.

B.1.3.7 CrosscuttingFeature

Uma característica transversal especifica a estrutura ou o comportamento que melhora a estrutura ou o comportamento de instâncias base no contexto de um relacionamento de *crosscutting*.

As melhorias caracterizam-se como *adições*, *refinamentos* ou *redefinições*. As adições são melhorias que oferecem nova estrutura ou comportamento aos elementos base. Os refinamentos são melhorias que estendem o comportamento base. As redefinições são melhorias que sobrepõem o comportamento base. As características transversais são declaradas em uma interface transversal.

`CrosscuttingFeature` é uma metaclasse abstrata.

Atributos

– *extension*

Um valor do tipo `ExtensionKind`. Especifica se a característica transversal melhora a estrutura (adiciona) ou comportamento (refina ou redefine) da instância base.

Associações

– *owner*

Especifica uma `CrosscuttingInterface` que possui a característica.

B.1.3.8 StructuralCrosscuttingFeature

Uma característica transversal estrutural é uma característica transversal que especifica melhorias estruturais para instâncias base no contexto de um relacionamento de *crosscutting*. As melhorias estruturais caracterizam-se como *adições*.

`StructuralCrosscuttingFeature` é uma metaclasse abstrata.

Associações

– *feature*

Uma referência à característica estrutural.

B.1.3.9 BehavioralCrosscuttingFeature

A behavioral crosscutting feature is a crosscutting feature that specifies behavior to enhance the behavior of base instances in the context of a crosscutting relationship.

The enhancements are characterized as *additions*, *refinements* or *redefinitions*.

Behavioral crosscutting features are specified as parameterized elements, i.e., they contain one or more unbound parameters that represent operation name and signature. A base operation may be explicitly invoked inside the aspect operation's body using the pronoun *base*.

Uma característica transversal comportamental é aquela que especifica comportamentos para melhorar o comportamento de instâncias base no contexto de um relacionamento de *crosscutting*.

As melhorias caracterizam-se como adições, refinamentos ou redefinições. As características transversais comportamentais são especificadas como elementos parametrizados, isto é, elas contêm um ou mais parâmetros livres que representam a assinatura e o nome da operação. Uma operação base pode ser explicitamente invocada dentro do corpo da operação do aspecto usando o pronome *base*.

As características transversais comportamentais possuem colaborações aspectuais associadas que descrevem como o comportamento *crosscutting* e o comportamento base são compostos. As colaborações aspectuais são apresentadas na Seção B.3.

`BehavioralCrosscuttingFeature` é uma metaclasse concreta.

Atributos

– *where*

Valor do tipo *CrosscuttingKind* que define a estratégia de composição usada para combinar o comportamento crosscutting ao comportamento base (before, after, around).

Associações

– *feature*

Uma referência a `BehavioralFeature`. Especializa *feature* de `CrosscuttingFeature`.

– *compositionStrategy*

Uma `AspectualCollaboration` que realiza uma estratégia de composição usada para combinar o comportamento *crosscutting* ao comportamento base (before, after, around).

B.1.3.10 Standard Elements

A Tabela B.1 contém uma lista de elementos padrão predefinidos para o metamodelo `aSide`.

Elemento padrão	Elemento base	Tipo	Descrição
Precedence	Dependency	Stereotype	Precedência entre aspectos
Require	Dependency	Stereotype	Requirement entre aspectos
Xor	Crosscutting	Constraint	Exclusive-or entre aspectos

Tabela B.1: Elementos Padrão

B.2
O Pacote Data Types

B.2.1
Visão Geral

O pacote *Data Types* especifica os diferentes tipos de dados usados no metamodelo `aSide`.

B.2.2
Sintaxe Abstrata

A sintaxe abstrata do pacote *Data Types* é expressa na Figura B.5.

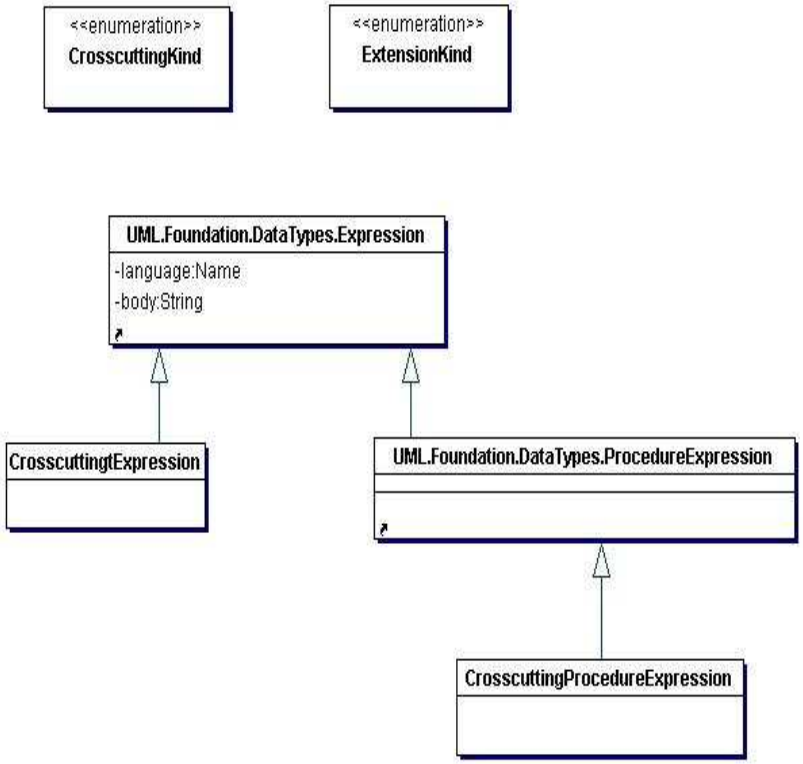

Figura B.5: Data Types.

B.2.3
Descrições de Classe

B.2.3.1 CrosscuttingKind

`CrosscuttingKind` define uma enumeração que denota como o elemento ao qual ele se refere será composto em um elemento base no espaço de nome. Seus valores são:

- undefined
- before
- after
- around

B.2.3.2 ExtensionKind

`ExtensionKind` define uma enumeração que denota como o elemento ao qual ele se refere estenderá o elemento base no espaço de nome. Seus valores são:

- undefined
- add
- refine
- redefine

B.3 O Pacote Behavioral Elements

B.3.1 Visão Geral

O pacote *Aspects.BehavioralElements* define construções de metamodelo concretas e abstratas para a criação de modelos comportamentais. As construções concretas especificadas nesse pacote incluem `AspectInstance`, `Aspectual Collaborations` e `Aspectual Interactions`.

B.3.2 Sintaxe Abstrata

A sintaxe abstrata para o pacote *Aspects.BehavioralElements* é expressa em notação gráfica nas Figuras B.6, B.7 e B.8.

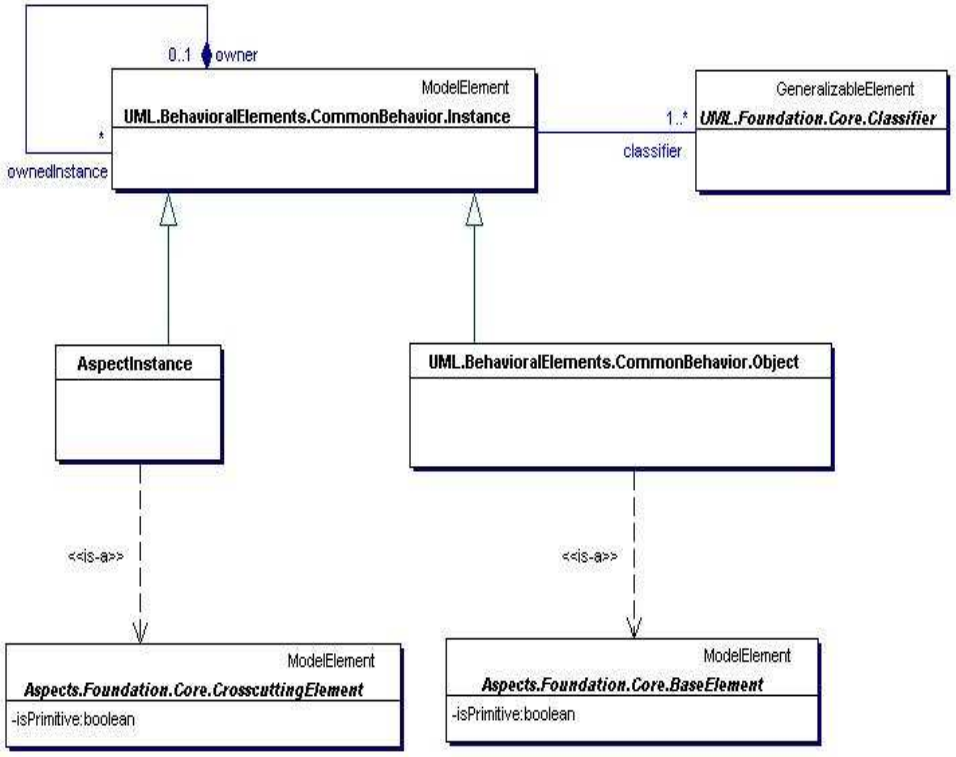

Figura B.6: Instâncias de aspecto.

B.3.3
Descrições de Classe

B.3.3.1
AspectInstance

Uma instância de aspecto é uma instância originada de um aspecto.

No metamodelo, uma `AspectInstance` é uma `Instance` (de UML) originada de exatamente um `Aspect`. `AspectInstance` é uma metaclasse concreta.

Associações

Não há associações adicionais.

Tempo de vida da instância de aspecto.

Uma instância de aspecto possui um tempo de vida restrito. Ela é implicitamente criada, é ativa ao longo do ciclo de vida do(s) elemento(s)

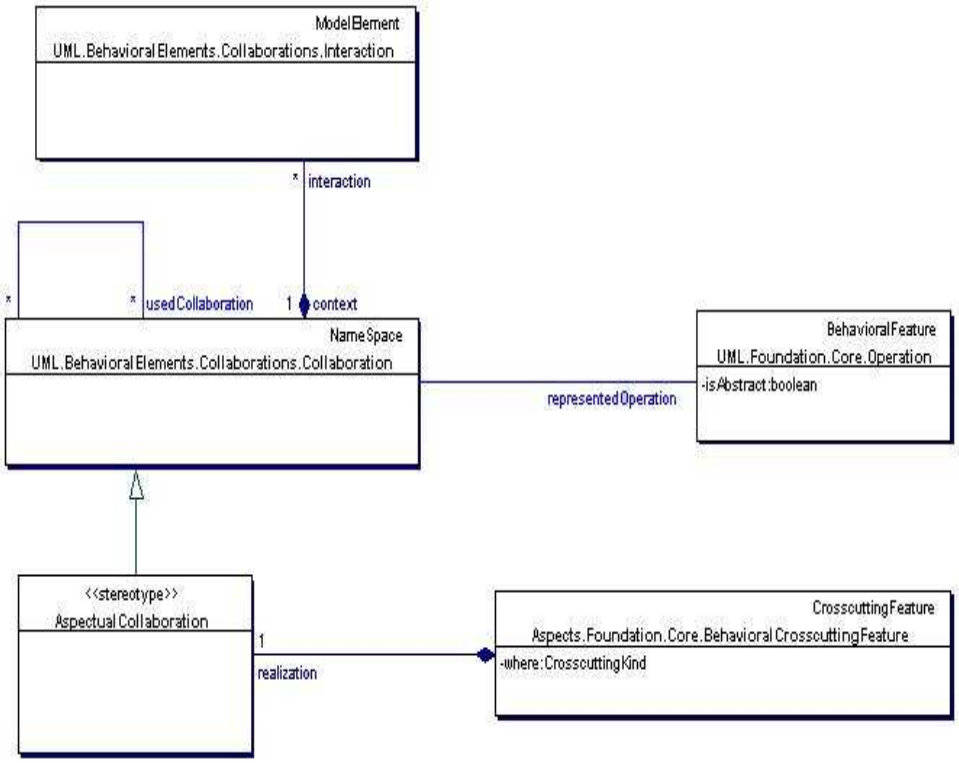

Figura B.7: Colaborações aspectuais.

base associado(s) e é implicitamente destruída. O ciclo de vida de uma instância de aspecto depende do escopo do aspecto:

- *Singleton aspects* provêem uma única instância de aspecto que, uma vez criada, é ativa e está disponível para cada objeto base no sistema;
- *Instance-level aspects* provêem uma instância de aspecto para cada objeto das classes base que afeta. A instância de aspecto é criada quando o objeto base é criado e é destruída quando o objeto base é descartado. A instância permanece ativa ao longo de todo o ciclo de vida do objeto base.

B.3.3.2 Aspectual Collaboration

Uma colaboração aspectual é uma descrição de uma organização geral de objetos e instâncias de aspectos que interagem dentro de um contexto a fim de implementar o comportamento *crosscutting* de uma característica transversal comportamental. Uma colaboração aspectual modela a realização de uma característica transversal comportamental.

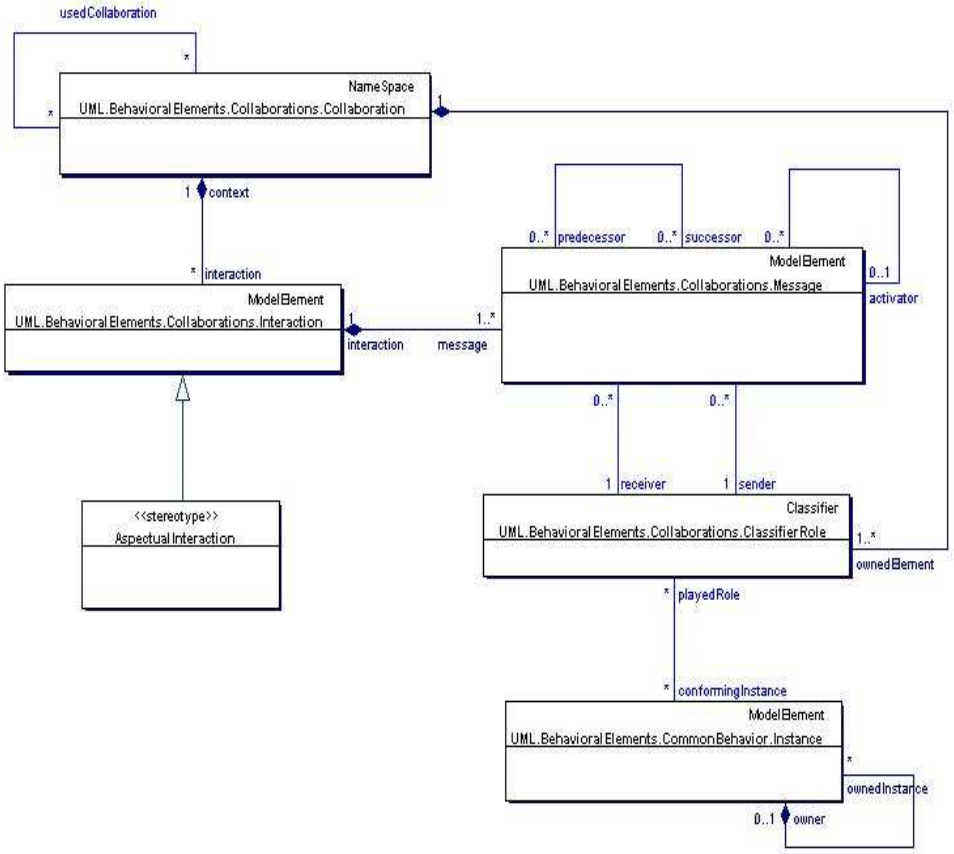

Figura B.8: Interações aspectuais.

A colaboração aspectual define um conjunto de papéis que devem ser exercidos pelas instâncias (objetos e instâncias de aspecto) e *links*, bem como um conjunto de interações que descreve a comunicação entre instâncias quando exercem os papéis definidos. Ela é uma colaboração que define pelo menos três papéis diferentes que devem ser exercidos: *Sender* (o emissor de uma mensagem), *Receiver* (o destinatário de uma mensagem) e *Aspect* (a instância de aspecto). Sender ou Receiver podem ser denotados pelo pronome *base*.

`AspectualCollaboration` é uma metaclasse. Ela estende `Collaboration` (de UML) e oferece algumas novas restrições.

Atributos

– *enhancesReceiver*

Especifica se o comportamento *crosscutting* melhora o Sender ou o Receiver. O valor padrão é *true*.

– *where*

Valor do tipo `CrosscuttingKind` que define a estratégia de composição usada para combinar o comportamento crosscutting ao comportamento base (before, after, around).

Associações

– *interaction* (herdada da Collaboration de UML)

O conjunto de Interações definido dentro da `AspectualCollaboration`.

– *representedOperation*

A operação *crosscutting* que é uma realização da `AspectualCollaboration`. Especializa *representedOperation* da Collaboration a fim de conter uma `BehavioralCrosscuttingFeature`.

B.3.3.3 Aspectual Interaction

Uma interação aspectual é uma especificação comportamental que incorpora uma seqüência de comunicações trocadas entre um conjunto de objetos e uma instância de aspecto a fim de conseguir a implementação de uma característica transversal comportamental que refina ou redefine o comportamento base. As interações aspectuais são definidas no contexto das colaborações aspectuais (Section 6.3). Elas se baseiam em quatro padrões de interação diferentes.

Um padrão de interação é uma interação parametrizada. Há quatro tipos de padrão de interação: *refine-before*, *refine-after*, *refine-around* e *redefine-around*. Mais precisamente, uma AspectualInteraction é uma interação que contém um conjunto de mensagens e pelo menos dois papéis: um para o elemento base e outro para a instância de aspecto.

`AspectualInteraction` é uma `Interaction` que contém um conjunto de `Message`s que especifica a comunicação entre um conjunto de `Instance`s em conformidade com os `ClassifierRole`s da `AspectualCollaboration`.

`AspectualInteraction` é uma metaclasse. Ela estende `Interaction` (de UML) e oferece algumas novas restrições.

B.4 O Modelo de Componentes

B.4.1 Visão Geral

O pacote *UML* define as construções de metamodelo concretas e abstratas básicas usadas como o modelo de componentes do metamodelo `aSide`. Os elementos de modelagem de UML são avaliados e alguns deles são selecionados como elementos base e também como pontos de combinação.

B.4.2 Sintaxe Abstrata

Neste trabalho, adotamos um subconjunto do metamodelo de UML como modelo de componentes, que inclui partes de seus pacotes `Core` e `Behavioral Elements`.

Para maiores detalhes, consultar o documento de especificação da versão 1.4 de UML [134], a mais recente disponível na época em que este trabalho foi realizado.

B.5 O Pacote Join Points

B.5.1 Visão Geral

O pacote *Join Points* define as construções de metamodelo concretas e abstratas básicas que incorporam o (sub)modelo de pontos de combinação do metamodelo `aSide`.

B.5.2 Sintaxe Abstrata

A sintaxe abstrata para o pacote *Join Points* é expressa na notação gráfica na Figura B.9.

B.5.3 Descrições de Classe

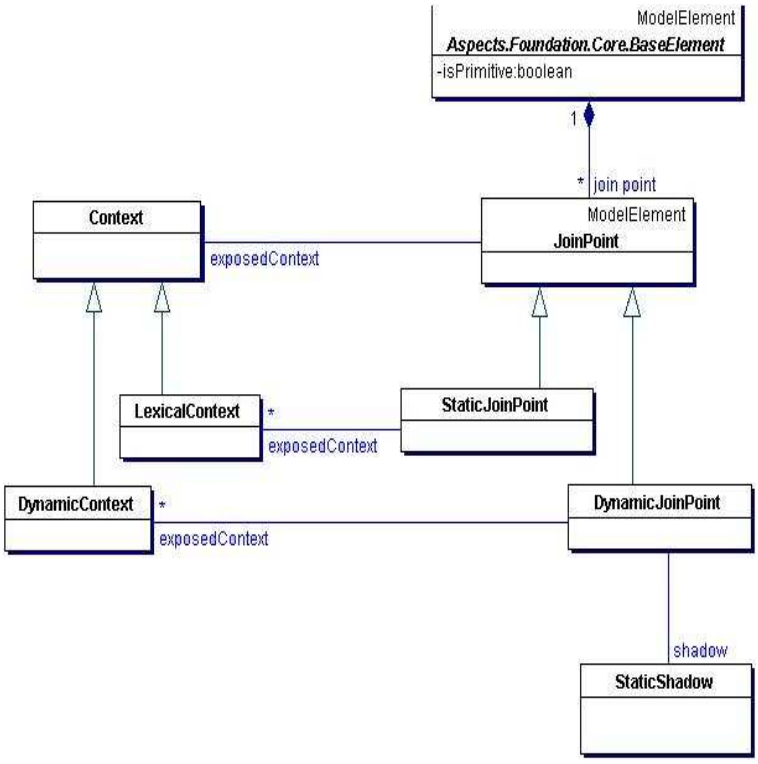

Figura B.9: Modelo de pontos de combinação.

B.5.3.1 JoinPoint

Um ponto de combinação é um local em um elemento base que pode ser melhorado por um elemento *crosscutting*. Os pontos de combinação estáticos são locais na estrutura de elementos enquanto os dinâmicos são locais relacionados à execução dinâmica de elementos.

`JoinPoint` é uma metaclasse abstrata.

Associações

- *name* (herdado de ModelElement)
- *exposedContext*

  Oferece o Contexto disponível no ponto de combinação. Deve ser especializado pelo tipo de ponto de combinação (estático ou dinâmico). kind of joint point (static or dynamic).

B.5.3.2 StaticJoinPoint

Um ponto de combinação estático é um local na estrutura de um elemento base que pode ser melhorado por um elemento de *crosscutting*. Ele pode expor o contexto léxico.

`StaticJoinPoint` é uma metaclasse abstrata.

Associações

- *name* (herdado de ModelElement)
- *exposedContext*

  Especializa *exposedContext* de `JoinPoint` para conter `LexicalContext`.

B.5.3.3 DynamicJoinPoint

Um ponto de combinação dinâmico é um local na execução de um elemento base que pode ser melhorado por um elemento de *crosscutting*.

Ele possui uma *sombra estática* (*static shadow*) correspondente no modelo de origem. Uma sombra estática é a região na visão estática que corresponde a um ponto de combinação dinâmico em uma visão de interação. Um ponto de combinação dinâmico pode expor o contexto dinâmico.

`DynamicJoinPoint` é uma metaclasse abstrata.

Associações

- *name* (herdado de ModelElement)
- *exposedContext* Especializa *exposedContext* de `JoinPoint` para conter `DynamicContext`.
- *shadow*

  Uma referência a `StaticShadow`.

B.5.3.4 StaticShadow

Um ponto de combinação dinâmico possui uma *sombra estática* correspondente na visão estática que é exibida no modelo comportamental. Uma sombra estática é a região na visão estática que corresponde a um ponto de combinação dinâmico em uma visão de interação.

Por exemplo, as execuções de método correspondem a corpos de métodos e são expressas por uma linha da vida de um objeto que é uma instância do classificador que possui o método. As chamadas de método correspondem a chamadas dentro de corpos de métodos que são expressas por setas que conectam o emissor de uma mensagem ao destinatário, rotuladasc com o nome do método.

Os efeitos do processo de combinação podem ser observados a partir das sombras estáticas. Uma sombra estática possui um tipo, uma assinatura e uma extensão em um modelo comportamental.

Tipo de PC	Assinatura	Contexto	Local (estrutura)	Local (comportamento)
chamada de método	nome do método	emissor, destinatário, parâmetros	chamada de método dentro de um corpo de método	setas do emissor de uma mensagem ao destinatário, rotuladas com o nome do método
executação de método	nome do método	destinatário, parâmetros	corpo de método	a ativação sobre a linha da vida

Tabela B.2: Sombras Estáticas.

B.6 O Pacote Weaving

O processo de combinação de modelos de aspectos e modelos de projeto orientados a objetos descritos nesta tese baseia-se na combinação de alguns elementos *crosscutting* e elementos base em um *elemento orientado a objetos combinado*.

O relacionamento de *crosscutting* relaciona aspectos e objetos. Ele oferece suporte a uma estratégia de combinação no nível do sistema. A estratégia de combinação é a estratégia usada para gerar elementos

combinados. Pode ser modificação local (*inPlace*) ou migração cliente (*clientMigration*). O valor padrão é *inPlace*.

B.6.1 Visão Geral

O pacote *Weaving* define as construções de metamodelo concretas e abstratas básicas usadas como o modelo de processo de combinação do metamodelo `aSide`.

As construções abstratas definidos no modelo de processo de combinação incluem *elementos combinados*. As construções concretas especificados no modelo de processo de combinação incluem *classes combinadas* e *colaborações combinadas*.

B.6.2 Sintaxe Abstrata

A sintaxe abstrata para o pacote *Weaving* aparece representada na Figura B.10.

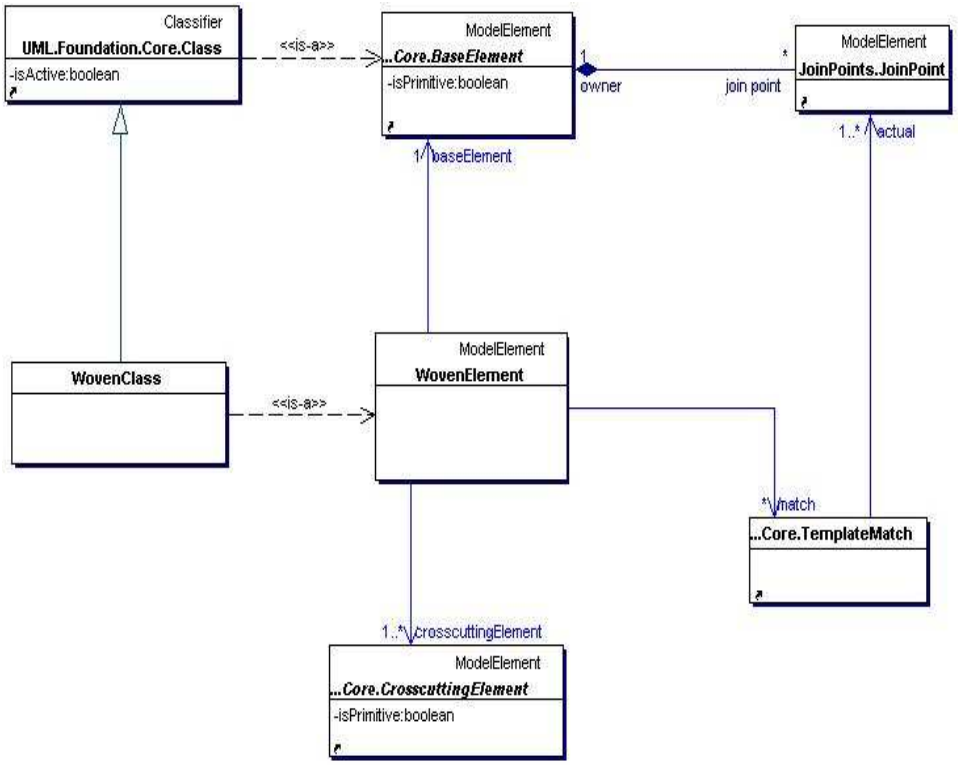

Figura B.10: Modelo de Weaving.

B.6.2.1 WovenElement

Um elemento combinado é um elemento gerado a partir de um elemento base e um ou mais elementos *crosscutting* relacionados pelos relacionamentos de *crosscutting*.

Associações

– *baseElement*

O `BaseElement` original.

– *crosscuttingElement* (associação derivada)

O conjunto de `CrosscuttingElement`s relacionado ao *baseElement* por Crosscutting.

– *match* (associação derivada)

O conjunto de `TemplateMatch`s obtido a partir dos relacionamentos `Crosscutting` que alcançam o *baseElement*.

B.6.2.2 WovenClass

Uma classe combinada (*woven class*) é um elemento gerado a partir de uma classe (elemento base) e um ou mais elementos *crosscutting* (aspectos) relacionados pelos relacionamentos de *crosscutting*.

Associações

– *baseElement*

O `BaseElement` original. Especializa *baseElement* a partir de `WovenElement` para conter uma referência a `Class`.

– *crosscuttingElement* (associação derivada)

O conjunto de `CrosscuttingElement`s relacionado ao baseElement por Crosscutting. Especializa *baseElement* a partir de `WovenElement` para conter uma referência a `Aspect`.

– *match* (associação herdada)

O conjunto de `TemplateMatch`s obtido a partir dos relacionamentos `Crosscutting` que alcançam o baseElement.